А.С.Кронрод

БЕСЕДЫ О ПРОГРАММИРОВАНИИ

А.С.Кронрод

БЕСЕДЫ О ПРОГРАММИРОВАНИИ

Предисловие Л.А.Кронрод
Послесловие В.Л.Арлазарова

Издание второе, стереотипное

УРСС
Москва • 2004

ББК 22.18, 32.973-018

Кронрод Александр Семенович

Беседы о программировании / Предисл. Л. А. Кронрод. Послесл. В. Л. Арлазарова. Изд. 2-е, стереотипное. — М.: Едиториал УРСС, 2004. — 248 с.

ISBN 5–354–00565–5

Эта книга была написана замечательным ученым-математиком А. С. Кронродом почти 40 лет тому назад, но публикуется впервые. Один из зачинателей программирования у нас в стране и основателей целой школы программирования, А. С. Кронрод в доступной и интересной форме обсуждает вопросы организации системного программирования, отладки программ и архитектуры ЭВМ. Значительная часть книги посвящена различным задачам искусственного интеллекта и использованию «интеллектуальных» подходов для решения вычислительных задач.

Книга будет интересна программистам, историкам науки и специалистам в области искусственного интеллекта.

Издательство «Едиториал УРСС». 117312, г. Москва, пр-т 60-летия Октября, 9.
Лицензия ИД № 05175 от 25.06.2001 г. Подписано к печати 03.11.2003 г.
Формат 60×84/16. Тираж 400 экз. Печ. л. 15,5. Зак. № 2-1129/339.

Отпечатано в типографии ООО «Рохос». 117312, г. Москва, пр-т 60-летия Октября, 9.

ИЗДАТЕЛЬСТВО УРСС
НАУЧНОЙ И УЧЕБНОЙ ЛИТЕРАТУРЫ
E-mail: URSS@URSS.ru
Каталог изданий
в *Internet:* http://URSS.ru
Тел./факс: 7 (095) 135–44–23
Тел./факс: 7 (095) 135–42–46

ISBN 5–354–00565–5

Кронрод Л. А.

Предисловие

«Беседы о программировании» — это книга скорее не о том, как надо программировать, а о том, как развивалось программирование в нашей стране.

Если взглянуть на «Беседы» с современной точки зрения, то их, очевидно, следует отнести к разделу: «История развития программирования». Дело в том, что когда у нас стали появляться ЭВМ, многие не поняли, какое место в жизни людей займут компьютеры. А те, кто понял, разделились на две части: идти своим путем или копировать заграницу. Между этими группами людей шла ожесточенная борьба. Она-то и отображена в «Беседах о программировании». Скорее даже — «Беседы» являются элементом этой борьбы.

За кадрами книги остались бесконечные обращения автора «Бесед» в высокие инстанции — буквально крики души о том, что надо у НАС развивать строительство своих машин, надо привлекать сильных конструкторов и математиков — а они у нас есть — к разработке новых компьютеров. Было продемонстрировано не на словах, а на деле, что мы это можем, что мы сильнее. Например, наша машина М-20 обыграла в шахматы со счетом 3 : 1 машину из Стэнфорда.

Но победили те, которые решили копировать заграницу. Теперь все компьютеры иностранные, все «говорят» по-английски. И даже теорему Адельсон-Вельского и Ландиса, которую у нас коротко называют АВЛ, теперь читают АБЛ.

Но в «Беседах о программировании» рассказано не только о борьбе разных направлений. И даже это не главное. Мы упомянули об этой борьбе, пожалуй, только для того, чтобы читателю был понятен тон или стиль, которым написана книга.

На наш взгляд, главное в «Беседах» — это задачи, о которых говорил А. С. Кронрод. Книга была написана в 1963 году. Теперь уже видно, по какому пути пошло программирование — по предложенному Кронродом или нет. А называл А. С. эти задачи — ИГРАМИ.

«Беседы о программировании» были посланы на рецензии разным людям. Рецензии было бы тоже интересно прочитать. Сторонники А. С., естественно, хвалили эту книгу. Противники — полемизировали. Но надо сказать, что все, даже разгромные, рецензии кончались словами: «Тем не менее, книгу надо напечатать!»

«Беседы о программировании» прошли обязательную тогда цензуру, и было напечатано 10 сигнальных экземпляров. И вдруг книга была запрещена. Кем? Математиками! Как говорил Кронрод «Иногда такие люди занимали должности даже и довольно высокие». Похоже было, что они не выдержали полемики, поднятой в этой книге. Но они не запрещали всю книгу: кто-то требовал уничтожить какой-нибудь параграф в «Беседе», кто-то — написать слово не с большой, а с маленькой буквы, и т. п.

Кронрод отказался им подчиниться. Книга издана не была.

Теперь — по прошествии более 30 лет с момента написания этой книги — многое будет в ней непонятно. Чтобы избежать этого, можно было бы в конце «Бесед» к каждой странице дать пояснения. Но, скорее всего, можно обойтись и без них. Просто: «Имеющий уши слышать — да слышит».

Теперь можно переходить к чтению «Бесед о программировании» Александра Семеновича Кронрода.

ОТ АВТОРА

Эти беседы можно назвать застольными. Если, конечно, стол письменный, а собеседник-программист — профессионал. В остальных случаях читателю рекомендуется ограничиться оглавлением и первой беседой.

Беседа первая

Как А. Л. Брудно придумал программирование в содержательных обозначениях

А ищи всех паче
Разума в задаче.

Магницкий

§ 1. Что было сперва

Сперва Александр Львович Брудно поступил на работу в Институт электронных управляющих машин. Было это в 1954 году. К тому времени в институте построили отличную трехадресную машину М-2. Строил М-2 талантливый инженер М. А. Карцев. Он придумал очень удобную систему команд. Вероятно, система команд чем-то похожа на стихотворение — ее сочинение не терпит соавторства. Может быть именно поэтому система команд М-2 получилась такой цельной — не сборная солянка, а поэма в кодах.

По нынешним временам машина М-2 показалась бы очень убогой — всего 512 ячеек электронной и 512 барабанной памяти (со сквозной нумерацией). Но в 1954 году такая машина считалась отличной: у нее была плавающая запятая, а скорость доходила до 2000 действий в секунду.

У Брудно не было ни помощников-математиков, ни лаборантов. Все приходилось делать самому. Немировича-Данченко тоже не случилось под рукой. Поэтому никакой исторической встречи, хотя бы в соседней столовой, не произошло, а просто Брудно начал составлять программы и решать на машине задачи.

Обстоятельства сложились благоприятно — Брудно забыл к этому времени в равной степени и английский, и немецкий языки (французским он и раньше не владел), на русском прочесть про програм-

мирование было нечего, и поэтому он просто не знал, как, например, программисту полагается обозначать сложение. В простоте душевной он писал так:

$$a + b = c.$$

Когда случалось две величины перемножить, Брудно по необразованности не додумался до обозначения

005105135775516,

а написал

$$a \times b = c.$$

Когда дело дошло до пересылки чисел с передачей управления, вместо научного

056200566060121

появилось

$$a \rightarrow b;\ c,$$

а то и так:

$$a \rightarrow b \urcorner\!\downarrow$$

А команда

577020233445670

выглядела и вовсе наивно:

стоп.

Поскольку Брудно все делал сам, он сперва, естественно, писал всякий раз программу в содержательном виде, т. е. примерно так:

$$a + b = c,$$
$$c \times c = c^2,$$
$$c^2 + 3 = R,$$
$$\ldots\ldots\ldots,$$

а уже потом переводил ее на язык машины (кодировал).

Так родилось

ПРОГРАММИРОВАНИЕ В СОДЕРЖАТЕЛЬНЫХ ОБОЗНАЧЕНИЯХ.

Перед каждой командой в кодированной форме Брудно писал еще ее адрес. Получилось так:

					A	
	$0+0=S$	1003	001	0	0	2570
	$1 \to n$	1004	056	0101	2571	1006
	$n+1=n$	1005	001	2571	0101	2571
	$n \times n = R1$	1006	005	2571	2571	0011
	$1 : R1 = R2$	1007	004	0101	0011	0012
	$S+R2=S$	1010	001	2570	0012	2570
	$n<10$	1011	036	2571	0112	1005
	Стоп	1012	017	0	0	0
	1	0101	101	4000	0	0
	10	0112	104	5000	0	0

Теперь, чтобы запомнить, какую ячейку отвели данной букве, и чтобы ячейки не налезали друг на друга, понадобилась шпаргалка (роспись памяти).

Она выглядела так:

1260	1261	1252	1263	1254
1300 *a*	1301 *b*	1302 *c*	1303 17,5	1304
1320 Δ	1321 *R1*	1322 *R2*	1323	1324
1340	1341	1342	1343	1344
1360	1361	1362	1363	1364
1400	1401	1402	1403	1404
1420	1421	1422	1423	1424

Блок подготовки

Если у человека есть подчиненные, он всегда твердо помнит, что разделение труда между ними ускоряет работу. Почему-то значительно реже вспоминается другая избитая истина — о роли личного труда во всяком деле. Брудно работал один. Чтобы меньше приходилось помнить, он стал обозначать ячейку-аргумент всегда одной и той же буквой α и закрепил за ней номер в памяти. Теперь уже всякому, кто хоть немного знаком с греческим, легко было додуматься до того, чтобы ячейку-ответ тоже закрепить и назвать β.

Дальше Брудно заметил, что, хотя у каждой подпрограммы есть свое отдельное начало, конец (который тоже нужно помнить) можно у всех подпрограмм сделать общим. После некоторых колебаний было решено [1] называть его Ω.

Теперь предстояло написать библиотеку подпрограмм. Эта первая библиотека А. Л. Брудно содержала, как, впрочем, и все библиотеки в мире, программы вычисления элементарных функций, перевода из 10 в 2 и из 2 в 10 с печатью. Кроме того, в ней, естественно, были константы. Для обозначения констант Брудно придумал специальную систему:

Наименование константы	Условное обозначение по системе А. Л. Брудно
1	1
2	2
π	π
1/4	1/4
0	0
...	...

Ничего лучшего ему в голову не пришло.

Для обозначения единицы левого, правого и среднего адресов появилось:

$$1,0,0 \qquad 0,0,1 \qquad 0,1,0$$

Создав свою систему программирования, А. Л. Брудно не заметил этого, а просто стал решать на машине разные задачи. Нужно сказать, что в те идиллические времена ввод большой программы на М-2 занимал минут 20. Поэтому, если программа сбивалась целиком, приходилось снова в течение 20 минут вводить ленточку. Может быть, при наличии штата подчиненных это обстоятельство осталось бы

[1] Не забывать, что Брудно работал один!

вещью в себе. Но когда человек работает сам, ему приходится (если он не аскет и не подвижник) думать о собственных удобствах.

Ход мысли был примерно таков: если бы программу можно было начать всегда сначала, не приходилось бы всякий раз заново вводить ленточку. Так появилась

САМОВОССТАНАВЛИВАЮЩАЯСЯ ПРОГРАММА.

Это значит: программа, которую можно остановить в любом месте, а затем, *ничего не поправляя*, начать сначала.

Вот и все, что было сперва.

§ 2. Что было потом

Потом на огонек на Большую Калужскую 18 стали собираться некоторые математики. Это были — А. Г. Пантелеев, Г. М. Адельсон-Вельский, Е. М. Ландис, автор этих строк, а затем и другие, в их числе В. Л. Арлазаров и М. М. Бонгард.

Читать математическую литературу из нас никто, кроме Е. М. Ландиса, не любил. Работать на машине нам, наоборот, всем нравилось. Брудно выучил нас программировать, как умел сам, и никто (!) не заметил всей антинаучной направленности обозначения деления:

$$a : b = c. \tag{1}$$

В это время на М-2 стали ходить студенты из МГУ. Будучи более просвещенными, они писали под руководством своих преподавателей и профессоров вместо (1)

$$\bar{3}05\bar{2}73\bar{1}0016. \tag{2}$$

Формы (1) и (2) мирно соседствовали, и каждая сторона воспринимала чужую систему как милое чудачество. Тогда еще никто не знал, что форма (2) породит АЛГОЛ.

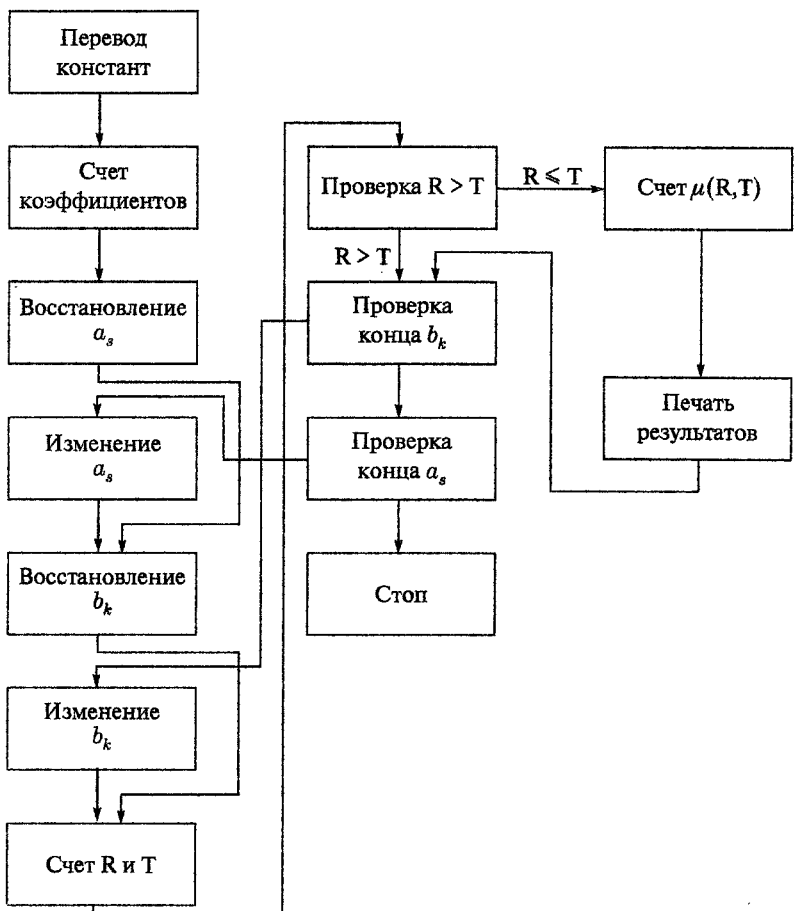
Между тем на машине считалось все больше и больше задач. Полностью разделилась работа по *написанию программы* (т. е. левой ее части в содержательных обозначениях), по кодировке и по набивке ленты (появились кодировщицы и перфораторщицы). От этого произошли такие два последствия:

1°. Пришлось выработать твердые обозначения для левых частей программы, чтобы ее мог кодировать другой человек.

2°. Стандартизовались приемы и порядок написания часто повторяющихся частей программы. Теперь математик действовал в шаблонных задачах по шаблону. Скорость работы возросла, а ошибки стали

случаться реже. Кроме того, такую программу легче читать другому человеку. Даже и сам автор через некоторое время (месяц, год) с трудом разбирает свою программу, если она написана непривычным способом.

Довольно скоро все заметили, что удержать в голове схему достаточно сложной программы трудно. Поэтому стали рисовать блок-схему программы. Выглядело это примерно так:



Но программы становились все сложнее, а блок-схемы все больше и непонятнее.

Разобраться во всех стрелках и линиях было уже почти невозможно. Однако мучились и разбирались. И вот, наконец, сразу две группы запутались окончательно. Это случилось у Вайнштейна и Доброчаевой (программа Гигра—Тигра) и у Адельсона-Вельского, Ландиса и меня, когда мы программировали игру в подкидного дурака. Последней надеждой были цветные карандаши. Но когда кончились цвета, мы пришли к Брудно. И он сказал:

— Не рисуйте больше блок-схем.

— Пишите блок-программы.

И мы стали писать блок-программы.

Беседа вторая

Что такое блочное программирование, которое велел нам придумать Брудно

А ищи всех паче
Разума в задаче.

Магницкий

§ 1. Структура больших программ

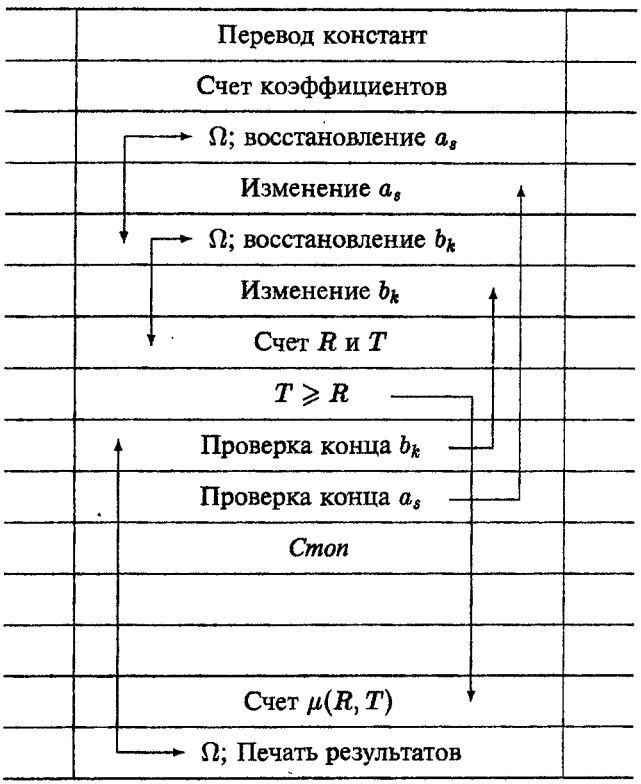
Только очень упрямый (глупый) человек выписывает все команды в сколько-нибудь серьезной программе. Например потому, что считает машина в двоичной системе, а печатать числа хочется в десятичной. И программу для перевода чисел из 2 в 10 не пишут каждый раз заново, а пользуются стандартной подпрограммой. Конечно, так же поступают и с программой e^{α}, $\ln \alpha$ и т. д.

Заметьте, что всякий раз мы теряем на этом одну-две ячейки (обращение к подпрограмме, возврат).

Если некоторый кусок программы, например выбор максимального из 20 чисел, повторяется много раз, его без особых душевных мук тоже выделяют в подпрограмму. А вот стоит ли выделять в качестве подпрограммы такой кусок, если он встречается в программе *один* раз? От того, как мы ответим на этот вопрос, зависит структура нашей программы.

Блочная программа — это такая программа, где каждая самостоятельная ее часть выделена в отдельную подпрограмму. И чем мельче куски, выделенные в подпрограммы, тем более блочной является собранная из них программа.

Для того чтобы из этих кусков-блоков сделалась сама программа, нужно, конечно, написать еще «собирающую» — программу, которая по очереди будет передавать управление нужным блокам. Собирающая — она же блок-программа — получается такой:



Сравните эту программу с блок-схемой на с. 13. Между ними только одна разница: блок-схема состоит из нарисованных прямоугольников, а блок-программа из команд. То есть ее можно закодировать и ввести в машину. И следить за работой программы «в целом», имея перед глазами только один этот листок.

Вот что заставил нас сделать А. Л. Брудно.

Понятно, что сложный блок сам должен разбиваться на подблоки. А значит, он, в свою очередь, управляется блок-программой областного ранга. Дальше его подблоки тоже могут оказаться сложными. Такой подблок мы разобьем на подподблоки. И напишем блок-про-

грамму районного значения. Только, чтобы не ломать язык, будем все подпод...подблоки называть просто блоками.

От этого еще никто ни разу не запутался.

§ 2. Что мы на этом потеряли?

Человек — ничего. Ангел-программист [1] — время на лишние передачи управления и возвраты. И — ячейки памяти, содержащие команды блок-программы. Но даже и ангел потерял бы времени относительно мало. А свободные ячейки памяти, хотя бы и в большом массиве, сами по себе все равно не приносят процентов.

§ 3. Что мы при этом выиграли?

Ангел-программист — ничего. Человек — время. То, которое он тратит на поиск ошибок. И то, которое идет на исправление этих ошибок.

Блочная программа позволяет относительно легко заменять куски программы — нам незачем писать программу слитно. Более того, если программа легко влезает в память, блоки стоит сразу кодировать с некоторым зазором — неровен час, придется исправлять блок и он удлинится.

Блочную программу очень удобно отлаживать. Прежде всего можно пройтись по Главной собиралке, ставя стоп по адресу при возврате из очередного блока. Если программе суждено вылететь на Авост [2] или зациклиться, мы сразу увидим, в каком блоке это случилось. Пусть нам не повезло, и этот плохой блок сам является сложным. Что ж, начнем сначала, поставив на сей раз стоп по адресу по входу в блок. А теперь перейдем к подчиненной блок-программе самого этого блока и пойдем по ней так же, как шли по главной собиралке. Переход к следующим рангам подчинения мы оставляем читателю в качестве упражнения.

Кроме облегчения поиска ошибок, блочная программа облегчает проверку. Благодаря четкому выделению блоков, нам легче составить контрольные задачи для проверки работы каждого из них.

И, наконец, блочная программа легче эволюционирует. А ангелы-заказчики, к сожалению, почти столь же редки, как и ангелы-

---

[1] *Определение* 1. *Ангелом* мы будем называть программиста, не делающего ошибок.

Определение 2. Программиста, не являющегося ангелом, мы будем называть *просто программистом*, или *человеком*.

[2] *Авост* — аварийная остановка. Случается при невозможности выполнить указанное в команде действие (деление на нуль; переполнение при сложении, умножении и т. д.).

программисты. От этого в реальной жизни программы приходится переделывать. Поскольку блоки более или менее развязаны, переделки эти происходят куда как легче и быстрее. Как и в биологии, побеждает вид, более склонный к эволюции. Точнее — выживает не сама программа, а ее модификация. Опять-таки, как и в биологии...

§ 4. Как обращаться к блокам. Зачем нужна команда Ω = КОНЕЦ

Блок-программа обращается к блокам ровно так же, как вообще обращаются к подпрограммам. Для трехадресной машины это обращение выглядит примерно так:

$$u \leftrightarrow \Omega; \quad \text{Счет коэффициентов,}$$

т. е. заслать в Ω команду безусловной передачи к ячейке u и уйти на блок счета коэффициентов.

Поскольку возврат чаще всего происходит к ячейке $Я + 1$, т. е. следующей за той, где лежит команда обращения, стоит условиться вместо

$$(Я + 1) \leftrightarrow \Omega; \quad \text{Счет коэффициентов}$$

писать короче:

Счет коэффициентов.

Кодировщицы без всякого напряжения будут кодировать такую запись. А математик избавится от лишней работы (и лишних описок), во-первых, и будет легче читать программу, во-вторых.

При обращении к блоку соблазнительно, на первый взгляд, сэкономить ячейки. Такой соблазн проявляется двояко.

Грубое падение состоит в том, чтобы в конце блока, к которому обращаются *однократно*, поставить команду безусловной передачи управления в должное место блок-программы. Ангелу это сойдет с крыльев, а человеку стоит задуматься, что случится, если придется изменить блок-программу.

Более тонкий соблазн таков:

$(Я + 1) \leftrightarrow$ конец счета коэффициентов; Счет коэффициентов.

Это сходит легче уже и человеку. Но изменение блока повлечет и изменение всех команд обращения к нему. Это не радостно — такие команды могут встречаться и там, где мы о них уже забыли.

Поэтому следует жестко придерживаться обращения к блокам с возвратом через стандартный конец Ω. А тогда каждый блок нужно начи-

нать командой

$$\Omega \rightarrow \text{конец блока;} \quad (Я + 1)$$

и писать ее так:

$$\Omega = \text{конец.}$$

Команда эта, помещенная в начале каждого блока, сама по себе сильно способствует программированию: известно, с чего начать работу.

Беседа третья

Кое-что об отладке программ

А ищи всех паче
Разума в задаче.
Магницкий

§ 1. Зачем программисту барабан?

Очень редко затем, зачем написано в книгах. Если прочесть 2–3 книги по программированию, то станет ясна примерная длина средней программы: 4096 слов в ячейках оперативной памяти плюс 3–4 магнитных барабана, по 5000 слов каждый, плюс 1–2 магнитные ленты по 100 000 слов. Если всего этого не хватит, то, как известно из любой книги, следует очередные куски программы вводить с перфокарт на временно не нужные куски старой программы.

И действительно, такие программы-гиганты существуют. Один из моих друзей — человек совершенно надежный — слышал от своего товарища-профессионала, что тот лично знаком с группой программистов из Новосибирска, которая почти закончила такого типа программу. Так что все это правда.

К счастью, повседневные трудности обычно все-таки гораздо меньше. И рядовая программа свободно умещается в ферритной памяти. И даже остается свободное место.

А барабан нужен совсем для другого дела.

Когда в программе случается ошибка, она может привести до своего обнаружения к удивительно ехидной порче всей программы. Самая мелкая неприятность — это, например, стирание 90 % программы. Такая гадость обнаруживается сразу, и программу вводят заново. Гораздо

хуже обстоит дело, если ошибка привела к тому, что, скажем, чуть-чуть подправились 2–3 слова в различных местах — на 1 изменились два эталона окончания цикла и в одном из разветвлений сменились условия передачи управления.

Теперь, хотя мы и исправили найденную ошибку, ее следы остались в экземпляре программы оперативной памяти. Мы продолжаем отладку, но я даже затрудняюсь сформулировать, какую именно программу мы в этот момент отлаживаем. Это — не та программа, которая была до исправления ошибки. И не та, которая случится завтра, когда мы введем программу с исправлением. В этот момент программа может оказаться в состоянии практически неповторимом.

Чтобы освободиться от такого сорта раздумий, и пользуются магнитным барабаном. Делается это так:

1. После ввода программы в машину *до всего последующего* всю оперативную память целиком перематывают на магнитный барабан. Это, конечно, делается специальной программой, которую математики называют ТУДА. Программа ТУДА сопровождается контролем перемотки и циклическим суммированием слов в оперативной памяти. Полученная контрольная (циклическая) сумма понадобится нам при следующих выходах на машину.

2. Теперь начнем отладку. Если программа собьется, то мы можем, не вводя заново карт, вернуть ее к первоначальному состоянию. Для этого нужна программа ОБРАТНО.

3. Когда нашлась ошибка, естественно поступить так:

A) Дать ОБРАТНО.

B) Исправить ошибку.

C) Выполнить ТУДА.

Новую контрольную сумму нужно обязательно зафиксировать. Это гарантирует нас не от того, конечно, что мы забудем в картах исправить ошибку, а хотя бы от того, что наша забывчивость останется незамеченной.

Е. М. Ландис указал, что пункты A)–C) при исправлении ошибки можно удобно объединить в программу ПОПРАВКА С ПУЛЬТА.

Получается такая инструкция.

Чтобы исправить ошибку, надо:

1. Набрать во II адресе ДЗУ-1 адрес поправляемого слова*.

* ДЗУ (долговременные запоминающие устройства) — *клавишные* регистры, заменяющие ячейки памяти. Изменение их содержимого может производиться без остановки машины.

2. Набрать в ДЗУ-3 правильное слово.
3. Уйти на ПОПРАВКУ С ПУЛЬТА.

ПОПРАВКА С ПУЛЬТА работает так:

A) Происходит ОБРАТНО.

B) В экземпляр программы в оперативной памяти вносится поправка.

C) Происходит СТОП. В регистрах горит:

1. Адрес поправляемого слова.
2. Что туда теперь записано.

Практика показала, что человек еще гораздо дальше от ангела, чем он сам думает. Чуть ли не в *четверти* случаев при поправке он ошибается — производит поправку не по задуманному адресу или не тем словом. Самая частая неприятность: поправляя, мы набираем *не новое* слово, а именно *старое*, с ошибкой. Поэтому обязательно не просто полюбоваться на огни регистров, а вслух прочесть и адрес, и поправленное слово. Если что-нибудь вышло не так, то мы еще не успели испортить программу на барабане. Значит, можно поправиться на ДЗУ-3 и снова начать ПОПРАВКУ С ПУЛЬТА. Теперь пусть пункт C) благополучно закончен. Нажимаем ПУСК.

Происходит печать исправленного слова в такой форме:

```
0705
 056  2005
3751  2007
```

и управление уходит на программу ТУДА, на барабан переносится поправленная программа и фиксируется новая контрольная сумма.

Теперь мы уже полностью готовы к отысканию и исправлению очередной ошибки.

Нужно сказать, что введение такой системы работы, вернее, *запрет делать иначе* привел практически к почти 10-кратному выигрышу во времени отладки.

Вот для чего в первую очередь нужен программисту барабан (конечно, магнитный).

Задачи

1. Зачем программисту 2 барабана?
2. Зачем программисту 3 барабана?
3. Зачем программисту 4 барабана?

§ 2. О росписи памяти и о контрольном суммировании

Сперва о контрольном суммировании. Часто проводят контрольное суммирование на входном устройстве. При таком контроле остается незамеченным:

a) Что забыли ввести нужные части библиотеки.

b) Что вообще отключена ферритная память.

c) Что ферритная память хоть и включена, но испорчена.

Кроме того, последняя карта остается во входном устройстве; от этого лишних удобств не возникает.

Поэтому нужно проводить контрольное суммирование *циклическим сложением всей оперативной памяти* (кроме нескольких первых рабочих ячеек). Удобно иметь специальную ячейку КΣ для контрольной суммы (например, с адресом 7767 для машины М-20). Если полученная контрольная сумма совпала с записанной в КΣ, то программа ТУДА заканчивается стопом, а на регистрах горит

0, *F*, 0, т. е. 000 0000 7777 0000

0, *F*, 0, т. е. 000 0000 7777 0000

Сигнал 0, *F*, 0 значит «хорошо». Если же накопленная сумма не совпала с КΣ, то зажжется:

новая сумма

старое содержимое КΣ

и будет стоп. При нажиме пуска новая сумма перенесется в КΣ и программа ТУДА выйдет на 0, *F*, 0.

Естественно, новая КΣ пропечатывается, и притом в уже знакомой нам адресной форме, например:

```
+++0  7767
+++0   205  3771
+++0  1664  4550
```

Карточку с КΣ нужно обязательно приложить к программе (на следующий раз).

Контрольное суммирование предполагает стандартное состояние памяти перед вводом программы. Каким оно должно быть?

В некоторых машинах специальная кнопка расписывает оперативную память пустым словом. Как число это слово — нуль, а как команда — передача управления дальше. Редкий случай осуществленного в металле памятника глупости.

В самом деле, чего желать от ячеек, куда мы сами ничего не положили? Часто встречающаяся ошибка — использование посторонней ячейки. Если ей передано управление, надо, чтобы, по возможности, произошел немедленный стоп. Итак, стандартное слово в ячейках до ввода программы должно, как команда, означать СТОП. Далее, если мы пытаемся плавающим образом сложить (умножить и т. д.) содержимое такой ячейки с чем-нибудь, самым желательным исходом является немедленная аварийная остановка (Авост). Но это значит, что стандартное слово росписи не имеет права получаться в результате никаких плавающих действий. Подходящим для этого словом является *минус нуль*, т. е., например, для М-20, это

200 0000 0000 0000

Отсюда возникает важное следствие: *код команды* СТОП *не является произвольным*. Фактически для этого нужно кое-что переделать в машине. Что поделаешь, за глупость приходится платить. Обидно только, что за чужую.

Задачи

4. Что делать, если инженеры не умеют переделать машину? Чем тогда расписывать память?

§ 3. Программы для работы с пульта

Три таких программы мы уже видели: ТУДА, ОБРАТНО и ПОПРАВКА С ПУЛЬТА. Что нужно еще?

Во-первых, печать массива ячеек — ПЕЧАТЬ ЧИСЕЛ С ПУЛЬТА (ПЧП); информацию, с какой по какую ячейки надо напечатать, можно разместить, например, в ДЗУ-1 так:

0, 0, ОТ, ДО

Только если ОТ > ДО, то нужно позаботиться, чтобы в этом случае печаталось единственное число с адресом «ОТ». Жалко, конечно, не бумагу, а время.

Во-вторых, печать массива команд с пульта — ПЕЧАТЬ ПРОГРАММЫ С ПУЛЬТА (ППП). Информация та же, что и для чисел.

Печать должна иметь вид:

Числа		Команды		
+++0	2250	+++0	6225	
+++02	600000000	+++05	075	0000
++−03	107529056	+++0	0000	0724
++−04	965519773	+++06	275	0000
+−+01	245460012	+++0	0631	0141
+−−02	331751180	+++07	436	0501
+++0		+++0	6254	0140
+++0	2255	+++0		
+++02	650000000	+++00	6230	
++−03	108924522	+++0	452	0000
++−04	963004707	+++0	0044	6252
+−+01	245459836	+++01	414	0050
+−−02	330965438	+++0	0141	0015

Нужно обязательно позаботиться, чтобы, скажем, не более чем через 8 ячеек получались интервалы — иначе читать трудно. По той же причине перед каждой группой должен печататься адрес первого слова в группе. Подробно об этом смотри, например, в [1].

Последняя пультовая программа — ВВОД ЧИСЕЛ С ПУЛЬТА (10сП). Эта программа берет число в 10-форме с ДЗУ-3, переводит в 2-форму и кладет в ячейку с адресом, указанным в ДЗУ-1 в среднем адресе. Кроме того, она должна напечатать на табулограмме, куда и какое число мы занесли. На барабан поправленное число переписывать нет нужды, зато зажечь после стопа на огнях адрес поправленной ячейки и занесенное туда число в 10-форме полезно — меньше будет ошибок.

§ 4. Какими должны быть программы печати

Прежде всего — *работающими по требованию*, т. е. включаемыми и отключаемыми с пульта одним нажатием заданной клавиши ДЗУ. Такая программа будет обладать по сравнению с неотключаемой колоссальным преимуществом при отладке (ср. радио и телевизор). Информация о номере ДЗУ и номере разряда должна задаваться прямо рядом с указанием массива. Например, строка информации может быть такой:

453, . . . , 2005, 2076

т. е. перевести из 2 в 10 и напечатать числа с 2005 по 2076-ю ячейку. Нажим $53_8 = 43_{10}$ разряда ДЗУ-4 включает печать. В I адресе можно поместить указание о том, как часто делать интервалы. Заодно для машин с буфером можно указывать, делать ли печать немедленной (с *салютом*) или копить слова на буфере (*без салюта*). Скажем,

$$453, 4012, 2005, 2076\ ^{1)}$$

означает: интервал через $12_8 = 10_{10}$ слов, печать немедленная, а

$$453, 0011, 2005, 2076$$

будет значить: интервал через $11_8 = 9_{10}$ слов, копить на буфере без печати.

Естественно, надо иметь возможность и для такой печати: «Печатать, начиная от 2005, столько слов, сколько указано в среднем адресе ячейки N, названной в III адресе строки информации». Подробней об этом см. [1].

Для печати команд должны иметься те же возможности.

§ 5. Об организации контрольных просчетов

Как ни странно, во многих вычислительных заведениях этого вопроса не возникает — контрольных просчетов просто не делают.

О результатах счета судят по правдоподобности. К сожалению, некоторым математикам по роду службы приходится выдавать не приблизительно верные, а просто верные ответы. Мы оказались именно в этом положении. Этот параграф написан не для всех читателей, а только для товарищей по несчастью. Следующий — тоже.

Прежде всего легко убедиться, что контрольные просчеты лишь тогда гарантируют от ошибок, когда они проведены с большим (6–8) числом знаков. Если знаков взять мало, то останется незамеченной ошибка в маленьких слагаемых. Когда же мы перейдем к полному счету, маленькие слагаемые могут сильно подрасти. Рост ошибок при этом почему-то, как правило, напоминает атомный взрыв.

Пусть теперь контрольный просчет немножко разошелся с просчетом на машине. Как искать ошибку? Для этого нужно локализовать место, где она произошла. Машина работает в двоичной системе. По-

[1)] Для реальной машины М-20 в Б-61 адреса меняются местами, и та же строка выглядит так:

453, 2005, 4012, 2076

этому, хотя мы и видим на пульте результаты выполнения операций, нам трудно сверить их с результатами ручного просчета.

В индивидуальном, неизвестном нам заранее, месте помогает программа ПЧП. Кстати, удобно иметь возможность переведенное в 10-форму число видеть зажженным на пульте — не придется из-за одного числа бегать к печати.

Стоит зафиксировать некоторый (у нас 41_{10}-й) разряд определенного ДЗУ (у нас ДЗУ-2), нажим которого превращает ПЧП-ПЕЧАТЬ в ПЧП-ОГНИ.

Однако это все мелочи. Важно суметь в точности проверить правильность работы большой вычислительной программы и в то же время не превратить в вечную каторгу[2] работу по контрольным просчетам.

Общего рецепта, к сожалению, я не знаю. При отладке отдельных блочков можно пользоваться расчетом по искусственным входным данным. Здесь стоит рекомендовать вещи простые:

A) Не делайте ни одно из входных чисел в искусственном контрольном варианте ни 0, ни 1 — легко прозевать ошибку.

B) Не делайте двух чисел равными — могут быть перепутаны ячейки в программе.

C) Не берите чисел чересчур маленькими и чересчур большими — ошибка проявится в слишком далеких знаках.

А главное — относитесь к этому делу с полным уважением. В вычислительных задачах *организация контрольного счета*, кажется, *самая квалифицированная часть работы*.

§ 6. Один пример организации контрольного счета

Нам нужно вычислить:

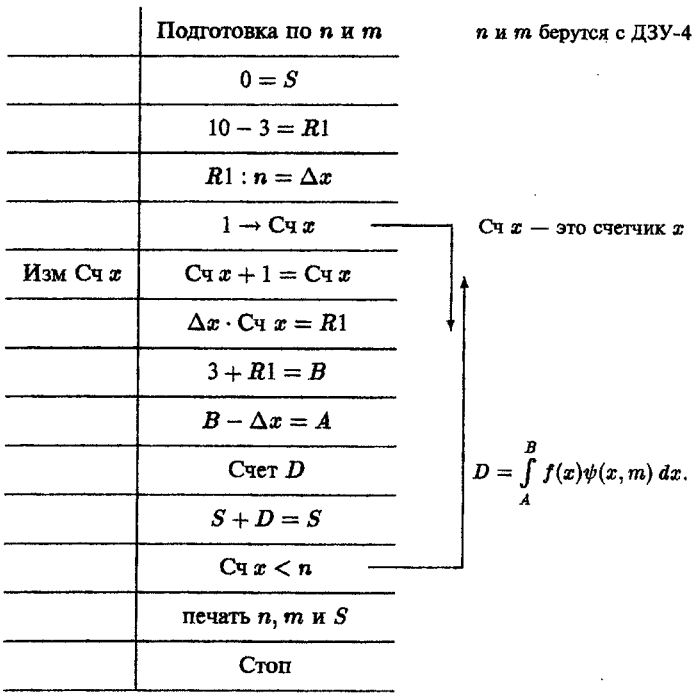
$$S = \int_3^{10} f(x)\,dx \int_{a(x)}^{b(x)} \varphi(x, y)\,dy.$$

[2] Как для лаборантов, проводящих расчеты, так и для себя.

Пусть мы и внешний и внутренний интегралы хотим вычислять так. Разобьем при заданном x отрезок y от $a(x)$ до $b(x)$ на m равных интервалов и на каждом из них вычислим интеграл от $\varphi(x, y)$ по 5 гауссовым точкам. Назовем результат такого вычисления $\psi(x, m)$.

Теперь разобьем отрезок от 3 до 10 на n равных интервалов и опять-таки вычислим на каждом из них интеграл от $F(x) = f(x) \cdot \psi(x, m)$ по 5 гауссовым точкам. Полученный ответ назовем $S(n, m)$.

Для разумной организации контрольного просчета поступим так. Составим блок-программу вычисления $S(n, m)$ по схеме:



(Здесь интеграл от A до B берется по 5-ти гауссовым точкам, а функция $\psi(x)$ равна интегралу $\int_{a(x)}^{b(x)} \varphi(x, y)\, dy$, взятому по m равным отрезкам, на каждом из которых интегрирование идет по 5-ти гауссовым точкам.)

Блок счета D будет выглядеть так:

Для машины без регистра адреса

СЧЕТ D	$\Omega =$ конец D	
	$B - A = \delta x$	
	$0 = D$	
	$\star^\circ = \star$	
	$\Delta^\circ = \Delta$	
	1 → Сч Уз x	
$\star^\circ$	Уз $1 \cdot \delta x = R1$	
Δ°	$v1 \cdot R1 = R1$	
Изм $\star$	$\star +, (1, 0, 0) = \star$	
	$\Delta +, (1, 0, 0) = \Delta$	
	Сч Уз $x + 1 =$ Сч Уз x	
$\star$	Уз $k \cdot \delta x = R1$	не перфорируется
	$A + R1 = x$	
	СЧЕТ $f(x)$	
	СЧЕТ $\psi(x)$	
	отв $f(x) \cdot$ отв $\psi(x) = R1$	
Δ	$vk \cdot R1 = R1$	не перфорируется
	$D + R1 = D$	
	Сч Уз $x < 5$	
	$D \cdot \delta x = D$	
	Конец D	

Для машины с регистром адреса

СЧЕТ D	$\Omega =$ конец D	
	$B - A = \delta x$	
	$0 = D$	
	$0 \to$ РА; (восстановление старого РА) $\to$	предконец
	Уз $1^* \cdot \delta x = R1$	модификация по 1 адресу
	$A + R1 = x$	
	СЧЕТ $f(x)$	
	СЧЕТ $\psi(x)$	
	отв $f(x) \cdot$ отв $\psi(x) = R1$	
	$v1^* \cdot R1 = R1$	модификация по 1 адресу
	$D + R1 = D$	
	РА $< 4 \mid$ (РА $+ 1$) $=$ РА	
	$D \cdot \delta x = D$	
	Предконец	не перфорируется
	Конец D	не перфорируется

Здесь Уз 1–Уз 5 — гауссовы узлы, а $v1$–$v5$ — гауссовы веса для отрезка [0, 1].

Программа счета $\psi(x)$, в свою очередь, будет выглядеть так:

Счет $\psi(x)$	$\Omega =$ конец $\psi(x)$
	СЧЕТ $a(x)$ и $b(x)$
	$b(x) - a(x) = \Delta y$
	$\Delta y : m = \delta y$
	$0 =$ отв $\psi(x)$
	$1 \to$ Сч y
Изм Сч y	Сч $y + 1 =$ Сч y
	$\delta y \cdot$ Сч $y = R1$
	$a(x) + R1 = b_{\text{раб}}$
	$b_{\text{раб}} - \delta y = a_{\text{раб}}$
	СЧЕТ M
	отв $\psi(x) + M =$ отв $\psi(x)$
	Сч $y < n$
	Конец $\psi(x)$

$$M = \int_{a_{\text{раб}}}^{b_{\text{раб}}} \varphi(x, y)\, dy$$

Наконец, блок СЧЕТ M приобретет вид:

Для машины без регистра адреса

СЧЕТ M	$\Omega =$ конец M	
	$0 = M$	
	$0 = D$	
	☆☆° =☆☆	
	$\Delta\Delta^\circ = \Delta\Delta$	
	$1 \to$ Сч Уз y	
☆☆°	Уз $1 \cdot \delta y = R1$	
$\Delta\Delta^\circ$	$v1 \cdot$ отв $\varphi = R1$	
Изм ☆☆	☆☆+, (1, 0, 0) =☆☆	
	$\Delta\Delta+, (1, 0, 0) = \Delta\Delta$	
	Сч Уз $y + 1 =$ Сч Уз y	
☆☆	Уз $k \cdot \delta y = R1$	не перфорируется
	$a_{\text{раб}} + R1 = y$	
	СЧЕТ $\varphi(x, y)$	
$\Delta\Delta$	$vk \cdot$ отв $\varphi = R1$	не перфорируется
	$M + R1 = M$	
	Сч Уз $y < 5$	
	$M \cdot \delta y = M$	
	Конец M	

Для машины с регистром адреса

СЧЕТ M	$\Omega =$ конец M	
	$0 = M$	
	$0 \to$ РА; (восстановление старого РА) $\to$	предконец
	Уз $1^* \cdot \delta y = R1$	
	$a_{\text{раб}} + R1 = y$	
	СЧЕТ $\varphi(x, y)$	
	$v1^* \cdot$ отв $\varphi = R1$	
	$M + R1 = M$	
	РА < 4 | (РА $+ 1$) $=$ РА	
	$M \cdot \delta y = M$	
	Предконец	
	Конец M	

Теперь посмотрим, как организовать контрольный расчет. Выясним прежде всего, какие из блоков являются самыми «глубокими». При этом нужно думать не о том, сколько блоков обнимают извне данный блок, как матрешки, а сколько «*матрешек*» *содержит* одну в другой наш блок. С этой точки зрения блоки младшего ранга, ничего (кроме, быть может, программ библиотеки) не обнимающие, суть ПОДГОТОВКА ПО n и m, СЧЕТ $f(x)$, СЧЕТ $a(x)$ и $b(x)$ и, наконец, СЧЕТ $\varphi(x, y)$.

Это — блоки 1-го ранга. Следующий по глубине (1 матрешка внутри) блок один — СЧЕТ M. Это — блок 2-го ранга. СЧЕТ M является подблоком блока СЧЕТ $\psi(x)$. Кроме него, подблоком СЧЕТа $\psi(x)$ служит еще и СЧЕТ $a(x)$ и $b(x)$. Но ранг придается по максимуму, т. е. СЧЕТ $\psi(x)$ получает 3-й ранг. Теперь, очевидно, СЧЕТ D получает 4-й ранг, и на этом дело кончается. Если угодно, можно самой внешней блок-программе придать 5-й ранг, но это уже будет иметь значение чисто философское.

Из блоков 1-го ранга раньше всех в программе встретится блок ПОДГОТОВКА ПО n и m. Реально этот блок берет слово из ДЗУ-4

и, скажем, по его II и III адресам определяет n (II адрес) и m (III адрес), после чего превращает числа единиц в этих адресах в целые числа в плавающей форме и кладет их в ячейки n и m. Здесь все настолько просто и легко проверяемо, что разве лишь педанту придет в голову для полноты и законченности поставить печать по требованию.

Дальше в программе случается блок СЧЕТ $f(x)$. Здесь уже к месту контрольная печать (по требованию). Что напечатать? По-видимому, величины x и $f(x)$. Так и сделаем в конце этого блока. А для включения такой печати отведем самый левый (45_{10}-й) разряд ДЗУ-4.

Следующим по ходу программы встретится из блоков 1-го ранга СЧЕТ $a(x)$ и $b(x)$. Не поленимся в контрольной печати в конце этого блока, кроме самих $a(x)$ и $b(x)$, повторить и печать x. Не забывайте, что в живой программе ошибка могла испортить x за время, пока мы шли от конца блока, считавшего $f(x)$, к концу блока СЧЕТ $a(x)$ и $b(x)$.

Включением печати по требованию в порядке очереди поручим заведовать 44_{10}-му разряду ДЗУ-4.

Теперь последним блоком 1-го ранга является СЧЕТ $\varphi(x, y)$. Печать по требованию в конце этого блока должна давать x, y и $\varphi(x, y)$. Если блок СЧЕТ $\varphi(x, y)$ — сложный, то можно вывести и другие существенные величины. Как, впрочем, и в блоке, считавшем $a(x)$ и $b(x)$, и в блоке счета $f(x)$. А вообще-то говоря, блок, скажем, СЧЕТ $\varphi(x, y)$ мог иметь сложную блок-программу со многими подблоками разного ранга. Но этот параграф не рассчитан на педанта. И, как сказано в одной довольно популярной книжке, имеющий уши да слышит.

Итак, вернемся к контрольной печати блока СЧЕТ $\varphi(x, y)$. Для ее включения по требованию используем очередной 43-й разряд все того же ДЗУ-4.

Блоки 1-го ранга исчерпаны. Перейдем к следующему, 2-му рангу. Здесь есть только один блок — СЧЕТ M. В конце его приделаем печать необходимых величин: x, $a_{\text{раб}}$, $b_{\text{раб}}$ и M. Включение этой печати поручим 42_{10}-му разряду ДЗУ-4.

Блок 3-го ранга (тоже единственный) СЧЕТ $\psi(x)$ снабдим контрольной печатью, включаемой 41_{10}-й клавишей ДЗУ-4. А на печать выведем x; $a(x)$, $b(x)$, m и отв $\psi(x)$.

Теперь остается в блоке СЧЕТ D поставить печати A, B и D, включающиеся нажатием 40_{10}-го разряда ДЗУ-4. Сведем данные о введенных нами печатях по требованию в таблицу. Такую таблицу наряду с инструкцией *необходимо при отладке держать на пульте.*

Блок	Что печатается (по порядку)	Кто включает печать	
СЧЕТ $f(x)$	x, $f(x)$	45_{10}	(ДЗУ-4)
СЧЕТ $a(x)$ и $b(x)$	x, $a(x)$ и $b(x)$	44_{10}	—
СЧЕТ $\varphi(x, y)$	x, y, $\varphi(x, y)$	43_{10}	—
СЧЕТ M	x, $a_{\text{раб}}$, $b_{\text{раб}}$, M	42_{10}	—
СЧЕТ $\psi(x)$	x, $a(x)$, $b(x)$, m, отв $\psi(x)$	41_{10}	—
СЧЕТ D	A, B, D	40_{10}	—

Теперь при отладке естественно поступить так:

Зададим для себя, скажем, $n = 2$ и $m = 3$.

Вычислим для случая счетчик $x = 2$ величины A и B. Это будут 6,5 и 10. Далее вычислим на отрезке [6,5; 10] 2-й гауссов узел. Это будет $x = 7{,}3076\,7869$.

Теперь для одного этого x проведем все дальнейшие расчеты. Величины $f(x)$, $a(x)$ и $b(x)$ вычислим непосредственно.

Далее придадим счетчику y, скажем, значение 3 и сосчитаем δy, $a_{\text{раб}}$ и $b_{\text{раб}}$.

На отрезке $[a_{\text{раб}}, b_{\text{раб}}]$ выберем опять-таки 2-й гауссов узел. Это будет контрольное значение y. Вычислим $\varphi(x, y)$ в этой единственной точке.

Дальше поступим при отладке так:

1°. Не будем пока включать никаких печатей по требованию. Поставим стоп по команде Изм Сч x. Пойдем на автомате с начала. Остановка произойдет в начале второго цикла по x.

2°. Теперь поставим стоп по команде Изм ☆ и пойдем дальше. Стоп будет в начале второго цикла, т. е. *перед* тем как будет взят 2-й гауссов узел. Теперь *включим* печать $f(x)$ (нажмем 45_{10}-й разряд ДЗУ-4). Стоп можно поставить по команде СЧЕТ $a(x)$ и $b(x)$ в блоке СЧЕТ $\psi(x)$. Пустили машину. До стопа пропечатаются x и отв $f(x)$. Проверим их. Если x получился неправильным, проверим прежде всего ячейки Сч x и A. Если они хороши, проверим команду ☆ и следующую за ней. Наконец, проверим число Уз2 (например, с помощью, ПЧП-ОГНИ, если, конечно, у нас нет числа Уз2 в восьмеричной форме). Если и теперь все хорошо, то, вероятно, x портится программой СЧЕТ $f(x)$. Чтобы проверить это, дадим ОБРАТНО, повторим пункт 1°, а затем поставим стоп по команде Изм ☆ и на шагах выполним две команды. Мы будем иметь x *до* работы блока СЧЕТ $f(x)$. Если же x получился правильным, а отв $f(x)$ нет, уместно, снова дойдя до СЧЕТА $f(x)$, перемотаться на свободный барабан программой ТУДА и затем провести отладку блока СЧЕТ $f(x)$, не начиная с самого начала.

3°. Пусть с $f(x)$ все в порядке, и стоим на СЧЕТе $a(x)$ и $b(x)$. Выключим печать $f(x)$, включим печать $a(x)$ и $b(x)$ (44-й разряд) и, дав стоп по $b(x) - a(x) = \Delta y$, пустим машину. Проверим печать x, $a(x)$ и $b(x)$. Если x испортился, виновен блок СЧЕТ $a(x)$ и $b(x)$.

4°. Теперь, когда $f(x)$, $a(x)$ и $b(x)$ уже верны, после 3° можно дать стоп по Изм Сч y. Стоп случится в начале второго цикла. Не меняя стопа, пойдем дальше. Теперь стоп произойдет в начале нужного нам третьего цикла. В этот момент удобно перемотаться на свободный второй барабан — доходить до этого места всякий раз заново сложновато (стоп по Изм Сч x, стоп по Изм ☆, стоп по Изм Сч y и еще раз тот же стоп).

Далее дадим стоп по Изм ☆☆. Мы попадем на начало второго цикла по узлам. Включим печать $\varphi(x, y)$ (43-й разряд ДЗУ-4) и дадим стоп по $\Delta\Delta$. Сверим печати.

а) Если x и y верны, а $\varphi(x, y)$ нет, можно дать ОБРАТНО со II барабана; затем выключить печать $\varphi(x, y)$, дать стоп по Изм ☆☆ и пойти с команды Изм Сч y. После стопа хорошо перемотаться (ТУДА) на свободный барабан и провести отладку $\varphi(x, y)$, всякий раз при сомнении давая ОБРАТНО и начиная с Изм ☆☆.

b) Если x верен, а y — нет, следует искать ошибку в блоке СЧЕТ $\psi(x)$. Прежде всего сразу проверим δy. Если δy неверно, посмотрим Δy и m. В случае чего, проверим, были ли они верны *до* работы блока СЧЕТ $\varphi(x, y)$ в первом цикле. Для этого придется перемотаться программой ОБРАТНО с барабана, где лежит первоначальная программа. Затем, отключив печати, дойти до Изм Сч x, затем стоп по Изм ☆ и, наконец, стоп по $b(x) - a(x) = \Delta y$. Выполнив две команды на шагах, проверим δy. Если есть ошибка, она видна в этих командах. Если же δy верен, то он дальше портится на первом цикле СЧЕТа M. Следовательно, нужно локализовать место порчи δy. Можно последовательно ставить стопы, скажем, по СЧЕТу M, СЧЕТу $\varphi(x, y)$, $\Delta\Delta$ и т. д. и проверять всякий раз δy. Еще проще поставить *стоп по записи* в δy, если, конечно, в машине есть такая возможность.

Итак, пусть δy уже верен, а y все равно неверен. Перемотаемся со второго барабана и дойдем до ☆☆. Теперь, не снимая стопа, пройдем до той же команды еще раз. Она должна иметь вид:

$$\text{Уз2} \cdot \delta y = R1.$$

Выполним команду. Проверим полученное $R1$ (эта величина есть в контрольном просчете!). Если $R1$ не совпала с просчетом, остается еще проверить Уз2. Если Уз2 в порядке — врет машина или неверен контрольный счет. Пусть $R1$ в порядке. Еще шаг — и мы получили y.

Если он верен, то дальше его портит СЧЕТ $\varphi(x, y)$. Даем ТУДА на II барабан и проводим поиск порчи y. Если есть стоп по записи

в ячейку, то даже ТУДА излишне — мы мгновенно найдем плохое место в СЧЕТе $\varphi(x, y)$. Если же y неверен, а команда $a_{\text{раб}} + R1 = y$ верна, смотрим $a_{\text{раб}}$, — именно эта ячейка неправильна. Поиск ошибки описывать не будем: если уже написанного недостаточно, то ненаписанное — излишне.

c) Теперь рассмотрим случай, когда x неверен. Напомним, что мы стоим на команде ΔΔ на втором цикле по x, втором цикле по УЗЛАМ x, третьем по y и втором по УЗЛАМ y. Следовательно, нам нужно решить вопрос: кто испортил x? Что он получался поначалу верным, мы уже проверяли в печати $a(x)$ и $b(x)$.

Последний раз x получался командой

$$A + R1 = x$$

в блоке СЧЕТ D. Значит, перемотаемся с основного барабана и пройдем так: стоп по Изм Сч x, стоп по Изм ☆, стоп по $A + R1 = x$ или на шагах до этой команды. Еще раз проверим полученный x (все бывает!), а затем, задав *стоп по записи в ячейку* x, дадим стоп по команде Изм Сч y, снова стоп по той же команде, стоп по Изм ☆☆ и, наконец, стоп по ΔΔ. Мы поймаем команду, портящую x. Не удивляйтесь, если мы придем на $A + R1 = x$ (!). Это может случиться, например, если в результате ошибки *при некотором условии* блок СЧЕТ $\varphi(x, y)$ выбрасывает управление в середину старшего блока (безусловной передачи быть не может — мы бы не дошли до ΔΔ). Если стопа по записи в x нет, придется после получения x следить за ним, например, так: стоп по команде СЧЕТ M, проверили x. Стоп по команде отв $\psi(x) + M =$ отв $\psi(x)$, проверили x. Если x испорчен, начнем снова с $A + R1 = x$ (перемотавшись, конечно, перед началом поиска ТУДА, а в этот момент ОБРАТНО). Поиск пройдет теперь по внутренней части СЧЕТа M — стоп по СЧЕТу $\varphi(x, y)$, затем по ΔΔ. Смотрим оба раза x (заметим, что y не контрольный (!), но это сейчас не мешает). Возможна порча x *не на первом цикле* M. Чтобы уловить это место, удобно x вызвать на огни, если это предусмотрено в машине, и по ходу дела следить за x прямо по регистру.

d) Остается рассмотреть самый легкий случай: x и y верны, а $\varphi(x, y)$ нет. Дело сводится к отладке блока СЧЕТ $\varphi(x, y)$. Можно прямо дать ТУДА и начать отладку.

5°. Пусть, наконец, отлажен блок СЧЕТ $\varphi(x, y)$. Теперь нам остается немного — проследить совместную работу программы. Проще всего поступить так:

a) Дали ОБРАТНО с основного барабана. Отключили печати, стоп по Изм Сч x, стоп по Изм ☆, стоп по Изм Сч y, еще стоп по Изм Сч y. Теперь включили печать $\varphi(x, y)$ и M (43_{10}-й и 42_{10}-й разряды) и дали стоп по команде отв $\psi(x) + M =$ отв $\psi(x)$.

Полученная табулограмма даст значения $\varphi(x, y)$ в 5-ти точках и $M = \sum_{s=1}^{5} v_s \cdot \varphi(x, y_s)$. Второе значение $\varphi(x, y)$ — контрольное. Ценой 5-ти умножений мы проверим (отойдя от машины) правильность работы блока M.

b) Снова отключим печати, дадим ОБРАТНО с основного барабана. Стоп по Изм Сч x, стоп по Изм ☆. Теперь включим печать M и $\psi(x)$ (44_{10}-й и 41_{10}-й разряды). Дадим стоп по Δ. Полученная табулограмма содержит три значения M и их сумму $\psi(x)$, причем последнее значение M мы имеем из печатей пункта A.

c) ОБРАТНО с основного барабана. Выключили печати, дошли до Изм Сч x. Включили печати $f(x)$, $\psi(x)$ и D (разряды 45_{10}-й, 41_{10}-й и 40_{10}-й). Поставили стоп по $S + D = S$. Мы ведем отладку блока СЧЕТ D. На полученной табулограмме будет 5 групп значений $f(x)$ и $\psi(x)$ и величина D. Поскольку $D = \delta x \cdot \sum_{s=1}^{5} v_s \cdot f(x_s) \cdot \psi(x_s)$, мы легко проверим правильность работы блока СЧЕТ D.

d) Дали ОБРАТНО. Выключили печати $f(x)$ и $\psi(x)$, оставили только печать D (40_{10}-й разряд) и пошли без стопа. Получим печать двух значений D и их суммы S. Если все хорошо — программа отлажена.

Ответы на задачи

1. Пусть в нашей программе произошло зацикливание, замеченное на *десятой* минуте работы программы. Мы перемотались ОБРАТНО и пошли по блокам. На *шестой* минуте мы дошли до блока, где, как мы подозреваем, происходит зацикливание. Но сам этот блок управляется блок-программой. Как теперь быть? Как только мы доберемся до зацикливающегося места, нам придется начинать сначала, чтобы локализовать район ошибки. А каждый раз это обойдется в 6 минут. Идти по шагам нельзя — может быть, до ошибочного места добираться еще 4 минуты работы на автомате. Вот в этот момент можно перемотаться с помощью ТУДА на *второй* барабан. Теперь мы сможем, дав при нужде ОБРАТНО со *второго* барабана, начинать прямо с этого места, не теряя шести минут. Дальнейшее усовершенствование оставляем читателю.

2. На одном барабане удобно хранить библиотеку, а про остальные два см. предыдущую задачу.

3. Одному программисту — незачем. А трем программистам вместе поможет. Программист A записал свою программу на II барабан (на I — библиотека), немножко поотлаживался, нашел и исправил пять ошибок, а на шестой увидел, что ему нужно подумать. Программист B использовал таким же способом III барабан. Теперь программисту C необходим IV барабан. Заметим, что если он сотрет программу A, то, когда

дойдет очередь до А, тому придется не просто ввести свою программу, а еще и сделать все поправки. Случай двух программистов с четырьмя барабанами опять-таки оставляем читателю.

4. Все-таки выгоднее максимальным возможным в машине числом — от этого часто происходит через несколько действий Авост. Такое число мы называли Щ. Конечно, если повезет, то хорошо бы, чтобы Щ было стоп-командой. Только надо помнить, что действие Щ $\times \frac{1}{2}$ к Авосту не приводит (!).

Беседа четвертая

Еще кое-что об отладке программ

А ищи всех паче
Разума в задаче.

Магницкий

§ 1. Об отладочных программах

Без крайней нужды их не надо помещать на места, занятые какой-либо частью основной программы. Наоборот, надо считать *основную программу плюс отладочные единой программой* с разными входами. Только так мы можем быть уверены, что отлаживали ту программу, которую собирались. Если же отладочная программа (программы) составляет отдельную колоду, то основная программа, введенная без отладочных, может отлично выйти на Авост, зациклиться или — хуже — неверно работать из-за ошибок, которых нет в колоде-сумме. Уже не говоря о том, что контрольная сумма КΣ основной колоды нам неизвестна. Обратное, конечно, неверно. Попадаются ошибки такого типа: некоторая неперфорируемая ячейка восстанавливается в отладочной программе, а в основной забыта. Что от этого получится — сообразите сами.

Именно по таким причинам желательно в отладочной программе не допускать вмешательства в основную. Поясню на сознательно тривиальном примере. Пусть нам надо сосчитать

$$S = \sum_{k=87}^{10\,000} \frac{k+3}{k(k+1)(k+2)}.$$

Напишем программу так:

	$0 = S$
	$87 \to k$
Изм k	$k + 1 = k$
	$k + 3 =$ Числ
	$k + 1 =$ Знам
	Знам $+ 1 = R1$
	Знам $\cdot R1 =$ Знам
	Знам $\cdot k =$ Знам
	Числ : Знам $= R1$
	$S + R1 = S$
	$k < 10\,000$; Изм k
	Печать S
	Стоп

Для отладки мы сосчитали (с 8–9 знаками!):

$$S_{\text{отл}} = \sum_{k=87}^{89} \frac{k+3}{k(k+1)(k+2)}.$$

Тогда можно заменить команду

$$k < 10\,000; \quad \text{Изм } k$$

командой

$$k < 89; \quad \text{Изм } k$$

и сверить результат работы с просчетом. Так делать плохо — могли остаться неверными замененная команда и число 10 000. Можно поступить иначе: команду

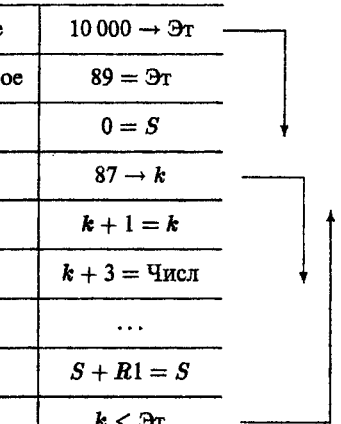
$$k < 10\,000; \quad \text{Изм } k$$

сохранить, а число 10 000 заменить на 89. Теперь стало лучше, но число 10 000 все-таки не проверяется. Конечно, в конкретном примере можно отдельно вывести его на печать, а команду

$$k < 10\,000; \quad \text{Изм } k$$

прочесть с пульта. Однако канонический способ таков:

Начало основное	10 000 → Эт
Начало отладочное	89 = Эт
	$0 = S$
	$87 \to k$
Изм k	$k + 1 = k$
	$k + 3 =$ Числ
	...
	$S + R1 = S$
	$k <$ Эт
	Печать Эт и S
	Стоп



Пусть теперь мы при отладке некоторой программы хотим отдельно проверить блок счета функции $F(x)$ и отдельно — остальную программу, заменив $F(x)$ простой функцией $f(x)$. Не стоит забивать ячейку обращения к $F(x)$.

Лучше пожертвовать тремя ячейками и поступить так:

Введем неперфорируемую ячейку U. В рабочей части программы всюду вместо команды ухода на $F(x)$:

$$\text{Продолжение} \leftrightarrow \Omega; \quad F(x)$$

поставим команду ухода на U:

$$\text{Продолжение} \leftrightarrow \Omega; \quad U$$

Теперь достаточно в начале основной программы поставить

$$F(x) \leftrightarrow U$$

а в отладочной программе

$$f(x) \leftrightarrow U$$

Систематическое проведение такой чуть громоздкой системы окупается не уменьшением числа ошибок, а ускорением их поиска.

Все по тем же причинам не следует без крайней нужды менять содержимое ячеек с пульта. Вспомним пример, разобранный в предыдущей беседе. Пусть мы хотим сосчитать S с заданной точностью ε. Реально мы вычисляем $S(n, m) \approx S$. Заранее неизвестно, как должны быть велики n и m. Для контрольного расчета мы примем $n = 2$, и $m = 3$. Эти числа не следует писать в ячейки n и m. Мы сознательно удлинили программу, введя блок подготовки по n и m. Теперь мы меняем n и m нажатием клавиш ДЗУ, а ***программу оставляем нетронутой.*** В конкретном случае мы за это получаем еще премию в таком виде:

Мы уже отладили программу при $n = 2$, $m = 3$.

Начнем подбор n и m.

Положим $n = 1$, $m = 1$ и проведем счет (все дополнительные печати выключены).

Дальше, сохраняя $n = 1$, будем добавлять к m по 1 или удваивать m. Найдем, при каком максимальном $m = m_0$ число $S(1, m)$ стабилизировалось с заданной точностью.

Теперь вернем $m = 1$ и поиграем с n. Найдем число n_0. Вычислим $S(n_0, m_0)$. Много шансов, что $S(n_0, m_0)$ и есть то, что нам нужно. К слову сказать, проверка

$$|S(n_0, m_0) - S(n_0, 1) - S(1, m_0) + S(1, 1)| < \varepsilon$$

делает это утверждение чрезвычайно вероятным. Если все же сомнения есть, можно сосчитать $S(n_0 + 1,\ m_0 + 1)$. Заметьте, что переход от варианта к варианту по n, m вовсе не требовал времени на ввод чисел в машину — мы просто перебирали клавиши на ДЗУ, причем могли готовить новый вариант во время счета старого.

§ 2. Стандартизация обозначений

Ангелу-программисту [1] читать программу не придется — она верна. Человеку [2] — приходится. Поэтому ему, как и всякий текст, программу быстрее читать не по буквам и не по слогам, а целыми словами и даже фразами. А значит, из двух возможностей

$$S + x = S;$$
$$x + S = S$$

стоит остановиться на любой (я выбрал первую) и не баловать взор разнообразием.

---

[1] См. определение 1 на с. 17 .

[2] См. определение 2 там же.

Точно так же команды

$$(Я + 1) \leftrightarrow \Omega; \quad \ln \alpha$$

$$(Я + 1) \leftrightarrow \Omega; \quad \text{Счет коэффициентов}$$

отвлекают внимание на чтение и тратят время на письмо. Им следует предпочесть

$$\ln a$$

Счет коэффициентов.

Если нам надо заслать 2 в A и перейти к следующей команде, можно написать:

$$2 \to A \ \neg$$

или $0 + 2 = A$
или $2 + 0 = A$
или $2 \vee 2 = A$
или $2 \wedge 2 = A$
или $0 \vee 2 = A$
или $0 \not\sim 2 = A$
. . . . . .

Хорошо остановиться на одном способе и его придерживаться. Я выбрал для себя

$$0 \vee 2 = A$$

учитывая, что приходится засылать не только числа, но и команды (отпадает возможность $0 + 2 = A$, как стандартная) и что команда

$$2 \to A; \quad Я + 1$$

содержит лишний соблазн ошибки для кодировщицы (три ненулевых адреса вместо двух).

Конечно, раз стандарт установлен, написание $0 \vee \ldots$ излишне и следует писать просто:

$$2 = A.$$

Короткая запись $2 \to A$ нехороша, так как она легко получается от того, что в команде

$$2 \to A; \quad B$$

не дописано B.

Именовать величины лучше так, чтобы их начертание напоминало о назначении: Сч x, Эт y, Знам — выразительнее, чем $A2$, M и $R5$.

Передачу управления на близкую ячейку тоже выразительнее метить стрелкой. Однако никогда не тяните стрелок на другой лист. Да и на одном листе множество параллельных и встречных стрелок затруднит чтение. Часть из них надо заменить обозначением места передачи, скажем,

$$n < 5; \quad \text{Изм } x$$

Ячейку, куда происходит передача управления, не указанная стрелкой, обязательно следует пометить слева символом — Изм x; $a > 0$; ХОРОШО и т. д., если это соответствует смыслу, или ☆, Δ и т. д., если выразительного короткого обозначения не нашлось.

Индексы при программировании следует писать «в рост»: $A1$, $R5$, $L2,3$, а не A_1, R_5, $L_{2,3}$. Спутать не с чем.

Не употребляйте в программе одной и той же буквы с *разным* числом индексов. Это затрудняет кодировку (правильную), и, кроме того, недописанный индекс приводит к ошибке. Нуль тоже число, значит, A и $A2$ в одной программе несовместны.

Использовать ДЗУ следует тоже по привычным стандартам. Скажем, если ДЗУ-2 — библиотечное, а ДЗУ-1 и ДЗУ-3 употребляются при поправке с пульта, все блокировки печати лучше делать на ДЗУ-4, причем закрепить для этого в первую голову разряды кода. Если мы считаем тройной интеграл, естественно для числа точек распределить адреса ДЗУ-4 так: левый адрес — для внешнего интеграла, средний — для среднего, правый — для внутреннего. Если почему-нибудь хочется поступать иначе — пожалуйста. Но старайтесь не менять собственного стандарта.

Для машин без регистра адреса рекомендуется такая структуру цикла:

Не слишком страшно переставить изменение вниз, после рабочей части. Но тогда уж закрепите такую схему как стандартную (!). Для машин с РА, пожалуй, следует пойти на то, чтобы начинать счет кругов не с 1, а с 0. Правда, это заставляет проверять окончание не по n, а по $n-1$. Хорошего во всех смыслах решения я не вижу. Приходится останавливаться на каком-нибудь, но стандартном.

В предыдущих параграфах во всех командах мы пользовались плавающими константами для счетчиков. Это неудобно: плавающую константу на регистре читать дольше, чем фиксированную. Стоит остановиться на каком-либо определенном адресе. Если машина работает с РА и обмен идет через средний адрес, естественно его и выбрать для счетчика.

Команды работы со счетчиком приобретают вид:

$$(0, 1, 0) = \text{Сч}$$
$$\text{Сч}+, (0, 1, 0) = \text{Сч}$$
$$\text{Сч}-, (0, n, 0) = \text{Сч}$$

Для машин без РА, пожалуй, приятнее III адрес.

Не жалейте места на введение счетчика в случае цикла *не по регистру адреса*. Экономия на уменьшении числа ошибок окупится многократно. ЭТАЛОН-КОМАНДА неприятна сама по себе и затрудняет отладку. Подумайте, как бы все это выглядело без счетчиков применительно к примеру из прошлой беседы!

Если в машине есть РА, следует подумать, как мы будем отлаживаться, когда пишем цикл по РА.

Начало

$$\text{РА}1, 0, 0, 0$$

с проверкой

$$\text{РА} < 53\ldots$$

потребует при отладке изменения команд проверки (всюду, где они есть), если мы сменим числа m, n, от которых зависит длина цикла.

Наоборот, начало

$$\text{РА}2, 0, (0, n, 0), 0$$

с проверкой

$$\text{РА} \geqslant 2\ldots$$

не требует изменений — меняется только $(0, n, 0)$, а его мы берем с ДЗУ.

Для машины с РА, кроме молитвенного начала блока

$$\Omega = \text{конец},$$

есть еще стандартная команда

РА1; 0*; *m*; предконец

или

РА2; 0*; *m*; предконец

при первом употреблении РА в блоке. Каждый блок (где менялось содержимое РА) кончается обязательно двумя пустыми ячейками ПРЕДКОНЕЦ и КОНЕЦ.

§ 3. Ячейка Ω при отладке

Разумный оптимизм при отладке заставляет начинать отладку без стопа. Это называется «В радужной надежде» (применительно к отладке — В. Арлазаров). Программа стала на Авосте в библиотечной подпрограмме e^{α}. Посмотрим ячейку Ω. Сразу выяснится, какое из обращений к e^{α} устроило Авост (в Ω будет лежать команда возврата!). То же получится и с остановкой в любом блоке. Именно поэтому для всех блоков надо иметь стандартный выход Ω.

§ 4. Немножко о кодировке

Программа должна быть написана и правила кодировки должны быть устроены так, чтобы переход к машинному языку (кодировка) был *однозначным*, т. е. чтобы два различных человека, кодируя программу независимо, получили совпадающие знак в знак результаты с точностью до собственных ошибок.

Тогда возникает возможность эти ошибки почти исключить: можно проверить кодировку второму человеку. В особо важных случаях можно провести повторную кодировку и сверить результаты.

При хорошей организации дела кодировка занимает по сравнению с написанием программы очень мало времени — раз в 10 меньше. И может делаться людьми, вовсе не имеющими математического образования.

Что для этого нужно сделать?

Во-первых, абсолютно стандартизовать обозначения кодов команд в левой части программы.

Для этого составляются ПРОПИСИ разрешенных обозначений, например:

a U c	056 *a U c*
Б*U*	056 0 *U* 0

или

Здесь показано, какие разрешены обозначения пересылки слов с безусловной передачей управления для машины М-20.

Во-вторых, шпаргалка (роспись памяти) должна давать полную информацию о всех ячейках программы, чтобы ничего не надо было добавлять «на словах».

Для ангела-кодировщицы[3] этого уже и достаточно. Для человека — мало.

Пусть в программе участвуют рабочие ячейки 11, 12, ..., 17. Обозначим их так:

*R*11—2011
*R*12—2012
*R*13—2013
*R*14—2014
*R*15—2015
*R*16—2016
*R*17—2017

а теперь так:

*R*11—2036
*R*12—2037
*R*13—2040
*R*14—2041
*R*15—2042
*R*16—2043
*R*17—2044

Понятно, что кодировать стало много тяжелее. Но ведь можно сделать

*R*11—2011
*R*12—2012
*R*13—2013
*R*14—2015
*R*15—2014
*R*16—2016
*R*17—2017

Это уже почти полная гарантия того, что ошибка при кодировке случится, и притом с большой вероятностью *не будет замечена при проверке.*

[3] Определение ангела-кодировщицы вдумчивый читатель даст по аналогии с определениями 1 и 2 на с. 17.

Составление шпаргалки — дело отнюдь не механическое. Про него должен подумать математик. В том числе и про такое, что, например, произойдет, если для вектора $\bar{u}(u_1, u_2, \ldots, u_7)$ мы отвели ячейки $2031, \ldots, 2037$, для вектора $\bar{v}(v_1, v_2, \ldots, v_7)$ — ячейки $2041, \ldots, 2047$, а потом порядок векторов увеличился до 12. Кодировщице догадаться, с какими массивами это может случиться, — трудновато. Даже — ангелу.

Несколько чисто технических замечаний.

Сокращенные обозначения помогают программисту и не затрудняют кодировщиц. Поэтому, например, удобно для часто встречающегося адреса 7777 закрепить обозначение F. Тогда F, O, F будет закодировано $7777, 0, 7777$.

Обращение к печати идет со строкой информации. Поэтому оно выглядит, например, так[4]:

```
(Я+1)   Пq1   Ω
453  a1 4005  a20
```

Но вместо этого можно писать просто:

```
        Пq1   Ω
453  a1 4005  a20
```

Только это должно быть раз навсегда оговорено.

Левую часть нужно писать очень разборчиво. Неаккуратное обозначение команды сверки

$$a\chi b = c$$

приведет к тому, что ее воспримут как

$$a \times b = c$$

Не ленитесь, работая на машине с РА, при модификации адресов

$$a^* : M = b^*$$

звездочки делать красными — сэкономите время и нервы и себе и кодировщицам.

И — не употребляйте трудно различимых обозначений: a и A не спутаешь, а как насчет c и C?

В правой части надо обязательно метить адреса команд. Однако достаточно делать так:

---

[4] В библиотеке Б-61 в действительности происходит передача управления с возвратом не к $Я+1$, а к $Я+2$.

2145	001	3011	3012	3001
46	452	0	0001	2063
47	...	...	...	...
2150				
51				
...	...	...	...	...

Так излишне:

2145
2146
2147
2150

А так недостаточно:

2145
6
7
50
1
2

И последнее. Кодировать нужно только карандашом. Тогда исправлять помогает резинка.

Беседа пятая

Про библиотеку стандартных программ

А ищи всех паче
Разума в задаче.

Магницкий

§ 1. Что должно быть в библиотеке

Во-первых, обслуживающие программы. Это ТУДА, ОБРАТНО, ПОПРАВКА С ПУЛЬТА, ПЕЧАТЬ ПРОГРАММЫ, ПЕЧАТЬ ПРОГРАММЫ С ПУЛЬТА и т. д.

Для удобочитаемой печати команд нужно еще иметь программу, превращающую слово

503257103752715

в два слова, дающие *при десятичной* печати результат:

503 2571
0375 2715

В двоично-десятичной записи, если, скажем, тетрада $e \equiv 1110$ дает на печати пропуск, они должны иметь вид[1]:

01*ee*503*e*2571
01*e*0375*e*2715

[1] Для десятичной печати машины М-20. Левые 9 разрядов слова воспринимаются ею слева направо так: 3 разряда — знаки, затем 2 разряда — четверичное число, и только правые 4 разряда дают тетраду, в которой можно сделать пропуск.

Такая программа A — отличный повод для упражнений на наиболее экономное ее написание. Для машины М-20 рекорд долго был за В. Арлазаровым и Г. М. Адельсоном-Вельским (9 команд), но был побит Е. Леферовым (8 команд). (См. Приложение 1 к этой книге.)

Полезно еще иметь программы для перевода констант типа $a, 0, 0$; $0, b, 0$; $0, 0, c$ в плавающую форму (ФП-1, ФП-2, ФП-3) и обратно (ПФ-1, ПФ-2, ПФ-3), программы групповых пересылок слов и, наконец, очень часто употребляемую программу условной передачи управления по состоянию указанного разряда в указанном ДЗУ (программа ПО ДЗУ).

Все такие программы следует объединить в 1-ю часть библиотеки (Б-1 — обслуживание).

Они нужны и в вычислительных и в невычислительных программах. Кроме того, к Б-1 хорошо отнести программы перфорации. В эту же 1-ю часть естественно отнести набор констант.

Чистота идеи здесь нарушается — плавающие константы не нужны в невычислительных программах. Но некоторый оппортунизм облегчает жизнь. Поэтому все-таки в Б-1 хорошо включить, скажем,

$$10^{-6}, 10^{-5}, \ldots, 10^{-1}; 2^{-4}, 2^{-3}, 2^{-2}, 2^{-1}; 1, 2, \ldots, 10; \frac{1}{3}; \frac{\pi}{2}; \pi \text{ и } \frac{\pi}{180.}$$

Довольно часто бывает нужна константа Эйлера $C = 0{,}57721566\ldots$. Зато ни разу не встретилась надобность в e.

Из фиксированных констант необходимы комбинации единичек в адресах:

$$0, 0, 1$$
$$0, 1, 0$$
$$0, 1, 1$$
$$\ldots\ldots$$
$$\ldots\ldots$$
$$1, 1, 1$$

и высекатели адресов:

$$0, 0, F$$
$$\ldots\ldots$$
$$\ldots\ldots$$
$$F, F, 0$$
$$F, F, F$$

Адреса этим константам удобно придать хорошо запоминающиеся:

0, 0, 1	0121
0, 1, 0	0122
.....	
1, 1, 1	0127
0, 0, *F*	0131
.....	
F, *F*, 0	0136
F, *F*, *F*	0137

Вслед за Б-1 идет по частоте употребления группа программ для перевода $2 \to 10$ и печати (Б-2).

Дальше хорошо иметь такие части библиотеки:

Б-3 Программы для работы с барабанами и лентами.

Б-4 Группа программ, связанная с $10 \to 2$.

Б-5 Элементарные функции.

Б-6 Линейная алгебра.

Б-7 Работа с таблицей.

Б-8 Интегралы.

Б-9 Интегрирование систем дифференциальных уравнений.

Б-10 Работа со случайными числами.

Б-11 Корень функции.

Кроме того, бывают еще нужны программы для специальных функций, программы для работы с числами с двойной мантиссой и большим диапазоном порядков (Библиотека дальнего плавания), реже — для работы с комплексными числами и функциями.

§ 2. Стандартные ячейки

Для аргумента мы условились закрепить ячейку α. Учитывая, что числа бывают и комплексными, полезно иметь две аргумент-ячейки: $\alpha \equiv \alpha 0$ и $\alpha 1$, идущие подряд. Если предполагается работа с библиотекой дальнего плавания, хорошо еще добавить $\alpha 2$.

Для ответа нужны две ячейки: $\beta 0$ и $\beta 1$ [2]. Программы для вычисления cos и sin (и некоторые другие) чаще всего выгодно объединить и иметь общую программу

cos sin,

[2] По причинам чисто историческим эти ячейки называются в Б-61 (см. [1]) $\gamma 0$ и $\gamma 1$.

черпающую аргумент из α и кладущую ответы:

$\cos\alpha$ в $\beta 0$

$\sin\alpha$ в $\beta 1$ (именно так по четности).

Для стандартного конца Ω надо выбрать удобную ячейку, например 0007, 0077 и т. д. Если в машине есть РА, то обязательно еще возвести в ранг стандартной ячейку $\Omega-1$.

Из ячеек α и β особенно часто берутся числа для сравнения их с контрольным просчетом. Значит, программа 2 → 10 должна иметь *свои* ячейки аргумента и ответа $\alpha_{\text{печ}}$ и $\beta_{\text{печ}}$.

§ 3. Что случится с программами, если мы изменим библиотечные подпрограммы?

Хотелось бы, чтобы ничего не случилось. Потому что время от времени библиотечные программы хочется улучшить. И совсем неприятно менять всякий раз все рабочие программы. Общий конец Ω освобождает от хлопот об изменении возврата.

К сожалению, по глубоко принципиальным причинам нельзя устроить для всех подпрограмм общее начало. Зато за каждой программой можно закрепить *постоянное начало* (заголовок). Заголовки следует собрать в кучу в одном из концов оперативной памяти (для М-20 в библиотеке Б-61 это 7600–7766). И поклясться *никогда не менять их адресов*. А рабочие части библиотечных программ теперь можно менять как угодно.

Остается одно — позаботиться об инвариантности контрольных сумм. Сделать это нехитро. Все равно библиотечные программы вводятся целыми частями: Б1, Б2 и Б5 и т. д. Значит, надо не пожалеть к каждой части Бk сделать *добавку* Дk, меняющуюся с изменением программ данной части.

Заголовки образуют еще одну часть библиотеки Б-0. Добавки Дk можно положить в ячейки, примыкающие к массиву заголовков. Но вводить их нужно, конечно, каждую со своей частью библиотеки.

§ 4. Библиотечное ДЗУ

Одно из ДЗУ закрепим для обслуживания библиотеки. В Б-61 это ДЗУ-2.

Разряды в нем использованы так:

45 44 № барабана, с которым будут работать программы ТУДА, ОБРАТНО, ПОПРАВКА С ПУЛЬТА.

[43 42] № магнитофона.

[41] Огни. Если этот разряд нажат, при каждой печати будет получаться остановка. На Р1 зажжется адрес слова, а на Р2 — само слово, которое машина собирается напечатать. По пуску программа пойдет дальше.

[40] Выключение перфорации.

[39] Выключение печати. На ангел-машине (ср. определение 1 и 2 на с. 17) не требуется. На просто машине позволяет продолжать отладку, пока чинят сломанную печать.

[38 37] Дополнительная печать. Если нажат 37-й разряд, каждая печать сдвоится, 38-й — утроится, 37-й и 38-й вместе — учетверится (см. замечание к 39-му разряду).

[36] Выключение печати адресов. Если этот разряд *не* нажат, при печати каждой группы слов перед ней будет напечатан адрес первого слова.

[35] Выключение печати при работе программы 10 → 2.

[34] Автосалют. Нажим 34-го разряда превращает все программы печати в программы с немедленной печатью (вместо занесения впрок на буфер).

Разряды ДЗУ-2 с первого сколько нужно влево служат для вызова частей библиотеки (коротко — библиотек) с барабана.

§ 5. Вызов библиотек

Библиотеки Б0, Б1... можно вводить с карт. Но если нет особых причин, их хорошо завести на один из барабанов, скажем на первый. Туда же надо положить программу ВЫЗОВ.

Нам нужны библиотеки Б0, Б1, Б2, Б5 и Б8. Нажмем разряды 1-й, 2-й, 5-й и 8-й в ДЗУ-2. Наберем номер барабана, с которым будем работать (45-й и 44-й разряды).

Введем одну карточку ЗАПАЛ. Дадим пуск. Программка ЗАПАЛ вызовет в оперативную память программу ВЫЗОВ. ВЫЗОВ разберется, какие нам захотелось иметь библиотеки. Вызовет Б0, Б1, Б2, Б5 и Б8 (с контролем, конечно). Распишет остальные ячейки стандартным словом Щ (−0, если инженеры не ленивы!). И перейдет к программе ТУДА, которая закончится сообщением:

0, *F*, 0 — все отлично
0, *F*, 0

или

$F, 0, F$ — не получилось
$F, 0, F$

Конечно, себя ВЫЗОВ тоже сотрет (в оперативной памяти).

Теперь можно вводить свою программу.

В программе ВЫЗОВ нужно позаботиться о том, чтобы зависимые части библиотеки вызывались, даже если они не указаны на ДЗУ (для специальных функций — элементарные функции, Б0 и Б1 всегда и т. д.).

§ 6. Вызов со сдвигом

Удобнее всего работать с библиотеками, лежащими на постоянных местах. Но все-таки случается (не часто), что свободные ячейки образуют при этом неудобно разбитые массивы. Поэтому существует программа ВЫЗОВ СО СДВИГОМ. Она смыкает вызываемые рабочие части библиотек в единый массив по требованию программиста.

Чтобы устроить ВЫЗОВ СО СДВИГОМ, нужно ввести про каждое слово в библиотеке информацию, надо ли адреса этого слова менять при сдвиге и какие именно.

Как это сделать, предложил В. Арлазаров, а сделали Г. М. Адельсон-Вельский и М. З. Розенфельд. На каждое слово нужна восьмеричная цифра-характеристика. 0 означает, что ни один адрес не нуждается в изменении, 1 — изменяется правый адрес, 2 — средний, 5 — левый и правый, 7 — все три.

Эта информация — таблица характеристик (ТХ) достаточна для того, чтобы ВЫЗОВ СО СДВИГОМ разобрался, кого как менять. А записывается она довольно компактно — по 15 характеристик в 45-разрядной ячейке.

Искушенный профессионал найдет случай, когда таблица характеристик недостаточна для организации сдвига. В практике это значения не имеет.

§ 7. Библиотечные карты

Колода библиотечных карт распадается на естественные части — по номерам библиотек. Добавки и КΣ должны прикладываться к каждой части — на случай непосредственного ввода в память библиотек (а не через вызов с барабана). Затем в виде отдельных колод (тоже с добавкой и КΣ) существуют ВЫЗОВ и ВЫЗОВ СО СДВИГОМ — естественно, расположенные в памяти так, чтобы они могли сосуществовать.

Кроме того, нужны еще отдельные карты:

1. РОСПИСЬ Щ.
2. ЗАПАЛ.
3. ЗАПАЛ СО СДВИГОМ.
4. АВАРИЙНАЯ. Эта карта вот зачем. Мы отлаживались. Ошибка в программе испортила, кроме всего прочего, программу ОБРАТНО. Введем карту АВАРИЙНАЯ и дадим пуск. Прочтется с ДЗУ-2 номер барабана, вся информация с этого барабана (даже без контроля) перемотается в оперативную память и управление перейдет на ОБРАТНО. Так же можно поступить, если сломалась машина, ее починили, а информация на барабане не пострадала.

Беседа шестая

Про библиотеку стандартных программ (продолжение)

А ищи всех паче
Разума в задаче.

Магницкий

§ 1. Программы с информацией

Нам нужно сосчитать $\ln x$ и положить ответ в y. Напишем:

$$x = \alpha,$$
$$\ln$$
$$\beta = y.$$

Итак, по α—β-системе Брудно мы потратим три ячейки. Можно ли сделать короче?

Заведем ячейку информации ИНФ. Укажем в ней адреса x и y. Теперь мы сможем, обращаясь к программе $\ln$, заставить ее переслать число из x в α, а ответ β переслать в y. Однако нам опять понадобились три ячейки:

$$\begin{array}{l} \text{☆} \to \text{ИНФ} \\ \text{☆}\colon\ x, y, 0 \\ \ln \end{array} \quad \Big\downarrow$$

Стало даже хуже — ячеек столько же, а времени пойдет заметно больше (на расшифровку информации и формирование команд).

Однако можно поступить иначе:

$$\begin{array}{ll} \mathit{Я}\colon & (\mathit{Я}+1) \leftrightarrow \Omega;\ \ln \\ \mathit{Я}+1\colon & x, \qquad y,\ 0 \\ \mathit{Я}+2\colon & \ldots\ldots\ldots\ldots\ldots \end{array}$$

Теперь истрачены только две ячейки. Правда, чтобы извлечь и использовать информацию, придется потратить несколько команд: сперва по ячейке Ω, где хранится команда возврата к $(Я+1)$, надо восстановить адрес $Я+1$, а затем сформировать команды:

1°. перекладывающую число из x в α;

2°. перекладывающую число из β (в конце программы) в y и, наконец,

3°. передающую на выходе управление $Я+2$.

Такое занятие слишком дорого в сравнительно быстрых и часто встречающихся в цикле программах вычисления элементарных функций. Несмотря на потерю ячейки при обращении, выгодней остаться при α—β-системе (см. эпиграф к этой беседе). В беседе седьмой мы увидим, как введением специальных команд можно выиграть во всех отношениях.

Не так обстоит дело, скажем, при перемножении двух матриц. Пусть множатся прямоугольные матрицы $||A||$ порядка $k \times l$ и $||B||$ порядка $l \times m$. Результат — матрица $||C||$. Тогда естественно задать информацию так:

$$\begin{matrix} a_{11} & b_{11} & c_{11} \\ k & m & l \end{matrix}$$

где ячейки k, m, l имеют вид, скажем, $(0, k, 0)$ и т. д. $a_{11}\ldots$ — адреса первых ячеек матриц, а в каждой матрице все элементы считаются выписанными *подряд* строка за строкой.

Программу перемножения матриц назовем ММ. Обращение к ней будет выглядеть так:

$Я$: ММ
$Я+1$: a_{11} b_{11} c_{11}
$Я+2$: k m l
$Я+3$: продолжение программы

Здесь ММ, как всегда, — сокращенная запись команды:

$$Я+1 \leftrightarrow \Omega;\ \text{ММ}^{1)}$$

---

[1] $(Я+1) \leftrightarrow \Omega$... не обязательно: можно обратиться

$$u \leftrightarrow \Omega;\ \text{ММ}$$

программа ММ будет черпать информацию из u и $u+1$, а по окончании отдаст управление $u+2$. Мы эту возможность дальше всюду будем подразумевать молчаливо.

или, для машины М-20,

$$\overbrace{\qquad\qquad}$$
$$Я + 1;\ \text{ММ};\ \Omega$$

Аналогично нужно задавать информацию в программах переводов $2 \to 10$ и $10 \to 2$, ИНТЕГРАЛ, решение системы линейных алгебраических уравнений, интегрирование системы обыкновенных дифференциальных уравнений и т. д. Особняком стоят программы групповой пересылки слов. Здесь жалко времени, и я не умею дать универсального рецепта.

§ 2. Система ИНФО-ГФК

Программы с описанным только что обращением называют программами с сопутствующей информацией. Естественно, в каждой такой программе необходимо:

1°. Извлечь адреса ячеек, где лежит информация.

2°. Сформировать после этого необходимые команды, зависящие от этой информации.

Чтобы не терять места, я предложил поступиться временем и ввести две стандартные подпрограммы обслуживания:

1°. Программа ИНФО. Она по ячейке Ω догадывается про адрес $Я + 1$ и переносит слова из $(Я + 1), \ldots, (Я + 4)$ в *стандартные ячейки* ИНФ-1, ..., ИНФ-4.

Программа ИНФО в некотором смысле совершенно особая: у нее принципиально *не стандартный конец* — нельзя портить Ω, где как раз и лежит информация. Конец ИНФО так и называют КИНФО, и обращение к ИНФО выглядит так:

$$Я + 1 \leftrightarrow \text{КИНФО};\ \text{ИНФО}^{2)}$$

2°. Для формирования команд я предложил создать программу группового формирования команд ГФК. После поправок Г. М. Адельсона-Вельского, В. Арлазарова и А. В. Ускова она сделалась такой:

Обращение обычное

$$Я + 1 \leftrightarrow \Omega;\ \text{ГФК}$$

[2)] Фактически все равно в содержательной части пишут просто ИНФО, а про отдельный конец отлично помнят кодировщицы.

После этого обращения идет *подряд* произвольное число пар ячеек. Каждая пара И1; И2 содержит сведения о создании одной команды.

Информация про это такая:

	Код	I адрес	II адрес	III адрес
И1:	0		S	адрес создаваемой команды M
И2:	код новой команды	A	B	C

S — это адрес ячейки, где лежит слово-основа адресной части новой команды. Часто S — нулевая ячейка (пустое слово).

A, B и C содержат дополнительные сведения для создания соответственно I, II и III адресов новой команды M. Я буду говорить только про I адрес.

Итак, первый адрес MI новой команды M получается из двух объектов: SI–I адреса S и I адреса И2, т. е. A. Если мы хотим использовать *сам адрес* A, т. е. 12-разрядное число, то мы поставим в *некоторой триаде* 0, коль скоро нам нужно сложить A с SI, и поставим в этой триаде 4, если мы хотим иметь $M\mathrm{I} = A - S\mathrm{I}$ [3)].

Если же мы хотим использовать не 12-разрядное число A, а один из адресов *слова, лежащего в ячейке с адресом* A, то мы соответственно напишем в упомянутой триаде 1, 2 или 3, если мы желаем иметь

$$M\mathrm{I} = A\mathrm{I} + S\mathrm{I},$$
$$M\mathrm{I} = A\mathrm{II} + S\mathrm{I}$$

или

$$M\mathrm{I} = A\mathrm{III} + S\mathrm{I},$$

и 5, 6 или 7, если нам нужно получить:

$$M\mathrm{I} = A\mathrm{I} - S\mathrm{I},$$
$$M\mathrm{I} = A\mathrm{II} - S\mathrm{I}$$

или

$$M\mathrm{I} = A\mathrm{III} - S\mathrm{I}.$$

[3)] Реально на машине М-20 в ИТЭФ сделано наоборот: $M\mathrm{I} = S\mathrm{I} - A$. Мы заранее не сообразили, что нужно было сделать, как в тексте.

Под эту информационную триаду мы займем три разряда в I адресе И1. Еще две триады пойдут для информации про MII и MIII. Осталось три разряда в I адресе И1. В них ставят 1, если нужно блокировать взятие соответствующего адреса из S.

Если обозначить через $\gamma 1, \gamma 2, \gamma 3$ триады, рассказывающие про MI–MIII, а через $\gamma 0$ — триаду, сообщающую о блокировках, то I адрес И1 заполнится, естественно, так:

I адрес

И1: $\ldots |\gamma_0\gamma_1\gamma_2\gamma_3| \ldots$

Как программа ГФК догадывается, что информационные пары ячеек уже кончились? Именно для этого и служит 0 в первом адресе И1. Конечно, нужно запретить при этом пользоваться кодом 0 в команде, идущей впритык за ГФК. У нас инженеры не ленивые, и 0 служит СТОПом. А где это не так, нужно в коде И1 ставить стоп-код.

Программы ИНФО и ГФК входят, понятно, в состав библиотеки Б1.

§ 3. Про программу ИНТЕГРАЛ. Зачем эта программа должна быть совершенно стандартной

Нам нужно сосчитать $\int\limits_a^b F(x)\,dx$. Мы уже остановились на методе интегрирования. Пусть он таков: отрезок $[a, b]$ разбивается на n равных частей и на каждом маленьком отрезке берется гауссова квадратура, скажем, пятого порядка.

Тогда необходима такая информация для стандартной программы: адреса a и b, адрес x, адрес F — начало блока счета $F(x)$, наконец, число n; итого получилось 5 адресов. В одно слово мы не уместимся, даже если потребуем $x \equiv \alpha$. А раз все равно понадобились 2 слова, удобно не ограничивать аргумент для $F(x)$.

Обращение к программе получается, например, таким:

$$\begin{aligned} &\text{Я}: \quad \int \\ &\text{Я}+1: \; F\ a\ \ b \\ &\text{Я}+2: \sim n\ x \end{aligned}$$

Теперь пусть нам нужно сосчитать

$$\int_a^b f(x)\,dx \int_{c(x)}^{d(x)} \varphi(x, y)\,dy.$$

По каждой переменной алгоритм интегрирования оставим прежним, а числа отрезков разбиения возьмем n_x и n_y соответственно.

Для стандартной программы нам нужна теперь информация: a, b, x, f, n_x, c, d, y, φ, n_y — итого 10_{10} адресов, умещающихся в 4 слова. Беда, однако, в том, что нам для внешнего интеграла не воспользоваться уже имеющейся стандартной программой ИНТЕГРАЛ — эта программа нужна для вычисления интеграла по y. Такое обстоятельство дискредитирует стандартность программы ИНТЕГРАЛ. Между тем дело вполне ясно: программа ИНТЕГРАЛ обращается внутри к свободно заданному блоку счета функции $F(x)$. В двойном интеграле эта функция была бы

$$F(x) = f(x) \cdot \int_{c(x)}^{d(x)} \varphi(x, y)\,dy,$$

т. е. внутренний блок [счет $F(x)$] сам должен обращаться к программе ИНТЕГРАЛ. Произошло нарушение табели о рангах. В программе это тоже недопустимо.

Тем не менее обидно создавать для каждого варианта кратных интегралов отдельную стандартную программу. Ведь может понадобиться и такое:

$$\int_a^b dx \left[f(x) + \varphi(x) \int_c^d \psi(x, y)\,dy - \int_{k(x)}^{l(x)} \chi(x, y)\,dy \right],$$

и такое:

$$\int_a^b dx\, f(x) \left\{ \int_c^d dy\, \varphi(x, y) \left[\int_{k(x,y)}^{l(x,y)} \psi(x, y, z)\,dz \cdot \int_{m(y)}^{n(x)} \chi(x, y, z)\,dz \right] \right\} \text{и т. д.}$$

С другой стороны, человеку-то достаточно точно знать, что такое однократный интеграл, и он не затруднится в понимании вышенаписанного. Нельзя ли все же объяснить то же самое и программе?

Некоторое время это было для нас голубой мечтой. А потом Марианна Розенфельд придумала

СОВЕРШЕННО СТАНДАРТНУЮ ПРОГРАММУ ИНТЕГРАЛ.

А сделала она так. Пусть речь идет о том двукратном интеграле, про который мы рассуждали. Если использовать для внешнего интегрирования стандартную программу однократного интеграла, то когда произойдут неприятности?

Очевидно, в тот момент, когда мы уйдем на счет функции $F(x)$. В самом деле, $F(x)$ сама перейдет на счет $\int \ldots dy$. При этом она переделает команды программы ИНТЕГРАЛ так, чтобы счет шел не по x, а по y, изменит все эталоны, связанные с n_x, на эталоны по n_y, и, наконец, само обращение к $F(x)$ переделает на обращение к $\varphi(x, y)$.

Как со всем этим справиться?

Вспомним популярную задачу о столбе, на котором умещалась карточка с единственной фамилией. К столбу приходили люди, и каждый заменял карточку на столбе карточкой с собственной фамилией. Часов ни у кого не было. Можно ли в этих условиях догадаться, кто был первым? Ответ таков: достаточно, чтобы каждый из посетителей, оставляя карточку со своей фамилией, брал себе карточку с фамилией предшественника.

Вот так же и поступила М. З. Розенфельд. Она завела в программе ИНТЕГРАЛ ячейку КЛАД. Это — карточка на столбе. А при каждом обращении из основной программы к программе ИНТЕГРАЛ наряду с информационными ячейками заведем в основной программе свою ячейку КЛАД МЕСТНЫЙ. Это — карточка, которую взяли со столба.

В начале работы стандартной программы ИНТЕГРАЛ обменяем местами слова в ячейках КЛАД и КЛАД МЕСТНЫЙ. Адрес ячейки КЛАД МЕСТНЫЙ *данного обращения* нами пока не утрачен — он известен благодаря содержимому Ω. Только после такого обмена перейдем к настройке команд ИНТЕГРАЛа по информации. Теперь посмотрим, в какой момент нужно исправить программу ИНТЕГРАЛ.

Очевидно, это нужно сделать по окончании счета подынтегральной функции, т. е. по возвращении в саму программу ИНТЕГРАЛ из ухода на счет функции — именно тут и могла оказаться многократно перестроенной вся программа. А для того чтобы можно было восстановить настройку, достаточно одного — чтобы в ячейке КЛАД лежало то самое слово, которое там было при уходе на счет подынтегральной функции. Но теперь уж легко сообразить, как этого добиться. Достаточно перед

концом программы ИНТЕГРАЛ из КЛАДА МЕСТНОГО перенести слово в ячейку КЛАД.

Теперь уже нехитро осуществить перенастройку программы. (Подробно об этом см. [1] и [2].)

Наряду с перестройкой команд программы нужно, конечно восстановить и еще кое-что. А именно, надо запоминать при интегрировании по данной переменной, до какого узла и на каком из отрезков мы уже дошли, а также не без надобности и сумма, которую мы уже накопили. Значит, кроме ячейки КЛАД МЕСТНЫЙ (точнее — одного ее адреса), нужно еще держать на тех же правах ячейку-число (накопленная сумма) и ячейку с указанием, где мы находимся — номер точки и номер максимального отрезка, подо что хватит и одного адреса. Значит, хватит всего двух дополнительных ячеек. Но при желании считать интеграл не по заданному числу отрезков разбиения, а с заданной абсолютной или относительной точностью, появляется потребность еще в одной ячейке. Всего их оказывается три. А обращение к программе ИНТЕГРАЛ приобретает вид:

$$\begin{array}{lccc} \mathit{Я}: & & \int & \\ \mathit{Я}+1: & F & a & b \\ \mathit{Я}+2: & c1 & \varepsilon & x \\ & & (\text{или } n_x) & \end{array}$$

где $C1$—$C3$ — три идущие *подряд* ячейки для хранения клада, накопленной суммы и т. д.

В коде $(\mathit{Я}+2)$ помещаются сведения об интегрировании с абсолютной или относительной точностью ε или по заданному числу отрезков n_x и т. д.

Живая программа имеет чуть иное расположение информации. (Подробно обо всем этом см. [1] и [2].)

СОВЕРШЕННО СТАНДАРТНЫЕ[4] ПРОГРАММЫ образуют особый очень важный тип: эти программы имеют право обращаться к самим себе произвольное число раз. Поэтому к слову «стандартный» и добавили эпитет «совершенно».

Кроме ИНТЕГРАЛА, совершенно стандартной уместно сделать еще программу КОРЕНЬ ФУНКЦИИ, чтобы иметь возможность искать корень системы

$$f(x, y) = \varphi(x, y) = 0$$

[4] Иначе — *рекурсивные*.

как повторный

$$f[x, y(x)] = F(x) = 0,$$

где $y(x)$ — корень функции $\varphi(x, y)$. Если, конечно, не жалко времени.

Табличную функцию тоже хотелось бы видеть совершенно стандартной. Но здесь встречаются практически трудности другого типа — неодинаковое задание сеток по y при разных x, экзотическая форма области задания и т. п.

§ 4. Работа со случайными числами

Как это ни странно, но первые журнальные статьи о применении метода Монте-Карло в машине, по-видимому, принадлежали людям, на машинах не работавшим. Потому что для выработки случайных чисел предлагались радиоактивные датчики и тому подобное.

Это полностью исключало возможность повторения расчетов (например, для контроля), разве что все случайные числа записывать — с машинной точки зрения такое нужно считать запрещенным приемом.

Между тем средние знаки мантисс членов последовательности, скажем

$$A_n = K(n) \cdot L(n) \cdot M(n),$$

где

$$K(n) \equiv n (\mathrm{mod}\, \pi)$$
$$L(n) \equiv n (\mathrm{mod}\, e)$$
$$M(n) \equiv n (\mathrm{mod}\, \sqrt{2})$$

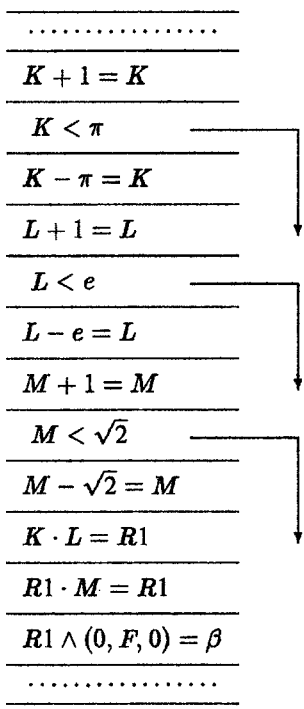
дают отличные равномерно распределенные числа. Теоретически (Г. М. Адельсон-Вельский) получается некоторая неравномерность — сдвиг в сторону малых значений. Фактически это роли не играет. В такой последовательности выдержано среднее, среднеквадратичное и корреляция.

Возможны и другие более быстрые алгоритмы получения квазислучайных чисел с равномерным распределением.

С точки зрения программирования важно обеспечить возможность повторения с заданного места.

Для этого библиотека для работы со случайными числами устроена так.

Пусть для определенности имеются три счетчика K, L, M; следующее состояние счетчиков и соответствующее значение случайной величины получаются по программе:



После чего, конечно, к β приписывается порядок и результат нормализуется.

Тогда АБСОЛЮТНОЕ НАЧАЛО С. В. состоит в засылке 1 в счетчики K, L и M. Одновременно K, L и M пересылаются в запасные ячейки $K_{зап}, \ldots, M_{зап}$. Дальше имеются еще два начала.

ПРОДОЛЖЕНИЕ С. В. перекладывает слова из счетчиков K, L и M в $K_{зап}$, $L_{зап}$, и $M_{зап}$.

ПОВТОРЕНИЕ С. В. слова из $K_{зап}$ ... перекладывает K, L и M.

Кроме того, имеется программа С. В., выдающая очередную случайную величину — она вычисляет по нынешнему состоянию K, L и M эту величину, как описано в приведенной программе.

Естественно, в начале и в конце серии расчетов хорошо бы пропечатать (*печатью программы*!) состояние счетчиков K, L и M.

Беседа седьмая

Что нужно переделать в трехадресной машине, чтобы было хорошо

А ищи всех паче
Разума в задаче.

Магницкий

§ 1. Машинно-выделенные ячейки

Одна такая ячейка обычно есть — незабиваемый нуль в нулевой ячейке. Сверх него, на наш взгляд, необходимы (ср. [3]):

1°. Машинно-выделенная ячейка Ω. Она дает возможность в ряде команд написать три адреса, кроме Ω. А запись в Ω будет диктоваться кодом (обычно в Ω будет класться передача к $Я + 1$).

2°. Машинно-выделенная ячейка S (в основном для записи результата действия при немедленной передаче управления, но не только для этого).

3°. Ячейка-секундомер. Эта ячейка должна быть сделана несколько хитрей обычной. В машину надо ввести электронный счетчик, отсчитывающий элементарные такты машины. Этот счетчик образует хвост нормальной ячейки-секундомера. Скажем, через каждые 2^{12} тактов выполнение программы прерывается [1] для автоматического прибавления в основную — ферритную — часть ячейки-секундомера единицы по правому адресу, после чего программа идет дальше.

Естественно, должна иметься команда, позволяющая передать электронный хвост ячейки-секундомера в нормальную ячейку. Например, это может быть команда, сдвигающая слово в ферритной части

[1] Очевидно, после окончания команды, в середине выполнения которой пришлось ровно 2^{12} тактов. Иначе сделать трудно.

секундомера на 12 разрядов влево и на их место вводящая 12 разрядов из хвоста. В 45-разрядной машине при элементарном такте 1 μ сек переполнение секундомера наступит примерно через год. Профилактика, вероятно, случится раньше.

Конечно, специальная команда как минимум должна уметь очищать и хвост секундомера. Главное назначение секундомера — хронометраж. В сложной программе бывает нелегко разобраться, куда ушло время. А резервы ускорения (даже в несколько раз) фактически находятся всегда. Лишь бы знать, что ускорять.

Кроме того, секундомер может быть полезен и для контроля машины.

Вероятно, впоследствии понадобятся и другие машинно-выделенные ячейки.

§ 2. Адреса для регистров

Обязательно следует придать регистрам *адреса в оперативной памяти*, т. е. пожертвовать несколькими ячейками оперативной памяти. В машине с РА самое необходимое — придать адрес РА. Тогда, например, можно быстро удваивать содержимое РА и т. д. Конечно, слово из ячейки-РА будет восприниматься так: один адрес — содержимое РА (в М-20, конечно, средний), остальные адреса (по-видимому, и код) — нули.

Во всех машинах нужно придать адрес регистру, где хранится выполняемая команда, и короткому регистру, где лежит ее адрес.

Первое дает, например, возможность выполнять команду на чужом адресе, не портя истинное содержимое этой ячейки (метод КУКУШКИНОГО ЯЙЦА), Второе еще более важно: если можно извлечь адрес *Я*, такое самопознание позволяет после выполнения любой программы с пульта (ПЕЧАТЬ ЧИСЕЛ, ПЕЧАТЬ ПРОГРАММЫ, ТУДА) продолжать как ни в чем не бывало основную программу с места, где мы остановились.

§ 3. Стоп при попытке передать управление

Отлаживая программу, мы вылетели на адрес, где лежит Щ. Кто нас выбросил сюда? Всякий профессионал знает, что именно эту ошибку найти очень трудно. Поэтому и нужно предусмотреть СТОП ПО ПОПЫТКЕ ПЕРЕДАТЬ УПРАВЛЕНИЕ задуманной нами ячейке. Этот стоп *не должен происходить* при передаче управления к этой ячейке *из предыдущей* в результате арифметического действия — иначе трудно будет поймать незаконную передачу, скажем, в середину цикла из постороннего блока.

Паллиативной мерой служит вывод на огни одного из регистров адреса предшествующей по выполнению команды при всякой передаче управления (т. е. именно адреса команды, *которая передает управление*).

§ 4. Больше ДЗУ!

Четырех мало. Неудобно напихивать в одно ДЗУ разные по смыслу вещи — число участков интегрирования вместе с печатью по требованию и т. д. Думается, хватит 16_{10}—32_{10}. Все ДЗУ должны иметь номера ячеек памяти. На это ячеек не жалко.

§ 5. Больше разрядов в коде команды!

Обычных шести (не считая разрядов РА) мало. Чтобы использовать возможности машины, хорошо иметь свыше ста (в десятичном смысле) кодов, т. е. минимум семь разрядов на код. Одной из неразрешенных загадок человеческого интеллекта служит то, как в живой машине М-20 при 64 возможных кодах сделали 4 стопа (!). И в том же духе кое-что еще [2]. Может быть этого не случилось бы, если бы состав кодов машин подвергался *предварительному печатному* обсуждению. За монополии всегда расплачивается потребитель!

§ 6. Какие команды нужно добавить и какие — убрать

Без вреда можно убрать округление во всех арифметических командах [3]. Попробуете — убедитесь. Уж больно дороги коды машины. А ведь только очень, мягко говоря, изощренный ум будет пользоваться различием в командах — извлечение квадратного корня без округления и с таковым. А код-то потерян!

И конечно, можно обойтись одним стопом. Зато код этот должен быть нулевым. Поскольку так редки ангелы-программисты. Для других команд тоже хватит по одному коду на штуку (ср. команды 40 и 60 в М-20!). И даже шестиадресная команда обращения к внешним устройствам (в М-20 — коды 50 и 70) обойдется одним кодом.

---

[2] Кое-что — это процентов 25 % всех команд.

[3] Наоборот, арифметические действия *без округления* строго необходимы. См., например, переводы 2 → 10, выделение целой части и т. д.

Зато нужно ряд команд добавить. Они распадаются на естественные группы:

1°. Обращение к функциям.
2°. Команды ФА и ФК.
3°. Дополнительные логические команды.
4°. Смешанная арифметика (плавающие действия с числами, частично заданными в фиксированной форме).
5°. Передачи управления.
6°. Тройная арифметика.
7°. Сверхдальнее плавание.
8°. Группа начальных команд и команд проверки окончания (только для машин с РА).

Конечно, все это требует доделок в реальных машинах. Но, во-первых, это вполне осуществимо и в живой машине (например, почти все, что написано в следующих восьми параграфах, введено на машине Института теоретической и экспериментальной физики). А тем более все это пригодно для новых машин.

Конечно, чтобы удвоить число кодов, приходится вводить еще один (46-й) разряд в ячейках памяти. Но мы уже имели беседу (третью) о том, что за глупость нужно платить.

§ 7. Обращение к функциям

Мы видели два способа обращения к программе вычисления функции $y = f(x)$:

$$\begin{array}{lll} x = \alpha & \text{и} & f \\ f & & 0, x, y \\ \beta = y & & \end{array}$$

Естественно, хочется совместить быстроту α—β-способа с краткостью обращения с сопровождающей информацией и, более того, вообще иметь обращение в одну ячейку.

На первый взгляд это невозможно из информационных соображений: нужно указать адреса $f, x, y, Я$ и Ω, т. е. 5 адресов. На это требуются две ячейки.

Однако А. Л. Брудно придумал, как все-таки обойтись одной ячейкой. Конечно, нам помогает существование двух машинно-выделенных ячеек Ω и S.

Итак, введем команду

$$F, a, b, c.$$

Она делает следующее:

1°. В ячейку S посылается слово из $\mathcal{Я}$, т. е. сама команда

$$F, a, b, c.$$

2°. В ячейку Ω поступает $0 \to 0$; $\mathcal{Я}+1$.

3°. Управление переходит к a.

Заметим, что все нужное для вычисления $y = f(x)$ запасено. Значит, мы в принципе решили вопрос об обращении в одно слово.

Теперь надо подумать о быстрой дешифровке. Брудно это сделал с помощью такой команды:

$$\text{ДЕШ}, k, l, m.$$

Она поступает так:

1°. Слово из ячейки, адрес которой написан во II адресе ячейки S, пересылается в k.

2°. В ячейку m записывается команда, лежащая в Ω, но с добавлениями:

1) По первому адресу добавляется l.

2) По второму адресу добавляется *адрес*, записанный в третьем адресе ячейки S.

3°. Управление уходит к $\mathcal{Я}+1$.

Реально для вычисления, скажем, $y = \ln x$ пишем в обращении:

$$F, \ln, x, y$$

или, кратко,

$$\ln x = y.$$

Про код F может помнить кодировщица. Теперь эта команда выполнится так:

1°. В S пойдет $F, \ln, x, y$.

2°. В Ω пойдет $0 \to 0$; $\mathcal{Я}+1$.

3°. Управление перейдет к $\ln$.

Дальше рабочая часть программы $\ln$ начнется так:

$$\text{ДЕШ}, \alpha, \beta, \text{конец } \ln.$$

Тогда:

1°. В α запишется x.

2°. В конец $\ln$ пойдет команда $\beta \to y$; $\mathcal{Я}+1$, где $(\mathcal{Я}+1)$ — ячейка, *следующая за той, где произведено обращение* $\ln x = y$.

3°. Управление вниз, т. е. к дальнейшим действиям рабочей части.

Итак, дешифровка тоже произошла за одну команду. В общем, см. эпиграф к этой главе.

§ 8. ФА и ФК

Про программу ГФК мы уже говорили. Она плоха одним — для создания каждой команды (по паре ячеек информации) ГФК тратит много времени. Поэтому, скажем, нельзя применять ГФК во внутреннем цикле.

Н. И. Бессонов, ныне покойный, придумал вместе с В. И. Виноградовым, как дешево ввести на действующей машине М-20 *команду* ФК — формирование команды, — выполняющую над парой ячеек И1, И2 ровно те же действия, какие делала ГФК.

Такая команда обходится по времени не дороже, чем два логических действия. Она была осуществлена под руководством Н. И. Бессонова и В. И. Виноградова при участии Н. Ф. Авдеева, В. В. Журкина и О. Е. Михайлова на машине ИТЭФ *без остановки машины* более чем на несколько часов. Теперь этой новой командой все пользуются очень охотно и часто.

Команда ФК по сравнению с программой ГФК имеет одно удобное добавление: если в III адресе ячейки И1, т. е. там, где написано, куда положить сформированную команду, стоит 0, то эта команда поступает непосредственно *на регистр выполняемой команды* и выполняется как якобы записанная в ячейке И2 (без порчи последней), т. е. по чужому адресу (метод КУКУШКИНА ЯЙЦА).

Команда ФА — формирование адресов — особенно нужна в машине без РА. Однако она полезна и в машине с РА. Это снова шестиадресная команда. Пусть написано:

$$\begin{aligned} Я: &\quad \text{ФА},\ l,\ m,\ n \\ Я+1: &\quad K,\ a,\ b,\ c \end{aligned}$$

Тогда при приходе управления к $Я$ выполнится команда, лежащая в $Я+1$, но с адресами: $a+l$II, $b+m$II, $c+n$II, т. е. *сформируются исполнительные адреса*[4]. При этом, конечно, содержимое $Я+1$ не меняется.

Ячейки l, m, n служат индекс-регистрами. Выбор в них II адреса для индекса зависел только от особенностей реальной машины.

Эта команда также осуществлена Н. И. Бессоновым, В. И. Виноградовым и Н. Ф. Авдеевым на машине ИТЭФ без остановки последней. Команда ФА берет времени столько же, сколько 2 логические команды.

Для успешного использования команды ФА приятно было бы иметь команды окончания цикла по индексу (ОЦИНД). Такая команда, по-видимому, должна быть тоже шестиадресной.

[4] Сверх того сохраняется обычная модификация по РА в обеих строчках команды.

§ 9. Дополнительные логические команды

Во-первых, нужно иметь логическое вычитание. Замечено, что такая операция случается чаще, чем извлечение квадратного корня с блокировкой округления.

Более сложной является команда

$$[\delta_1, \delta_2] \Uparrow b = c$$

Здесь δ_1 и δ_2 — секстады, т. е. группы по 6 разрядов. Выполнение этой команды поясним на примерах:

1°. $[22, 37] \Uparrow b = c$;

разряды с 22_8-го по 37_8-й включительно слова из ячейки b переносятся в те же разряды ячейки c. Остальные разряды c гасятся.

2°. $[37, 22] \Uparrow b = c$.

Разряды с самого старшего по 37_8-й включительно и с 22_8-го по 1-й слова из b переносятся в те же разряды c, а разряды c с 36_8-го по 23_8-й гасятся.

Управление в обоих случаях к $Я + 1$.

Всякий, кто занимался логическими задачами, оценит эту команду. Но и в более тривиальных случаях она помогает — например, не нужны высекатели адресов среди констант библиотеки.

Команды СБОРКА (А. Л. Брудно) и РАЗБОРКА (автор) сложны в осуществлении на машине. Именно из-за этого качества их, по-видимому, и стали вводить на новых машинах, так как нужны они в общем-то редко.

СБОРКА a, b, c

Берутся разряды ячейки a, где стоят единицы. Только эти разряды слова из b и принимаются во внимание. Они сдвигаются вплотную друг за другом к *правому* краю ячейки c и туда и записываются. Остальные разряды c — нули.

Пример.

a: 110 011 100 111 100 010
b: 011 101 100 010 100 000

c: 000 000 000 101 101 010

0 вне зависимости от b

РАЗБОРКА a, b, c

Команда, обратная сборке. По-прежнему единицы в a указывают, какие разряды принимать во внимание. Сколько единиц в a, столько

разрядов из правого края b будет взято. Каждый из них станет в c (с сохранением взаимного расположения) под 1 в a. Остальные разряды c — нули.

Пример. a: 110 011 100 111 100 010 всего 10_{10} единиц.
b: 011 110 110 101 101 010
учитываются только эти разряды

c: 01 01 1 010 1 0 это разбросали хвост b

И окончательно c: 010 001 100 010 100 000

Заметим, что

$$\text{РАЗБОРКА}\,[a, c \equiv \text{СБОРКА}\,(a, b, c),\ d]$$

положит в d слово $a \wedge b$.

Не удержимся от замечания, что программа A — перевод слова-команды в печатную форму для удобного чтения — с помощью РАЗБОРКИ получается очень просто.

Наконец, несколько довольно популярных команд, требующих только *двух адресов*.

СЧЕТ ЕДИНИЦ	$\sim b = c.$	В один из адресов c (реально-средний) кладется число единиц в слове b.
ПЕРЕВОРОТ	$\sim b = c.$	Слово в b зеркально отражается относительно середины ячейки.
ОБМЕН	$\sim b \rightleftarrows c.$	Содержимые ячеек b и c меняются местами.

§ 10. Смешанная арифметика

Начинающий программист в общем легко понимает адресный принцип. И тем не менее сплошь и рядом пишет

$$a - \quad 5 = b \quad 002\ 1003\ \underline{0005}\ 1004$$

или, особенно,

$$a +, (0, 1, 0) = b \quad 013\ 1003\ \underline{0001}\ 1004$$

Один знакомый мне профессор (по вычислительной математике) ровно на этом не сумел (отказался) выучиться программировать.

— Глупый, тупой автомат, просто раздражает, — резюмировал он. Так ли уж невероятно был он туп?[5]

Вовсе нет. Так, как этот профессор кодировал, поступать очень естественно. Особенно естественно писать

$$a + 23_8 = b \text{ и кодировать } 001\ 1003\ 0023\ 1004$$

(это он[6] тоже делал, не задумываясь, есть ли в библиотеке такая константа, а если есть — то где она лежит).

Психофизиологический опыт с профессором был оценен. И решили: раз такие ошибки часто повторяются (и даже у разных людей, профессор был просто выразителем дум), значит, рядовому человеку делать так свойственно. Посмотрели, выгодно ли делать так на машине. И нашли, что выгодно. Появилась

СМЕШАННАЯ АРИФМЕТИКА.

Сделана она так: напишем около числа-адреса ФП, и пусть это означает: воспринимай число единиц в данном адресе как *натуральное число в нормальной форме*. Машина это может сделать без труда. Появляются команды:

$$a + \overline{b}_{\text{фп}} = c$$

$$a - \overline{b}_{\text{фп}} = c, \quad \text{а также } \overline{a}_{\text{фп}} - b = c$$

$$a \times \overline{b}_{\text{фп}} = c$$

$$a : \overline{b}_{\text{фп}} = c \quad \text{а также } \overline{a}_{\text{фп}} : b = c \text{ и } \overline{a}_{\text{фп}} : \overline{b}_{\text{фп}} = c$$

(Черта над $\overline{m}$ значит всегда — не ячейка с адресом m, а *сам адрес* m. Конечно это — фулпруф.)

Получилось очень мило: теперь нам не нужны в библиотеке константы — натуральные числа[7]. И все натуральные числа до 4095_{10} включительно можно использовать. И даже в одно действие получить дроби типа $\frac{13}{37}$.

Теперь уже легко догадаться и до команд:

$$a+,\ \overline{b} = c$$

$$a-,\ \overline{b} = c$$

$$a\times,\ \overline{b} = c$$

$$a(\mathrm{mod}\,\overline{b}) = c \quad \text{(целая часть } a : \overline{b} \text{ идет в } S).$$

---

[5] Автомат.

[6] Профессор.

[7] Кроме 1. Как ни странно, единицу нужно оставить (из-за обхода изменения в начале цикла).

Эти команды выполняются так:

$a\pm$, $\overline{b}$ — к слову a добавляется (вычитается) по второму адресу b.

$a\times$, $\overline{b}$ — адресная часть a умножается как целое число на натуральное число b.

$a(\mathrm{mod}\,\overline{b})$ — a приводится по натуральному модулю $\overline{b}$.

Эти команды тоже бывают полезны, а соответствующие условные передачи управления просто очень удобны.

Конечно, приоритет во всех этих командах (кроме разве что $a(\mathrm{mod}\,\overline{b}) = c$) тоже за профессором.

§ 11. Передачи управления

На первой советской машине «Стрела» (Б. И. Рамеев и др.) условная передача управления была довольно неудобной — по сигналу ω. На машинах БЭСМ-2 (С. А. Лебедев) и М-2 (М. А. Карцев) чуть-чуть позже была сделана человеческая команда условной передачи управления

$$a < b; \quad c$$

(уход к c, если выполнено условие). Почему на более поздней машине снова вернулись к передаче по сигналу ω — еще одна загадка (с грустной разгадкой).

Однако жемчужное зерно в ω-куче все-таки есть.

Можно устроить условные передачи так:

$$\begin{array}{ll} a < b; c & |a| < |b|; c \\ a \geqslant b; c & |a| > |b|; c \\ a < b_{\text{фп}}; c & a_2 < \overline{b}; c \\ a \geqslant b_{\text{фп}}; c & a_2 > \overline{b}; c \end{array}$$

(Здесь a_2 — II-й (можно III-й) адрес ячейки a; $\overline{b}$ — II-й адрес самой команды условной передачи; $b_{\text{фп}}$ — натуральное число в плавающей форме (см. § 10), равное фиксированному числу $\overline{b}$.) Передача управления c — во всех случаях при выполнения условия. А вот разности $a - b, \ldots, a - b_{\text{фп}} \ldots$ хорошо записывать в ячейку S. Правда, при этом можно потерять время на лишнюю запись. Но ведь ячейку S можно сделать и на активных элементах...

Кроме сравнений плавающих чисел, полезно иметь еще группу условных передач управления по результатам логических действий.

Она может выглядеть так:

$$\begin{array}{ll} a \vee b \neq 0;\ c & a \vee b = 0;\ c \\ a \setminus b \neq 0;\ c & a \setminus b = 0;\ c \\ a \wedge b \neq 0;\ c & a \wedge b = 0;\ c \\ a \not\equiv b;\quad c & a \equiv b;\quad c \end{array}$$

К ним еще очень полезно добавить:

$$[\delta_1, \delta_2] \Uparrow b \neq 0; c$$
$$[\delta_1, \delta_2] \Uparrow b = 0; c$$

Эти команды позволяют осуществить передачу управления по состоянию нужного разряда ДЗУ и вообще любой ячейки ($\delta_1 = \delta_2 =$№ разряда, b — адрес ДЗУ или ячейки).

Во всех случаях результат операции над a и b попадает в S (операцию $\equiv$ можно воспринимать как сверку γ).

Ну, и, конечно, раз уж Ω — машинно-выделенная ячейка, стоит ввести две команды безусловной передачи управления:

$$a \rightarrow_{\Omega} b; c$$
$$a \leftrightarrow_{\Omega} b; c$$

Обе они выполняются как обычно, но дополнительно в ячейку Ω поступает команда *безусловной передачи управления* к $Я + 1$. Сколь такие команды необходимы, сказать трудно. Но удержаться от их введения еще труднее [8].

Все это сделано на машине ИТЭФ.

§ 12. Тройная арифметика

Мы считаем по формуле

$$Z = \{[(a + b) \cdot c - d]e + f\} : g;$$

получается так:

---

[8] Практика показала, что

$$a \rightarrow_{\Omega} b; c$$

— чрезвычайно полезная команда, а

$$a \leftrightarrow_{\Omega} b; c$$

почти никогда не применяется.

$$a + b = R1$$
$$R1 \times c = R1$$
$$R1 - d = R1$$
$$R1 \cdot e = R1$$
$$R1 + f = R1$$
$$R1 : g = Z$$

А можно бы сделать и так:

$$(a + b) \cdot c \quad (= S)$$
$$(S - d) \cdot c \quad (= S)$$
$$(S + f) : g \quad = Z$$

и вместо 6 команд получилось 3. Обращений к ферритной памяти (включая вызов команды) было 24, стало 11. Если, конечно, S — электронная ячейка.

Так сделать придумал и в своей машине осуществил в 1958 году Б. И. Шитиков. Машина Б. И. Шитикова *не принята в серию*. Она за 14 суток испытания дала два сбоя, из которых один — поломка винта. Скорость ее — 10 000 действий в секунду. Потребная площадь — 30 м^2. Обслуживающий персонал — два техника, по одному на смену. Таких машин сделано в разных местах уже штук 6.

Решение вопроса о том, какую машину надо строить, а главное, какую запретить, обычно поручается специальных дел мастерам.

В данном случае в комиссию входили М. Р. Шура-Бура, Р. А. Мямлин, В. М. Сухов, Э. А. Глузберг[9] (Б. И. Шитиков, Л. И. Дроздов и В. О. Максютенко в комиссию не входили). Руководил операцией Д. А. Жучков.

Для реализации тройной арифметики можно взять такие команды:

$a + b + c$	$(a \bar{\times} b) + c$,	т. е. $-a \cdot b + c$
$a + b - c$	$(a \bar{\times} b) - c$,	т. е. $-a \cdot b - c$
$(a + b) \cdot c$	$(a \bar{:} b) + c$,	т. е. $-\frac{a}{b} + c$
$(a - b) \cdot c$	$(a \bar{:} b) - c$,	т. е. $-\frac{a}{b} - c$
$a \cdot b + c$	$a \times b \times c$	
$a \cdot b - c$	$(a \times b) : c$	
$(a : b) + c$		
$(a : b) - c$		

[9] Ср. Послесловие.

Кроме того, если позволят свободные коды, удобно иметь и вариант тех же команд с записью результата не в S, а в C — по последнему адресу (эти команды в ИТЭФ не введены но причинам непринципиальным — было мало места под дополнительное оборудование и пожертвовали смешанной арифметикой в пользу действий сверхдальнего плавания).

§ 13. Запятая сверхдальнего плавания

Как-то, грустя о том, что вопросы дел машинных дано решать только специалистам-мастерам (см. § 12), мы с Г. М. Адельсоном-Вельским рассудили так. Все знают, что запятая бывает фиксированная и плавающая. И что фиксированная — это плохо, а плавающая — хорошо. А что, если объявить в важном собрании, что впредь нужно строить машины, ну, скажем, с ЗАПЯТОЙ ДАЛЬНЕГО ПЛАВАНИЯ? Найдется профан, который спросит, что это, собственно, значит? Решили — не найдется. ТАКОЙ ЭКСПЕРИМЕНТ БЫЛ ОСУЩЕСТВЛЕН. И действительно, не нашлось. Хотя, разыгравшись, я даже объявил, что нужна запятая СВЕРХДАЛЬНЕГО плавания, и спросил, все ли знают, что это значит. Сказали, что все.

А между тем шутка-шуткой, а нам потребовался счет с большим числом знаков, однажды даже со 150 десятичными (!) знаками. Тут-то я вспомнил про разговор с Г. М.

И была сделана *библиотека сверхдальнего плавания* (арифметические действия, корень функции, линейная алгебра — остальное тогда не понадобилось).

А когда стали модернизировать машину, В. Л. Арлазаров и особенно В. И. Виноградов нашли, как дешево осуществить на живой машине *действия* СДП, т. е. с запятой сверхдальнего плавания.

Таких действий четыре:

$$a +_{\text{сдп}} b = c$$

$$a -_{\text{сдп}} b = c$$

$$a \times_{\text{сдп}} b = c$$

и, наконец, $a \rightarrow_{\text{сдп}} b;\ c$.

Хорошо бы, конечно, иметь еще

$$a :_{\text{сдп}} b = c.$$

Но у нас дешево это не получилось. А в новых машинах можно и сделать. Хотя деление случается и не так-то часто.

Числа в форме СДП записываются конкретно так:

В ячейке-заголовке a указано:

1) Знак числа в обычном месте (44-й разряд).
2) Число единиц I адреса означает, сколько ячеек, идущих *вслед за* a, содержит мантиссу числа.
3) II и III адреса a дают порядок плюс старшая единица II адреса.

В ячейках, идущих за a $(a+1, a+2\ldots)$, лежит мантисса числа. При этом в живой машине голова мантиссы лежит в адресной части $a+1$, ее продолжение — в адресной части $a+2$ и т. д. Коды этих ячеек во внимание не принимаются.

Ответ в ячейки $c, c+1, c+2\ldots$ записывается в той же форме. Длина мантиссы c — минимум из длин мантисс a и b. Про это, конечно, в c кладется нужное указание.

§ 14. Специально про машину с регистром адреса

Всякий блок, как мы уже знаем, начинается командой

$$\Omega = \text{конец}.$$

Если в блоке меняется РА, то, кроме неперфорированной ячейки КОНЕЦ, нужно еще иметь такую же ячейку ПРЕДКОНЕЦ, куда записывается восстановление первоначального значения РА (того, какое было при входе в блок).

Итого получаются четыре ячейки, например:

$\Omega =$ конец
РА1, 0, $\bar{2}$, предконец
. . . . . .
. . . . . .
Предконец
Конец

Можно свести их к двум. Для этого служит команда (Г. М. Адельсон-Вельский, В. Арлазаров, автор)

$$\text{Нач } 1, a, b, c,$$

выполняемая так:

1°. В c поступает команда:

α) Для серийной машины М-20

$$\text{РА} \geqslant 0;\ \text{уход на } b\text{II},\ \text{старое РА} \to \text{РА}$$

β) Для машины с передачей по III адресу

$$\text{РА} \geqslant 0;\ \text{старое РА} \to \text{РА};\ \text{уход на } b\text{III},$$

где, как обычно, bII и bIII–II и III адреса b [10].

[10] Обычно b — это Ω, а c — конец блока (неперфорируемый).

2°. В РА идет $\bar{a}$.

3°. Управление к следующей команде.

В паре с Нач 1 естественна команда

$$\text{Нач } 2, a, b, c.$$

Ее выполнение отличается от выполнения Нач 1 единственным пунктом:

В РА идет не $\bar{a}$, а некоторый адрес (в живой машине — II ячейки a). Аналогия проведена с командами РА1 и РА2.

Наконец, учитывая заголовки, полезно иметь команды:

$$\text{Нач } 3, a, b, c$$

и

$$\text{Нач } 4, a, b, c.$$

Здесь запись в c идет так:

$$c\colon\ \text{РА} \geqslant 0,\ \text{старое РА} \to \text{РА};\ d.$$

Только адрес d берется из машинно-выделенной ячейки Ω. Роль a прежняя (у Нач 3, как и у Нач 1, а у Нач 4, как у Нач 2).

А освободившийся адрес b служит для безусловной передачи управления [11].

Затем в машине с РА надо добавить к командам:

$$\text{РА} < \bar{a}\ldots$$

и

$$\text{РА} \geqslant \bar{a}\ldots,$$

сравнивающим величину РА прямо с написанным адресом $\bar{a}$, еще и команды:

$$\text{РА} < a\ldots$$

и

$$\text{РА} \geqslant a\ldots,$$

где РА будет сравниваться с некоторым (в машине М-20, видимо, II) адресом *ячейки* a. До этого додумались во многих местах.

Наконец, для машины с РА выгодно (по времени) иметь двухадресную команду ОЗВЕЗДВЛЕНИЕ $\sim$, $b = c$, переписывающую в c команду b с заменой написанных адресов *на исполнительные*.

Соответствующая программа берет много времени.

---

[11] Запись в c для реальной машины такая:

$$c\colon \text{РА} \geqslant 0, d, \text{старое РА}.$$

Кроме того, для машины с передачей управления по III адресу ячейки b и c можно поменять ролями.

Беседа восьмая

Как в трехадресной машине устроить длинную память

А ищи всех паче
Разума в задаче.

Магницкий

§ 1. Как это, по-моему, нужно сделать

Пусть в адресной части команды 36 разрядов, т. е. по 12 на адрес. Тогда строго трехадресная машина может одновременно работать с памятью не более чем из $10\,000_8 = 4096_{10}$ ячеек.

Можно, однако, пожертвовать ради большой памяти строгой трехадресностью. И сделать так.

Пусть наша машина располагает памятью, в $100\,000_8 = 32\,768_{10}$ ячеек. Придадим им всем адреса *подряд*, т. е. от 00 000 до 77 777. Для записи такого адреса нужно 15 разрядов. Значит, три адреса в 36 разрядов не влезут. А два влезут. И даже еще останется 6 разрядов.

Вот этими-то 6 разрядами мы и воспользуемся.

Шести разрядов хватит для записи номеров ячеек с 0 по $77_8 = 63_{10}$. Пусть теперь в команде два адреса длинных, а один — короткий. Только, к сожалению, нельзя считать коротким всегда определенный адрес, так как, например, в командах

$$a - b = c,$$

$$a : b = c$$

даже первый адрес не переставочен со вторым. Поэтому каждую команду придется писать в трех видах, указывая, какой из трех адресов короткий.

Адресная часть соответственно разбивается так:

Код	6 разрядов	15 разрядов	15 разрядов	I адрес короткий
Код	15 разрядов	6 разрядов	15 разрядов	II адрес короткий
Код	15 разрядов	15 разрядов	6 разрядов	III адрес короткий

Беда в том, что нужно под работу с длинной памятью занять уйму кодов.

Однако, раз совершив грехопадение, получим по крайней мере удовольствие. Применительно к нашему случаю займем для кода 3 разряда из адресной части, например:

3 разряда	15 разрядов	15 разрядов	3 разряда

И аналогично остальное. Теперь в нашем распоряжении для короткого адреса осталось всего 7 номеров (педантично — 8, но одна из ячеек с номером нуль — нулевая).

Зато теперь число возможных кодов *увосьмерилось*: каждое число в кодовой части (код в старом смысле) за счет приданных ему трех разрядов в адресной части обслуживает 8 команд.

Машина у нас стала $2\frac{1}{2}$- или, если угодно, $2\frac{7}{4096}$-адресной. Зато хочется при работе с длинной памятью иметь полноценный набор команд. То есть прямо-таки все команды, включая те, что описаны в седьмой беседе. Ну, конечно, кроме требующих под адрес именно 12, а не 3 разряда. Команды типа ФА, ФК придется, конечно, переделать. Быть может, ФК надо сделать трехячеечной. И так далее.

Еще недурно заметить, что осложнения случатся с командами СДП — в 7 ячейках может просто не разместиться слишком длинное число. А команды СДП для длинной памяти необходимы: такая комбинация очень важна для задач экономики[1)].

Наконец, как заметил Г. М. Адельсон-Вельский, приятно сохранить наряду с $2\frac{1}{2}$-адресными командами и чистые 2-адресные типа

$$a + b = b.$$

Теперь, поскольку III адрес всегда тождествен II, нам нужно под два 15-разрядных адреса 30 разрядов. Освободилось 6 разрядов вместо

[1)] Симплекс-метод!

3-х в $2\frac{1}{2}$-адресных командах. А значит, истинный код порождает вместе с приданными ему освободившимися 6 разрядами уже не 8, а целых $64_{10} = 100_8$ команд. Этого и хватает.

Быть может, как раз команды СДП следует делать двухадресными.

§ 2. Как это сделано на М-2

На М-2 память содержит 4096_{10} ячеек при разрядности адресной части 30_{10}, т. е. по 10_{10} на каждый адрес.

М. А. Карцев поступил так: всю память он разбил на 8 блоков, по 512_{10} ячеек в каждом. Блокам приданы номера $0, \ldots, 7$.

С пульта можно включить тумблерами любые два блока, причем самими тумблерами указывается, который блок будет младшим, т. е. его ячейки будут в памяти иметь номера от 0 до 777_8, а какой — старшим (с номерами ячеек 1000_8–1777_8). При этом, скажем, 5-й блок может оказаться младшим, а 2-й — старшим.

Дальше переключение блоков происходит с помощью специальной быстрой команды (время — как на логическое действие).

Но имеется возможность тумблером фиксировать один из блоков в качестве старшего или младшего или фиксировать оба блока.

Можно, наконец, один и тот же блок включить и как младший, и как старший одновременно.

Оказалось, что, хотя принципиально все, как надо, действовать так не очень удобно. Под давлением требований жизни пришлось ввести сквозной перенос, т. е. команду переноса заданной ячейки заданного командой же блока в нужную ячейку другого указанного блока, причем *блоки, включенные в качестве младшего и старшего, не меняются*, хотя бы перенос шел и не из них и не в них.

Тем не менее употребление блочной памяти:

а) замедляло работу и

б) главное, *усложняло программирование и увеличивало количество ошибок*.

Поэтому я бы не рекомендовал идти по этому пути.

Что в нем все же соблазнительно, так это возможность неограниченного наращивания памяти при полном сохранении набора команд. И еще — возможность работать на блоках 2 и 5, если, скажем, блоки 0 и 1 испорчены.

§ 3. Стоит ли все же применять блочную память?

Вероятно, да. Во всяком случае на первых порах стоит оставить разбиение длинной памяти на блоки, каждый длиной в *половину* объема, разрешенного длиной адреса в команде.

Но это следует делать СВЕРХ, а НЕ ВМЕСТО введения $2\frac{1}{2}$-адресных команд для работы с длинной памятью.

А дальше жизнь рассудит, что есть истина.

§ 4. Какие изменения вносит длинная память в библиотеку

Длинная память всегда имеет тенденцию к еще дальнейшему увеличению. Да и вводить ее можно постепенно. А трогать библиотеку не хочется. Вернее, не хочется трогать заголовки и библиотечные константы. Рабочие же части библиотечных программ перенести на другое место не хитро. Только, конечно, команда не должна использовать *по существу* специального вида адреса ячейки, где она находится.

И, конечно, нельзя трогать такие ячейки, как Ω, $\Omega - 1$, S, α, β. Программисты часто пользуются рабочими ячейками библиотеки в самых внутренних блоках, которые уже не обращаются к библиотечным подпрограммам. Поэтому придется не трогать эти ячейки при переводе библиотеки па длинную память. Но вообще-то лучше *запретить* раз навсегда *пользоваться библиотечными рабочими ячейками в небиблиотечных программах*. Кроме увеличения числа ошибок, от такой «экономии» происходит еще масса осложнений, когда в процессе отладки мы хотим, скажем, перевести в 10-форму содержимое $R2_{\text{библ}}$–$R7_{\text{библ}}$.

Поэтому желательное расположение ячеек в начале памяти примерно таково:

$$0 - 0$$
$$1 - S$$
$$\dots$$
$$6 - \Omega - 1$$
$$7 - \Omega$$

Первые ячейки до 7-й очень хорошо иметь быстрыми (ячейки-регистры), особенно ячейку S [2].

Что касается ячеек $\Omega - 1$ и Ω, я не совсем уверен, что, например, им не следует придать номера 0076 и 0077, оставив, наоборот, ячейки 2–7 пустыми для работы с длинной памятью.

ДЗУ следует придать адреса тоже где-то вначале. Например, с 10_8-го по 27_8-й; или по 47_8-й. Если, конечно, мы не так богаты, чтобы иметь, скажем, 37_8 ячеек-регистров. В этом случае до 37_8-й ячейки мы используем регистр-ячейки, а для ДЗУ оставим адреса с 40-го по 75-й. Три

[2] Ячейка-секундомер не входит в наши избранные 8.

адреса, вероятно в этом же месте, надо оставить для регистра, содержащего выполняемую команду, регистра, содержащего ее адрес, и РА.

Дальше можно расположить библиотечные константы и заголовки.

А вот рабочие части библиотеки надо разместить в *конце памяти*. Причем, чем меньше номер библиотеки, тем ближе к концу должна лежать соответствующая рабочая часть, ибо тем чаще она бывает нужна либо тем чаще она требует (линейная алгебра) наличия большого сплошного массива.

Проще было бы, конечно, разместить рабочие части библиотеки в начале памяти, скажем, вслед за заголовками, тогда не пришлось бы при наращивании памяти смещать библиотеку (рабочие части). Однако зто будет препятствовать *расширению библиотеки* (увеличению числа заголовков). И в результате нам придется не реже, а чаще передвигать рабочие части библиотеки (ср. эпиграф).

Беседа девятая

Пофантазируем о машинах нашего завтра или переменная адресность и микропрограммирование

А ищи всех паче
Разума в задаче.

Магницкий

§ 1. Как быть с памятью?

Математик-программист отлично знает, что в сегодняшних машинах ему гораздо чаще приходится страдать из-за маленькой оперативной памяти, чем из-за недостаточной скорости машины.

А те из математиков, кто интересовался подробней, на что машина тратит время, знают еще и другое: на выборку слова из ферритной памяти и на запись в нее слова времени идет в 4–6 раз больше, чем на выполнение логического действия в арифметическом устройстве.

Отсюда возникает сразу два желания:

ЖЕЛАНИЕ 1: Надо сделать оперативную ферритовую память большой, например не 4 096 ячеек, а 32 768, 65 536 и даже больше.

ЖЕЛАНИЕ 2: Если уж нельзя всю память сделать быстрой, — на регистрах, — то хоть сколько-нибудь ячеек, например 200_8 или 100_8, устроить скоростными. Если и это дорого — то еще меньше. Но сколько-нибудь — обязательно.

Можно, конечно, сделать память трехступенчатой — большая память сравнительно медленная (как сейчас), затем, скажем, 2000_8–1000_8 — более быстрая память, наконец, сколько можно ячеек — сверхбыстрая память.

Адреса при этом надо придавать ячейкам так, чтобы *меньшие* номера соответствовали *более быстрым* ячейкам.

§ 2. А как же быть с разрядностью ячейки?

В машине с 4 096 ячейками памяти на адрес идет $12_{10} = 14_8$ разрядов. Значит, адресная часть ячейки в трехадресной машине имеет длину 36_{10} разрядов. Если мы теперь перейдем к памяти длиной в 32 768 слов, нам понадобится на каждый адрес 15_{10} разрядов. Адресная часть ячейки выросла до 45_{10} разрядов. Не забудем, что еще есть и код. В современных машинах под собственно код отводят обычно слишком мало разрядов, а именно 6. Мы уже говорили, что как минимум на код нужно 7 разрядов. А 8 или даже 9 еще лучше — следует ожидать появления все новых команд, развитие идет именно в эту сторону.

Теперь вспомним, что бывают на свете еще и регистры адреса, или индекс-регистры. И делают их несколько — 3, 7, иногда 15_{10} и даже еще больше. Правда, я думаю, что чрезмерное увлечение большим числом индекс-регистров не очень оправдано. Но, может быть, это мне кажется из-за отсутствия опыта работы на такой машине. И уж во всяком случае 7 индекс-регистров иметь приятно. Остановимся на этой цифре: 7 индекс-регистров. Теперь для указания модификации каждого адреса нам понадобится 3 разряда. А на 3 адреса — 9 разрядов. Итого на код плюс указатели модификации пошло, скажем, $9 + 9 = 18$ разрядов.

Итак, ячейка получилась длины $45_{10} + 18_{10} = 63_{10}$ разрядов. Нужны ли столь длинные ячейки?

Смотря для чего. Для чисел, пожалуй, что и нет. Все равно если в счете 10–11 знаков (десятичных) не хватает, то еще 2–3 знака спасают редко. Тогда уж нужно удваивать, устраивать и т. д. число знаков и делать это программно или специальными командами (см. про СДП-действия в седьмой беседе).

А вот для команд и особенно для *слов-информации* я бы, пожалуй, сделал ячейку длинной. И даже хотелось бы иметь не 63_{10}, а не меньше 64_{10} разрядов. Может быть, потому, что на свете есть шахматы. А *игра в шахматы на машине — это серьезное дело*. Скорей уж *вычисления по сравнению с этим — детская игра*.

Об этом мы еще побеседуем дальше.

Но иметь длинную ячейку — довольно дорого. Тем более, что если память дойдет, скажем, до $262\,144 = 2^{18}$ или, еще лучше, до $2\,097\,152 = 2^{21}$ ячеек, длина ячейки трехадресной машины вырастет (при той же длине кода 9 разрядов и 7 индекс-регистрах) до 72 или 81 разряда соответственно.

Я думаю так. До 66_{10} разрядов длину ячейки довести хорошо. А как быть дальше или как быть, если упрямые инженеры не захотят заходить, скажем, за 48 разрядов, поговорим в следующем параграфе [1].

Только заметим сразу, что на длине мантиссы больше чем 36 разрядов настаивать незачем — оборудование возрастает, а пользы не прибавится. Для порядка чисел стоит иметь 8–9 разрядов (диапазон $10^{\pm 38}$–$10^{\pm 76}$). Остальные разряды пусть пустуют — не жалко.

§ 3. Об адресности машины

Сегодня перед нами одно-, двух- и трехадресные машины. Представители этих классов — машины серий «Урал», «Минск» и, скажем, БЭСМ-2 или М-20 соответственно.

Какая адресность нужна в машинах будущего?

По-видимому, выбор адресности машин во всем мире производят главным образом инженеры. А математики приспосабливаются к тому, что дают. Или в этом выборе в лучшем случае участвуют не лучшие математики. Если, конечно, судить по результатам их деятельности. А инженеров, естественно, тянет в сторону одноадресных машин. Уж больно простой получается структура машины.

Поглядим, во что обходится эта простота.

Сравнивать машины нужно по времени выполнения одной и той же работы. Но прямое сравнение времен не пройдет — одна и та же задача по-разному программируется, скажем, для трех- и для одноадресной машины.

И все-таки рассудим. Команда, производящая арифметическое действие, например

$$a + b = c,$$

требует на трехадресной машине четырех обращений к памяти (вызов самой команды, вызов a, вызов b, запись в c).

По сравнению с этим продолжительностью самого сложения можно пренебречь. Если, конечно, вызов слов идет из ферритной памяти. Потому что быстрая ферритная памяти сегодня (1964 г.) требует $\approx$ 2 микросекунды на одно обращение. А для действий типа сложения или даже умножения относительно легко достигается время в 1 микросекунду и меньше.

---

[1] К этому месту автор возвращался неоднократно. И остановился на 66_{10} скорей всего из свойственных ему, автору, робости мысли и консерватизма. А совсем уж по чести я думаю, что именно и надо переходить к 72-разрядным, 81-разрядным и даже еще более длинным ячейкам. И что к ним и перейдут после целого леса времянок и паллиативов.

Итак, цена сложения на трехадресной машине — 4 обращения к памяти.

На одноадресной машине та же операция получится так:

$$\begin{aligned} a &\rightarrow \sum \quad \sum \text{ — сумматор} \\ +b & \\ \sum &\rightarrow c, \end{aligned}$$

т. е. потребовалось 3 команды. Значит, вроде бы, надо 6 обращений к памяти (вызов 3 команд, вызов a, b и запись в c). Однако в одноадресных машинах обычно в *ячейку удается уместить не одну*, а 2 *одноадресные команды*. Тогда нужно считать не 6, а только $4\frac{1}{2}$ обращения к памяти. Пока еще все-таки хуже, чем у трехадресной машины.

Защитники одноадресных машин не согласятся, однако, на такой счет. Они предложат сравнить машины, например, по времени вычисления такой формулы:

$$z = [(a + b) \cdot c + d] \cdot e.$$

В одноадресной машине получится:

$$\begin{aligned} a &\rightarrow \sum \\ +b & \\ \times c & \\ +d & \\ \times e & \\ \sum &\rightarrow z, \end{aligned}$$

т. е. 9 обращений к памяти (3 пары команд, 5 чтений и 1 запись).

В трехадресной машине вместо этого будет:

$$\begin{aligned} a + b &= z \\ z \cdot c &= z \\ z + d &= z \\ z \cdot e &= z, \end{aligned}$$

т. е. 16 обращений к памяти, и победа за одноадресной машиной.

Но взглянем на дело поглубже. Откуда взялся выигрыш у одноадресной машины? В обеих программах мы имеем по 6 *необходимых*

обращений к памяти (вызов a, b, c, d и e и запись в z). От этого нам никуда не деться. Но в трехадресной машине мы устроили 3 промежуточные записи в z (в 1–3-й командах) и 3 дополнительных чтения из z (во 2–4-й командах). Всего 6 явно *лишних обращений к памяти*. С ними все ясно. Но ведь проигрыш-то был не в 6, а в 7 обращений. Откуда взялось это седьмое обращение? Посмотрим на дело вот с какой точки зрения. Каждой команде (адресной части) припишем *коэффициент использования*, т. е. отношение числа адресов, занятых полезной информацией, к числу всех адресов.

В одноадресной программе коэффициент использования был всюду 1. А в трехадресной:

$a+b=z$ коэффициент $\frac{2}{3}$ (z — излишняя промежуточная запись);

$z \cdot x = z$ коэффициент $\frac{1}{3}$ (лишний вызов из z и лишняя запись в z);

$z+d=z$ коэффициент $\frac{1}{3}$ (лишний вызов из z и лишняя запись в z);

$z \cdot e = z$ коэффициент $\frac{1}{2}$ (вызов из z лишний, а запись дельная).

Вот за счет того, что мы заняли 6 адресов ненужными информационно вещами, мы и проиграли 2 команды.

В тройной арифметике дело выглядело бы так:

$$(a + b) \cdot c,$$
$$(S + d) \cdot e,$$
$$S = z,$$

т. е. мы имели бы те же 9 обращений к ферритной памяти, что и для одноадресной машины. Напомним, что ячейка S, куда записывается результат действий в тройной арифметике, конечно, регистр-ячейка, и вызов и запись в нее не учитываются.

Но, более того, мы в последней команде имеем свободный адрес, т. е. ценой еще *одного* обращения к ферритной памяти мы могли бы ввести, скажем, добавление к результату числа f. А в одноадресной машине на это ушло бы $1\frac{1}{2}$ обращения.

Откуда теперь взялся выигрыш в $\frac{1}{2}$ обращения?

А взялся он вот откуда. В *одноадресной команде код приходится на один адрес*. А в трехадресной — *на три адреса*.

Следовательно, с информационной точки зрения мы расточительно используем разряды в одноадресной машине.

Более того. Ведь раз ячейка S постоянна, можно было бы сложение с S, умножение на S и т. д. делать *без упоминания адреса* S, а перенести это в код. Тогда мы получили бы такую программу:

$$(a+b)\cdot c(=S)$$
$$[(S)+d]\cdot e=z,$$

т. е. мы имели бы даже не 9, а 8 обращений к ферритной памяти.

Вот именно по таким причинам я осмеливаюсь сказать, что *направление развития машин, как одноадресных, неэкономично информационно.*

Это в действительности еще далеко не главная сторона дела. Но сперва поговорим о трудностях, уходя от которых делают большие машины одноадресными.

Ведь память-то в больших машинах длинная. Лучше всего, конечно, сделать и ячейку длинной. Об этом шла речь в предыдущем параграфе. Ну, а как быть, если все-таки ячейка короткая? Например, пусть в ячейке 36 разрядов, а объем памяти $2^{15_{10}}$.

Тогда, мне представляется, следует пойти на $2\frac{1}{2}$-адресные команды. Короткий адрес (100_8 первых ячеек с 0 до 77_8) соответствует ячейкам-регистрам. Запись туда идет «без времени». При такой системе получится для нашей контрольной формулы программа:

$$a+b=R \quad \text{(быстрая)},$$
$$R\cdot c=R,$$
$$R+d=R,$$
$$R\cdot e=z.$$

Получилось опять 10 обращений к памяти. Но теперь уже легко видеть, как улучшить дело. Раз мы пошли на разбиение ячеек на ранги, сделаем еще шаг. Выделим в совсем особый ранг, скажем, первые ячейки с номерами от 0 до 7. Их упоминание требует трех разрядов. Значит, мы можем такие ячейки упоминать в одной команде *дважды*, лишь бы хватило кодов на указание, что делать.

Теперь программа выйдет такой:

$$a+b=R \quad (\text{адрес } R\leqslant 7)$$
$$(R)\cdot c+d=R$$
$$R\cdot e=z.$$

Мы вышли на уровень одноадресной машины.

А вот в формуле

$$[(a + b) \cdot c + d] \cdot 2e$$

мы даже превзойдем одноадресную машину, заменив последнюю команду на такую:

$$(R + R) \cdot e = z.$$

Заметим еще, что разрешив переменную адресность в командах, мы получим *возможность создавать для ячеек с маленькими адресами* и *вовсе многоадресные команды*.

При длине адресной части 36_{10} разрядов команда может быть относительно первых 7 ячеек 12_{10}-адресной! Для первых 77_8 ячеек умещается до 6 адресов и т. д. Это, конечно, создает большие возможности экономного использования разрядов.

Важность таких многоадресных команд особенно возрастает в машинах со ступенчатой памятью. Во-первых, ячейки в быстрой и тем более сверхбыстрой памяти крайне дефицитны. И если уж их использовать под команды, то пусть такие команды несут максимальную нагрузку — понадобится меньше ячеек. Во-вторых, если мы даже черпаем команду из более медленной памяти, то, коль скоро она работает, скажем, с 6 адресами, на один вызов ячейки-команды придется 4 действия, а не одно, как в трехадресной, или 2, как в одноадресной машине.

И теперь самое существенное.

Программировать на одноадресной машине несложно, но обычная математическая (арифметическая и логическая) символика по самому своему существу почти всегда трехкомпонентна. Значит, трехадресная запись программы является естественной и происходит с большей легкостью, чем, скажем, одно- или двухадресная.

Пренебрегать этим нельзя. По американским данным, *больше половины расходов* на математические машины в целом приходится на долю *программирования* (что-то около 1 миллиарда долларов в год). А удобное программирование означает: меньше ошибок, выше производительность.

Да и что греха таить. Ведь *больше половины* машинного времени идет, по моим наблюдениям, на отладку программ и *меньше половины* — на счет. (Правда, это относится к тем местам, где есть сильные программисты. Там, где программисты слабые, отладка берет мало времени, почти все время — «полезное»: идет счет по чужим программам. Но только почему-то какую там задачу ни поглядишь, сразу видно ускорение в 3–5–10, а бывает и 100_{10} раз. Так что совсем-совсем полезным это время не назовешь.)

Недаром именно *на одноадресных машинах особенно распространились автокоды*. По существу главная радость от автокода — возможность *писать программу как бы для трехадресной машины*, а получать переработанную программу для одноадресной.

Кто видит главную цель автокода в другом, пусть подумает, почему не сделались автокоды для трехадресных машин.

Правда, для трехадресных, как и всяких прочих, машин есть АЛГОЛ. Но про это мы, быть может, когда-нибудь побеседуем особо.

Пока же мы пришли к тому, что

МАШИНА ДОЛЖНА БЫТЬ ПЕРЕМЕННОАДРЕСНОЙ.

Что же касается памяти машины, то она должна быть разбита на иерархию типов не только по скорости обращения к ячейке, а и по числу разрядов в номере ячейки (не считая нулей слева).

§ 4. Откуда взять столько кодов?

Неоткуда их взять. Ведь если адресов всего только шесть, а действий лишь четыре $(+, -, \times, :)$, то и тогда, чтобы составить всевозможные комбинации операций, учитывая, что последний адрес — адрес результата, нам понадобится $4^4 = 2^8 = 256$ кодов. А если адресов 8? Или 12? И, кроме того, разбиение ячейки на адреса переменно?

Итак, понятно, что нельзя завести такое число *постоянных* кодов, чтобы использовать в полном блеске информационные возможности адресной части ячейки.

Но ведь коды-то могут быть и *непостоянными*!

Главное было осмелиться высказать эту мысль. Я не знаю, кто ее высказал первым. Кто-то из иностранцев. Несколько лет назад. И применительно к другим надобностям. Но слово было сказано. Это слово значит

МИКРОПРОГРАММИРОВАНИЕ.

Почему выбран такой термин? По смысловой (и притом очень глубоко продолжающейся) аналогии с обычным программированием. В самом деле, перенесемся из завтра во вчера. В век счетно-аналитических машин (табуляторы, мультипляйеры и т. д.). Каждая такая машина имела для хранения *чисел* счетчики. То, что эти счетчики были механическими, как счетчики арифмометра, не принципиально — это сказывается на скорости, а не на логической схеме работы. Итак, машина имела счетчики для чисел. А вот программа — довольно жалкая, но тем не менее программа, хотя ее так тогда и называли, —

итак, значит, программа в счетно-аналитической машине задавалась с помощью перестановки шнуров на КОММУТАЦИОННОЙ ДОСКЕ.

У табулятора было немного возможностей. Это — сложение, вычитание, передача из одного счетчика в другой (всего счетчиков до 8), чтение чисел с одного из входов, печать. Однако из этого набора получаются относительно сложные комбинации. И были специальные люди, которые занимались именно подготовкой задачи для табулятора. Главной частью этой работы было составление СХЕМЫ КОММУТАЦИИ. По-теперешнему это была самая настоящая программа.

Схема коммутации набиралась на вкладыше коммутационной доски шнурами, втыкавшимися своими концами в дырочки, вроде как на телефонной станции. А схемы для часто повторяющейся задачи просто даже *запаивались* на вкладыше. И тогда получалась *постоянная программа* (пока не сменялся вкладыш).

В современной программно-управляемой машине поступают вроде бы совсем по-другому. Команды программы никуда не напаивают. А заводят их в ячейки ровно так же, как числа. И оперируют с ними, как с числами. Это было большим и трудным достижением. Принадлежит оно одному из крупнейших математиков нашего века — Джону фон Нейману. Именно с этого и пошло развитие программирования. Деятельность Джона фон Неймана можно сравнить с работой Х. Колумба. А деятельность Н. Винера — с работой Америго Веспуччи. Х. Колумб открыл новую страну и не знал, как ее надо называть. А вот Америго Веспуччи *описал* эту открытую Колумбом страну. И в его, Америгину, честь страну стали называть Америкой. То же случилось с Джоном фон Нейманом и Н. Винером. Только программирование стали называть почему-то не винерологией, а наукой-кибернетикой.

Вернемся, однако, к машинам. С высоты своего электронного величия мы с презрением взираем на табулятор с его паяной (в лучшем случае — набранной шнурами) программой. А так ли уж во всех отношениях мы превзошли старину?

В табуляторе было мало элементарных операций. Сложение и вычитание (фиксированные!), передачи из счетчика в счетчик. Но ведь зато эти операции *коммутировались любым способом.*

А в современной машине? *Элементарных* операций в ней побольше, но тоже не так уж много: логическое сложение, логическое умножение, переход к обратному коду, выделение единиц переноса, сдвиг на 1 разряд влево-вправо, передача из регистра в регистр. Вот ведь, пожалуй, и все. Ну еще, конечно, проверка разряда (0 или 1), чтобы по его состоянию определить дальнейший ход выполнения операции. А, скажем, выборка из ферритной памяти с точки зрения арифметического

устройства и устройства управления (АУ и УУ или вместе АУУ) — операция внешняя: АУУ просто передает из своего регистра в регистр памяти нужный адрес. И затем принимает из регистра памяти выбранное слово на свой регистр.

Теперь подумайте, как в ЭВМ реально выполняется та или иная команда, ну, например, команда условной передачи управления.

Не сомневайтесь, что команда

$$a < b; c$$

складывается из *последовательности элементарных машинных операций*, примерно в таком стиле [2]:

1. Вырезать I адрес (a) из РЕГИСТРА КОМАНДЫ.
2. Направить в память запрос о слове с таким адресом и отдать управление устройству памяти.
3. Полученное из устройства памяти слово-ответ послать в РЕГИСТР I.
4. Вырезать II адрес (b) из РЕГИСТРА КОМАНДЫ.
5. Направить в память запрос о слове с таким адресом [3].
6. Полученное из устройства памяти слово-ответ послать в РЕГИСТР II.
7. Вычесть содержимое РЕГИСТРА II из содержимого РЕГИСТРА I.
8. Проверить знак разности в регистре СУММАТОР.

9 а. Если это знак МИНУС, то
10 а. Вырезать III адрес из РЕГИСТРА КОМАНДЫ и
11 а. Передать результат 10 а в РЕГИСТР АДРЕСА ВЫПОЛНЯЕМОЙ КОМАНДЫ.
12 а. Перейти к выполнению следующей команды.
9 б. Если в регистре СУММАТОР знак оказался ПЛЮС, то
10 б. Прибавить 1 к содержимому РЕГИСТРА АДРЕСА ВЫПОЛНЯЕМОЙ КОМАНДЫ.
11 б. Перейти к выполнению следующей команды.

Посмотрите, как похоже получилось описание выполнения команды

$$a < b; c$$

на обычную программу. Между прочим, у нас по дороге случилась и подпрограмма — это был пункт 7. Если вы вдумаетесь, то сообразите,

---

[2] В действительности операции в машине чуть другие, но это не принципиально.

[3] Заметьте, что пункты 1 и 5 различаются еще сдвигом на 12 разрядов!

что вычитание само есть довольно длинная цепочка операций: ведь чтобы вычесть из одного числа другое, нужно:

a) Вычесть из порядка 1-го числа порядок 2-го.

b) Ориентируясь по знаку разности, передать меньшее число, скажем, в сумматор.

c) Сдвинуть вправо содержимое сумматора на число разрядов, определяемое абсолютной величиной разности, полученной в пункте a), и положить результат в регистр, где поначалу лежало меньшее число.

d) Произвести вычитание мантисс чисел с выравненными порядками, причем следует разобраться еще, кто из кого вычитается, и учесть знаки чисел.

e) Полученный результат нормализовать[4].

Теперь сообразите, что если в машине элементарной операцией является сдвиг на один разряд, то пункты c) и e) снова подпрограммы, да еще циклические. И пункты a) и d) тоже не элементарные операции, даже если отбросить вопрос о знаках.

Вдумчивый читатель поймет, что хотя «в принципе» можно сделать команду

$$a < b; c$$

«элементарной машинной операцией», фактически это абсолютно нереально. Ну, тут мы подошли к вопросам определений. Чистый математик (определение см. в последней беседе) спросит немедленно: «А что значит элементарная операция?» На это автор еще сумеет ответить: «...операция, выполняемая за один машинный такт». Но ведь настоящий математик (определение настоящего математика см. в последней беседе) на этом не успокоится. Он скажет: «Что это значит "нереально"? Я, математик (настоящий), могу *доказать*, что это можно сделать за один такт с КОНЕЧНЫМ оборудованием. Более того, такие задачки может решать любой студент, знакомый с переключательными схемами»[5].

И на это автор-программист ответить уже не может. Потому что и сказать ему — автору — нечего. Так как для *настоящего* математика понятия *много* оборудования и *мало* оборудования — звук пустой. А различаются лишь *конечное* и *бесконечное*.

А, скажем, 100 000 000 ламп — величина конечная, и автор признает свое поражение и умолкает.

---

[4] Если вычитание производилось в ходе сравнения, этот пункт излишен.

[5] Ср. беседу 10, § 5.

Итак, мы увидели, что команда

$$a < b; c$$

реализуется в машине, как некоторая *внутренняя программа, составленная из элементарных машинных операций.*

Это — общий случай.

В ЭВМ каждая команда — это *последовательность элементарных операций* (включая условные переходы), *постоянным образом встроенная* в машину. То есть самая настоящая *постоянная программа команды.* Чтобы отличать ее от обычной программы, будем называть *программу команды* МИКРОПРОГРАММОЙ.

Теперь мы не только объяснили, почему кошка называется кошкой (то бишь микропрограммой), но и увидели, надеюсь, в чем мы уступаем временам табуляторным.

Ведь изменить *микропрограмму*, т. е. команду, в ЭВМ можно только с паяльником в руках. Да еще, быть может, придется вставить несколько новых блочков в АУУ. И они не захотят сходу работать (наводки, кто-то генерит, кто-то другой не тянет, в номинале идет, а на режимах дохнет — кто из нас, программистов, не слыхал этих загадочных манящих слов, пахнущих ферритной солью Зурбагана и транзисторными магнолиями Лисса!).

А ведь в табуляторе, вставив новый вкладыш, никто не сомневался в том, что все пойдет и потянет.

Итак, чтобы не отставать от дедовских колымаг, машина нашего завтра должна допускать наряду с программированием простое и легкое *микропрограммирование.*

И вот теперь-то кодов нам хватит. Если, конечно, под код будет отведено не нищенских 6, а, скажем, королевских 9 разрядов, ну, в крайнем случае 7.

§ 5. Задание микропрограммы

Самый простой способ — сделать так, как на табуляторе. То есть иметь коммутационную доску (доски) со сменными вкладышами. И шнурами или паяльником задать на таком вкладыше нужные нам микропрограммы. Конечно, вместе с кодами, которые при данном вкладыше будут соответствовать созданным нами микропрограммам, т. е. новым операциям машины. Все это как-то ужасно не электронно. Но зато, ей-богу, надежно. И в каком-то смысле вернее, чем все остальное, — психологически такой способ задания заставляет программиста тщательно выверять свою микропрограмму. Потому что каждая ошибка

будет очень наглядна — исправлять ее придется паяльником. Да и для инженерной фантазии будет меньше простора: «вкладыш генерит» — не литературно.

Трудно, однако, поверить, что на этом остановятся. Гораздо легче — что вообще не сдвинутся. А уж если сдвинутся, то обязательно захотят отказаться от паяльника. И микропрограммы станут устраивать не на пассивных элементах (коммутационные доски), а на активных. Вероятно, следующий этап — это микропрограммы на медленных и надежных активных элементах. Может быть, на реле.

Теперь уже микропрограмма, как и обычная программа, будет писаться, кодироваться и вводиться в память. А затем специальные команды переключат в соответствии с нашей информацией реле. И коды приобретут тот смысл, который мы им придали микропрограммой. Конечно, с точностью до ошибок.

Наконец, скорее всего захотят устроить задание микропрограммы на быстрых элементах. Потому что реле — это тоже очень не электронно. Почти не лучше коммутационной доски. И, конечно, сделают. Как — не знаю, я математик, а не инженер.

§ 6. Распределение кодов у машины с микропрограммированием

Вероятно, в машине нашего завтра не стоит все сплошь команды делать программируемыми. Как *программист* пользуется *библиотекой стандартных программ*, так *микропрограммисту* естественно пожелать *иметь микробиблиотеку стандартных микропрограмм*. Проще говоря — постоянные команды. Ну, хотя бы такие, как сложение, логическое умножение, пересылка с безусловной передачей управления.

Более того, я думаю, что библиотека стандартных команд будет *много богаче*, чем сегодняшние унылые наборы кодов. Может статься, даже постоянные команды будут разбиты на ранги.

Скажем, малый набор (ну, например, 32_{10}–64_{10} команды) для повседневного употребления математиками невысокой квалификации. Потому что лучше медленная верная программа, чем быстрая никакая. А для ряда задач все время — время отладки.

Затем набор побольше, — например, вдвое — для более квалифицированных программистов.

И, скажем, еще вдвое больший набор для эстетов, работающих с трудными программами. Для эстетов получилось 128_{10}–256_{10} постоянных кодов.

И еще столько же ходов оставим свободными — для микропрограммирования.

Все-таки до чего легко давать указания! Хотя бы и ценные. Все равно делать-то придется другим. А делать придется. Потому что переменная адресность плюс микропрограммирование даст примерно $5–10_{10}$ выигрыша скорости для машины с микропрограммированием против машины на таких же элементах, но без микропрограммирования.

А десятка на улице не валяется.

§ 7. Еще о пользе микропрограмм

Передо мной бланк, взятый наудачу из большой игровой программы «крестики-нолики» (см. [4]). Ну, как в этих случаях берут бланк наудачу, знает всякий. А уж что правда, то правда — это бланк *внутреннего цикла*. На нем 16_{10} трехадресных команд. Все действия — логические. Программа эта производит РАЗБОРКУ (см. беседу седьмую) адресной части двух ячеек $\alpha 0$ и $\alpha 1$ по ячейке $R0$ и результаты кладет в $\beta 0$ и $\beta 1$ соответственно.

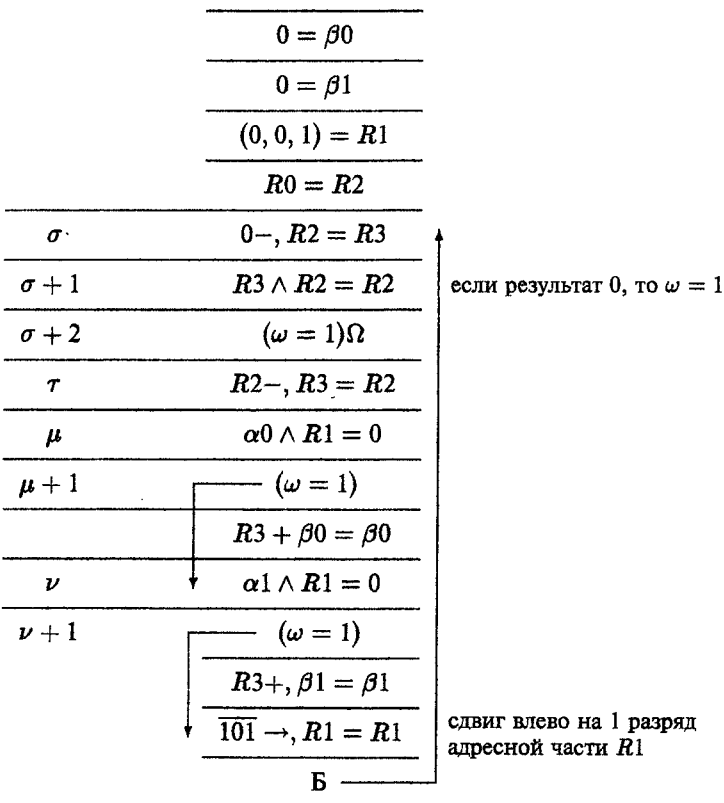
В нашей программе есть цикл. Легко усмотреть, что число кругов — это просто число единиц в ячейке $R0$. Ну, пусть в $R0$ половина нулей, а половина единиц. Тогда при равномерном распределении единиц в $R0$ математическое ожидание числа кругов будет 18. Вне цикла выполняются 4 *команды* (перед циклом). *Внутри* цикла в среднем выполняется 11_{10} команд (всего-то их 12_{10}, но 2 выполняются в «среднем» в половине случаев). Итого наша программа берет $18 \times 11 + 4 = 202$ логических действия. При стандартном цикле в 24 μ сек на выполнение этих операций идет 4 848 μ сек[6] — чистое время 808 обращений к ферритной памяти с временем одного обращения 6 μ сек. Сама же логическая операция занимает 1,5 μ сек и успевает выполниться между делом (пока вызывают следующую команду).

Считаю *истинные* обращения к памяти, т. е. отбрасываю запись в ячейки для промежуточных результатов и взятие из них записанного обратно. Оказывается, их 5 (вызов $\alpha 0$, $\alpha 1$ и $R0$, запись в $\beta 0$ и $\beta 1$)[7]. Если не верите, то вот программа:

---

[6] Время выполнения сдвига мы посчитали, как для обычного логического действия. В машине М-20, принадлежащей ИТЭФ, благодаря КОММУТАТОРУ БЕССОНОВА это действительно так. В обычной машине сдвиг на один разряд возьмет не 24, а 25,5 микросекунд. Только команды сдвига по адресу надо заменить на сдвиг по порядку. А 1,5 микросекундами мы в счете пренебрежем.

[7] Запись (0, 0, 1) в $R1$ не считается истинным обращением к памяти, так как (0, 0, 1) — библиотечная константа.



Значит, строго необходимое время обращений — 30 μ сек. Заменим наши 16_{10} команд одной командой — микропрограммой. К пяти обращениям к памяти добавится еще вызов самой этой команды (6 μ сек) и время выполнения 202 логических операций $\approx$ 303 μ сек. Итого получилось 339 μ сек. То есть мы выиграли в 14,3 раза. А обещали 10.

Но в действительности сэкономится кое-что еще. Команды σ и $\sigma + 1$ в цикле посвящены выделению младшей единицы по Р. А. Иоффе. А команда τ заменяет логическое вычитание, т. е. гашение этой единицы в $R2$. При наличии коммутатора Бессонова все эти 3 операции можно, конечно, выполнять как одно логическое действие. И команда $\sigma + 2$ — условная передача в случае отсутствия этой единицы приплюсуется в это же действие без потери времени (в параллель). Ну,

пусть на все понадобится не 1,5, а 3 μ сек. Все же два действия мы выиграли.

Действия μ и $\mu + 1$ уложатся оба в 1,5 μ сек. И то же верно для ν и $\nu + 1$. Еще 2 действия сэкономлены. Наконец, последняя команда пойдет вовсе без времени. Всего вместо 11_{10} мы имеем в среднем 7 операций. Т. е. нашу экономию времени в цикле можно умножить на 1,5.

Только, конечно, накопление надо вести в электронных ячейках, а перенос в $\beta 0$, $\beta 1$ осуществлять вне цикла, заменив команду $\sigma + 2$. Окончательно получается такая микропрограмма:

	$0 \rightarrow$ РЕГ 0; $0 \rightarrow$ Рег 1; $(0, 0, 1) \rightarrow$ РЕГ 5	1,5 μ сек
	$R0 \rightarrow$ РЕГ 4	6 μ сек
	$\alpha 0 \rightarrow$ РЕГ 2	6 μ сек
	$\alpha 1 \rightarrow$ РЕГ 3	6 μ сек
ЦИКЛ	Выделение и отброс в РЕГ 4 младшей единицы, сама единица записывается в РЕГ 6	3 μ сек
	РЕГ 6 $\equiv 0$	
	РЕГ 2 $\wedge$ РЕГ 5 $\equiv 0$	1,5 μ сек
	РЕГ 6 $\vee$ РЕГ 0 = РЕГ 0	1,5 μ сек
	РЕГ 3 $\wedge$ РЕГ 5 $\equiv 0$	1,5 μ сек
	РЕГ 6 $\vee$ РЕГ 1 = РЕГ 1	1,5 μ сек
	$\overline{101} \rightarrow$, РЕГ 5 = РЕГ 5	1,5 μ сек
	РЕГ 0 $\rightarrow \beta 0$	6 μ сек
	РЕГ 1 $\rightarrow \beta 1; \Omega$	6 μ сек

и расчет времени дает: 6 μ сек на вызов команды + 31,5 μсек на действия вне цикла + 18 циклов $\times$ 9μсек/цикл = 199,5 μсек. Т. е. мы сэкономили время в 24 раза.

Теперь обратимся к другому примеру. Пусть мы хотим сосчитать в ячейке β число единиц в адресной части ячейки $\alpha 0$, а соответствующей команды в машине нет.

Программа будет выглядеть, скажем, так:

$\alpha 0 = \alpha 1$	чтобы не портить $\alpha 0$
$0 = \beta$	
$0-, \alpha 1 = R1$	
$\alpha 1 \wedge R1 = R2$	выделили младшую 1 из $\alpha 1$
$(\omega = 1)\Omega$	если $R2 = 0$, то $\omega = 1$ и идем на выход Ω
$\beta+, (0, 1, 0) = \beta$	
$\alpha 1-, R2 = \alpha 1$	
Б ⟵	

В цикле мы имеем 6 команд на каждую единицу. Т. е. в максимуме 216 команд + 2 входных. А «в среднем» $6 \times 18 + 2$ входных, т. е. 110 команд.

Каждая команда — 24 μ сек. Итого 2 640–5 232 μ сек. А микропрограмма даст на каждую единицу по 3 действия — сдвиг на 1 разряд, проверка на нуль и, если надо, добавление 1 в счетчик. Это уложится в 3 μ сек (добавление в счетчик идет параллельно со следующим сдвигом), т. е. всего понадобится максимум 108 μ сек плюс перенос слова из α_0 в регистр-ячейку, запись результата в β и, наконец, вызов самой команды. Это еще 18 μ сек. Итого — 126 μ сек. Вот она, наша десятка, и даже три десятки [8].

Кого все сказанное не убедило, пусть попробует для тренировки спрограммировать операции ФА и ФК или ОЗВЕЗДВЛЕНИЕ (беседа седьмая). Конечно, эти команды можно и спаять. Но ведь про то и речь!

В вычислительных программах будет, я думаю, поменьше. Но невычислительные программы (логические, игровые) так энергично теснят вычислительные, а главное — требуют высоких скоростей (и приносят зато весомые результаты), что машины нашего завтра будут, вероятно, ориентированы именно на них.

А, впрочем, выигрыш, скажем, тройки в вычислительных задачах тоже достаточен для того, чтобы с лихвой окупить микропрограммирование. А столько-то выиграется.

[8] Приведенную программу можно ускорить в 2–3 раза, высекая группы разрядов и по таблице определяя прибавку к β — число единиц в группе. Однако тогда нужны ячейки памяти под таблицы. Дальнейшее ускорение требует нереальной растраты ячеек памяти.

§ 8. Об этом же

Представим себе, что нам необходимо программу, написанную на одну машину, переделать так, чтобы она шла на другой машине. В действительности такая потребность возникает не так уж часто. Даже просто редко. Но если судить по писаниям о *программировании*, получится, что едва ли не главная задача программирования — переставлять программы с машины на машину. Кстати сказать, переставлять прямо так, не принимая во внимание особенностей машины, на которую переставляют программу, — занятие дорогое: мы не используем специальных возможностей машины и, глядишь, теряем время в 2–3 раза. А то и больше. Это по сравнению с машиной, на которую писалась первоначальная программа.

АЛГОЛ позволяет избегнуть этой неприятности. Здесь время теряется в те же 2–3, а то и в 5 раз уже прямо сразу. *И на новой машине дополнительных потерь нет.*

Так вот, пусть мы перешли с машины на машину. И пусть первая машина была без микропрограммирования, а вторая — с таковым. Теперь нам не нужна программа-переводчик. Нужно составить *словарь команд* — на машине с микропрограммированием в точности микропрограммировать коды первой машины. И программу можно пускать. Конечно, если будет все в порядке с такими мелочами, как разрядность ячейки, объем памяти и т. д.

§ 9. Чего не знал автор, когда писал первые 8 параграфов этой главы или как (и можно ли) обойтись без микропрограммирования

За год, прошедший со времени написания этой книги[9], мне привелось работать на машине с многими индекс-регистрами. А еще гораздо больше на этой машине работал (составил библиотеку) мой сын В. А. Кронрод. И вот, глядя на библиотеку В. А. Кронрода (Б-65) [18], да и на программы задач, которые мы решали на этой машине, я заметил одну удивительную вещь: *программы вовсе не содержат переадресуемых* или *формируемых команд*. То есть завершился круг: великое изобретение — операции с командами, краеугольный камень зари программирования — изжило себя за 15 лет.

Казалось бы, ну и что? А между тем это дает возможность вернуться к донеймановским временам: *числа хранить в одном месте, а команды —*

[9] Этот параграф написан ровно годом позже (октябрь 1965), чем все остальное.

в другом. И вот в ту память, где хранятся команды, запись может быть вовсе и не быстрой — туда ведь мы будем писать только при вводе программы, а на ввод команды извне идет что-нибудь в 1 000 раз больше времени, чем на ее оперативное чтение. Так вот, если из такой памяти, куда *нельзя вовсе писать во время работы*, можно зато *очень быстро читать* слова-команды, то микропрограммирование с точки зрения скорости работы машины делается излишним. Точнее: микропрограмму можно расписать как самую настоящую программу, где операция — элементарная машинная операция, а ячейки промежуточного хранения — самые быстроходные регистры. Ну, конечно, раз таких регистров мало, адреса им можно придать короткие, а в ячейку-команду напихать таких микрокоманд несколько. Но это уже детали, а вот машинные устройства перестраивать командами не придется вовсе. При такой системе работы индекс-регистров потребуется порядком. Теперь уже семи штук мало.

Как в обычной библиотеке существуют *библиотечные рабочие ячейки*, так в такой машине В. А. Кронроду пришлось завести *библиотечные индекс-регистры*. На это пошло 3 индекс-регистра, а для программ осталось уже 12. ...В общем, с 15_{10} индекс-регистрами он обошелся. Но доверительно сообщил автору, что 31_{10} не мешало бы. Откуда автор уже вполне самостоятельно додумался, что истина состоит в $63_{10} = 77_8$ регистрах[10].

В 72-разрядную ячейку (для команд) все это вполне уложится. Все, что сказано в § 1-2 этой главы и про адресность машины (§ 3), сохраняет силу. И если бы не некоторая деталь, то из главы этой можно было бы вычеркнуть почти все про микропрограммирование. Некоторая же деталь состоит в том, что такой памяти пока толком никто не сделал. Хотя, казалось бы, ничего хитрого здесь нет: если пропустить через феррит очень короткий сильный импульс, феррит все равно не перемагнитится (грубо говоря, играет роль It). А вот ток во вторичной обмотке будет заметно разным для ферритов, хранивших ноль, и для тех, где хранилась единица.

Так что все сделано. Кроме того, что автор понимает в этом деле меньше, чем в числе индекс-регистров. К чему приводят такие суждения, см. последнее подстрочное примечание.

А вот попробовать сделать такую память, товарищи инженеры, нужно. Так что выбирайте сами, что вам легче: память с быстрым считыванием (хотя бы и без быстрой записи) или микропрограммирование. Что-нибудь из этого сделать придется. Решайте!

---

[10] Автор не прозевал противоречия с § 2 этой главы, но сознательно не стал припудривать § 2 — пусть читатель видит, как дешево стоят умозрения, и поучится на чужих ошибках. Покойный Бисмарк говорил, что именно этим умные отличаются от дураков. Последние учатся на *своих* ошибках.

Беседа десятая

О работах Н. И. Бессонова

§ 1. Зачем написана эта беседа

Николай Иванович Бессонов (21/X 1907–6/X 1963) был, возможно, самым талантливым из всех людей, с которыми мне в жизни довелось сталкиваться лично. Сын бедного священника, он с большим трудом получил высшее образование. Жизнь не баловала Николая Ивановича. Долгие годы ушли на — пусть полезную, отличную, даже талантливую — работу на заводе. А все, что делал Бессонов, было неизменно талантливо, сколь бы ни был мелок сам предмет работы. Видно, таково уж свойство истинного таланта. Но служебные рамки инженера заводского ОТК дают все-таки слишком мало простора даже и для гения. Да, жизнь Николая Ивановича не баловала...

Более широкие возможности открылись было перед ним в 40 лет. Инженер-механик по специальности, он познакомился со счетно-аналитическими машинами (других тогда не было) на заводе САМ, где служил в ОТК. И стал одним из лучших знатоков этих машин. И — в тот момент это казалось почти несущественным — делал удивительные коммутационные схемы. Табулятор (об этом уже говорилось в прошлой беседе) управляется вместо программы схемой, набранной шнурами на коммутационной доске. Так вот эти-то схемы под руками Бессонова засверкали гранями таких возможностей, которые создателям табулятора даже и не снились [1].

Н. И. Бессонову очень понравился бы эпиграф, который есть во всех предыдущих главах. Потому что во всей своей работе он руководствовался именно этой мыслью, если не самими словами. Мне всегда было удивительно, какими иногда до головокружения неожиданными, а иногда до обидного простыми путями решал он самые, казалось бы, неприступные задачи; про одну из них я расскажу чуть позже.

---

[1] Например, он, создал целую серию схем для умножения на табуляторе 4-, 5-, 6-, 7- и 8-значных чисел. Эти схемы следовало бы напечатать как образец виртуозного комбинаторного решения. Для чисел каждой значности решение сделано особо и поражает свежестью идеи.

Так вот, в 1947 году безвестный 40-летний инженер Н. И. Бессонов нанялся в один академический институт заведовать станцией счетно-аналитических машин, которую ему же самому и предстояло наладить с помощником-механиком, человеком довольно добродушным. Бессонов быстро создал свою станцию. И увидел, что табуляторы и мультипляйеры, предназначенные для бухгалтерского счета, совсем не подходят для решения инженерно-физических задач.

Но в научном характере Николая Ивановича не было и грана уважения к инструкциям и авторитетам. Не колебнувшись, он разрезал счетчики табулятора[2]. Соединил табулятор и мультипляйер в единую схему. А так как этого было все же недостаточно (решалось параболическое уравнение в частных производных), он с легким сердцем добавил к ним автомат из 300 реле. Который сам же на ходу придумал и построил, отрегулировав, между прочим, каждое реле собственными руками. И все это — в подвале, полном стеклянной пыли от работавшей рядом шлифовальной машины. На такие мелочи, как на туберкулез легких, а у Бессонова был туберкулез легких, внимания он не обращал.

Итак, получился КОМБАЙН БЕССОНОВА. Он работал много месяцев, по 3 смены, в 7 раз превосходя по производительности исходный комплект машин. Патент на это изобретение Николай Иванович не взял. И статьи тоже не опубликовал. Кажется, он вообще не опубликовал ни одной статьи в жизни. Мотивировал это Бессонов тем, что он не Лев Толстой. И это была правда.

Странно, но в служебном смысле Комбайн Бессонова никак не помог его автору. Даже скорее повредил. И он не получил за это никакого ордена и даже медали. Зато его Непосредственный Начальник за работу, выполненную на этом комбайне, получил Довольно Высокую Награду. Хотели было дать Н. И. Бессонову крупную премию, на нынешние деньги что-то тысячи две. Но в последний момент вышла бумага, что толку от комбайна Бессонова не было. И денег не дали. Бумагу эту подписал довольно известный профессор — физик-теоретик. А комбайн был первой математической машиной в СССР.

Так вот, хотя в служебных делах комбайн и не помог Бессонову, в научном смысле это был очень важный этап. Теперь Н. И. Бесоннов твердо знал, что машину нужно строить не на механических счетчиках, а на реле. И еще — что математические машины нужны. И примерно представлял, какие у них должны быть возможности.

С этого момента начинается восемнадцатимесячный штурм: Бессонов изобретает релейную вычислительную машину — РВМ. Закончил он эту работу 31 декабря 1949 года.

---

[2] Правда, он это делал и раньше — когда заставлял табулятор множить.

Именно с созданием РВМ и связана задача, о которой я говорил в начале параграфа. Бессонов, конечно, очень быстро выбрал для РВМ двоичную систему. И задумался, как построить быстрый сумматор.

Дело казалось очевидным: пока самые младшие разряды не сложатся, нельзя догадаться — случится или не случится перенос единицы в следующий, второй справа разряд. Поэтому сложение во втором разряде можно начинать только с запаздыванием в один такт. А третий разряд должен дожидаться результатов сложения во втором разряде и т. д. Если всех разрядов n, то и тактов надо не меньше n. Не правда ли, это очень похоже на теорему?

А. Л. Брудно, с которым работал Бессонов, тоже так думал. Но справился на всякий случай у самого, быть может, крупного в СССР математика. Тот сказал, что, по-видимому, это можно даже доказать. Точнее — не совсем это, а более слабое утверждение: не существует релейной схемы, производящей сложение n-разрядных чисел за фиксированное (не зависящее от n) число тактов, если удельный объем оборудования на разряд не растет с ростом n. Еще один тоже крупный математик-академик был того же мнения.

Но инженер Бессонов, хотя и очень уважительно относился к ученым, в своих поисках верил себе и, пожалуй, только себе. И потому, нисколько не усомнившись, придумал сумматор Бессонова, обладающий ровно теми свойствами, которых, как почти доказали академики, быть не может.

Нужно сказать, к чести академиков, что, узнав только сам факт изобретения такого сумматора Бессоновым, оба они воспроизвели этот сумматор. Более курьезно то, что у обоих произошла амнезия — оба искренне забыли, что это казалось им невозможным. Но это уже из области психоанализа.

Бессонов на сей раз получил на свой счетчик авторское свидетельство. Только поэтому я могу привести в этом запоздалом и довольно жалком некрологе рассказ о сумматоре Бессонова. Потому что годом позже знаменитый Шеннон изобрел свой знаменитый, известный всему миру

СЧЕТЧИК ШЕННОНА,

который ничем не отличается от запатентованного за год перед тем

СУММАТОРА БЕССОНОВА.

Сам Бессонов, правда, по этому поводу никогда не огорчался — другие у него были дела и другие огорчения.

Итак, к концу 1949 года Бессонов изобрел свою РВМ. Именно изобрел, а не сконструировал. Почему так — будет сказано в § 2–4.

К сожалению, Сергей Львович не понял всей важности сделанного Н. И. Бессоновым. И не поддержал строительство РВМ. А один человек построить такую машину не мог. Даже Бессонов. Даже с помощью добродушного механика. И дело застопорилось почти на 3 года.

Ох, эти 3 года! В какое время они были потеряны... Только в 1952 году удалось добиться решения о строительстве РВМ Бессонова. При яростном сопротивлении к сему касавшегося Министерства (кажется, тогда оно называлось Министерством Приборостроения).

На этот раз Непосредственный Начальник оказал энергичную поддержку (сопротивлению). Бессонов свалился с тяжелым воспалением легких.

Но постановление есть постановление. Рабочее проектирование машины началось. Предполагалось придать Бессонову десяток инженеров. Придали одного. И тем не менее за 4 месяца работы появился проект — 700 листов ватмана. И что еще важнее — действующий макет. Правда, по работе Непосредственный Начальник снизил Н. И. Бессонову зарплату на 15 %. И не дали квартиры, хотя жил Бессонов за городом. Чтобы впредь не перечил. Но беда была все-таки не в этом.

Неприятности обрушилась совсем с другой стороны. Проект был сделан, макет работал отлично. И на кальках появилась вместо фамилии Бессонова в графе «Ответственный руководитель» другая фамилия. А Бессонов с тяжелым легочным кровотечением оказался на несколько месяцев в туберкулезной больнице.

И потянулись еще 4 года. Которые даже я вспоминаю как болезненный кошмар. А каково же было Бессонову... РВМ была построена. Но как! Из 5 000 реле у тысячи оказалось короткое замыкание в обмотках. Шестерни болтались на валах. Провода отваливались от наконечников, а наконечники от клемм. В общем, хотели доказать, что РВМ, как оно и говорили, построить нельзя. И... не доказали. Машину перевезли в ИТЭФ, куда еще в 1953 году перешел работать Бессонов. И 5 человек — Николай Филиппович Авдеев, Виктор Иванович Виноградов, Иван Семенович Евдокимов, а позже Лев Львович Рыбников и Владимир Георгиевич Бобров — под руководством Бессонова за год пересмотрели и перепаяли более 100 000 контактов, отрегулировали 5 000 реле. И сделали еще много такой же работы. Только в одном месте нам было не справиться: нужно было заменить негодную механику на годную. Это сделать мог только завод. Но не хотел. И все же сделал (когда, переходя от инстанции к инстанции, мы дошли до инстанций очень высоких). Механика стоила на теперешние деньги 800 рублей.

И вот 3 ноября 1957 года была выверена последняя схема, найден и перепаян последний ошибочный конец. И машину включили. На скорости двойной против проектной. И РВМ пошла. *Без отладки.*

Просто она стала решать задачу. И впервые испортилась недели через две. Вероятно, специалистам-инженерам это тоже должно бы быть поучительно.

РВМ появилась на свет с опозданием на 7 лет, когда уже сериями пошли электронные машины. И вот это действительно было трагедией. Личной трагедией автора и трагедией машины. И еще это помешало понять инженерам-электронщикам, как много в РВМ Бессонова есть такого, что им необходимо использовать в электронных машинах уже сегодня, а особенно — завтра. Помешало потому, что РВМ — машина релейная. И это — еще одна драма Н. И. Бессонова. Да, путь его трудно назвать устланным розами.

В 1957 году Бессонову исполнилось 50 лет. И он снова сел за учебу: нужно было превращаться в инженера-электронщика, как в свое время в 40 — в инженера-релейщика. Это было нелегко. Правда, теперь у него была уже настоящая квартира и необходимые сотрудники. А вскоре ему даже дали и ученую степень. Здесь, правда, Николаю Ивановичу не повезло снова. Четыре оппонента (М. В. Келдыш, Л. В. Канторович, Л. А. Люстерник и М. Р. Шура-Бура) высказались за присуждение Бессонову сразу степени доктора. Но это все были математики. А единственный оппонент-техник, член-корреспондент Академии наук СССР, высказался против. Мотивировал он это тем, что «сама диссертация написана плохо».

Не машина плоха, а *бумажка*, в которой эта машина описывалась. Спорить было нельзя. А диссертацию Н. И. Бессонов даже и такую написать все равно не мог. Как он говорил, он не был Львом Толстым. И мог только придумывать схемы, устройства машины... Текст же забракованной диссертации от слова до слова написал автор этих строк. Конечно, под руководством Николая Ивановича. Что ж, я тоже не Лев Толстой...

В общем, последние 4 года жизни Николай Иванович занимался не релейными, а электронными машинами. И снова последовал целый каскад остроумных, ярких и всегда удивительно экономных решений. Принцип работы у Бессонова был такой: исходить не из того, что можно, а из того, что нужно. Первыми осуществленными на машине ИТЭФ бессоновскими устройствами были новые команды ФА и ФК (см. седьмую беседу). Кстати, их вводили *без перерыва* в эксплуатации машины. Не считая, разумеется, нескольких часов на подключение проводов. Стиль чисто бессоновский.

При работе над ФА и ФК Николай Иванович задумался над логической структурой машины М-20. И результатом этих раздумий стал коммутатор Бессонова. О нем мы поговорим в § 5. На мой взгляд, коммутатор Бессонова в завтрашних машинах необходим. Многие инженеры этого пока не видят. Что ж, зрение и прозрение недаром так близки

лингвистически... А вот в авторском свидетельстве на коммутатор Бессонову отказывали трижды. (Его товарищи настояли на подаче заявки.) По самым разным поводам. И только уже после его смерти добросовестный и квалифицированный эксперт понял, что эта вещь прекрасна.

Заседание соответствующего изобретательского комитета было триумфальным. Жаль только, что самого изобретателя на нем уже не было.

Последним, доведенным до конца, изобретением Бессонова было устройство, позволяющее быстро осуществлять некоторые операции, необходимые при программировании шахмат. Сам Николай Иванович называл его СЛОН.

В самом напряженном по времени месте (определение возможных ходов данной фигуры с учетом расположения прочих фигур) СЛОН Бессонова сэкономит время в сто раз.

Всем нам, кто стоял рядом, создание такой конструкции казалось невозможным. И тем не менее Бессонов решил эту задачу. И решил блистательно, даже если сравнивать с его собственными работами. Может быть, созданные им три варианта СЛОНа — самое изящное из того, что он придумал вообще. И пока что совсем не поняты скрытые возможности этого изобретения, которыми всегда так богаты работы Н. И. Бессонова. Потому что когда человек интеллектуальной мощи Н. И. Бессонова выбирает для работы самое трудное и самое нужное место и пробивается к решению всем напряжением духовных и физических сил, то подтекст, который в технике значит не меньше, чем в литературе, звучит торжественной симфонией.

Имеющий уши да слышит.

В отзыве заведующего отделом Вычислительного центра Академии наук СССР[3] В. М. Курочкина было сказано:

«...Поэтому специальное устройство для определения всех возможных ходов шахматных фигур, предложенное Н. И. Бессоновым, с практической точки зрения абсолютно никому не нужно и делать его незачем.

С теоретической стороны работа также не представляет большого интереса, ибо известно, что из элементарных схем, реализующих основные логические операции, можно составить переключательную схему для любой логической функции (в том числе и для предлагаемых автором); более того, задачки подобного типа и трудности могут решать студенты, овладевшие элементарными приемами составления переключательных схем».

И в авторском свидетельстве на СЛОНа отказали. Это был уже последний отказ в изобретательской карьере Бессонова.

[3] Директор ВЦ АН СССР — акад. А. А. Дородницын.

В ночь на 6 октября 1963 года Николай Иванович Бессонов скоропостижно скончался от ночного инфаркта.

* * *

Когда выйдет в свет эта книжка, СЛОН будет уже действовать. А коммутатор Бессонова работает с осени 1964 года. И посмертное авторское свидетельство на СЛОНа, конечно, тоже будет выдано. И не этим жив был великий инженер. Хотя, быть может, целая вереница неталантливых или недобросовестных людей, мешавших ему жить, а особенно, работать и ускорила на несколько лет печальную октябрьскую ночь.

Последнее, над чем упорно думал Николай Иванович, было микропрограммирование. Судя по некоторым беседам с ним, он далеко продвинулся в решении этой задачи. К сожалению, Бессонов почти ничего не записывал, пока не кончил работы — тогда на стол клалась готовая схема. Львом Толстым он действительно не был.

А беседа эта написана все-таки не для того, чтобы рассказать историю работы большого мыслителя. Главным образом мне хочется познакомить читателя с несколькими идеями, осуществленными Н. М. Бессоновым в металле. Эти идеи актуальны для ЭВМ уже сегодня и еще актуальней станут завтра. Про это пойдет речь в следующих параграфах. Но, быть может, несколько страниц, потраченных на рассказ об авторе этих идей, все-таки не пропали совсем даром. Потому что программисту, как и всякому другому настоящему работнику, хочется чувствовать, что и другие люди рядом в тягчайшем труде творят новое. Это ощущение всегда было у меня очень сильно, когда я работал бок о бок с Николаем Ивановичем.

Может быть, хоть отблеск этого чувства сумеет согреть когонибудь в трудную минуту. Если это хоть раз случится, не будем жалеть о нескольких потраченных страницах.

§ 2. РВМ. Каскадный принцип

Электромеханическое табуляторное реле завода САМ имеет время срабатывания и время отпускания по семь *миллисекунд* (7 мсек) каждое. Еще 7 мсек надо отнести на включение реле, которое работает от контактов данного — рабочее время. Итого получается $\approx$ 20 миллисекунд.

Если бы каждое реле работало непрерывно и притом один раз при любой арифметической операции, если бы не шло время на считывание и запись в память, на обработку команды и т. д., максимум-максиморум чего можно было бы достичь по производительности, — это 50 действий

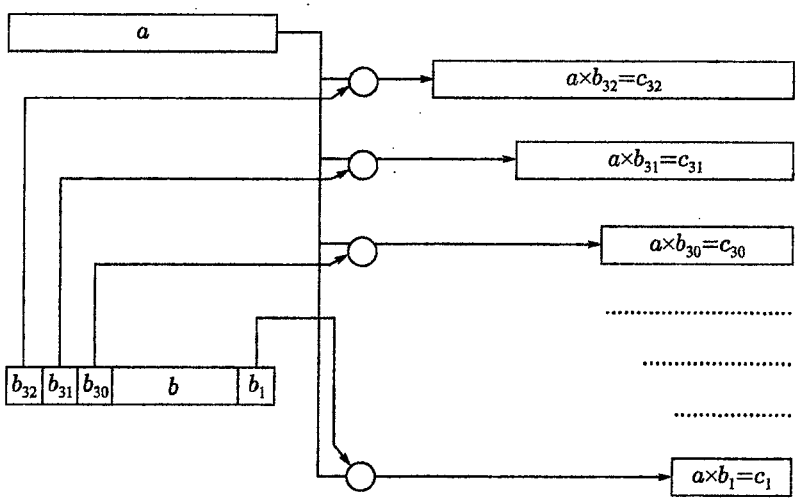
(например, умножений) в секунду. Так как фактически в разных релейных машинах (США, ФРГ, Швеция) реле работает на одной операции не один раз, много уходит на промежуточные операции и т. д., вместо 50 действий (реле там с теми же параметрами, что и у РВМ) получается 2–3 умножения в секунду. Во всяком случае — не более четырех.

А у РВМ Бессонова максимальная производительность — 20 умножений в секунду.

Для того чтобы это стало возможным, Бессонову понадобилось придумать КАСКАДНЫЙ ПРИНЦИП. А значит это вот что.

Пусть нам надо перемножить 2 двоичные мантиссы a и b. Длины мантисс, скажем, 32_{10} разряда у каждой.

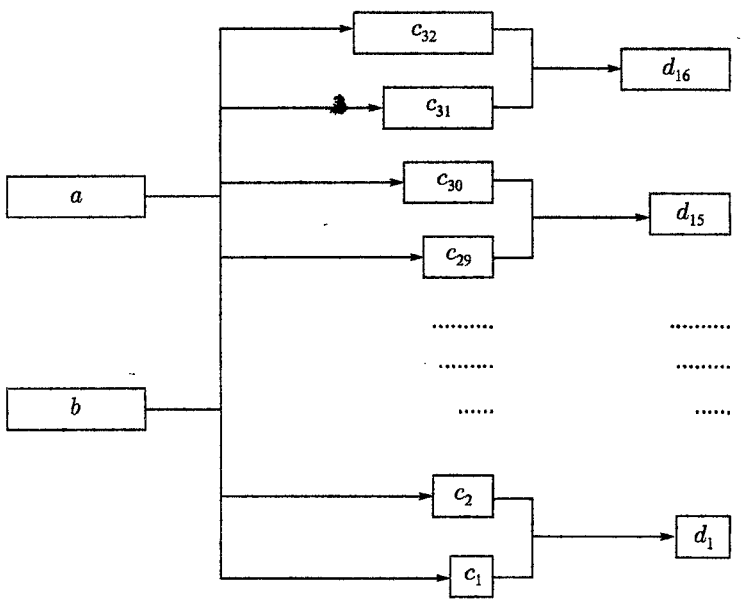
Помножим мантиссу a (I множитель) на каждую двоичную цифру мантиссы b (II множитель), Получится 32 частных произведения. Фактически, конечно, все умножение сводится вот к чему: если цифра-множитель — 0, результат — 0. Если же цифра-множитель — 1, результат — сама мантисса a, только сдвинутая на некоторое число разрядов вправо. Запишем, наши частные произведения в 32 релейных регистра. Получается так:



Сами релейные регистры с a и b образуют ПЕРВЫЙ КАСКАД, а числа — частичные произведения, точнее, регистры с ними (32 строки) — ВТОРОЙ КАСКАД. Понятно, что при длине сомножителя 32 разряда нет нужды смотреть знаки за запятой после 33-го разряда (33-й разряд еще нужен, кроме округления). Поэтому каждый следующий регистр короче предыдущего на один разряд.

Само получение частных произведений сводится к передаче числа a со сдвигом в регистры $c_{32}, \ldots, c_1$. Сдвиг осуществляется просто: провода, скажем, 32-го разряда a идут: в c_{32} — к 31-му разряду, в c_{31} — к 30-му и т. д. А роль числа b сводится к *управлению клапанами*, разрешающими передачу числа a в c_s если в должном разряде b_s числа b стоит 1.

Итак, во II каскаде мы получили частные произведения. Сложим их попарно и результаты пошлем в 16 регистров III каскада d_{16}—d_1. Будет так:

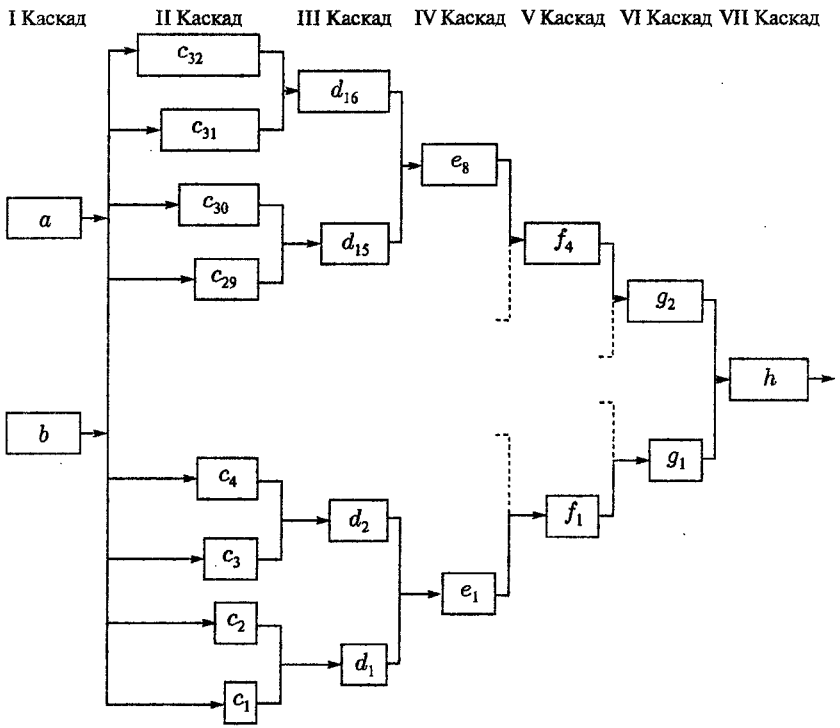


Из регистров d передадим парные суммы в 8 регистров $e_8, \ldots, e_1$ IV каскада, отсюда — в 4 регистра $f_4, \ldots, f_1$ V каскада, затем в 2 регистра g_2 и g_1 VI каскада и, наконец, сложив числа из g_2 и g_1 в единственном регистре h VII каскада, мы получим в этом регистре число-произведение (см. схему на с. 116).

Это, конечно, упрощенная схема арифметического устройства РВМ. Хотя бы потому, что уже во II каскаде сделано вдвое меньше строк, так как от I ко II каскаду сразу идет умножение не на 1, а на 2 разряда числа b. Схема такого умножения на 2 разряда — один из примеров изящного инженерного решения. А экономится-то оборудование вдвое!

Ну, конечно, РВМ работает с плавающей запятой. И потому рядом с каскадами мантисс надо бы нарисовать квадратики для хранения порядков *a* и *b* (сперва их 2, а затем по 1 — на сумму). И еще квадратики для *знаков* чисел.

И — кольцо передачи из последнего каскада в предпоследний, чтобы можно было копить сумму произведений.



И еще парочку НОРМАЛИЗАТОРОВ — ведь, кроме умножения, бывает еще и сложение. А людям склада Бессонова просто не может прийти в голову устроить выравнивание порядков *последовательным* сдвигом по одному разряду зараз. Таков уж у этих людей дефект мышления: они плохо отличают свое от общественного. И жмутся над миллисекундами в машине, как если бы им самим расплачиваться в такси.

Не нужно им давать за это орденов. Но и обижать их доносами тоже не нужно. Они этого не заслужили.

Вернемся к каскадной схеме. Мы начали множить числа a и b. На *первом* такте в регистры a и b нужно было положить числа (из памяти или, например, результат с выхода-регистра). На *втором* такте частные произведения передались из I каскада во II. В этот момент заняты оба этих каскада. А вот на *третьем* такте происходит попарное сложение чисел из регистров II каскада и передача сумм в регистры III каскада. Заняты II и III каскады. Но I *каскад уже свободен*! И можно принимать в него новые сомножители. На *четвертом* такте суммируются частные произведения сомножителей a и b из регистров III каскада в IV каскад. И в *это же время* частные произведения новых сомножителей передаются из I каскада во II. Первая пара на *пятом* такте переходит из IV каскада в V. Вторая пара — из II в III. А в I регистр *уже поданы следующие числа*.

Вот как работает арифметика по каскадному принципу!

Конечно, реле имеют перед электронными или полупроводниковыми элементами одно несомненное преимущество (удобство). В реле *абсолютно разделены* обмотки, включающие реле, от контактов, выдающих импульсы (точнее — включающих цепи для передачи этих импульсов). Электронные и полупроводниковые приборы более клоачны — в них в некоторой степени все смешивается: и сигнал, управляющий триггером, и сигнал на выходе.

Некоторые инженеры говорят, что это исключает использование каскадных идей в электронных схемах.

Но мне всегда вспоминается в таких случаях история с сумматором Бессонова, рассказанная в § 1.

А ведь при времени срабатывания элемента, скажем, в 0,5 μсек электронная машина дает 10 000 умножений в 1 сек. Т. е. производительность — одно умножение *за* 200 *времен срабатывания*. А в РВМ — за 7. Неужели реле во столько раз удобнее?!

§ 3. РВМ. Система управления

До одной идеи Н. И. Бессонов все-таки не дошел. Это — идея программного управления, когда команды хранятся в ячейках памяти наряду с числами и перерабатываются как числа. Я этого тоже не смог подсказать Бессонову — не додумался. Додумался до этого Джон фон Нейман.

У РВМ управление идет от *внешней программы*, набитой — команда за командой — в строках перфокарты. Колоды таких перфокарт вкладываются в программный вход РВМ и при чтении каждой строки берется на исполнение одна команда. Так как каскады работают при одном умножении (сложении, вычитании) по одному разу, но заняты

в течение двух тактов, чтение строки-команды жестко синхронизовано, конечно, с тактами арифметики — по одной команде на два такта.

Интересно, однако, другое.

Как скомандовать машине, чтобы она перемножила два числа a и b? Для этого нужно:

1°. Подать эти числа на регистры 1-го и 2-го сомножителей (в том числе их порядки на регистры порядков I-каскада и знаки — в регистры знака).

2°. Сказать машине, чтобы она стала множить мантиссы.

3°. Сказать машине, чтобы она сложила порядки.

4°. Сказать, чтобы в должный момент из суммы порядков вычлось 40_8 (в РВМ порядок записан в форме $40_8 + P$, где P — истинный порядок).

5°. Повелеть нормализовать произведение и соответственно, если надо, подправить порядок.

6°. Если надо — произвести выдачу результата-произведения в некоторую ячейку C или, скажем, на вход одного из сомножителей. Или — в место, куда надо направлять слагаемое (это — III каскад).

Из всего описанного 2° излишне — каскадное АУ в РВМ действует как мельничные жернова: раз втянув на некотором такте на I каскад сомножители (например, нули, если мы ничего не подали), оно через 8 тактов выплюнет произведение.

Остальное — нужно. Можно, конечно, дав одно-единственное указание «множь!», остальные операции, связанные с порядками, нормализацией и т. д., сделать выполняемыми *автоматически* в нужный момент. В команде их тогда указывать не надо. Но, более того, даже и указание «множь!» излишне. Нужно только подать числа. И — указать, куда послать результат.

Однако как быть в таком случае с суммированием произведений? Можно бы, понятно, в команде второго умножения оговорить, что результат надо добавить к уже накопленному произведению. А в первом умножении — не указывать записи в ячейку. Но что, если мы хотим вычислить

$$a \times b - c \times d?$$

Теперь нужно указать, что второе произведение следует не сложить, а вычесть из уже накопленного.

Ну, а если так:

$$a \times b - c + d \times e + f?$$

Пусть еще при этом числа a, b и c велики, а d, e и f малы. Но разность $a \times b - c$ мала. Тогда, пожалуй, хорошо нормализовать

$a \times b - c$, а потом уже продолжать счет. Это во-первых. А во-вторых, недурно для подаваемых слагаемого f и вычитаемого c иметь тоже указания о добавлении (вычитании) к накопленному (из накопленного) в машине. И еще — о выдаче или невыдаче результата. А вот в случае

$$S = a_1 \cdot b_1 + a_2 \cdot b_2 + \ldots + a_{30} \cdot b_{30}$$

нормализация произведений излишня — от этого мы только можем потерять 1–2 разряда мантиссы[4]. Если, конечно, все произведения одного знака. А между тем на это пойдет лишнее время — можно произвести единственную нормализацию в самом конце.

И, наконец, кроме сложения, вычитания и умножения, в РВМ было предусмотрено быстрое программное деление. И — вычисление $\sqrt{x}$, e^x, $\ln x$, $\sin x$ и $\cos x$. А сверх того — особо быстрые переводы $10 \to 2$ и, что особенно важно, $2 \to 10$.

Чтобы все это сделать действительно быстрым, Бессонов поступил истинно по-бессоновски.

Он придал РВМ многие дополнительные возможности. Некоторые из них требовали сложения без участия порядков, без нормализации и т. д.

И все это решилось одним ударом.

Бессонов *отказался* в РВМ от *команд* $\times$, $+$, $-$ и т. д. Вместо этого математику была предоставлена «вся внутренность» машины. Он сам указывал (в должный момент), что нужно сделать нормализацию, дать выдачу и т. д.

Теперь, скажем, команда умножения выглядела схематично так:

$a \times b$
поправь порядок
нормализуй
$\to c$

Но зато идущие подряд действия, скажем, вида

$$a \times b^2 = d$$
$$e \cdot f + g \cdot h - kl = m$$
$$-p^2 = r$$

[4] В РВМ предусмотрены в кольце запасные разряды *слева*.

записывались так:

$b \times b$; $a+$	
$e \times f$	
$g \times h$ и поправь порядки	(порядки для $a + b^2$)
$k \bar{\times} l$ и нормализация	(для $a + b^2$)
$p \bar{\times} p$; АУ $\rightarrow d$	
поправь порядки	уже для m
нормализация и поправь порядки	(нормализация по m: поправка порядка для r)
АУ $\rightarrow m$; нормализация	(для r)
АУ $\rightarrow r$	

Если теперь нам нужно было бы переслать числа из ячейки в ячейку, мы могли бы это делать одновременно с командами, где не указано адресов ячеек. И — начать еще новые действия, пока доканчивается операция по вычислению r.

Но как реализовать все это?

В команде было введено 6 адресов. Впоследствии добавилось еще 2, и их стало 8. Чуть-чуть отходя от истины, я расскажу, что в РВМ было 4 магистрали (фактически 2 полноценные, а 2 с уменьшенными возможностями, но это — дело экономии оборудования. Скуповат был Николай Иванович. На народные деньги. Может быть, потому, что тяжело они доставались ему самому. И никак он не мог понять, что государство-то у нас богатое. Не было у него все-таки широты!).

Итак, будем считать, что магистралей 4. А магистраль — это вроде водопроводной трубы. Подсоединим один ее конец к любому способному к выдаче месту — и наполнится магистраль тем, что в этом месте содержалось. Такими источниками могут быть ячейки памяти, АУ, блоки паяных констант (такое тоже было — паяные-то константы дешевы!). А открыть магистраль можно в любое место, куда происходит запись. И даже в несколько мест. И число из магистрали втечет во все места, куда ему открыта дорога (ячейки памяти, сомножители,

слагаемое — это место на III каскаде). Ну, магистраль, конечно, — это не один, а 34 провода — по числу разрядов в РВМ, включая знаковый.

А математику нужно только открывать клапаны в магистрали и из них. И еще — давать дополнительные команды: нормализуй, смени знак, поправь порядок и т. д.

Клапаны сами имели номера. Их указывали в адресах команды наряду с ячейками. И команда на РВМ писалась любая, какую можно вписать в 8 адресов с помощью любых клапанов и 4 магистралей.

Конечно, для наиболее шаблонных команд существовали прописи: когда программист хотел получить

$$a \times b = c,$$

он больше не задумывался, какие именно клапаны надо открыть. Он просто писал

$$a \times b$$

и через интервал

П
М
АУ → c

А кодировщица превращала это в

$a \times b$	06	Н14	02	Н53	
П					П
М					М
АУ → c	16	Н12			

тоже не задумываясь, а действуя по шаблону.

У кого зрение хорошее — поймет, что все это значит. А кто не видит — может переходить к следующей беседе. Или даже вовсе закрыть книгу. Потому что то, что сделал Бессонов на РВМ, это — МИКРОПРОГРАММИРОВАНИЕ. Только выглядит оно иначе, чем в будущих ЭВМ. Всего лишь оттого, что на перфокарте в строке можно сделать 80 дырок. И еще оттого, что маленькая (всего 50 ячеек[5]) активная память позволила уписать много адресов в одну строку. И поэтому не нужна была *настройка* кодов. И без того любую возможную команду

[5] Позже, году в 60-м, была подсоединена к РВМ ферритная память на $1\,024_{10}$ ячейки.

можно было собрать из элементарных — клапанных. И этого хватало. А Николай Иванович всегда руководствовался эпиграфом к предыдущей главе. И лишних завитушек не любил.

§ 4. РВМ. Несколько слов о памяти

Скорость реле невелика. А Бессонову хотелось иметь максимальную *производительность* машины. И нужно сказать, что он получил ее реально более чем двойной против записанной в проектном задании. Как, впрочем, и во всех других своих работах, Николай Иванович удивительно не любил не выполнять своих обещаний. Это была еще одна его слабость. Хотя поначалу обещания выглядели так, что я считал его фантазером и авантюристом. И не я один. Только я не излагал этих своих мыслей письменно. А кое-кто излагал. К сожалению, на служебных бланках.

Может быть, гипертония, присоединившаяся к туберкулезу, в заметной степени и произошла оттого, что служебные бланки были с подписями (а иногда и с печатями) и поступали к директору, в партком и т. д. Кое-что удавалось оставить для Бессонова неизвестным. К сожалению — не все.

Так вернемся к малой скорости реле. Нельзя было терять время на *последовательную* выборку сомножителей из памяти. Отсюда возникла *секционированная* память. Теперь выборка происходила одновременно. Не сомневайся, читатель, в ЭВМ последовательная выборка тоже не ускоряет работы!

Ну, а уж коли так, то тем более постоянно встречающиеся константы следовало устроить как отдельную секцию памяти с параллельной выборкой. Что и было сделано. Тем более, что ячейки этой третьей секции были очень дешевы — по диоду на разряд, да и то только на разряд, содержащий 1. А на 0 пожалели (скупость, скупость!). Когда позже сделалась дополнительная (ферритная) память, она стала четвертой секцией. Вот, пожалуй, и все о памяти.

Имеющий глаза программист (микропрограммист) да видит. И конструктор будущих машин — тоже.

§ 5. Коммутатор Бессонова

На РВМ были поставлены нормализаторы. Они давали однотактную нормализацию с любым сдвигом. Чтобы обеспечить быстрое приведение к нормальной форме суммы и разности — при умножении-то больше, чем на разряд, сдвигаться не придется.

А когда Бессонову привелось создавать новые команды на ЭВМ, он как раз и вспомнил о нормализаторе. Потому что многие команды

требовали передачи группы разрядов из одного места слова в другое, например из адреса в адрес. И первые же команды (ФА, ФК), где это понадобилось, привели Н. И. Бессонова к мысли об *универсализации* этой операции.

Кажется, он понял принципиальный (информационный) смысл этой операции раньше нас, математиков. И предложил построить КОММУТАТОР. То есть устройство, позволяющее информационно-заданный отрезок слова пересылать за один такт в заданное место. За один такт — чтобы экономить на ЭВМ микросекунды, как раньше на РВМ экономились миллисекунды.

Чем больше мы думали впоследствии над новыми, нужными программистам, командами, тем чаще всплывала необходимость именно в такой передаче.

А польза коммутатора для нормализации при сложении (вычитании) — побочный результат. Как и для ускорения сдвига. Побочный, но не случайный. Многоплановость свойственна классическим решениям в технике так же, как и в искусстве.

Сейчас коммутатор Бессонова уже действует в ИТЭФ на машине М-20.

Думаю, что в машинах нашего завтра он будет обязателен. Более обязателен, чем, например, операция извлечения корня. И даже, чем деление.

§ 6. О стиле

Однажды во время создания рабочего проекта РВМ Бессонов предложил своему единственному инженеру подумать об устройстве перевода $10 \rightarrow 2$.

Десятичное число набивается, как обычно в счетно-аналитических машинах, на целой карточке — по цифре в строке. И колода с массивом таких карт ставится в числовой вход. Чего мы можем желать оптимально?

По-видимому, чтобы перевод $10 \rightarrow 2$ длился столько же, сколько идет карта. Тогда мы *совсем* не потеряем времени на это занятие. Ну, и, конечно, чтобы устройство потребовало не слишком большого оборудования.

Дело всерьез осложнялось тем, что числа, как десятичные, так и двоичные, конечно, давались (получались) в плавающей форме. Инженер потребовал оборудования еще в полмашины (2 500–3 000 реле). И иначе, говорил он, сделать это невозможно принципиально. Промежуточные диалоги день за днем я опускаю. А заключительный выглядел так. В руках у Бессонова был сложенный лист миллиметровки.

— Так как же, Александр Александрович, подумали еще?

— Не буду и думать. Это — химера.

— Слушайте, Александр Александрович, ну а вдруг вот на этой миллиметровке схема? Ведь стыдно будет.

— Чушь!

— Ну, а вдруг?

— Фантазия. Чего нельзя, того и нельзя. Это понятно любому *грамотному* инженеру.

— Открыть?

— Даже и смотреть неинтересно. Скорее всего чистый лист. Или полная ерунда.

— Тогда смотрите.

На стол легла схема. Из 28 реле. Схема решения была такой: каждую десятичную цифру переведем как *цифру целого числа*. И — направим в одну из строк II каскада. Благо там регистров хватает. Теперь сложим все. Остается домножить на подходящий множитель. Но по диапазону чисел (10^{-9}–10^{+9}) этих множителей всего 19! Так запаяем их задешево на диодах. И — последнее — цифры-то будут поступать в разное время (с разных строк карты). Что ж, пусть себе поступают. Жернов арифметического устройства все равно будет перемалывать (складывать) все, что под него попадет. И к концу интервала между картами мантисса будет готова. Ну, а тут уж повезло — удалось отправить результат на умножение раньше, чем пошли числа со следующей карты [6]! И пока в первые каскады приходят цифры нового числа, перевод старого заканчивается и результат отправляется в заданную ячейку. На n чисел понадобилось $n + 1$ карт.

Это и есть стиль.

А как быть с переводом 2 → 10? Всякий, кто программировал такой перевод, знает, какой продолжительности получается программа. А вот на РВМ цена перевода с плавающей запятой — всего *четыре* последовательных умножения. Потому что, если множить двоичное число < 1 на 100_{10}, целая часть даст 2 очередные цифры результата. И за 4 умножения получаются все 8 цифр. Только вот порядок... Но ведь двоичных-то порядков в диапазоне 2^{-40_8}—2^{+40_8} всего 65_{10}. Напаяем переходные множители (включая первое умножение на 100_{10} в множитель) и будем выбирать 10-порядок результата и переходный

[6] Не пришлось к слову рассказать, что интервал между картами (2 строки) был, конечно, тоже использован для выполнения действий. Только их нужно было, естественно, записать в предыдущих строках в дополнительных столбцах. Просто не мог Бессонов спокойно пережить, что на каждые 12 работающих строк 2 будут пропадать задаром. Зато и поплатился 15 % зарплаты. Непосредственный Начальник тоже жалел народные деньги...

множитель по 2-порядку. Теперь все хорошо, если не считать одной детали — двоичный порядок 4 иногда соответствует десятичному порядку 1, а иногда 2. Все зависит от мантиссы.

Решение было такое: в печати результатов *позволить* иметь мантиссу и с 0 и с 1 перед запятой. Чистота идеи нарушена. Но зачем же бороться за чистоту, лучше просто подметать. И яйцо поставить на острый конец тоже легче всего, надколов его. А *четыре умножения на перевод* $2 \to 10$ *стоят надколотой скорлупы*!

Это тоже стиль.

Про деление. Оно сделано в РВМ программно. Только для первого приближения мантиссы частного Николай Иванович устроил специальную ячейку памяти. Нормально она вела себя, как все ячейки. А по специальной команде из нее выдавались первые 4 разряда мантиссы числа, обратного к мантиссе ее содержимого. И теперь уже деление происходило быстро. Не без того, конечно, что кой-какие детали были для этого предусмотрены в АУ.

А знаете ли Вы, читатель, что $\sqrt{x}$, e^x и $\ln x$ можно вычислить за такое же время[7)]? По крайней мере в РВМ. Ну, напаяв кое-какие константы. И введя микрооперацию получения целой и дробной части — нормализатор-то в РВМ есть! И еще кое-что для работы с порядками при вычислении корня (и выдачи при нужде дополнительного множителя $\frac{\sqrt{2}}{2}$. А вот для $\sin x$ и $\cos x$ понадобилось времени в 1,5 раза больше. Сравните с этим, товарищи программисты, свои самые быстрые программы.

И это — тоже стиль.

Ну, и, наконец, кое-что для инженеров. У реле есть паспорт. В нем написано, сколько раз должно сработать реле до износа. Эту цифру сильно *занижать* завод не будет.

А в РВМ в массе они (реле) не износились за несколько лет вообще. Были только поломки небольшого числа недоброкачественных реле. А между тем на самых нагруженных участках *сотни реле выполнили работу*, в 150_{10} *раз превосходящую паспортные нормы*. И не поломались.

Как это получилось?

Просто Бессонов поразмыслил о том, отчего реле ломаются. И сообразил, что провода изнашиваются мало (так что зря в одном из рассказов Зощенко писал: «...свет, конечно, не горит — провода экономят...»). А изнашиваются контакты реле. Но и контакты реле изнашиваются не оттого, что включаются и выключаются. А оттого, что включаются, а особенно выключаются — *под током*. И что если этого избежать, то сносу реле не будет. Тогда Бессонов устроил временну́ю диаграмму

[7)] Т. е. чуть меньше, чем время 4-х последовательных умножений.

так, чтобы ни одно реле не включало и не выключало контактов под напряжением[8]. А цепи стали включать и рвать мощные импульсаторы главного вала.

Это — для инженеров.

И последнее — тоже для инженеров. Хотя, быть может, и не только для них. Николай Иванович твердо верил, что верно нарисованная схема обязана работать. И что проверять нужно только то, что заранее предусмотреть нельзя. И еще — что такие вещи, как разброс параметров элементов, надо предусмотреть как неизбежные и дать на них внушительный запас по времени. И исходить из устойчивости работы, жертвуя скоростью. Тогда *номинальная скорость машины понизится, но зато производительность повысится.*

И все это — тоже стиль. Стиль Николая Ивановича Бессонова.

[8] В табуляторе так велось испокон веку. Но в релейных вычислительных машинах это куда как не просто.

Беседа одиннадцатая

Человек и машина

В этой беседе автор излагает только свои собственные мнения. То есть он совершенно уверен, конечно, что все здесь сказанное верно, но не так уж уверен в том, что через несколько лет он будет так же уверен во всем сказанном, как сегодня.

В этом смысле одиннадцатая беседа отличается от предыдущих десяти. Потому что в них я не только излагал лишь то, в чем был уверен, но еще и был уверен в том, что эта уверенность постоянна. Что, конечно, не мешало мне где-нибудь и ошибиться.

А что до категоричности тона, который я себе позволю в этой беседе, то это чистая экономия бумаги. И собеседник волен в любом подходящем месте от моего лица мысленно добавить в начале фразы: «Я думаю, что...». Если кому-нибудь от этого будет приятно, то пожалуйста.

§ 1. Может ли машина думать?

Конечно, да. Кто не верит в это, пусть попробует побеседовать с арифмометром. Если арифмометра под рукой не случится — годятся обычные канцелярские счеты. Только этот эксперимент нужно *проделать фактически*. Теперь умножьте результаты эксперимента на 1 000 000 — таково соотношение скоростей ЭВМ и арифмометра. А для счетов можно взять даже 10 000 000. И вы получите представление о мыслительных способностях современных ЭВМ.

Если говорить не шутя, вопрос этот и очень серьезен и не существует вовсе. Смотря по тому, как посмотреть на дело.

Я, конечно, убежден, что вычислительная машина думать может. То есть, точнее, что нет такой логической, математической, теорфизической и т. д. и т. п. задачи, которая не окажется *со временем* под силу машинам будущего. Только не машинам нашего завтра. И, быть может, не послезавтра. Скорее — машинам будущей недели. Или месяца...

Это моя вера в некотором смысле *бездоказательна*. В самом деле, что мы знаем о человеческом мышлении? Огромное большинство научных работников твердо считает, что процесс мышления сводится к простым законам физики. Но откуда, собственно, берется эта уверенность?

Из единственного источника — опыта предыдущих экспериментов. Какой бы процесс мы ни рассматривали, в конце концов всегда находились материальные носители, подчиняющиеся ровно тем же законам, что и классические шарики и материальные точки (конечно, со всеми релятивистскими, квантовыми и иными, еще грядущими, поправками...).

Правда, в случае мышления мы сталкиваемся с явлениями столь поразительными, с аппаратом столь необычайно совершенным и непостижимым уму, что, право, иной раз придет в голову: а не проще ли всего объяснить все эти чудеса силами божественными и не искать механизмов? К сожалению, автор — закоренелый атеист. И даже более того — ему кажется крайне *маловероятным* (хотя и не вовсе исключенным), чтобы в *процессе думания играли роль законы природы, существенно отличные от уже известных сегодня*. Но это уже — вопрос веры (научной).

Ну, а если поверить, что мышление осуществляется механизмами (конечно, физическими, химическими и т. д. и т. п.), то, и правда, ведь нет причин, почему бы эти механизмы нельзя было повторить в металле — пусть в миллион раз более громоздкими, чем их создала природа. Пусть даже более медленными и неуклюжими — но ведь повторить-то можно!

А значит — машина думать сможет. Только не машина нашего сегодня (см. эксперимент в начале параграфа).

И — чтобы покончить с проклятыми вопросами, — а как же быть с самосознанием? Кто сказал А, тот скажет и Б. Буду отвечать откровенно. Да, если мир устроен так, то и самосознание — это всего лишь состояние, цикл, сложный комплекс состояний механизма думания — все что угодно, но нечто вполне вещное. И как таковое, оно — самосознание — подлежит повторению в машине. Как говорится: Верую, Господи, помоги неверию моему! Потому что теорема-то теоремой, а представить себе это, прямо скажем, несколько трудно. Но это — ситуация в науке не такая уж и необычная.

Вообще, вторая половина XX века характерна, я бы сказал, не то чтобы верой, а как бы удивительной готовностью ученых поверить в возможность «чуда». Действительно, появление на протяжении двух–трех десятилетиий таких вещей, как атомная энергия, пенициллин, ЭВМ, лазеры, хоть кого заставит ждать все новых и новых чудес. И параллельно с научными чудесами расцветают научные легенды: летающие тарелки, антигравитация, телепатия, партеногенез у человека... Заметьте, однако, что в этих научных легендах нет *ничего противоречащего известным законам природы*. Так что и сами легенды создаются на высоком научном уровне!

Что же касается самосознания машин, то это уже похоже не на научную фантастику, а на сказку. Ну, скажем, русскую народную... Или — сказку тысячи и одной ночи. Потому что речь идет о волшебном слове.

Ведь чем отличается «Сезам, отворись!» от *программы, дающей самосознание*? Только длиной!.. Так-то.

Н. Винер во время пребывания в Москве в одной из бесед прямо высказался *против активного поиска «думательных» алгоритмов во избежание возможности «бунта машин»*. Возражение от лица всех собеседников привел ему М. А. Карцев: «Мы не собираемся бунтовать ЭВМ. Хотелось бы знать только дорогу к машинному думанию. Хоть направление... А там мы клянемся, что остановимся вовремя...». На это уже Винер не ответил ничего.

§ 2. Кто работает скорее — машина или мозг?

Смотря по тому, про какую работу мы говорим. Если множить числа — то машина. В миллион раз. И даже в миллиард, если числа многозначные. А если знаков взять 20, то человеческий мозг без бумажки вообще не справится с такой работой.

Но разве мозг только и делает, что множит большие числа? Даже в области математики мы можем найти другие, более благородные примеры. Ну, хотя бы задачи на построение циркулем и линейкой. Или доказательства теорем. Или — задачи о взвешивании монеток. Почему-то получается так, что сколько ни бьются математики — «научить машину» делать эти вещи как-то не удается. А точнее, «в принципе» удается, но скорости оказываются ошеломляюще малыми: отношение их примерно такое же, как при умножении чисел, но в обратную сторону — машина начинает уступать мозгу в миллион и даже в миллиард раз. И вообще не справляется с задачей, если она еще чуть посложней (то есть за время, пока не погаснут звезды. А вообще-то, конечно, справится. Так что принципиально все решено).

Отвлечемся от математики. Человек вошел в комнату. Просто человек, без высшего образования. В комнате за столом сидело трое собеседников. И сразу же погас свет. Ну, скажем, через секунду. А вошедший снова вышел. Вряд ли он не сможет теперь ответить на целую кучу вопросов. Например, сколько людей было за столом? Какого они были пола? Возраста? В каких позах сидели и что делали — пили, ели, разговаривали? И — какой формы был стол? Был ли он накрыт? И стояло ли на нем что-нибудь? Например, еда или цветы? А если вошедший был человеком наблюдательным, он сможет довольно подробно описать еще и костюмы собеседников, цвет глаз и волос, обстановку в комнате... Не правда ли, немножко много для тех скоростей нервных процессов, которые называют нейрофизиологи, — десятки циклов в секунду (пресловутые α-, δ- и т. д. ритмы)?

Ребенок посмотрел в окно. Ребенку 9 лет. О, какие подробности он сможет рассказать об увиденном! А в окно он смотрел, оторвавшись

от таблицы умножения. Таблицу же ему задали учить в школе. Не всю, а тот столбик, где трактуется об умножении однозначных чисел на 7.

В наши годы модный термин — объем информации. Как Вы думаете, читатель, что несет больше информации: картина в окне или злосчастный столбик произведений на семерку?

И — последнее. Сколько времени нужно человеку, чтобы узнать хорошо знакомое лицо? Даже если оно закрыто наполовину газетой. Или измазано сажей. Или, если опознаваемый не стоит, а лежит? Доли секунды будет достаточно.

А теперь попробуем заставить ту же работу проделать машину. Ответ на технический вопрос — как задать картинку — труда не представляет. Его давно решили в телевидении. Там это называется растром. И в машине тоже. Только в машине растр — совокупность нулей и единиц. Несколько ячеек-слов дадут строку. А массив строк — это картинка. Только непонятно, что с такой картинкой делать — попробуйте-ка создать программу, опознающую человека по фотографиям!

Поставим задачу точнее — опознавать не человека по фотографии, а фотографию по фотографии. Скажем, удовлетворимся программой, которая сумеет работать так:

Программе показывают фотографии нескольких людей. По десятку фотографий на душу. И позволяют ей трудиться над каждой хоть сутки. Или — еще лучше — прямо записать себе в память все эти фотографии целиком. А теперь покажем ей новую фотографию одного из этих людей. Но только человек может быть сфотографирован в другом костюме, с иным поворотом головы, а то и с частью лица, прикрытой другими предметами. Но так, чтобы для человека, видевшего те же фотографии, что и машина, не составило труда опознать лицо на новой фотографии.

Если теперь программа тоже сумеет справиться с такой задачей, будем считать сражение выигранным. Только, знаете, мало шансов в близкие годы достичь этого уровня. Если уж быть откровенным до конца, то автор не вовсе уверен, что кто-нибудь в мире умеет сегодня написать программу, которая решила бы этот вопрос хотя бы с точки зрения чистого математика, то есть за любое конечное время (черт с ними, со звездами, пусть гаснут!). И при нынешних скоростях машин вовсе нет шансов сделать это со скоростью, идущей в сравнение со скоростью узнавания человеком.

Вот вам и ответ на заголовок этого параграфа.

§ 3. Два способа думания

Все, что написано в этом параграфе, — моя чистая фантазия, потому что следует различать между тем, *как устроено* нечто, и тем, *как могло бы быть* устроено это нечто. А говорить я буду именно о втором.

Итак, в § 2 мы поговорили на тему о том, что в счете мозг сильно уступает машине по скорости, а в других областях мышления так же сильно ее превосходит. Как это можно объяснить? Да самым простым образом.

Мозг человека НЕ ПРЕДНАЗНАЧЕН ДЛЯ АРИФМЕТИЧЕСКОГО СЧЕТА. И когда человек считает в уме, он работает так же производительно, как тогда, когда пытается поднимать тяжести, ну, скажем, с помощью ресниц. Очень маленький груз можно поднять и ресницами.

Обращает на себя внимание еще одно обстоятельство: когда человек множит в уме

$$237 \times 415,$$

то поступает он примерно так:

Пятью семь — тридцать пять. Пятью три — пятнадцать. Да еще нуль. Сто пятьдесят. Да запомнено тридцать пять. Сложим. Итого сто восемьдесят пять. Запомним сто восемьдесят пять. Пятью двести — тысяча. И сто восемьдесят пять было. Итого — тысяча сто восемьдесят пять. Десять на 237 — это 2370. Сложим с запомненным. Это уже длится подольше и само подразделяется на этапы. И так далее.

Итак, человек множит (да и складывает) медленно и притом *может точно рассказать шаг за шагом, как именно он это делал.*

А теперь пусть попробует тот же человек рассказать, как он узнает знакомого. За долю секунды (!). Только пусть не рассказывает, что узнает его по цвету лица или длинному носу. Потому что — загори этот знакомый — и всего-то и будет сказано: «Ишь, как Гена загорел». А вовсе не «...черт его знает, кто это. Среди моих знакомых таких черных нет». И длинный нос тоже может быть вовсе и не виден. Узнавание от этого не пострадает. Но ведь так происходит не одно узнавание.

Человек решал и решил нехитрую задачу на построение. Ну, так, минут на 10 размышлений.

Спросите его, как он нашел решение. Только пусть он не приводит доказательства правильности своего решения, а расскажет, *как* он к этому решению пришел.

Вряд ли ответ будет вразумительным. Внешне процесс мышления выглядит очень странно. Что человек делает, когда он напряженно думает? Да ничего. Отбивает согнутым пальцем такт по краю стола. Или мурлычет что-то совсем невнятное: «Тум-турурум, тум-турурум, тум-турурум...». Куда как не похоже на мыслительный процесс высшего уровня... А если вспомнить, что во время решения было в голове, получится еще удивительней. Отрывки неясных и простых образов и... пустота. Какой напрашивается вывод? Мне кажется возможным только одно:

Человек думает над трудным и считает не одним и тем же органом, а разными.

Может быть, это различные участки мозга. Хотя, вообще говоря, неясно, где собственно топографически размещаются механизмы мышления. Что они находятся именно в головном мозгу и только там, можно доказать лишь экспериментом. Вот что руки и ноги тут ни при чем — ясно: безрукие и безногие думают ровно так же, как и неискалеченные люди. А вот почему не может участвовать в этом деле, скажем, солнечное сплетение, столь уважаемое индусскими йогами? Или — любой другой орган, без которого человек не живет (а потому и не думает) нисколько? Ну, хотя бы сердце. Ведь в конце концов категорическое утверждение о том, что мышление происходит в головном мозгу, никем и никак не проверялось. Вроде как сообщение Аристотеля о самоубийстве скорпиона в огненном кольце. Скорпион же, между тем, самоубийством вовсе и не кончает.

Собственно говоря, нам и не важно (узко для целей нашей беседы), каким именно местом думает человек.

А важно лишь, что ДУМАЕТ он одним органом или способом (быть может, здесь, в свою очередь, есть целая иерархия).

А СЧИТАЕТ — другим.

И про работу той части думательного механизма, которая ведает счетом (а также еще некоторыми вещами, например — проведением формальных доказательств), мы узнаем очень подробно, все здесь происходит «на виду». И — невероятно медленно.

А ведь, наверное, именно поэтому и получились вычислительные машины! В самом деле, некоторую работу (умножение) человек делает удивительно плохо. И знает точно, каким путем он ее, эту работу, делает. Условия для создания механизма совершенно идеальные. Ведь даже самому препаршивейшему подъемному крану нехитро обогнать грузчика, трудящегося с помощью ресниц.

Вот какой слабый конкурент был побежден электронными машинами.

Это — про медленное думание. Будем называть его сознанием. А теперь — про быстрое думание. Назовем его условно подсознанием. Подчеркнув этим, что мы почти или совсем не контролируем процессы, происходящие при таком «быстром думании». Совпадение названий с фрейдовскими сознанием и подсознанием почти случайное. Хотя в одном месте у Фрейда есть удивительное наблюдение: «Во сне люди никогда не считают...». Если то, что написано в этом параграфе, правда, объяснение фрейдовского наблюдения (Фрейд его не дает) получается очень естественное: во сне, надо полагать, сознание не работает.

Итак про подсознание. Про него мне сказать почти нечего. Кроме веры в то, что скорости работы здесь сказочные. Превосходящие скорости ЭВМ в миллиарды, а может, и миллиарды миллиардов раз.

Опять-таки нейрофизиологи говорят, что таких частот они не наблюдают, а потому их нет. На это я могу возразить только одно:

Мы видим работающую современную ЭВМ. Попробуем определить частоты, на которых она работает. А в качестве инструмента возьмем набор маятников. Мы действительно уловим ряд частот — главную частоту колебаний, происходящих от работы вентилятора в системе охлаждения, и еще других — от моторов входных, печатающих и перфорирующих устройств. И, скажем, от мотор-генераторов энергопитания.

Может быть, так и возникают α- и δ-ритмы? А истинные процессы думания идут на мегамега... герцовых частотах, недоступных пока нашим приборам? Если, конечно, эти процессы думания носят именно характер электромагнитных колебаний. Это, конечно, чистый домысел автора. Покуда этих сверхбыстрых процессов никто еше не наблюдал. Но ведь происходит же колоссальная работа, требующая переработки необъятной информации и создания грандиозных логических построений за секунды, минуты и часы. Значит, кто-то эту работу делает! Конечно, в мозгу миллиарды нейронов. Так что возможна *параллельная* работа. Но, во-первых, *ускорения только в миллиарды раз вроде бы маловато*. А во-вторых, параллельная работа предполагает еще пирамидальную надстройку для центрального управления этими параллельно работающими устройствами. И пирамида эта представляется весьма и весьма многоэтажной. Быть может, только математик-программист, работавший с игровыми и логическими задачами, и представляет себе, хоть приблизительно, о каком объеме работы идет речь. Нет, миллиардами здесь не обойдешься...

§ 4. Как связаны сознание и подсознание

Прежде всего — входом и выходом. Иначе мы вообще не могли бы думать. Если я прав, то «думательная задача» подается сознанием в подсознание. Там и происходит настоящая работа. А потом на выход в сознание подается результат. Каковой и оформляется в этом слое в приемлемую (скажем, словесную) форму. Кстати, кто из математиков не испытывал этого ощущения — задача уже решена, сейчас скажу как. И понемногу приходят слова доказательства. И хотя они еще не выговорены, слова эти будут, потому что задача и правда решена. Может быть, кстати, от этого и требуется большое время — несколько дней, недель, месяцев, лет — для решения трудной проблемы. Ведь так и говорят: пусть отлежится.

А что это должно, собственно, значить? Если мы *ничем* не думаем все это время про задачу, то отчего бы она продвинулась? А если думаем — то она вовсе и не отлеживается. Между том, пока задача «отлеживается», сознание явно видимым образом занято другим.

Вероятно, в это время и идет работа подсознания. Быть может, очень интенсивная. А голова у нас, когда мы напряженно думаем про трудную задачу, пуста не случайно: именно так, заняв сознание всякими «турум-турурум», мы и не позволяем сознанию задавать подсознанию новые задачи, не отвлекаем его. Это, наверное, и значит — сосредоточиться.

§ 5. Почему так медленно обучается человек

А он вовсе не медленно обучается. По крайней мере тому, что идет через подсознание. Еще раз вспомним, какой грандиозный объем информации успевает восприняться за секунды, когда мы глядим, просто глядим на мир. Только вот левые притоки Днепра, время царствования Капетингов, да еще все та же таблица умножения никак не хотят укладываться в голове. С чего бы это?

Я-то думаю, что дело обстоит просто до крайности. Школьная премудрость — от Капетингов до трех признаков равенства треугольников — потому именно и является школьной премудростью, что она сама по себе не захотела укладываться в голове человека. То есть, это как раз и есть то, что не берется подсознанием, а потому воспринимается с помощью сознания — значит, медленно и трудно. А поэтому такие вещи и пришлось человечеству выделить как *предмет обучения*.

Про все в общем виде фантазировать не буду. Что происходит, скажем, при зубрежке таблицы умножения, я не знаю. Быть может, преодолев сопротивление подсознания, ее (таблицу) зубрежка как раз в подсознание и загоняет. А может статься, что подсознание здесь и минуется, просто идет запись в память непосредственно из сознания. И такая запись происходит медленно.

А вот про математиков в широком смысле автор хочет кое-что сказать. Потому что, хотя в настоящее время он, автор, и является программистом, но был когда-то и чистым математиком (ср. определения в последней беседе), так что воспоминания о тех временах у меня еще живы.

Все хорошо знают разницу между учеником (студентом, профессором...), *вызубрившим*[1] доказательство и *понявшим* его. А в чем здесь, между прочим, разница? Как раз «понявший»-то помнит доказательство *хуже вызубрившего*. Но только он извлекает его из головы как-то очень быстро. И — не всегда одинаковыми словами. А также может провести еще и доказательство близких теорем, решение близких задач, которые он, «понявший», раньше вовсе и не решал.

Думается мне, что «понимание по существу» — это как раз и есть преодоление барьера, введение вопроса в подсознание. Ну, а уж если

[1] Мягкая форма: понявшим формально.

удалось привлечь к работе подсознание, то чему ж удивляться. Конечно, теперь и скорости и возможности изменились фантастически. И самому понявшему стало все «просто и ясно». Еще бы! Какой силы аппарат заработал...

И еще попробуйте вспомнить. Вы никогда не улавливали момента, когда учитель вдруг добился понимания от своего ученика? Именно самого момента перехода от формального понимания к «пониманию по существу»? Ведь что в этот момент делает учитель? Иногда он, учитель, в этот момент произносит одно–два слова. Например: «...ну же, точка *B*, точка *B*...». Или даже — просто делает какой-то жест рукой. Как бы захватывая горстью и приподнимая средней величины предмет. Но притом всегда и слова и жест происходят с большим нервным напряжением. Похоже, что учитель несколько гипнотизирует ученика. И вот в этот-то момент и совершается «прорыв» в подсознание — наступает, если, конечно, получилось, как бы мгновенное озарение. Ученик *вдруг* понял окончательно. А учитель отирает со лба выступивший пот. Потому что работа его была очень тяжелой.

Быть может, преимущество живого учителя (профессора, лектора) перед книжкой в том как раз и состоит, что хоть и в малой степени, и неуклюже, почти всегда интуитивно, но находит преподаватель-человек какую-то узенькую щелочку, ведущую не в сознание, а непосредственно в подсознание обучаемого. Ну, а книжке такого не дано. Здесь все идет через сознание. А работу по передаче сведений в подсознание обучаемый вынужден провести сам, без помощи учителя. Или — остаться при «формальном понимании». Конечно, здесь всегда имеют место нюансы: что-то пошло в подсознание, а что-то нет. Но во всяком случае никакой, даже самый хороший, учебник живого преподавателя (тоже, конечно, хорошего) не превосходит. А казалось бы, можно ведь уж учебнику единожды раз написать самым-самым наилучшим образом. Однако не получается. Хотя, нужно сказать, талантливо написанный учебник, по-видимому, тоже как-то упрощает дорожку к подсознанию. И тем он и отличается от абсолютно точного, но тем не менее плохого учебника. В общем здесь, по-моему, и лежит тайна педагогического мастерства.

И — последнее. Все гипнотические пассы и убедительность тона учителя, все постепенные подходы хорошего учебника — это все паллиативы. Нет у нас дороги к *прямому введению* сведений в подсознание обучаемого. И приходится «опытным путем» доходить до того, как помочь преодолению барьера.

Представьте себе ЭВМ. И, скажем, уборщицу, обтирающую поутру машину тряпкой, сметающую с пульта окурки и т. д. и т. п. Пусть теперь эта уборщица ничего не знает ни об устройстве ЭВМ, ни о способе ввода в нее информации. За долгие годы уборки она, тем не менее, заметит связь между своими действиями и реакцией на них машины.

И, передавая эти сведения из поколения в поколение, научится, вероятно, понемногу, нажимая то одну, то другую кнопку, на ощупь, экспериментально, вводить потихоньку в машину несложную информацию. А между тем, через входное устройство широким потоком и с большой скоростью в ЭВМ можно подать и реально подается информация огромного объема и любой сложности...

Не в таком же ли положении мы, учителя? Целые потоки информации о мире льются в головы наших учеников через их входные устройства — зрение, слух, мышление. А мы неуклюжими потираниями и подталкиваниями по месяцу бьемся над тем, чтобы ввести жалкие крохи в виде таблицы чисел, кратных семи, и перечня четырех–пяти левых притоков все того же Днепра. Еще бы, ведь нет нам входного устройства. То есть видим мы, видим, как это происходит, когда нерадивый ученик смотрит в окно. Но ведь и уборщица видит, как программист вводит перфокарты! Только не знает она, *что на них нанесено.* И мы — тоже.

И все-таки нельзя исключить того, что найдет человечество это входное устройство. И тогда обратятся в часы и даже минуты долгие годы трудной учебы. Это будет ПРЯМОЕ ОБУЧЕНИЕ.

§ 6. Как же быть машинам?

Во-вторых, увеличивать скорость. И, конечно, оперативную память. Быть может, у нас просто не хватает размаха. Например, машина, на которой реально впервые осуществится мышление, может оказаться влезающей не в зал, как современные (1964) большие машины. И не в дом. И даже не в квартал. А, например, будет такая машина величиной с город Серпухов. Может статься, что, как для атомной бомбы необходимо некоторое минимальное количество плутония (критмасса), чтобы вообще мог произойти взрыв, так и для машины нужен некоторый минимальный объем оборудования (с город величиной), чтобы в *принципе можно было* организовать мышление. И что мы пока находимся в докритической области. Скажем, в миллион раз уступающей по объему критической. А оборудование в такой машине может работать, например, в параллель. Скажем, 10^5–10^6 параллельных арифметических устройств. Конечно, вряд ли понадобится умножение. И даже — плавающее сложение. Скорее — логические действия. И разнообразные информационные и обменные операции. С разветвленной сетью взаимного обмена между параллелями. И колоссальной системой управления, объединяющей всю систему. И память разного вида. С длинными и очень длинными и переменной длины ячейками. И — так называемая ассоциативная память, где идет не выборка слова по адресу, а, наоборот, выборка адреса по слову. Построить такую память можно уже сегодня. Только нужно многовато оборудования.

Все сказанное, конечно, фантазия. Даже относительно средней фантастичности этой — вообще слегка фантастичной — главы.

Все это во-вторых. А вот что во-первых. Даже и машина с Серпухов величиной будет всего лишь в 10^6 раз мощней по оборудованию, чем нынешние ЭВМ. Приблизительно, конечно. А ведь недодуманный алгоритм в игровой задаче влечет куда большую растрату времени. В 1960–1962 годах А. Л. Брудно занимался одномасткой (см. [5]). Через месяц у него был создан некий алгоритм. Дававший на трехминутном контроле времени возможность сыграть партию 6 : 6. За 2 года упорного труда Брудно и И. Я. Ландау систематически поднимали мощность алгоритма. К тому моменту, когда они бросили это дело, на той же машине, при том же трехминутном контроле игралась партия $16_{10} : 16_{10}$. Формально выигрыш здесь $\left(\frac{16!}{6!}\right)^2 \approx 7 \cdot 10^{20}$.

Фактически, учитывая сокращения первого алгоритма, ускорение получается поменьше — около 10^{10} (десять миллиардов!). Вот почем обходится незнание того, *как делать*. В упомянутой статье Брудно об этом подробно рассказано. Но ведь что любопытно — выигрыш скорости в 10^{10} раз достигнут за несколько этапов. И каждый из них почти очевиден. Только почему-то никому не приходил в голову, пока он не был рассказан. А Брудно рассказывал (подробно!) про эту работу — этап за этапом — каждые несколько месяцев. И слушатели у него были — семинар, в котором собрались Г. М. Адельсон-Вельский, В. Арлазаров, М. М. Бонгард, Е. М. Ландис, А. Усков, автор этих строк и многие другие. И слушали мы, честное слово, очень внимательно. Даже написали некий научный труд [6].

Итак, первое, самое необходимое — это настойчивый, упорный поиск новых алгоритмов решения логических, игровых и т. д. задач. Но как организовать этот поиск?

Еще один постулат. Где наше не пропадало! Все равно собеседник может закрыть книжку на любой строке этой главы. Потому что автор сознает, что высокопарное вещание никого не убеждает, даже если интонации будут очень убедительными. Но, к сожалению, ничем больше я не богат. Так вот, очередной постулат таков: новые решения, подобные тем, что нашел А. Л. Брудно в своей с И. Я. Ландау одномастке (грани и оценки, таблица, текущая справочная), *придут не из голого размышления о сущности мышления вообще, а из решения конкретных задач* (игровых, логических, экономических и т. д.). И притом — при решении задачи ее *решение обязательно нужно воспринимать как самоцель.* При выборе задачи подумайте сто (100_{10}) раз. И выберите такую задачу, какую захотите. А вот уж после этого забудьте о том, почему выбрана именно эта задача. И что рядом есть другие ничуть не хуже. Решайте ее так, как будто в ней сосредоточен весь смысл Вашей жизни. И тогда

(может быть) услышите шелест синекрылых птиц. А иначе (наверняка) не будет ничего. Все это тоже входит в постулат имени меня.

§ 7. Можно ли заглянуть в подсознание?

Для того, понятно, чтобы узнать, *как выполняет «думательную работу» самая совершенная машина* — наш мозг[2]. Хорошо бы, конечно. Только я не знаю, как это сделать. Нет причин, почему бы сознанию нельзя было включаться не только на входе и выходе работы подсознания. И тогда, быть может, мы бы узнали кое-что о путях и приемах, которыми мы думаем. Быть может, трудность здесь самая глупая — подсознание работает быстро и сознание не то что не может подключиться, а просто *не успевает* уследить за ходом дела. Ну, как близко стоящий человек не успевает разобрать ни слова в статье газетного листа наклеенного на стенку проносящегося рядом с ним вагона курьерского поезда. Если так — достаточно будет гетевского: «Остановись, мгновение!» И мы прочтем текст. Только как это мгновение остановить? Быть может, здесь когда-нибудь и правда помогут нейрофизиологи. Если, конечно, перестанут мерить маятником α–δ-ритмов мегамега... мегагерцовые частоты ЭВМ — подсознания.

§ 8. Что мы там увидим?

Вот уж чего не знаю, того не знаю. Может статься — такие удивительные логические схемы, что и не снились нашим мудрецам. А может случиться — и иное. Самую-самую обычную «сознательную» логику. Ту, которая в учебнике Киселева употребляется для доказательства школьных теорем. И — ничего больше. Кроме, понятно, скорости. В пользу этого есть тоже аргументы: если бы логика сознания и логика подсознания были принципиально различными, то с какой стати результаты работы подсознания кончались бы выдачей сознанию решения, с его, сознания, точки зрения безукоризненно верного?

Может статься, в подсознании идет точно такая же работа, скажем, перебор возможностей, какую делает человек, думая сознанием. И каждая возможность проверяется по обычной схеме доказательства. Негодное бракуется. И только когда найдется решение, подсознание замедляет свой бешеный бег, поезд останавливается, и на станции в сознание передается найденное решение. А вся-то разница и была в большой скорости подсознания. Ну, может быть, конечно, еще кое-какие профессиональные приемы ремесла. Не без того...

---

[2] Мозг — это, конечно, условность. Но мозг, или там солнечное сплетение — это ведь нам безразлично!

§ 9. Две точки зрения на неизвестное

Моя и Е. М. Ландиса. Как крайних выразителей этих взглядов. Остальные наши знакомые где-то посередине. Так вот, я думаю, что *человек думает по схеме полного перебора*. То есть буквально так, как в программе строится сегодня решение логической задачи. Конечно, со всеми отсечениями и улучшениями, которые на сегодня придумал Брудно. И теми, что придумали Адельсон-Вельский, Арлазаров и Усков. И даже теми, что они придумают завтра. И не только они. Доказывать это я не берусь. Но ведь я уже выговорил себе право в этой главе ничего не доказывать, а непосредственно вещать. Вот я и пользуюсь этим правом. Внутреннее чутье говорит мне, что, кроме такого прямого перебора, ничто не может привести к точным решениям. А думание-то ведь их находит! И все соображения, что на полный перебор никаких скоростей не хватит, меня хотя и смущают, но не сдвигают. Потому что я не вижу других возможностей для точных решений. В общем, как сказал Мартин Лютер: «Здесь я стою, и не могу иначе...».

А Е. М. Ландис думает по-другому. Он считает, что главное в мышлении (думании) — ассоциации. Ассоциации не только с объектами, но и с логическими схемами решений. И с более сложными конструкциями — ассоциации с ассоциациями и т. д. — на все более высоких уровнях. А что перебор возможностей, хотя и тоже есть, но играет, так сказать, подчиненную роль. Что ассоциации обрезают лишние ветви в дереве возможностей. И благодаря этому — углубляют возможную «этажность» рассмотрения-перебора. Выглядит все это очень складно, но для меня совсем неубедительно. Потому что — кто же тогда решает, что есть ассоциация, а что — нет? Впрочем, нет смысла опровергать собеседника, не давая ему слова. Боюсь, что я и так исказил его точку зрения. Хотя и старался не сделать этого.

К слову сказать, Е. М. Ландис — первый, кто (по крайней мере в СССР) создал программу полного перебора (весной 1959 года). Потому что я, например, сходу не видел — возможна ли *конечная* программа n-кратного цикла с произвольным n. Так что вера Е. М. Ландиса в ассоциации (неверие в полный перебор) не объясняется личными мотивами. А это ведь так часто бывает...

§ 10. Главные задачи программирования

Из всего предыдущего ясно, что именно автор считает главными задачами. Это — все то, что продвигает машины к думанию. А конкретно — к решению тех задач, с которыми сейчас легко справляется человек, и никак — программа. Сказать точно, решение какой именно задачи послужит ступенькой для перехода на следующий уровень, я не берусь. Потому что уже обжигался на этом.

В 1958 году была постулирована некоторая программа действий. Тогда мы не знали, как удивительно медленно действует машина. И думалось — достаточно дерзнуть. В качестве объекта я предложил игры. Чтобы можно было сравнить силу машины (программы) с силами человеческого мышления.

Выбор пал на «подкидного дурака». Мы — Адельсон-Вельский, Ландис и я — сделали и пустили программу на машине М-2. Пока игра шла с колодой, все было пристойно. Наверное, оттого, что человек тоже мало чего соображает в столь сложной ситуации. А может быть, все кругом просто плохо играли в эту игру. Или оттого, что уж слишком многое зависит от расклада. В общем, так или иначе, пока была колода, позора не было. Но что началось, когда колода кончилась! Именно с открытыми-то картами программа играла безумно, невероятно, непроходимо скверно.

А ведь, смешно подумать, нам казалось, что это-то как раз самый простой блочок, который мы доделаем потом, поючи и танцуючи, — ну, в общем, ленточка для украшения.

Вместо ленточки появился эндшпиль. И — финал эндшпиля, для ускорения. И ровно на этом мы и кончились. Потому что, считая эндшпиль 6 : 6[3)], машина обязательно ломалась. После нескольких часов работы... Что произошло от этой нашей неудачной деятельности? Прямо для думания — ничего. Или, вернее, негативный результат: с помощью машины мы могли думать много хуже, чем без нее. Точнее, *через машину мы думали медленнее, чем непосредственно.* Анализ показал неожиданное: мы просто не умели высказать всех своих соображений, как и почему мы выбираем ход (спрограммировать-то точные вещи мы уже умели). Именно тогда я задумался о сознании и подсознании.

Ну, это личное. А что для всех? Тоже кое-что. Переборная схема. Блок-программа (параллельно с Э. Доброчаевой и М. Вайнштейном). И — организация программы, в частности выяснение важности *способа задания информации.* Именно тогда мы пришли к *всемерному использованию машинного слова как множества единиц* в некоторых его разрядах.

Американцы, работая над шашками, пришли к тому же [8]. Совпадение совсем неслучайное.

Забавно, что так усиленно насаждаемая у нас система автоматического кодирования (пардон, программирования!) — пресловутый АЛГОЛ — рассчитана, универсальности для, на машину, работающую в любой системе счисления. Например, в десятичной. Поэтому в АЛГОЛе принципиально невозможно использование слова в указан-

[3)] То есть работая в ситуации, когда у каждого из партнеров по 6 карт, причем карты открыты, а колоды нет. Человек в такой обстановке играет почти безошибочно, причем думает одну–две минуты.

ном смысле. Впрочем, для серьезных задач АЛГОЛ малопригоден. Как и для несерьезных.

Вернемся, однако, к нашей теме. Являются ли игровые задачи тем рубежом, взятие которого откроет оперативный простор? Честно скажу, не знаю. Рубеж этот очень труден. Надо ли сосредоточивать все силы на нем — не ведаю. За время с 1958 года я стал осторожнее. И немножко образованнее. То есть увидел (прочел, услышал), что наряду с игровыми задачами есть еще задачи машинного поиска логического вывода. В СССР ими активно занимается группа логиков в Ленинграде и Вильнюсе под руководством Н. А. Шанина и, кажется, в Киеве. За границей подобные вещи делают Хао Ван, Дж. Маккарти, П. Гилмор, Дж. Робинсон, Дж. Фридман, М. Дэвис, И. Бар-Хиллел. А кроме того, есть ведь еще и разные комбинаторные задачи, на составление расписаний и т. п.

Все они, по нашему сегодняшнему уровню, необычайно трудны. И тот, кто ими займется, рискует с большими шансами потерпеть кораблекрушение. Но на то это и есть открытый океан! Потому что в безопасном каботажном плавании никто еще Америки не открыл.

И не откроет!

§ 11. О бессмертии

Автор сильно колебался — включать ли в беседы о программировании написанное в этом параграфе. Потому что он ясно отдает себе отчет в том, каким нападкам это написанное подвергнется. Но главное — что ему, автору, нечего противопоставить возражениям профессиональных деятелей соответствующих наук. Так как вопрос о бессмертии — вопрос экспериментальный. А у меня, кроме умозрения, ничего здесь за душой нет. Впрочем — как и у будущих оппонентов. Но мне приходится вторгаться в *чужую* область. А им — защищаться на родной почве. И в этом — большая разница.

Но все-таки автор решился включить этот параграф. Потому что он горячо верит, что написанное ниже с большой вероятностью правда. И раз уж вся эта беседа — авторское кредо (как еще и последняя), то «невместно мне страха ради иудейска» из символа веры исключать опасные места.

НЕСКОЛЬКО ЭКСПЕРИМЕНТАЛЬНЫХ ФАКТОВ

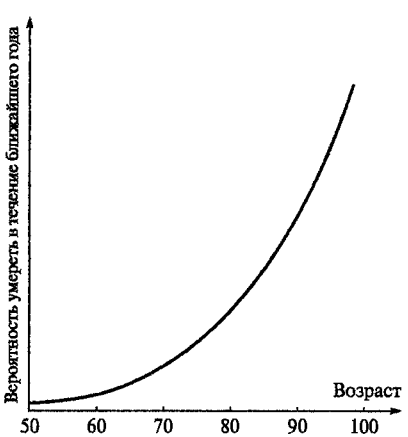
1°. Гидра живет в среднем около одного месяца. Но если от ее тела отщипнуть маленький кусочек, она регенерирует его и одновременно «обновит» свой организм. Ученые проделали такой эксперимент: раз в 2–3 недели отщипывали клочок гидриного тела. И эта гидра живет уже лет 20.

2°. Сыпной тиф — болезнь опасная. И выжившие часто страдают от осложнений. Но иногда — не так уж часто, но тем не менее не раз и не два — у выздоровевших от сыпняка людей сами по себе *вылечивались неизлечимые болезни* — порок сердца, например. Происходит как бы омоложение организма.

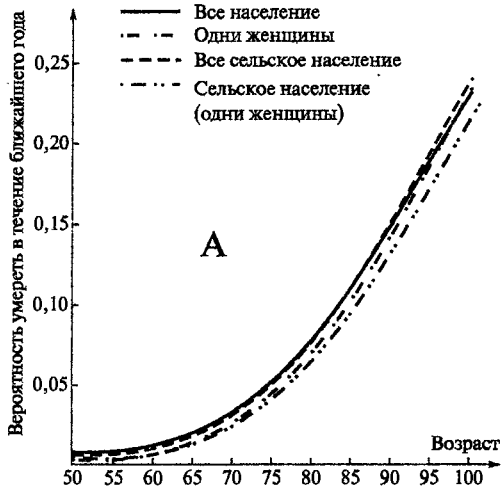
3°. Ткани в человеческом организме стареют. Чуть ли не все без исключения. Но физиологи научились поддерживать жизнь любой отдельной ткани *вне организма*, в пробирке. Подавая ей, ткани, искусственное питание. И в этих условиях *каждая ткань неопределенно долго не испытывает никаких изменений. Отдельно взятая ткань не стареет. А в организме та же ткань стареет.*

4°. Человек к старости дряхлеет. Склеротические процессы — кажется, вот сегодняшний символ старения. Примерно после 80 лет развитие склероза прекращается. Что случилось, то случилось, обратной дороги нет. Но и хуже не становится.

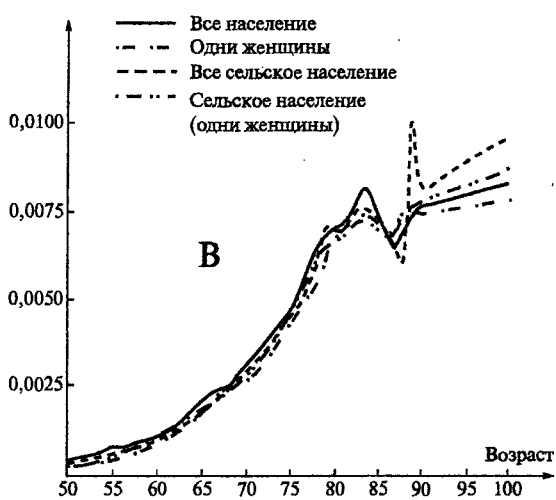
5°. И еще. Пусть старение идет так, как принято думать, — изнашивается организм, стареют ткани. Как тогда должна была бы выглядеть кривая смертности? Отложим по оси x возраст, а по оси y — % людей данного возраста, умирающих, скажем, за год, взятый от общего числа людей этого возраста. Тогда, наверное, эта кривая должна была бы выглядеть как-нибудь так:



Так примерно и выглядят в действительности кривые вероятности умереть в ближайший год для людей данного возраста. На графике А вы видите кривые, взятые из данных переписи населения СССР за 1959 год [7].



Но подумаем, как должна была бы выглядеть *производная* от этой функции. Можно предполагать, что не только сама вероятность умереть растет с годами — чего же и ждать: организм ведь не улучшается, — но и что скорость роста должна увеличиваться (пока вероятность далека, понятно, от 1). А между тем вместо ожидаемой гладко растущей кривой производная от вероятности умереть выглядит вот так:



Чем же вызвано «неправильное» поведение этой кривой где-то около 80–85 лет? Создается ясное ощущение, что лет до 78–80 мы имели дело с одним правильным процессом, с 78–80 до примерно 88–90 лет что-то происходило, а с 88–90 лет снова идет правильный процесс, но вовсе не тот, что был до 78 лет.

Как объяснить все это?

Мне представляется дело так: *ни один организм не стареет сам по себе из-за старения (снашивания) своих частей — тканей. Процесс старения* — если можно так выразиться — *посторонний, чуждый организму процесс.* Как бы *насильно вделанный* в организм. Для чего? *А для того, чтобы организм умер.* Оставив, конечно, и вырастив потомство. И освободив после этого для потомства место на земле. С этой точки зрения кривые ***B*** очень наглядны. Они рассказывают вот о чем: была совершена попытка убийства. Жертвам давали медленно действующий яд. В расчете на то, что в некоторый момент (примерно около 80 лет) они умрут из-за расшатанного здоровья. Но перенадеялись на силу яда, и кое-кто выжил. А яд по беззаботности давать перестали. И выжившие со своим сильно расстроенным здоровьем продолжают жить. И, поскольку здоровье испорчено, легко умирают. И более того, вероятность умереть продолжает увеличиваться с каждым годом, но только темп роста почти стабилизируется.

Для этого не требуется больше посторонних усилий — изношенная машина задевает за все выступы, ее трясет на всех ухабах и заносит во все ямы и кюветы далеко не ровной дороги жизни. И она продолжает потихоньку разрушаться, теперь уже сама по себе. И вероятность ей сломаться с каждым годом, конечно, растет (правая ветвь графика ***B***). Но вот бить по ней нарочно молотком где-то около 80 лет явно перестали. По-моему, график ***B*** сам говорит об этом убедительнее, чем любые рассуждения об этом графике.

Только в действительности сделано это немножко не так. *В организм встроен* некоторый механизм для убийства с помощью старения. *Конструкция* механизма нам пока неизвестна. А рассчитан он на то, чтобы убрать своего носителя годам к 80-ти. Ну, конечно, должны для этого еще быть в организме БИОЛОГИЧЕСКИЕ ЧАСЫ тоже неизвестной для нас природы. И механизм делает свое дело по большей части удачно. Но не контролирует результатов. Почему — вопрос его конструкции. А налицо сам факт: если убийства не случилось, механизм не повторяет попытки. И человек живет еще неопределенно долго — столько, сколько выдержит со своим испорченным здоровьем. От одной совсем простой, почти неграмотной, женщины я слышал как-то такие слова:

— Она пережила свой век, а не померла. Теперь будет жить вторую жизнь.

— Вера, а сколько же это — свой век?

— Век человеку положóн 80 лет. А кто пережил — живет второй век. Ну, худо живет, худо, конечно, уж года не те, но раз смерть миновалась, теперь можно жить...

Вот какие удивительные вещи бытуют в глубинах народных. Правда, только среди людей непросвещенных. Графика *B* они не видели.

ЕЩЕ ЧУТЬ-ЧУТЬ УМОЗРЕНИЯ

Что легче организму сделать: повторить при делении данную клетку в *точности* или чуть-чуть подправить ее в сторону старения? Думаю — первое. И то, что известно сейчас из работ о генетическом механизме (ДНК, РНК и т. д.), тоже, пожалуй, голосует за точное повторение. И еще, заметьте, «порча» должна начинаться с *определенного* возраста. А до этого должно идти точное копирование. Трудно это сделать без специальной *программы управления, связанной с часами* (биологическими). Но если так уж трудно, то зачем же было делать такую программу? Кому от этого хорошо?

Мог это сделать естественный отбор. И хорошо от этого биологическому *виду*, подверженному смерти. По сравнению с *«бессмертным»* видом у *«смертного»* огромное преимущество в легкости эволюции. Именно поэтому *выживает тот вид, отдельные особи которого умирают*. А тот вид, в котором организмы *не умирают естественной смертью*, сам обречен на *вымирание*. Вот какая здесь возможна диалектика.

А ТЕПЕРЬ ЕЩЕ НЕСКОЛЬКО ФАКТОВ

Поверим до конца параграфа в мою гипотезу. То есть, что смерть есть дело совсем чужеродное для живого организма. Так вот, всегда ли эта чужеродность так хорошо замаскирована «естественным старением», как это сделано у человека?

Оказывается, нет. Есть такая рыба — кета. Живет она пять лет. А потом — живая, здоровая и очень активная — идет из моря в реку метать икру. И уже по дороге перестает есть. Придет на место, вымечет икру и буквально сгнивает заживо. И — умирает.

Кета — это случай особенно откровенной, прямо-таки нахальной работы «программы смерти». Сделала свое дело — и дохни. Причем все это никак не припудрено и не замаскировано никаким старением. Иногда природа бывает удивительно откровенна...

И еще есть растение — бамбук. Он вырастает что-то примерно за месяц. И потом не увеличивается его ствол ни в высоту, ни в диаметре. Только стенки делаются толще.

Сколько живет бамбук? Здесь мы видим совсем странный случай. Как бы работу программы смерти в совокупности не с настоящими часами, а с игральной костью. Потому что бамбуковый лес (он является единым организмом) может прожить и 3 года, и 70 лет. И живет он, пока не зацветет. А потом немедленно погибает. Тоже случай большой откровенности...

МОГУТ ЛИ ПОРТИТЬСЯ БИОЛОГИЧЕСКИЕ ЧАСЫ?

Ну, хотя бы изредка? Да, могут. Хотя и, правда, изредка. Медикам известны случаи (кажется, несколько сот), когда жизнь человека протекала аномально быстро.

В 3 года такой ребенок производит впечатление юноши. В 5 лет может иметь детей и вообще всем своим внешним и даже (хотя и не полностью) интеллектуальным обликом является взрослым человеком. А в 10–12 — он умирает от старости. По всем признакам перед нами в этих случаях *действительно глубокий старик*. Вот вам и естественное старение! У природы иногда случаются очень знаменательные ошибки...

А в обратную сторону? Ну, это пореже. Как всегда, везет реже, чем не везет. И все-таки случаются женщины, рожающие детей в 90-летнем возрасте.

КАК УСТРОЕНА И ГДЕ РАСПОЛОЖЕНА ПРОГРАММА СТАРЕНИЯ?

Совсем не знаю. Если повезет — она центрального типа. Тогда хорошо. Испортим программу и — обретем бессмертие. И, более того — вечную молодость. Будет хуже, если программа «рассредоточена» по всем клеточкам тела. Но здесь есть некоторый довод против такого свинства. Если бы было так, то почему не стареют ткани отдельно от организма? Так что все-таки есть основания надеяться.

НУЖНО ЛИ ЭТИМ ЗАНИМАТЬСЯ?

Думаю, что да, нужно. Потому что если даже миллионная доля шанса есть за то, что я прав, то не жаль на проверку того, есть ли она — эта миллионная доля — никаких усилий. А заниматься такими вещами должны, конечно, физиологи, биофизики, биохимики... Математики, даже и программисты, здесь, конечно, ни при чем. Вот поэтому-то я и колебался, читатель, имею ли я моральное право на публикацию этого параграфа.

Но уж больно привлекательна вечная молодость... Хотя вряд ли придется она на долю моего поколения. А на вашу, более молодые собеседники, все-таки, может и прийтись. Потому что мы стоим перед неизвестным. Но, быть может, совсем излишне загипнотизированы

априорной верой в смертность человека. Так как, кроме предшествующего опыта, никаких аргументов против бессмертия у нас нет.

А условия опыта можно и нужно изменить и, быть может, — это не такое уж невероятно трудное изменение. Потому что сломать всегда легче, чем достроить. И если такой крайний оптимизм оправдается, бессмертия достичь будет легче, чем осуществить машинное думание. И если бессмертие свершится при моей жизни — я смогу умереть спокойно.

Беседа двенадцатая

Невычислительные задачи

Основная работа — детская игра.
Игры — вот основная работа.

Автор

Эта книжка называется — «Беседы о программировании». И это именно беседы, а отнюдь не энциклопедия по вопросам программирования. Поэтому автор не только оставляет за собой, но прямо-таки пользуется, и притом очень широко, правом писать не обо всем, а только о том, что он действительно знает. А это — вовсе не все. И, быть может, очень далеко не все.

Моему собеседнику необходимо помнить об этом. И не удивляться, когда он столкнется с очень интересными и важными задачами, о которых в этих беседах не сказано ни слова.

§ 1. Шахматы Г. М. Адельсона-Вельского, В. Арлазарова и А. Ускова

После нашего фиаско с подкидным дураком встал вопрос о том, какую игру выбрать для генерального наступления. Котировались крестики-нолики, шашки и шахматы. Самое важное, казалось нам тогда, — иметь игру, общую в международном масштабе. Вроде того, как у генетиков избраны муха дрозофила и кукуруза.

Порешили, что наиболее подходящим с этой точки зрения объектом являются шахматы. Может быть, играли роль личные вкусы Адельсона-Вельского и Арлазарова. Нам с Усковым (на первых порах я тоже принимал кое-какое участие в этом деле) было все равно. И — несколько односторонне, поскольку мы не вели про это переговоров ни с кем,

не только в Международном, но и в Союзном и даже в Московском масштабе — шахматы были признаны дрозофилой. Было это в 1960 году.

Не останавливаясь на истории вопроса, я сразу скажу в очень общих и приблизительных словах о том, что поучилось. Потому в общих и приблизительных, что вопреки обещанию я буду говорить здесь о том, что знаю плохо и не до конца. Если угодно, понаслышке. А желающие знать про это в самом деле пусть прочтут [9].

В результате двухлетней, тяжелой, а временами и вдохновенной работы трех математиков, чьи фамилии стоят в заголовке параграфа (для краткости впредь я буду обозначать их АВАУ), итак, в результате их работы сделалась программа для игры в шахматы.

Эта программа работает по следующей схеме.

Каждый раз машина воспринимает позицию как заданную непосредственно, не учитывая всего, что происходило перед тем (кроме, конечно, таких деталей, как возможность рокировки и взятие пешки на проходе — это учитывается). В заданной позиции машина пробует сделать все принципиально возможные ходы (полуход машины). На каждый из них она пробует все разрешенные правилами ответные ходы противника (полуход человека). На каждый полуход человека машина пробует все свои возможные ответы (снова полуход машины). Затем все возможные ответы человека. И так далее.

Эта часть работы программы — абсолютный перебор.

Глубина абсолютного перебора — суммарное число полуходов человека и машины, идущих друг за другом — конечно, кардинально определяет время рассмотрения позиции. Если ограничиться работой, измеряемой часами, а не сутками, на машине М-20 дальше 5 полуходов не пойдешь. И то с трудом. А этого для хорошей игры недостаточно.

Каждая веточка дерева перебора кончается некоторой позицией. Отнюдь не заключительной. Эту позицию нужно оценить. Для того, чтобы дальше действовать по уже хорошо к этому времени известному минимаксному канону. Какой этот канон в точности, поговорим в § 5. А сейчас ограничимся тем, что минимаксная схема в точности соответствует наилучшей игре *обоих* партнеров. При том, конечно, условии, что оценки заключительных (на концах веточек дерева) позиций принимаются за истинные.

Как именно устроены эти оценки и что получается при их изменении, можно прочесть у авторов программы (см. [9]). Чем эти оценки будут сделаны лучше с чисто шахматной точки зрения, тем, конечно, лучше будет играть программа. Но нас, вернее АВАУ (я к этому времени уже вышел из игры), интересовало в первую голову как раз не это, а то, *чего можно добиться прямым перебором.* И потому первоначальные оценки были попробованы нарочито грубыми (почти по одному только

материалу). К сожалению, оказалось, что мир устроен не так приятно — грубые оценки не искупаются даже 5-полуходовым абсолютным перебором, программа играет плохо.

Итак, абсолютный перебор с некоторой оценкой заключительных позиций определяет выбор хода. Таков был первый вариант программы.

Теперь проявилась главная слабость такой схемы: последний рассмотренный полуход мог оказаться взятием фигуры. Заключительная позиция теперь будет оценена очень высоко для взявшего. А в действительности следующим ходом противник отбирает, скажем, ферзя, схватившего перед тем ладью. И получается совершенная чепуха. Конечно, можно посмотреть, не отбирается ли что-нибудь следующим ходом обратно. Но, между нами, это уже будет шестой полуход!

В общем, из всего этого родился ФОРСИРОВАННЫЙ ВАРИАНТ (Ф. В.). А значит это следующее — рассмотрение дерева *не прерывается, если у противника есть возможность ответить взятием* фигуры или пешки. Но дальше (за обоих партнеров) рассматриваются только ходы-взятия. И даже, точнее, лишь такие, что съеденный материал оказывается в сумме *не меньше*, чем съеденный суммарно противной стороной (считая от начала Ф. В.). Иначе никаких времен не хватит. Зато с такими ограничениями Ф. В. удается протянуть на полтора десятка полуходов. Практически это — бесконечность.

Изобретение фирмой АВАУ форсированного варианта было крупным сдвигом. В очень приятную (для меня) сторону. Правда, Ф. В. не рассматривает столь коварных «тихих ходов». Но, к сожалению, трудность здесь очень принципиальна: в качестве тихого хода вроде бы надо рассматривать любой. И время сразу подскакивает в десятки раз. А выбор «тихого хода» по смыслу... Тут мы сразу подходим к границам дозволенного — если бы мы могли сказать, что означает смысл!

Не забывайте, дорогие читатели, что программа — жестко детерминированный автомат. И она поймет все что угодно — но это все что угодно должно иметь *программно-точный смысл*. А придать программно-точный смысл интуитивному понятию «по смыслу» — это, может быть, значит проникнуть в святая святых подсознания. Чего АВАУ и никому из нас покуда не удалось.

После изобретения Ф. В. и включения его на последнем полуходе игра, конечно, очень улучшилась. И — возросло время.

Но теперь уже можно было себе позволить скинуть полуход. Время стало снова приемлемым (в смысле, скажем прямо, довольно мягком — часов эдак до 5 на ход!), а игра много лучшей, чем была.

Следующим этапом была борьба за время. Она шла по двум направлениям: схемному и техническому.

§ 2. Борьба за время. Предварительная оценка. УХУДУ И ПУП

Можно, конечно, перебирать все ходы в любом порядке — ни меньше, ни больше их от этого не станет. Но вот время на перебор вариантов от этого зависит, И притом очень даже сильно. Потому что если первый выбранный ход окажется самым сильным, то следующие будут отброшены по большей части очень быстро. Как это сделать — догадался Брудно (см. [17]). И, значит, очень выгодно рассматривать ходы именно в порядке убывания их силы. Конечно, по приблизительной оценке — если бы мы знали истинную, мы бы даже и первого хода не рассматривали, а просто его бы и делали.

Итак, появилась *предварительная оценка*. Как она сделана — не суть важно. Фактически там фигурируют и Ф. В. за себя, и Ф. В. за партнера, и легкий перебор. Принципиально новым является то, что по грубым критериям производится предварительная оценка ходов.

Упорядочением ходов, удовлетворяющих данному условию, занимается особая программа, которая так и называется УХУДУ.

Возможные в данной позиции ходы вырабатываются специальными программами. Они тоже так и называются: ХОД СЛОНА, ХОД КОНЯ, ХОДЫ ПЕШЕК, ПЕШЕЧНЫЙ ПРЫГ и т. д. И объединяются тривиальной собиралкой, после чего все вместе называется: ВОЗМОЖНОСТИ МАШИНЫ. Возможные ходы получаются в различных ячейках и форме единичек в разрядах. А УХУДУ переводит их в иную форму, удобную для перебора вариантов. При этом УХУДУ отбирает те и только те ходы, которые удовлетворяют данному условию. Только в нашем случае это данное условие пока никакое — мы хотим покуда перебрать все ходы. Так что работа УХУДУ в этот момент чисто канцелярская — регистрация возможных ходов в порядке живой очереди.

Теперь начинается работа по предварительной экспресс-оценке силы каждого хода. И на основании этой оценки производится переупорядочение ходов по убыванию их ценности (по предварительной оценке). Этим переупорядочением ведает отдельная программа ПУП.

Введение системы УХУДУ — Предварительная Оценка — ПУП очень сократило время выбора хода. А это позволило в свою очередь при том же контроле времени увеличить глубину перебора. Программа стала играть лучше. От перестановки порядка ходов результат — время работы программы — меняется. Чем ходы и отличаются от слагаемых.

Заметим еще в заключение, что экспресс-оценка сама содержит, конечно, перебор. Только неглубокий. Скажем, на 3 полухода и, конечно, Ф. В. А иначе оценке будет грош цена. И еще — тот же прием нужно применить при выборе ответных ходов человека. Только, быть

может, еще несколько ускорить (огрубив) оценку. А если идет глубокий перебор, то та же метода может быть применена и на следующем полуходе. Как это и сделано у АВАУ.

§ 3. Борьба за время. Технические приемы

Всякое ремесло имеет свои приемы, которыми нужно владеть, если хочешь быть мастером этого цеха. Или хотя бы подмастерьем. Программирование совсем не служит исключением.

Рассказывают, что в Польше существовала среди программистов такая игра. Бралась нехитрая задача. Для условной машины (реальной не было) каждый писал кратчайшую программу решения этой задачи. И самый честолюбивый вешал свою программу на стенку. А плата за ложную славу была такой: если кто-нибудь умел сократить программу хотя бы на одну ячейку, честолюбец расплачивался чашечкой кофе. А программа на стенке заменялась более короткой. И так далее. Все это получило название польской игры. И может быть горячо рекомендовано начинающим программистам. Потому что всякое ремесло... и т. д.

Лучшими, известными мне игроками в польскую игру у нас являются Г. М. Адельсон-Вельский и В. Арлазаров. Сам я в нее играю более чем посредственно. И однажды расплатился за совместную с Адельсоном-Вельским программу ГФК никаким не кофе, а шестью бутылками отличного вина, которое радостно выпили В. Арлазаров и А. Усков. При участии, конечно, прочих заинтересованных и знакомых лиц. Но поработали они за эти 6 бутылок месяца полтора. Так что легким заработком такое не назовешь.

Польская игра и есть игра. На собственно технике много не наживешь. Разве что двойку в скорости. Да и то, если исходная программа была уж очень плоха. А так, нормально, экономится 10 % ячеек. От любой программы. Это — постулат Брудно [1]. Практически он выполняется почти с точностью законов природы.

Но есть место, где таится отнюдь не 10 %. Это — способ задания и переработки информации. В шахматных программах вопросы кодировки информации очень трудны. С одной стороны, привлекательно сделать информацию о позиции елико возможно компактной. Тогда можно быстро переносить сведения о позиции, скажем, на рабочее поле. Но, с другой стороны, компактная запись удлиняет *время переработки* информации. Далее, при совершении одного хода позиция мало меняется. Поэтому хотелось бы заменить, например, среди всех

[1] Легко видеть, что следствием постулата Брудно служит такое: максимальная длина программы — 9 ячеек. Иначе ее можно сократить. Ср. Приложение 1.

возможных из предыдущего положения ходов только те, которые оказались затронутыми сделанным нами одним ходом. Но ведь хорошо говорить. Человеку-то, конечно, ясно, что ход a2–a4 никак не отражается на возможностях, например, черных пешек, стоящих на вертикалях c–h. А вот как рассказать об этом машине? То есть рассказать-то, конечно, можно, но не потеряем ли мы при проверке того, зависит ли возможность разных ходов от уже сделанного, больше, чем на прямое получение всех этих ходов из новой позиции?

«В уме» сосчитать все это оказывается практически невозможным. И единственный способ избавиться от искушения (переделки программы) — поддаться ему. А потом посмотреть — что мы на этом заработали или потеряли. Чтобы позволить себе такую роскошь не единожды, снова нужна техника. Между прочим, еще и техника писания и отладки программы в конечный срок. Владеют такой техникой далеко не многие.

Ну, а способ задания информации в непосредственном смысле важен чрезвычайно. И притом — конкретно, применительно к возможностям данной машины. От его выбора зависит все конкретное содержание программы. И скорость — раз этак в 5–10. И ни степени, ни знания здесь не помогают. Даже и высокоученые. Примерно так же, как и в доказательстве теорем. Или в пении.

В частности, оказалось, что для пешек выгодна растровая информация (единички разрядов в паре слов). А для записи ходов — числовая, и притом помещающаяся именно в кодовой части слова. И нужны, скажем, ячейки с растровой записью положения тяжелых фигур. И отдельно — ячейки с растровой записью фигур диагонального боя (слоны и снова ферзь). А вот для возможных ходов фигуры нужно писать сразу — в кодовой части первого слова — ее место в форме числа. А в адресных частях пары слов — возможные ходы в растровой форме. И притом в эти возможные ходы обязательно надо включать фигуры (свои и чужие), которые бьются (защищаются) данной фигурой. Здесь сразу видно, как важен конкретный выбор формы записи информации.

В самом деле, пусть информация записана, как сказано, и мы хотим определить, какие из фигур противника бьются нашим белым ферзем. Что же, пересечем попарно ячейки ФИГУРЫ ЧЕРНЫХ — 1 и ФИГУРЫ ЧЕРНЫХ — 2[2)] с ячейками ХОД ФЕРЗЯ — 1 и ХОД ФЕРЗЯ — 2. Поскольку фигуры черных записаны в растровой форме и возможные ходы ферзя тоже, за два действия мы получили результат.

А теперь подумаем, как нам быть, если бы мы в ХОД ФЕРЗЯ *не включили упоров*, т. е. фигур, бьющихся ферзем, а только поля, куда он

[2)] Такие действительно есть.

может пойти. Ведь с точки зрения «настоящего математика» разницы здесь нет никакой. И правда — достаточно «замкнуть» множество полей, куда можно пойти, добавив по каждому лучу «граничное» поле, и все будет в порядке, т. е. информация у нас и там и там одинаковая...

Более деликатная штука — невключение в ХОД ФЕРЗЯ полей-упоров, куда ферзь и правда пойти не может, так как эти поля заняты *своими фигурами*. Представим себе ситуацию, в которой нам про некоторую свою фигуру хочется знать, кто ее защищает. Сравните обе ситуации (разнящиеся *формой записи информации*), и вы поймете, каким множителем она, эта форма, оборачивается...

Наконец, к техническим (ремесленным) вещам надо отнести непосредственно программы типа ХОД СЛОНА, ХОД КОНЯ и т. д. Здесь борьба идет за микросекунды. Приходится учитывать не только то, сколько действий берет данный алгоритм, но и почем каждое действие «в гектарах мягкой пахоты». Потому что на серийной машине М-20, скажем, сдвиг слова на два десятка разрядов обходился чуть ли не вдвое дороже логического умножения. Да и то, если это был «сдвиг по порядку». А иначе (сдвиг по адресу) цена была бы уже почти 4 гектара. (Все это пока не был введен коммутатор Бессонова, теперь-то сдвиг идет по цене логического умножения.)

Поэтому, например, ходы коня делаются бесцикловой программой (9 гектаров мягкой пахоты). А вот ходы слона — циклической (от 40 до 80 гектаров). Эти внутренние блочки отлизывались по лучшим канонам польской игры. Потому что цена этого дела — двойка-тройка множителем во времени создания всех возможностей (всех возможных ходов из данной позиции). А, в свою очередь, это — самая времяемкая часть программы (процентов 70). Вот отчего и возник социальный заказ на бессоновского СЛОНа. После его введения время на создание возможностей можно будет не учитывать.

§ 4. Ватерлоо АВАУ. Уроки Ватерлоо

Когда все сказанное было сделано, программа стала играть примерно в силу шахматиста III разряда (см. [9], где приведены партии). По крайней мере так оценили ее игру гроссмейстеры (Д. Бронштейн, М. Таль). Кажется, это самая сильная из шахматных программ в мире. По крайней мере ферзя форы гроссмейстеры ей давать не хотят. Это — в дебюте и миттельшпиле. Субъективное ощущение партнера (I разряд), игравшего партию по телефону, — что он играет с человеком. «Металлического привкуса» не чувствуется вовсе. Даже в дебюте, хотя никаких плодотворных дебютных идей в программу не закладывалось.

А вот что касается эндшпиля... Здесь ощущение игры не с плохим, даже не с очень плохим игроком, а, я бы сказал, с ребенком, который

не играет в шахматы. А двигает фигуры и пешки, просто хватая первую попавшуюся. И случайно переставляет ее на разрешенное поле.

Правда, фирма АВАУ эндшпилем н не занималась. Но я-то думаю, что это упорное (принципиальное) нежелание заняться эндшпилем имеет, кроме сознательной, еще и глубокую подсознательную подоплеку.

Однако дело не только в эндшпиле. Когда-то, на заре игровой деятельности, мы наивно полагали, что разница между игроком III разряда и чемпионом мира микроскопически мала по сравнению с разницей между абсолютным идиотом, только что и знающим правила игры, и игроком III разряда. И что если удастся любой ценой достичь игры, приличной в этом смысле, то, скажем, увеличив время на размышление в 100 раз, мы уже совсем бесплатно сможем обыгрывать гроссмейстера.

Как бы не так!

В то время мы совсем не представляли себе, что такое скорость работы подсознания. А по-видимому, где-то в районе II разряда происходит смена караула — вступает подсознание. Со всеми вытекающими отсюда последствиями, которые в форме множителя весят что-нибудь 10^9. Поди угонись. Наверное, от этого и происходит такая безнадежность при соревновании пижона с игроком. Не искупаемая ничем, даже присутствием любимой женщины. Один такой грустный случай реально наблюдался. А игрок-то и был всего лишь кандидатом в мастера...

Так вот, значит, Ватерлоо произошло на переходе от III разряда ко II. Даже если забыть об эндшпиле.

Поскольку выдерживался постулат имени меня, т. е. программу хотели усилить любыми средствами, произошло отступление от принципа по всей линии. То есть пожертвовали абсолютными вещами в пользу неабсолютных, чтобы выиграть время.

Ввели понятие *активного хода*. Ход называется *активным*, если даваемый за ним Ф. В. за себя ведет к выигрышу материала. Посельски это значит: пойду-ка я так. Забудем на минутку, что и партнер ответит. Вот если бы он пропустил ход, смогу я тогда после такого своего хода что-нибудь сразу выиграть? Если да — ход активен.

И теперь отказались от *полного перебора* всех ходов, кроме 2–3 первых полуходов. А оставили к рассмотрению только активные ходы и 1–2 лучших из пассивных (не активных). Лучших в смысле предварительной экспресс-оценки.

Программно здесь действуют с помощью Ф. В. и УХУДУ (см. § 2). Только теперь программе УХУДУ уже и действительно задаются условия (активность хода, максимальная цена по экспресс-оценке пассивного хода).

Время выигралось очень сильно. И игра стала лучше. Одновременно в оценку позиции ввели ряд чисто шахматных, мне малодоступных соображений. И опять получили усиление. Всего этого как не хватало, так и не хватило на преодоление пропасти между III и II разрядами. Можно было не тратить больше сил — идти на дно.

Дело в том, что в неабсолютной схеме, в отличие от абсолютной, дальнейшее углубление перебора очень мало улучшает игру. На 4 полухода программа играет не лучше, чем на 3 (может быть, из-за четности), а на 5 — мизерно лучше. Притом из рассмотрения деревьев видно, что дальнейшее углубление бесполезно вовсе — все глупости, которые она делает на 5 полуходов, она сделала бы и на 7. Действительно, забракованный первый полуход не будет рассматриваться, сколько бы ни углубляли перебор; если истина была в нем, то она утрачена безвозвратно. В результате в неабсолютной схеме «наибольший к. п. д.» получается, по-видимому, при игре на 3 полухода.

Зато очень хорошо видно, как было бы отлично, если бы можно было этот неабсолютный перебор на 3 полухода пустить в качестве форсированного варианта к абсолютному, хотя бы на 2–3 полухода. Разумеется, для этого его нужно во много раз ускорить.

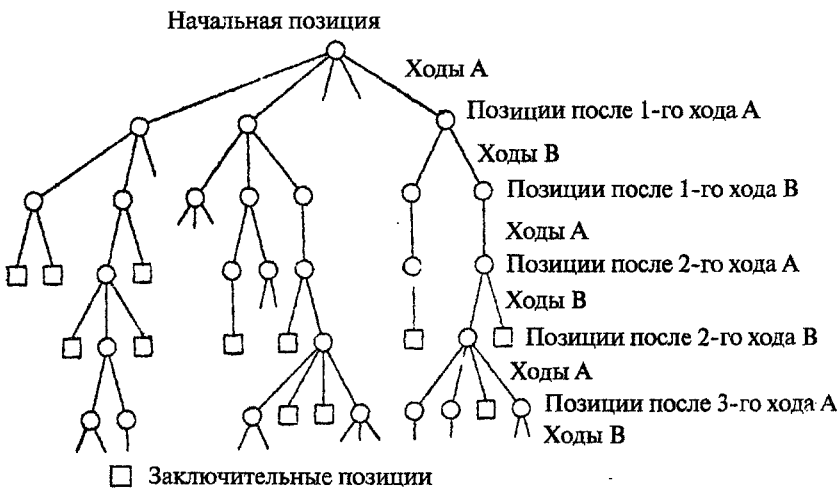
Вероятно, пуск бессоновского СЛОНа вольет струю свежей крови в старческие жилы АВАУ. Сейчас (1964) они делают больше вид, что готовят новый сверхмощный вариант программы. Но из своего Ватерлоо АВАУ извлекли один очень существенный урок.

Состоит он примерно вот в чем. В некоторой позиции нам (программе) показалось, что можно выиграть пешку на e4. Показалось — означает, что так решила экспресс-оценка. Ход, ведущий к такому псевдовыигрышу, основная программа проанализировала детально и увидела, что пешка не предлагается. Теперь в качестве первого полухода программа рассмотрела a2–a3. Последовал некоторый ответ черных. При экспресс-оценке 3-го полухода (второй ход белых) снова пометилось, что пешка на e4 берется. Результат — длинное рассмотрение в основной программе и новое разочарование. Теперь пойдем a2–a4. И — все сначала. Вот на что уходит время, — вещают АВАУ. А ведь между тем ходы a2–a3, a2–a4 *не имеют отношения* к событиям вокруг поля e4. И нечего было повторно десятки раз рассматривать все время одни и те же — буквально ход в ход, кроме a2–a3 и нейтрального ответа черных, — попытки выиграть пешку со всеми теми же опровержениями.

Вот здесь, кажется, АВАУ ухватили быка за рога. Им удалось найти место, *где человек радикально экономит время*, а машина этого сделать не умеет. От постановки вопроса до решения его далеко. Это знает всякий математик, даже настоящий (не программист). И тем не менее, быть может, этот урок Ватерлоо станет началом некоего прорыва. Дай-то Бог.

§ 5. Общая переборная схема

В игре двух партнеров с поочередным правом хода дерево игры получается таким:



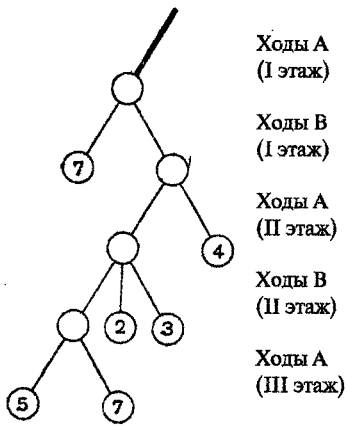
В заключительных позициях мы имеем числовую оценку — выигрыш 1, проигрыш 0, ничья $\frac{1}{2}$ (крестики—нолики) или число взяток ($0—16_{10}$ в одномастке) или — оценку позиции (по материалу и, например, активности в шахматах).

При этом заключительная позиция вовсе не обязана быть заключительной в смысле окончания партии. Программа считает заключительной (концевой) позицию, которая объявляется таковой ее собственным (программным) алгоритмом. Например — после 5 полуходов. Или — после первого взятия (на любом полуходу). Важно только, чтобы каждая ветвь нарисованного дерева — путь сверху вниз по черточкам-ходам A и B — на хвосте несла кружочек с числом-оценкой. Будем считать, что для A хороша наивысшая, а для B — наинизшая оценка.

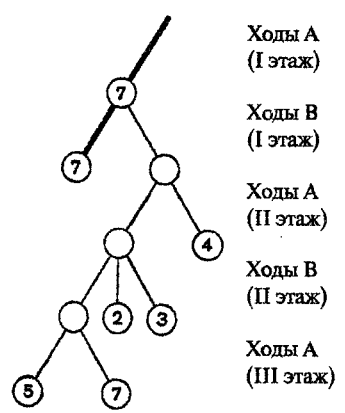
Теперь опишем алгоритм схемы полного перебора. Сперва — без сокращений.

Первым делом — надо упорядочить на каждом этаже ходы партнера. Пусть они перебираются на схеме *слева направо*.

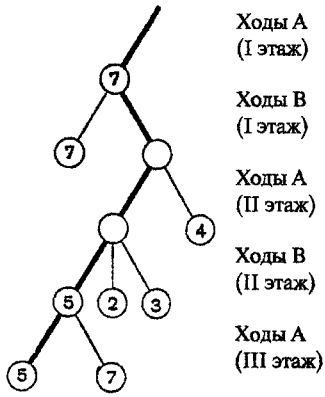
Итак, берем первый ход А. Если скажем, дело выглядит так:



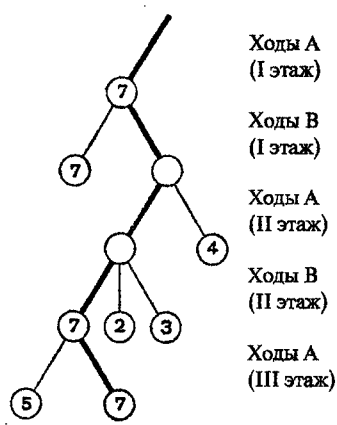
то мы теперь попробуем из получившейся позиции все ходы В по порядку. Первый же ход В приводит к заключительной позиции с оценкой 7. Перенесем эту оценку в позицию после первого хода А. Получится вот что:



Смотрим, не исчерпали ли мы все ходы В. Оказалось, нет. Пробуем второй из первых ходов В. Он привел к неконцевой позиции. Из нее делаем первый по очереди ход А. Позиция снова неконцевая. Берем первый ход В. Позиция опять неконцевая. Наконец, на III этаже за А делаем первый ход. Вышла концевая позиция с оценкой 5. Переносим ее вверх. Получилось так:

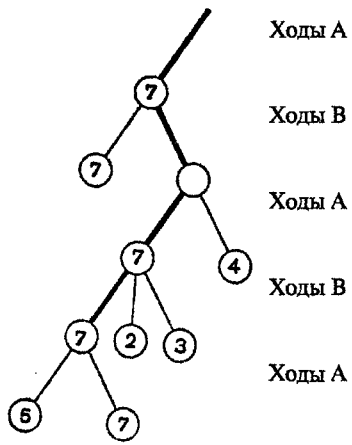


Жирным даны черточки ветки, которая сейчас рассматривается. После переноса вверх оценки мы смотрим — остались ли еще ходы из этого узла вниз (III этаж за A). Оказалось — да. Делаем этот ход. Мы снова в концевой позиции. Теперь оценка 7. Но ведь наш ход — ход A, т. е. A был волен выбирать ход. И уж, конечно, он выберет тот ход, который ему выгоден, т. е. на III этаже ход, приводящий не к оценке 5, а к оценке 7. Поэтому перенесем в узел между II этажом B и III этажом A оценку 7 (max за A). Получится так:

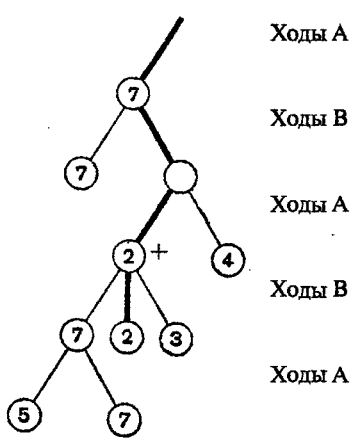


Опять пробуем пойти вниз (ход A на III этаже за A). Но ходов вниз *больше нет*. Придется подниматься вверх. Перенесем полученную оценку 7 в этот узел. Содержательно это значит: если я приду в такой узел (1-й ход A на I этаже за A, 2-й ход B на I этаже за B, 1-й ход A на II этаже

за А), а теперь из него пойду 1-м ходом В на II этаже за В, то при дальнейшей идеальной игре обоих партнеров (в данном случае одного только А, у В дальше ходов нет) я зарабатываю оценку 7, и мы получаем:

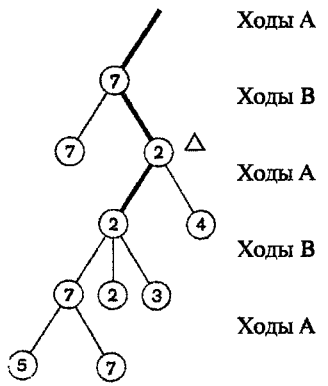


Теперь пробуем следующие ходы В из этого узла. Очередной ход приводит нас к заключительной позиции с оценкой 2. На этом этаже выбор был за В. Поэтому, конечно, В выберет наивыгоднейшую для себя (минимальную!) оценку. И, значит, 7 в узле между II этажом за А и II за В сменится на 2. Имеем:

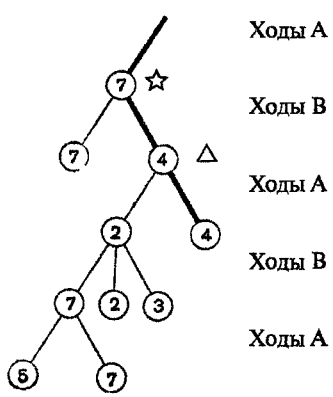


Из этого узла есть еще один ход. Он тоже приводит к концевой позиции, но с оценкой 3. Значит, этот ход В ни к чему. Оценка 2 в узле, помеченном крестиком, не меняется. Из этого узла больше ходов В нет. Пойдем вверх.

Переносим оценку 2 и имеем:

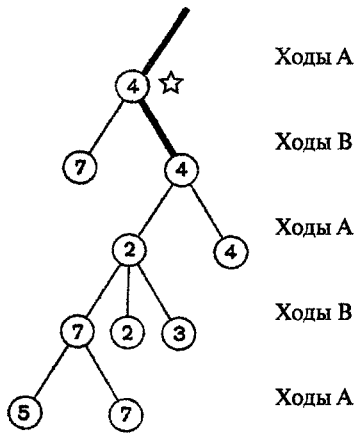


Из узла, отмеченного Δ, мы пока попробовали единственный ход, отмеченный жирной линией. Оценка 2 означает, что если сделать этот ход, то дальнейшая идеальная игра обоих партнеров принесет нам именно оценку 2. Пробуем теперь следующий ход из узла Δ. Концевая позиция дает 4. А выбор на этом этаже снова за А, то есть он пройдет по *максимуму* оценки. Значит, картина станет такой:



Из узла Δ больше ходов нет. Поднимаемся вверх. В узле ☆ стоит 7. Это произошло от рассмотрения 1-го хода В на этом этаже.

Но второй-то ход приносит более радостную для В оценку 4. Значит, выбирая по минимуму, В из такого узла сделает именно не первый, а второй ход. И потому мы заменим нашу картинку на такую:



Теперь уже из ☆ больше В никак пойти не может. Поднимаемся этажом выше. Мы пришли в начальную позицию. Оценку 4 туда и надо перенести из узла ☆. Содержательно это значит: если из начальной позиции А сделает первый из возможных ходов — первый в упорядочении нашего чертежа, — то при дальнейшей идеальной игре обоих партнеров он заработает 4.

Теперь сделаем из начальной позиции следующий ход. После рассмотрения всего дерева мы вернемся к узлу между I этажом А и II этажом В с некоторой оценкой α. Она означает: после того как А сделает на I этаже 2-й по порядку ход, идеальная игра обоих партнеров приведет в концевой позиции к оценке α. Это и есть «цена» такого хода А на I этаже. Так как в этом месте выбор принадлежит А, то, естественно, если $\alpha > 4$, он предпочтет второй по порядку ход, дающий α, первому, который приносит в конце концов только 4. Значит, в оценку начальной позиции надо записать $\max(4, \alpha)$. После этого мы перейдем к следующему ходу на I этаже А и т. д. — пока ходы не иссякнут. Записанная после всего этого оценка начальной позиции в точности означает: если оба противника будут играть идеально, то в концевой позиции оценка окажется именно такой.

Мы рассмотрели схему абсолютного перебора при игре двух партнеров. Еще ее называют минимаксной схемой.

Именно такую схему употреблял Брудно в одномастке, АВАУ — в шахматах, И. Л. Ильина и я — в преферансе, и т. д., и т. п.

С точки зрения программирования в переборной схеме есть нечто любопытное. А именно, переборная схема программируется как *цикл произвольной кратности*. Посмотрим, как такая программа выглядит.

Что нам нужно помнить в каждый данный момент? Прежде всего — где мы находимся. Т. е., поскольку в каждой позиции существует алгоритм упорядоченной выдачи всех возможных ходов, — иначе вообще нет алгоритма выдачи всех ходов, — нам нужно только знать номера ходов на каждом этаже, приведших нас в ту точку, где мы сейчас стоим. Плюс к этому — номер последнего хода, который мы уже сделали из данной позиции. Подчеркнем, что объем информации, которую мы должны хранить для перебора данной глубины k, пропорционален в нашей схеме *этой глубине* k, а не n^k, где n — «средняя кратность ветвления». Обстоятельство немаловажное для машин с конечной памятью.

Такой информации хватит, чтобы ориентироваться — что надо делать дальше. А именно, если из данной точки был сделан в качестве последнего s-й ход, то есть две возможности: либо всех возможных ходов было больше s, тогда мы находим $s+1$-й ход и делаем его (движение вниз или вперед), либо s — это был номер последнего возможного хода. Тогда нам нужно вернуться назад. Если, конечно, мы не находимся в начальной позиции по окончании последнего по счету хода на I этаже — т. е. когда перебор как раз и закончен.

Итак, номеров последних ходов, сделанных из позиций, лежащих на ветке, которая идет из начальной позиции в нашу, довольно, чтобы знать, что делать дальше. Но, кроме моциона, у движения взад-вперед по дереву есть и некоторая побочная цель — выбрать лучший ход А из начальной позиции. А для этого надо при возвращении назад переносить вверх оценку, если ее там не было вообще, или сравнивать ее по минимаксу с полученной и, если надо, менять.

Значит, нужно еще помнить *оценки всех позиций вдоль ветки*, ведущей из начала в данную точку.

Остается только заметить, что для ходов А из начальной позиции надобно знать, каков именно был ход-чемпион, принесший наивысшую оценку, а не только эту оценку. Потому что иначе в конце концов мы будем знать, сколько можно выиграть, но забудем, *как* это сделать.

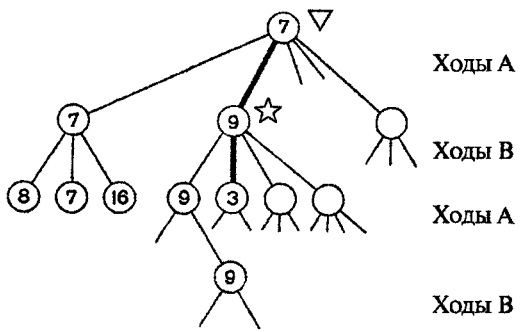
Ну, конечно, в реальной программе по большей части (не всегда!) помнят в каждой позиции, лежащей на рассматриваемой ветке, не только номер последнего сделанного хода, а еще и сами возможные ходы. Чтобы не надо было получать их всякий раз заново.

Затем стоит отметить некоторый чисто технический прием: перед началом работы выгодно во все позиции, откуда ходит А, положить невозможно низкую, а во все позиции с ходом В — невозможно

высокую оценку. Теперь не придется думать о случае, когда некоторая позиция еще не имеет оценки и ее надо переносить, а не сравнивать.

§ 6. Ускорение по Брудно

Пусть возникла ситуация:



Из позиции, отмеченной ☆, мы сделали два хода. На первом мы перенесли оценку 9. А вот второй ход дал нам оценку 3. Следует ли продолжать смотреть остальные ходы из ☆?

Конечно, нет. Потому что ведь из ☆ ходит В. И он выберет ход, дающий *минимальную* оценку. Она, может статься, будет 1 или даже (−5). Но ведь уж никак не больше 3 — этого-то мы уже достигли!

Значит, из позиции ▽ второй по счету ход А выбран не будет — первый был лучше. А значит, можно прекращать рассмотрение всего оставшегося еще куска дерева, ответвляющегося от ☆ вниз — это пустая потеря времени.

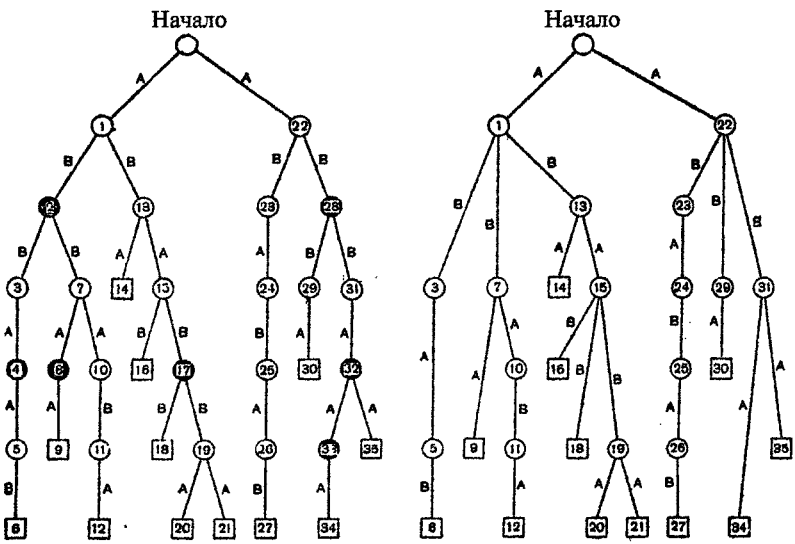
Вот это соображение (проведенное, конечно, на всех уровнях и за А и за В) и есть ускорение по Брудно. Которое, скажем, в одномастке привело к выигрышу по времени не то в 100_{10} не то в $1\,000_{10}$ раз.

Более глубокие (и трудные) возможности ускорения в этом же стиле читатель найдет в работе самого А. Л. Брудно [17].

§ 7. Переборная схема. Обобщения. Векторная оценка

Пусть снова играют два игрока А и В. Только порядок чередования ходов другой: начинает А, а В отвечает. Но в зависимости от этого ответа (в паре с ходом) следующий ход может принадлежать как А, так и В. Как теперь изменится описанная в § 6 переборная схема? Да

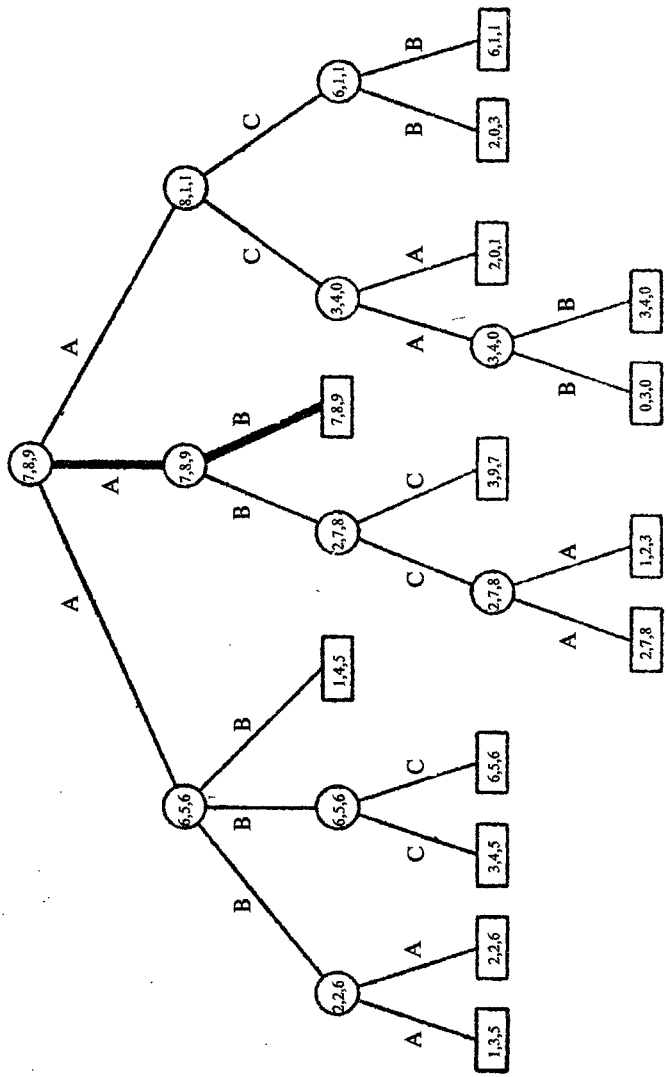
почти никак. Просто в дереве такого вида, как на следующем рисунке слева, надо считать стертыми зачерненные узлы, а идущие *подряд* два или более ходов одного и того же игрока считать за один. Получается дерево, нарисованное на правом рисунке. Для ясности соответствующие узлы на обоих рисунках обозначены одинаковыми цифрами.



А если играют три игрока: A, B, C? Каждый за себя. Ну, тогда, с точки зрения A, его партнеры — одно сообщество. Но не с точки зрения B и C. Они-то хорошо знают про себя, кто из них чего хочет. Как теперь будет выглядеть перебор?

Все дело в оценках, которые должны характеризовать концевые позиции. Теперь ведь эти оценки — не числа. То есть, с точки зрения A, конечно, числа, из которых он при случае будет предпочитать, скажем, большее. Но, с точки зрения B, такие числа, означающие величину выигрыша A, совсем недостаточны. Вообще говоря. Потому что, если выигрыш достался не A, то это, конечно, уже неплохо. Но ведь есть еще C на свете...

Итак, теперь концевую позицию надо характеризовать не числом, а, скажем, тройкой чисел ($Ц_A$, $Ц_B$, $Ц_C$). Эти числа значат: цена позиции (величина выигрыша) для A, для B и для C соответственно.


7,8,9
A
A
A
6,5,6
B
B
B
2,2,6
A
A
1,3,5
2,2,6
B
6,5,6
C
C
3,4,5
6,5,6
B
1,4,5
7,8,9
A
7,8,9
B
B
2,7,8
C
C
2,7,8
A
A
2,7,8
1,2,3
C
3,9,7
B
7,8,9
8,1,1
C
C
3,4,0
A
A
3,4,0
B
B
0,3,0
3,4,0
A
2,0,1
C
6,1,1
B
B
2,0,3
6,1,1

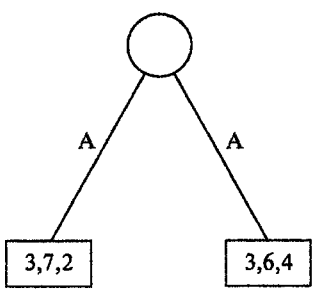
А перебор теперь пойдет почти по прежней схеме. Только в каждом узле, откуда начинаются, скажем, ходы В, выбор будет происходить по максимуму $Ц_В$.

Аналогично в узлах с ходами А и С. Ну, а при двух подряд ходах одного и того же партнера мы будем стирать узел между этими ходами.

Как и в случае двух партнеров. Теперь развитие оценки будет происходить так, как показано на рисунке. Прямоугольники здесь обозначают заключительные позиции.

Мы видим, что в этой ситуации после изучения дерева А изберет второй по счету ход. После чего В изберет правый из возможных ходов (оба эти хода обозначены жирными линиями). После этого игра кончится, и С вообще не получит слова.

Два слова о случае равных оценок.



Что здесь изберет А?

Ничего он не изберет, если это не оговорено условиями. Только эти условия должны быть функциями оценок. Так что ответ «бросим монетку» не подойдет — игра перестанет быть полностью определенной.

Случай $n > 3$ (игроков) оставим читателю. А вот к двум игрокам вернемся. Минимаксная схема для двух игроков — частный случай векторной оценки. Там молчаливо предполагается $Ц_А + Ц_В = \text{const}$. Вообще говоря, это не так. Игра может быть не абсолютно конкурентной. Ну, например, если А и В — штангисты. И ходы их — это установка веса на штанге. А цель — побитие личного или мирового рекорда каждым.

Последнее. Пусть идет игра с природой. Например, переборное доказательство теоремы. Появляется ли здесь переборная схема?

Очень даже просто. Пусть мы искали тремя взвешиваниями единственную фальшивую монетку среди 12 данных. Ход А — выбор монеток для и левой и правой чашек. А ход В? Ходом В здесь является *результат* взвешивания. То есть одна из возможностей — левая чашка тяжелей,

равновесие, правая тяжелей. А что будет оценкой заключительной позиции? Оценок две — скажем 0, если мы остались в невежестве, и 1, если по данной ветви узнали истину. Кстати, это — случай абсолютно конкурентной игры.

Конечно, я вовсе *не утверждаю*, что перебор — единственная форма игры с природой (доказательство теорем).

Но именно *не утверждаю*. Это ведь *не значит*, что я этого *не думаю*. И не удивлюсь, если в том или ином смысле окажется, что перебор именно и есть универсальная форма доказательства. И думания вообще.

А трудности, конечно, в конкретных случаях могут оказаться совсем-совсем не в организации перебора. Уж это-то и правда можно утверждать. С глубоким сожалением.

§ 8. ОБЩИЙ РЕШАТЕЛЬ американцев. Слово в защиту бюрократии

Не мы одни оказались такими умными. Американские программисты [8] тоже заметили очень общий характер переборной схемы.

И сделали свой ОБЩИЙ РЕШАТЕЛЬ (О. Р.).

Что такое О. Р.? Если угодно — *бюрократический аппарат* переборной схемы. Безразличный, как всякий бюрократический аппарат, к конкретному содержанию работы. К тому — будет ли эта работа игрой в шашки. Или — выбором размещения цементных заводов. Или — доказательством теоремы.

Неужто бюрократизм — это хорошо? А вот давайте рассудим.

Можно ли организовать трест вообще? Без того, чтобы предварительно решить: будет это трест по переработке свежемороженой рыбы или по изданию литературы для дошкольников? Так и представляешь себе фельетон Леонида Лиходеева, на корню уничтожающий Воинствующего Бюрократа (В. Б.), буде такой найдется, вздумавшего организовывать трест вообще.

А так ли уж был бы неправ В. Б.? Как вы думаете, машинописное бюро нужно в обоих трестах? А канцелярия? А экспедиция? А бухгалтерия, отдел снабжения, отдел кадров, библиотека, столовая, медпункт (если трест большой)? Заметьте, что все эти службы можно сформировать, не различая свежемороженую рыбу от дошкольной литературы.

Но пойдем дальше. Оба треста должны иметь директоров, заместителей, помощников и секретарей директора. И еще — отделы с начальниками, заместителями, главными специалистами и...

Позвольте, какими такими главными специалистами? По чему специалистами? По рыбе или литературе?

Не позволю. Я ведь не сказал, что мы их найдем заранее. А только — что нужны главные специалисты. То есть вакансии. А нанимать их

даже и В. Б. не станет заранее. Вот когда решится вопрос про рыбу и литературу, тогда и наймет. Как и начальников отделов. А что до штатного расписания — то, боюсь, его можно составить чуть ли не до последней должности. Разве что наименования придется сменить. Ну, там вместо главного рыбовода Г. Р. появится главный редактор (тоже Г. Р., но ведь какая разница в содержании!!!).

Все сказанное — вовсе не шутка. Заметьте, что я не говорил еще о таких вещах, как здание, котельная, гараж. В общем-то, если вдуматься, то развитие идет ведь в сторону *увеличения роли управленческого и обслуживающего аппарата*. И не надо этого так уж сильно бояться. Потому что хороший АППАРАТ обеспечивает такой подъем производительности труда, что общество в целом от этого только выигрывает. И немало. Тем более, что при переходе к заводам-автоматам с 4 сторожами и 4 сменными инженерами на завод трудно будет управленческий аппарат довести до «не свыше 10 % от числа занятых непосредственно на производстве». Если не отрезать от бюрократа-управленца некоторых деталей. Но это уже преступление уголовное. Даже по отношению к самому заядлому бюрократу. Не так ли?

Раз уж мы отвлеклись — два слова об аналогиях. С развитием (ускорением) программ роль управляющих команд — место (в смысле числа ячеек) и даже % времени, занятого ими от общего времени работы программы, — неуклонно растет. Анализ, проведенный Г. С. Разиной, показал, что команда *передачи управления с возвратом* (чисто бюрократическая!) стала *конкурировать по частоте выполнения* с непосредственно занятыми на производстве командами *сложения и умножения*. Ну, а если взглянуть на программу, написанную на бумаге, *трудно* бывает *найти место*, где вообще-то есть хоть одна не бюрократическая команда. И чем совершенней (производительней) программа, тем более она сдвинута в эту сторону.

Если посмотреть блок за блоком квалифицированную универсальную программу, то ведь что там делается!

Сперва все начинается с блок-программы (чистое управление, да еще от него можно и вовсе отказаться, а освободившиеся вакансии — пардон, ячейки — направить непосредственно на производство — умножать, делить, извлекать корни...).

Дальше пойдет блок настройки по заданным на ДЗУ параметрам. Это уже чистое безобразие. Не знали n и m — так нечего было и работу начинать! Ведь подумать только — вместо того, чтобы написать, где надо, 3 и 4. Прямо, по-деловому. Так вместо этого развели волокиту. Те же 3 и 4 почему-то набрали на клавишах. Да еще не в той форме. Затем сводку о том, что там набрано, переслали из одной инстанции в другую (из ДЗУ в $R1$). Да еще по дороге перенесли туда совсем уж

ненужные сведения о печатях по требованию (в коде). Конечно, целую простыню прислали.

Теперь, думаете, все? Как бы не так. Поскольку вместо честных 3 и 4 прислали простыню-сводку, где сказано про кучу ненужного для нашего дела, придется теперь тратить время на то, чтобы выковырять изюм из булочки (высечь II и III адреса $R1$). И выковыренные сведения разложить еще в отдельные папки ($R3 = (0, 3, 0)$; $R4 = (0, 0, 4)$).

Теперь, казалось бы, уже все. Так нет. Поскольку форма была, естественно, неподходящей, надо опять все переделать. То есть $0, 3, 0$ превратить в 3 настоящее. Думаете, как это будет происходить?! Припишем порядок, нормализуем и все? Как бы не так. Теперь $0, 3, 0$ перешлется в очередную инстанцию — ячейку α. И дальше пойдет сопроводиловка:

Направляем Вам в ячейке α для переработки некоторое число (это, конечно, чисто для важности, ведь число-то это просто 3, т. е. $(0, 3, 0)$), просим переработать в другую форму и ответ прислать. Наш адрес (это про $Я+1$) Вам... нет, не сообщаем, а всего-то направляем в общем порядке в положенной форме в должную инстанцию (Ω). И отдаем Вам управление. А пока Вы не ответите, больше ничего делать не будем. Привет — т. е. передача управления.

Думаете, куда передача управления — рабочей программе? Как бы не так. Всего лишь в общую канцелярию — к заголовку программы ФП-2. А заголовок передаст дело (т. е. управление) по инстанции — на рабочую часть программы. Думаете, теперь конец волоките? Нет. Оказалось, что присланная бумага $(0, 3, 0)$ написана хотя и по общезаводской (Ф. П.) форме, но не по цеховой. А цех требует не $(0, 3, 0)$, а $(3, 0, 0)$. Значит, перепишем ее по новой форме (сдвиг влево на адрес). Теперь годится? Конечно. Но только подождите — мы теперь должны передать управление цеху (безусловная передача управления рабочей части Ф. П.). Наконец, дело дошло до производственных команд. (Гм-гм, они что производят-то? Называют кошкой кошку? Ну, все-таки ближе к делу, чем пересылать с места на место). Приделали порядок. Нормализовали. И всего-то их 2. А управленческого персонала на их шее... В 3 этажа!

Ну-с, появилась нужная нам тройка — 3 плавающее. Но где? В ячейке β — общеканцелярской папке ответов. Теперь, думаете, приступим сразу к дальнейшей работе? Как бы не так. Ну, хоть управление передадим в прежнюю инстанцию — которая нам направляла $(0, 3, 0)$ для переделки? Опять нет. Управление мы передадим, но не той инстанции, а общей инстанции (Ω), где хранятся сведения об адресе этой самой инстанции, обратившейся к нашей инстанции с поручением о переделке тройки в тройку. А вот уж эта инстанция — Ω, она и точно передаст управление той инстанции, которая производила запрос.

Потому что ее, Ω, дело в том только и заключается, чтобы хранить сведения об адресе этой самой инстанции. Так-то.

Как Вы думаете, можно уже судить за все это как за головотяпство со взломом? Даже, пожалуй, злостное. А ведь заметьте, читатель, это мы обращались к программе, кончающейся командой

Б Ω

А еще же могло бы быть

Ω = конец

Но про это я рассказывать просто забоялся. А то ведь — неровен час — и правда засудят. И поделом.

Так вернемся все-таки к ОБЩЕМУ РЕШАТЕЛЮ? Давайте попробуем.

Итак, О. Р. — это бюрократическая часть переборной схемы. Как ее организовать — видно из прошлого параграфа.

И из него же видно, какие блоки надо написать, чтобы превратить трест вообще в трест по заготовке рыбы.

Прежде всего — блок создания ходов (с упорядочением их), т. е. конкретного содержания того, что в данной задаче перебирается (ход шашки, вариант размещения заводов). Но О. Р. до этого содержания дела нет. Ему только и надо знать — сколько этих возможностей в данном узле было. Следовательно, это число надо положить в приемник О. Р. Вернее, написать *команды передачи* числа в этот приемник, а сама передача будет происходить по требованию О. Р. Ну, затем рабочим является блок оценки. И опять — нужна стандартная связь с О. Р.

А теперь остается стандартизовать информацию о ходе. Что сделать не так-то просто: ход может быть и разрядом ячейки, и — целым массивом слов (в зависимости от задачи).

Естественно, если мы хотим ввести любые абсолютные сокращения по Брудно, это может сделать О. Р.

А вот если мы пожелаем ускорять, согласившись на *неабсолютную* схему (как в Ватерлоо АВАУ), блоки экспресс-оценок придется писать самим.

А на долю О. Р. придутся программы УХУДУ и ПУП.

Конечно, бюрократическую схему предварительных оценок в О. Р. можно ввести. Опять же стандартизовав информационную связь.

В общем, как видите, главная работа при использовании О. Р. состоит в конкретном выборе способа задания информации в ячейках машины. Так что, с точки зрения математиков, — публикации в журнале такое не подлежит. Потому что жалко бумаги. Не сговориться нам, программистам, с настоящими математиками...

Про О. Р. я кое-что придумал от себя. Как дело обстоит в точности, читатель может узнать в [8]. А сделать свой О. Р. полезно было бы и нам. Хотя очень больших выгод от этого ждать не следует. Потому что, несмотря на все сказанное, большое обобщение влечет большую растрату времени. Все-таки бюрократия — это не только хорошо...

§ 9. Шахматы и народное хозяйство

Строительство комбината состоит из большого числа отдельных работ. Одни из них — постройка корпуса ТЭЦ, монтаж колонн азотно-кислотного цеха — друг от друга не зависят и могут выполняться хоть последовательно, хоть в параллель. Другие — жестко связаны порядком: фундамент цеха неловко строить после возведения стен.

Если начало и конец каждой работы обозначить точками, а соединяющую их палку считать самой работой и надписать на этой палке ее продолжительность, получится сетевой график. Если, конечно, позаботиться о том, чтобы в точку — начало данной работы вели линии всех работ, которые *обязаны* предшествовать данной. При этом одна и та же работа может оказаться повторенной во многих черточках.

Теперь можно найти *лимитирующую* цепочку — ту последовательность работ, которая ограничивает снизу срок завершения постройки комбината.

Программа, которая это делает, называется ПЕРТ. Ее придумали американцы.

В 1962 году один проектный институт попросил нас решить для них такую задачу. Г. М. Адельсон-Вельский и Ф. М. Филлер за пару недель из свойственной им чистой любезности без труда сотворили такую программу и решили задачу. И отдали заказчикам. Труд и правда был невелик — взяли они из шахматной программы переборную схему (переписали, конечно, поскольку это все же не ОБЩИЙ РЕШАТЕЛЬ. хотя — и не так уж далеко от О. Р.). И осталось им всего-то приладить информацию. Что они не без блеска и сделали. Решили задачу и — перешли к очередным делам.

А между тем повторить программу ПЕРТ понадобилось в общехозяйственном масштабе. Поскольку, действительно, невредно знать, какие работы надо форсировать. А какие — могут без вреда подождать. Например, чтобы направить, куда надо, дефицитные материалы и сэкономить миллионы рублей.

И оказалось, что ВЦ и НИИ, которым поручили такую работу, за год с ней не справились (может быть, оттого, что не занимались играми, а только настоящей работой). И даже просили послать кого-нибудь в Америку — поучиться. Тут беднягам не повезло. Довольно

случайно (журналы — см. [20] — таких статей не печатают) узнали по начальству, что программа ПЕРТ уже с полгода в Институте теоретической и экспериментальной физики есть. И послали учиться не в Америку, а в ИТЭФ. То есть гораздо ближе.

Порадовались мы, напечатали программу на ротапринте, размножили колоды и разослали по градам и весям (Новосибирск, Киев и т. д. и т. п.).

А вот напечатать ПЕРТ все-таки не привелось.

В журнале [20] [3)] подумали-подумали и сказали, что бумаги на это жалко. Правда, не на все, а на рассказ про конкретную информацию: как что уложить в ячейки, поскольку это не АЛГОЛ [4)].

А других журналов по программированию у нас нет. Не сговориться программистам с настоящими математиками. А жаль!

[3)] Главный редактор — акад. А. А. Дородницын. Ср. примечание к с. 112.

[4)] Пока эта книга набиралась, статью о такой программе напечатать (в урезанном виде) все же пришлось [21]. Теперь там не печатают [13] и [14].

Беседа тринадцатая

О взвешивании монеток

§ 1. Постановка задачи

Из 12 монеток одна — фальшивая. Легче она или тяжелее хороших — не сказано. Только известно, что 11 монеток одного веса, а фальшивая — другого. Надо тремя взвешиваниями на чашечных весах найти эту фальшивую монетку.

Из 13 монеток можно тоже тремя взвешиваниями выделить фальшивую. Только не всякий раз мы одновременно установим, в чем ее дефект — легче она или тяжелее хороших. А для 12 узнается и это.

Из 14 монеток найти фальшивую тремя взвешиваниями нельзя. То есть можно доказать, что как бы мы ни производили взвешивания, результаты могут складываться так, что мы не отделим по этим результатам фальшивую монетку от хороших. Еще точнее: для такой заданной процедуры взвешивания можно указать непротиворечивые результаты каждого взвешивания и такую пару монеток A и B, что, будь одна из них фальшивой, — результаты всех трех взвешиваний будут именно такими, какие указаны. Такое уточнение слов «из 14 монеток за три взвешивания нельзя выделить фальшивую» будет необходимо, когда мы перейдем к программам.

Из 14 монеток тем не менее находится фальшивая, если нам, кроме этих 14, даны еще хорошие монетки (хватит и одной).

Из 15 монеток фальшивая тремя взвешиваниями не находится будь нам, сверх того, дан хоть мешок хороших монет.

Конечно, все эти результаты можно получить с помощью программы. Нормальной программы перебора в 6 этажей: три полухода А — это то, что мы сами делаем; полуход A состоит в выборе монеток для левой и правой чашек весов.

А три полухода B — это результаты взвешивания, причем на каждом из своих трех полуходов B имеет по три возможности [1]: левая

[1] Кроме случаев, когда мы добавляли монетки на чашки после предыдущего взвешивания, и еще некоторых других; тогда у B бывает и меньше возможных ходов.

чашка тяжелее, равновесие, правая тяжелее. Классический случай игры двух игроков.

Теперь перейдем к четырем взвешиваниям. Результаты здесь такие:

12 заменится на 39
13 —"— на 40
14 —"— на 41

Вместо 15 будет теперь невозможно выделить фальшивую из 42 монеток.

Это тоже может выяснить программа. Только полуходов будет не 6, а 8.

Ну, а если перейти от 3–4 взвешиваний к произвольному k? Человек это — худо ли, хорошо ли — делает.

§ 2. А как это можно осуществить в программе? Что значит получить ответ?

Прежде всего, как он, этот ответ, должен выглядеть? Например, ведь программа общепереборного типа со свободным заданием глубины перебора тоже в некотором смысле решает нашу задачу. При каждом *заданном* k она хоть и за долгий срок, а ответ выдает. Тем не менее такое «решение» нас не устраивает.

Мы хотим получить решение «в буквах». То есть доказательство того, что верна некоторая формула, зависящая от числа взвешиваний k. И притом — *для любого* k. Это, казалось бы, повелительно требует, чтобы ответ имел одну из следующих форм:

А. Арифметическая формула, в которой в некоторых местах *стоит не число, а буква* k. По-видимому, без ограничения общности можно считать, что формула задается следующим образом.

Про некоторые величины (у нас — k) *оговорено правилами* игры, что они считаются *буквенными переменными*, принимающими значения из области, тоже *оговоренной правилами* игры. То есть, скажем, адреса с 7 001 до 7 020 означают буквы $k, l, \ldots, t$ [2]. И как-то указана область изменения каждой буквы.

В нашем случае достаточно одного-единственного адреса — для k. А область задания — натуральной ряд. Теперь формализуем запись формул в машине. Ну, скажем, введем для простоты символы для:

1°. Арифметических действий $(+, -, \times, :)$.
2°. Скобок.

[2] Именно *адреса*. Фактически в ячейках с 7 001 по 7 020 при этом может лежать что угодно — числа, команды, Щ.

3°. Чисел.

4°. Символов-указаний начала и конца формулы.

Теперь, например, формула

$$\beta = k + (2 - k)(k^2 + 3k + 7)$$

может быть выписана так:

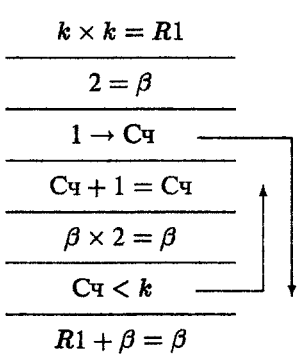
Начало формулы
$k \times k = R1$
$3 \times k = R2$
$R1 + R2 = R3$
$R3 + 7 = R4$
$2 - k = R5$
$R4 \times R5 = R6$
$R6 + k = \beta$
Конец формулы.

Один из двух символов «Начало формулы», «Конец формулы» излишен — о начале мы можем догадаться по *адресу первого слова* ($k \times k = R1$). А можно поступить и наоборот: в начальном слове указать длину формулы — число слов в ней. Теперь сделался излишним символ конца. Ну, это мелочи. А вот гораздо серьезнее такое. Пусть наша формула имеет вид:

$$k^2 + 2^k$$

Как записать ее в машине?

Нам не обойтись *без цикла*. С циклом же она будет, выглядеть так:



Конец формулы.

Формально дело обстоит ничуть не хуже, чем в первом случае. Однако, если мы вздумаем и в самом деле работать с формулами этого типа, возникают чрезвычайные трудности.

Впрочем, уже и в случае формулы, записанной в форме программы без цикла, эти трудности очень велики. Однако можно себе представить программу, которая была бы в силах проверить *эквивалентность* двух формул, заданных в форме бесцикловой программы, ну, скажем,

$$\begin{array}{rcl} k \times k = R1 & & k + 3 = R1 \\ 3 \times k = R2 & & R1 \times k = R2 \\ R1 + R2 = R3 & \text{и} & R2 + 7 = \beta \\ R3 + 7 = \beta & & \text{Конец формулы} \\ \text{Конец формулы} & & \end{array}$$

Преобразовывать такие формулы программно тоже можно, но неудобно. Несколько предпочтительней кажется запись по типу обычной арифметики. Например:

Открывающая скобка
k
$+$
3
Закрывающая скобка
$\times$
k
$+$
7
$=$
β
Конец формулы.

Или, для экономии места:

$(, k, +$
$3,), \times$
$k, +, 7$
$=, \beta$, конец формулы.

По-видимому, с такой записью работать будет легче.

В общем-то с бесцикловыми формулами на первый взгляд мы управимся. А вот что делать с формулой, содержащей цикл?

Представляется, что для ее доказательства *не обойтись без индукции по переменной букве*.

Ну, а тогда можно и сразу перейти от рассматриваемого вида записи А к такой записи, где формула-ответ так-таки и дается в «индуктивной форме».

Может быть, все же мы можем спастись, *введя прямо* в список разрешенных действий возведение в степень? То есть добавив к символам $+, -, \times, :$ еще, скажем, символ exp, означающий, что предыдущее число, букву или скобку следует возвести в степень с показателем — непосредственно следующим числом, буквой или скобкой?

Может быть. Но представляется (отнюдь не доказывается!), что выход этот довольно иллюзорный. Нам придется худо при проведении доказательств: нужно будет или вводить еще целый список правил действий со степенями, или все равно переходить к индуктивной форме В. Но все это — скорее ощущение, чем рассуждение.

В. Формула-ответ в «индуктивной форме». Примерно это должно выглядеть так. Пусть опять адреса с 7 001 по 7 020 предназначены для символов, соответствующих величинам при $k = n$. А, например, адреса с 7 101 по 7 121 — для величин при $k = n + 1$

Тогда индуктивное задание формулы

$$x_k = k^2 + 2^k$$

будет выглядеть так:

$$k_{\text{старое}} + 1 = k_{\text{новое}}$$

$$M_{\text{старое}} \times 2 = M_{\text{новое}} \text{ (это про } 2^k\text{)}$$

$$k_{\text{новое}} \times k_{\text{новое}} = R1$$

$$M_{\text{новое}} + R1 = x_{\text{новое}}$$

Конец формулы.

Ну и еще, конечно, придется написать, что $M = 2$ при $k = 1$.

Заметим, что индексы «старое» и «новое» в приведенной записи *не нужны в рабочей части*. Можно писать так:

$$k \times k = R1$$

$$M + R1 = x$$

Конец формулы

и лишь индуктивную часть

$$k_{\text{старое}} + 1 = k_{\text{новое}}$$

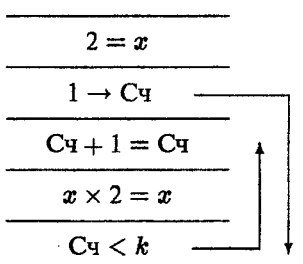
$$M_{\text{старое}} \times 2 = M_{\text{новое}}$$

снабдить этими индексами.

Запись формул в «индуктивном виде» представляется мне более перспективной в смысле реального проведения доказательств.

§ 3. Возражения по форме (ответа)

А ведь мы так и не сказали, чем нам не мила переборная схема. Смотрите — формула дает ответ и схема дает ответ. Правда, формула дает ответ при любом k *за одно и то же* число действий. А у схемы продолжительность работы зависит от k. Но это только для бесцикловых формул. И — безындукционных. А ведь уже в таком виде



число действий для получения $x(k)$ так-таки пропорционально k. И введение операции exp нас не спасет — подумайте о формуле вида:

$$x(k) = \frac{1}{1^2} + \frac{1}{2^2} + \ldots + \frac{1}{k^2}$$

И тем не менее «всякому ясно», чем последняя формула лучше переборной схемы. Мне представляется, что дело даже не в экспоненциальном характере роста числа действий в переборной схеме против степенного (линейного в нашем примере, кубичного для случая перемножения матриц и т. д.) характера «хороших» ответов. Сказать, в чем разница, — не умею. Но отчетливо чувствую ее. Может быть, математическая логика, которой я пока не знаю, может дать в этом месте вместо ощущений точные критерии? Мне кажется возможным пока еще двигаться вперед, не размышляя над этим местом. Но вообще-то ясно чувствуется, что мы подошли вплотную к границам чужой державы — Математической Логики. Правда, быть может, к ее окраинным владениям — элементарным областям. По-видимому, придется изучать нравы, обычаи и язык ее коренных жителей. Потому что, вероятно, они-то лучше приспособлены к природным условиям своей страны, чем пришельцы-программисты. Ну что ж — изучим при нужде. Когда нужда подгоняет — дело идет быстро, куда быстрей, чем если учить впрок! Впрочем, может быть, это кому как. Автору — так.

§ 4. Перебор возможностей. Молекулы

Итак, мы хотим взвешивать монетки. Пусть для всех $k < n$ мы уже знаем, из скольких монеток можно, а из скольких нельзя выделить фальшивую ценой k взвешиваний. Как в машине это наше знание выглядит — вопрос тоже не простой, но пока забудем об этом.

Мы теперь занимаемся случаем $k = n$. Причем n — буква!

Может ли переборная схема помочь нам в такой ситуации? Ведь на чашки весов-то мы должны положить *монетки в некотором количестве штук и, более того, — запомнить, какие монетки где лежали.* Иначе, если мы забудем, кто был легче, а кто — тяжелее, что будет нам дальше толку от проведенного взвешивания при следующих?

А ведь в *буквенном* случае не перепишешь монетки, какие были на левой чашке, какие — на правой...

Ну, а как это делает человек? Он ведь как-то обходится с буквами.

Мне представляется дело так. До всяких взвешиваний монетки образуют единый массив; мы имеем про каждую из них *одинаковую информацию.* Теперь произошло некоторое взвешивание. Множество монеток разделилось на три подмножества: монетки на левой чашке, на правой и — вне весов. Ну, еще с непринципиальной, хотя и важной технически, оговоркой, что число элементов-монеток в подмножествах левой и правой чашек одинаково. Что произойдет после взвешивания? Теперь про элементы этих подмножеств информация уже будет *разная.*

Т. е., скажем, при равновесии мы будем знать, что монетки левой и правой чашек — все хорошие. А среди оставшихся вне весов — есть одна фальшивая.

Но внутри каждого из подмножеств все монетки с точки зрения нашей информации *равноправны.* То есть они как бы образуют *единую молекулу* — мы не имеем права предпочесть один ее атом-монетку другой монетке-атому. Это, если угодно, переложение на такой химический язык недосказанного в условиях задачи требования: «... k взвешиваниями на чашечных весах, *не пользуясь больше ничем*».

В случае равновесия монетки левой и правой чашек можно «ссыпать в *мешок хороших*» — нам, во-первых, безразлично в дальнейшем, кто из них лежал слева, а кто — справа. Но более того, они ничем не выделяются в дальнейшем от остальных хороших монеток — буде такие к этому моменту уже имелись.

В случае *неравновесия* в мешок хороших можно ссыпать монетки-атомы из молекулы, оставшейся вне весов. А «левые» и «правые» монетки образуют теперь две *неравноправные молекулы: тяжелую и легкую.*

Итак, пусть у нас 2 молекулы (тяжелая и легкая) и — мешок с хорошими монетками (быть может пустой!). Что мы должны знать о них? Ну, во-первых, что одна молекула была «тяжелой», а другая

«легкой». И еще — *число* атомов в каждой молекуле. А больше — ничего: к конкретным номерам монеток, образующих данную молекулу, мы глубоко равнодушны.

Теперь мы переходим к новому взвешиванию.

В чем оно состоит? Очередь хода — за нами. Это — ход А: мы решаем, *какие монетки положить на левую, а какие — на правую чашку.* Но ведь монетки в молекулах равноправны! Молекул у нас пока 3 (тяжелая, легкая, мешок). И в силу этого нам нужно только решить, СКОЛЬКО (а не какие!) монеток тяжелой молекулы пойдет на левую чашку, а сколько — на правую. И то же про монетки легкой молекулы. И — про монетки мешка.

Теперь тяжелая и легкая молекулы разделились на 2 части каждая (конечно, одна из них могла и не делиться: может статься, мы брали для взвешивания монетки, например, только из одной легкой молекулы). И у нас стало перед взвешиванием, кроме мешка, уже 4 молекулы. Произвели взвешивание. Теперь мы имеем 4 неравноправные молекулы плюс мешок. Конечно, хорошие монетки при взвешивании играли роль катализатора и могут быть возвращены обратно в свой мешок. А вот легкая и тяжелая молекулы, и правда, поделились.

Конкретно в условиях нашей задачи дело будет выглядеть так:

1°. Случай равновесия при втором взвешивании.

Все монетки обеих чашек ссыпаются в мешок хороших. Мы по-прежнему остались при двух молекулах — легкой и тяжелой, только число атомов в них уменьшилось.

2°. Пусть *левая* чашка оказалась при втором взвешивании *легче правой.*

Теперь в мешок хороших можно ссыпать остатки легкой и тяжелой молекул, не попавшие на весы. Дальше туда же можно сбросить атомы тяжелой молекулы, присутствовавшие во втором взвешивании на левой чашке, и атомы легкой молекулы, лежавшие на правой чашке. Теперь мы снова пришли к *двум* молекулам — легкой и тяжелой. Опять-таки с уменьшенным (хотя бы в одной) числом атомов. Если, конечно, мы не взвешивали бесцельно одни хорошие монетки.

Мы пришли к некоторому новому, более перспективному, взгляду на задачу. Число-то монеток, конечно, осталось буквенным. А вот число *молекул* стало конкретным числом. Более того, их бывает либо *ровно две* — тяжелая и легкая, либо *одна*: тяжелая молекула, легкая или, наконец, «первозданная» — молекула, состоящая из монеток, *не подвергавшихся* взвешиванию, но заведомо содержащая фальшивую монету. Очевидно, что если не пуста первозданная молекула, то и тяжелая и легкая пусты. Это нам еще понадобится. А пока отметим, что во всех случаях после взвешивания у нас получается не больше двух молекул. Это уже сулит нечто.

§ 5. Перебор возможностей. Молекулярный вес. Ранги гипотез

С числом молекул мы справились. Это теперь честная двойка, а никакая не буква. А как быть с числом атомов-монеток в молекуле? Заметим, что задача решена тогда и только тогда, если у нас осталась, скажем, тяжелая (или легкая) молекула из одного атома, а легкая (соответственно тяжелая) — пуста.

Или — если в первозданной молекуле остался один атом. Только в этом последнем случае мы не знаем, в чем дефект фальшивой монеты — легче она или тяжелее хороших. А считать ли это решением задачи, зависит от условий игры.

Все это, конечно, хорошо, а все же как нам выбирать число монеток при создании новых молекул на своем ходе (очередное взвешивание)?

Ведь число-то атомов в молекулах пока что *буквенное*. Теперь мы примем некоторое содержательное решение. То есть, как обычно, *неабсолютное*. Обратимся к бездоказательной аналогии.

Как поступает человек, когда он решает эту задачу? С моей (см. § 9 в беседе одиннадцатой) точки зрения, человек должен *перебирать возможности*. То есть в нашем конкретном случае это — *различные молекулярные веса* — числа атомов в создаваемых для взвешивания новых молекулах.

Ну, а как справляется человек с буквенными затруднениями?

Думаю, что перебор идет по *рангам сложности гипотезы*. А громкие эти слова имеют следующее негромкое содержание.

Перед началом выбора человек имеет условия задачи. И в нем фигурируют буквы: n — сколько разрешено взвешиваний сейчас, N — сколько у него есть монеток, $N(n-1)$ — из какого максимального числа монет можно было выделить фальшивую за $n-1$ взвешиваний, $N(n-2)$ — то же для $n-2$ взвешиваний и т. д.

И когда человек начинает перебирать возможности, он первым делом берет *простейшие* возможности.

Ну, скажем, попробует класть на каждую из чашек весов число монеток, которое *прямо равно* одной из букв, фигурирующих в условиях, т. е.:

$$N;\ n;\ n-1;\ N(n-1) \text{ и т. д.}$$

И еще, конечно, придется добавить простейшее из натуральных чисел, т. е. 1. Фактически, быть может, выгодно включить еще 2, 3; но для нас это пока неважно — мы отвлекаемся от конкретной машины, ее скорости, объема памяти. Мы всего лишь ищем возможную *схему конечной программы*. И то не легко!

Итак, возможности *первого* ранга при создании молекулярных весов — это взять одну из перечисленных букв и еще число 1.

Попробовали — не получится ли? Т. е. не сведется ли задача после такого взвешивания к уже известному (по индукции) случаю? Получилось — отлично. Не получилось — переходим к *рангу два*.

Теперь комбинируем буквы и числа, встречавшиеся в *первом* ранге возможностей, по 2. У нас возникнут:

$N \pm N$; $N \pm n$; $N \pm (n-1)$; $N \pm N(n-1)$; $N \pm 1$; $n \pm n$; $n \pm (n-1)$; $n \pm N(n-1)$; $n \pm 1$; $(n-1) \pm (n-1)$; $(n-1) \pm N(n-1)$; $(n-1) \pm 1$; $N(n-1) \pm N(n-1)$; $N(n-1) \pm 1$; наконец, 1 ± 1.

Все эти гипотезы второго ранга (часть из них, конечно, вздорна, а часть — повторяется) мы снова проверяем: а что если молекулярные веса брать такими — не сведется ли задача к известному индуктивно случаю?

Дальше, используя разрешенные символы второго и первого рангов, мы из них конструируем символы третьего ранга и т. д.

Получается не очень чисто — нужно поаккуратнее сказать про вещи типа $n-2$; $N(n-3)$ и т. д. На каком они, собственно, ранге включаются? Но это особо, а пока заметим вот что.

РАДОСТНОЕ ОБСТОЯТЕЛЬСТВО состоит в том, что у нас на каждом шагу сделалось конечное число гипотез, подлежащих рассмотрению. Теперь мы можем производить перебор гипотез. Он стал конечным! И если набор операций для конструирования символов следующего ранга из символов предыдущего ранга достаточен (в нашем конкретном случае кроме $\pm$ еще полезно иметь такие вещи, как max, min), то вроде бы мы придем рано или поздно к решению задачи программой, если к нему приходит человек.

НЕСКОЛЬКО ГРУСТНОЕ ОБСТОЯТЕЛЬСТВО состоит в том, что человек догадывается еще, что, например, из 16 монеток тремя взвешиваниями фальшивой не найти (соответственно k взвешиваниями из, скажем, 3^{k+5} монеток). А если мы разрешим программе перебирать все более к более сложные гипотезы, то она будет бесконечное время работать над неразрешимой задачей.

И ЕЩЕ ГОРАЗДО БОЛЕЕ ГРУСТНО другое. Об этом мы скажем позже. А пока посмотрим внимательно, как решает задачу человек!

Рассудить окончательно он может, например, так.

§ 6. Решение задачи человеком

Сперва решим более простую задачу. Пусть нам разрешены k взвешиваний. Из какого максимального числа $N(k)$ монеток мы сможем найти фальшивую, если сказано, что она легче хороших?

При $k = 1$ непосредственно убеждаемся, что $N(k) \geqslant 3$. Для этого взвешиваем

$$1 : 1 \text{ и } 1 \text{ вне весов}$$

и все готово. Теперь остается показать, что из $\geqslant 4$ монеток фальшивую одним взвешиванием не найти. Но и это просто: при разбиении массива из более чем *трех* монеток одна из молекул получится более чем одноатомной. И тем все доказано.

Пусть мы уже имеем $N(n-1)$. Перейдем к случаю $k = n \geqslant 2$. Перебирая, подобно машине, возможности, уже на недалеком ранге мы придем к ситуации

$$N(n-1) : N(n-1) \text{ и } N(n-1) \text{ вне весов},$$

если только у нас монеток хватит.

Получается, что $N(n) \geqslant N(n-1) + N(n-1) + N(n-1)$. А для $N > 3N(n-1)$ в одной из молекул после первого взвешивания атомов будет *больше, чем* $N(n-1)$.

Все ли этим доказано? Отнюдь нет. Ведь мы теперь имеем несколько новую ситуацию. Кроме подозрительной молекулы, содержащей фальшивую монетку, у нас появился еще непустой мешок хороших монет. Правда, он нам не помогает. Но ведь это нужно было оговорить с самого начала! Следовательно, полное решение должно было бы выглядеть так:

Определим $N(k)$ как максимальное число монеток, из которых за k взвешиваний находится фальшивая, если у нас был пустой мешок хороших монет. А через $M(k)$ обозначим *минимальное* число монеток, из которых за k взвешиваний фальшивая монетка не выделяется, будь у нас хоть бесконечный мешок с хорошими монетками.

Нам повезло, потому что в действительности

$$M(k) = N(k) + 1.$$

А ведь могло бы быть и так, что, скажем,

$$M(k) = N(k) + 3k.$$

Для промежуточных случаев пришлось бы тогда вводить понятие $N(k, p)$ — максимальное число монеток, из которых можно найти фальшивую легкую монетку при условии, что дан еще мешок с p хорошими монетами.

Ну, так или иначе, с легкой задачей мы справились. И примерно видно, как в принципе с ней могла бы справиться программа. К сожалению, если ей, программе, *подсказать*, что наряду с числом $N(k)$ надо рассматривать $M(k)$ в том смысле, какой был оговорен выше — т. е. про бесконечный мешок хороших монет. Подсказка довольно неприятная...

А теперь вернемся к нашей исходной задаче. Сменим для задачи с легкой монеткой обозначения. Вместо $N(k) = 3^k$ и $M(k) = 3^k + 1$ будем писать $N_л(k)$ и $M_л(k)$. Здесь л — символ легкой задачи (известно, легче или тяжелее фальшивая монетка хороших), а не легкой монеты.

Через $N(k, 0)$ будем обозначать максимальное число монеток, из которых за k взвешиваний выделяется фальшивая при первоначально пустом мешке хороших.

$N(k, \infty)$ будет значить то же, но при условии, что мешок хороших монет уже первоначально бесконечен.

$M(k, 0)$ и $M(k, \infty)$ — минимальные числа монеток, из которых за k взвешиваний не находится фальшивая при пустом и бесконечном мешках хороших монет соответственно.

Теперь пусть все эти числа для всякого $k < n$ нам известны. Рассмотрим случай $k = n$ при $n \geqslant 2$.

Пусть наш мешок хороших монет: a) полон и b) пуст.

Прежде всего задумаемся, сколько монеток можно было при первом взвешивании отложить в сторону.

Очевидно, что, коль скоро равновесия не случится, число отложенных монеток несущественно: они все хорошие. Ну, а если наступило равновесие?

Тогда отложенные монетки содержат фальшивую. Следовательно, число отложенных было *не больше* чем $N(n-1, \infty)$. А вот могло ли оно быть ровно таким, пока неясно. В случае полного мешка — да. А в случае пустого? К этому пункту еще придется вернуться. Правда, наш мешок в случае равновесия пополнился хорошими монетками с чашек. Но ведь их *конечное* число.

Ну-с, а сколько монет могло быть на чашках?

Введем понятие СМВО — *суммарного молекулярного веса остатка*. Это — *число монеток в первозданной молекуле*, если она не пуста; и — *сумма чисел монеток в тяжелой и легкой молекулах*, если пуста первозданная.

Посмотрим, как меняется СМВО *после* взвешивания, т. е. по результатам хода В.

Пусть при данном взвешивании на левую чашку легло $Л_л$ легких атомов и $Т_л$ тяжелых. А на правую — $Л_п$ и $Т_п$ соответственно. И в отложенной группе осталось их $Л_о$ и $Т_о$.

ПЕРВЫЙ ИСХОД. Перетянула левая чашка. Теперь СМВО $= Т_л + Л_п$. А все остальные монетки мы сбросим по праву в мешок хороших.

ВТОРОЙ ИСХОД. Тяжелее правая чашка. Имеем СМВО $= Л_л + Т_п$.

ТРЕТИЙ ИСХОД. Равновесие. Теперь СМВО $= Л_о + Т_о$.

Отсюда сразу видно, какой ход В нам особенно опасен с точки зрения величины СМВО. Это — тот ход, при котором СМВО остается максимальным. Но СМВО до *хода* А был $Т_л + Т_п + Т_о + Л_л + Л_п + Л_о$. Значит, хоть одна из трех сумм — $(Т_л + Л_п)$, $(Л_л + Т_п)$ и $(Л_о + Т_о)$ составляла *не менее трети* СМВО до хода А. Теперь ясно, что *после каждого взвешивания* СМВО *понижается не более чем в три раза* [3]. А ведь нам нужно прийти к СМВО $\leqslant 1$. Только это и означает решение.

Отсюда сразу следует, что при первом взвешивании на чашки могло лечь в сумме *не более чем* 3^{k-1} монет, где k — общее число разрешенных взвешиваний.

Заметим в скобках, что этот пункт не кажется непроходимо трудным и для программы, если, правда, ей «подсказано» понятие СМВО.

Ну, а как быть с оценкой *снизу* максимального суммарного числа монет на чашках весов при первом взвешивании? Повезет нам, т. е. сгодится 3^{k-1} монет? Если да — задача решена. Пока же мы только и знаем, что больше, чем 3^{k-1}, их быть не может.

Если весы при первом взвешивании в равновесии — все ладно: все монеты на весах хорошие.

Пусть теперь перетянула левая чашка. Образовались легкая и тяжелая молекулы с суммарным числом атомов СМВО. Пусть СМВО $\leqslant 3^{k-1}$.

Разделим, если делится, тяжелую молекулу на 3 равные части: $М_л^т$, $М_п^т$ и $М_о^т$. Если получился остаток — отложим его пока в сторону. Так же поступим и с легкой молекулой. Разделим ее на равные части: $М_л^л$, $М_п^л$ и $М_о^л$. Опять-таки остаток отложим.

Теперь на чашки весов положим:

На левую — $М_л^т$, $М_л^л$ На правую — $М_п^т$, $М_п^л$.

Число монеток на чашках одинаково. Группы $М_о^т$ и $М_о^л$ оставим вне весов. Остается разобраться с остатком. Если в нем ровно 3 монетки, то две из них, взятые из *одной молекулы*, добавим по штуке на *разные* чашки. А последнюю — прибавим к группе вне весов. Если же оставалось всего 2 или 1 монетка, то число СМВО было $< 3^{k-1}$. В этом случае, коль скоро в остатке было 2 монетки и обе — из одной молекулы, положим их на *разные* чашки. Если же в остатке 2 монетки из *разных молекул*, добавим одну, скажем, из тяжелых, на левую чашку, а одну — в группу вне весов. А на правую чашку положим *дополнительно* хорошую монетку, благо взвешивание *не первое* и хорошие монетки у нас уже выделились. Если в остатке была всего одна монетка — оставим ее вне весов.

Ну, а теперь читатель без труда проверит, что при всех трех исходах второго взвешивания получится СМВО $\leqslant 3^{k-2}$.

---

[3] В невыгодном для нас случае.

И этим завершается решение задачи человеком. Вернее — почти завершается. Потому что мы действительно доказали, что СМВО после первого взвешивания не имеет права быть $> 3^{k-1}$. И что имеет право быть $\leqslant 3^{k-1}$. Но мы не заметили, что при первом взвешивании *нельзя* положить на чашки в сумме ровно 3^{k-1} монет. Просто и единственно потому, что 3^{k-1} — *число нечетное*. Значит, мы можем положить столько монеток на чашки лишь в том случае, если уравновесим чашку с меньшим числом монеток хорошими (причем хватит одной).

Поэтому если мешка хороших монет нету, то придется на первом шаге ограничиться $3^{k-1} - 1$ монетами на чашках (по половине на каждую). Что же до числа отложенных монеток, то их может быть столько, сколько разрешает индукция.

У нашей задачи — 4 варианта, так как:

1) можно требовать и не требовать, чтобы про фальшивую монетку было выяснено, чем именно она фальшива (тяжела или легка);
2) мешок хороших монет может быть заранее дан или не дан.

Присмотримся внимательнее к началу индукции. Пусть $k = 1$. Если монеток *одна*, то:

a) Мешка нет. Характер фальши знать надо. Задача неразрешима. $N(1) = 0$.
b) Мешка нет. Характера фальши не требуется. $N(1) \geqslant 1$.
c) Мешок есть. Характер фальши нужен. $N(1) \geqslant 1$.
d) Мешок есть. Характер фальши не требуется, см. b).

Теперь пусть монеток *две*.

a) Все по-прежнему.
b) Мешка нет. Характера фальши знать не нужно. Взвешивание нам не дает ничего. $N(1) < 2$. Окончательно имеем: $N(1) = 1$.
c) Мешок есть. Нужно знать характер фальши. Две монетки не проходят. Получаем окончательно $N(1) = 1$.
d) Мешок есть. Характер фальши нас не интересует. Теперь взвешивание одной из подозрительных с одной хорошей монетой решает дело $N(1) \geqslant 2$

Монеток *три*.

d) Мешок есть, а характер фальши безразличен. Остаемся при $N(1) = 2$. Действительно, если при взвешивании на каждой чашке окажется по подозрительной монете, то любая из них может обеспечить своей фальшивостью исход взвешивания — перевес. А если одна из чашек окажется свободной от подозрительных монет, то 3 монеты дадут после взвешивания хоть одну неодноатомную молекулу.

Посмотрим теперь первый шаг индукции — случай двух взвешиваний.

a) Отложить можно только одну монетку, так как после первого взвешивания при равновесии чашек мы придем к случаю c). На чашки мы можем положить $3^{k-1} - 1 = 2$ монетки, по одной на каждую чашку. Итого имеем $N(2) = 3$.

b) Теперь отложить можно 2 монетки, так как случай равновесия приведет нас на втором взвешивании к d). А на чашках попрежнему будет по 1 монете. Итак, $N(2) = 4$.

c) Теперь отложить можно опять-таки только 1 монетку, ибо случай равновесия дает c) при втором взвешивании. Но на чашки при первом взвешивании можно положить $3^{k-1} = 3$ подозрительные монетки: две — на одну чашку, а одну — на другую, добавив к ней хорощую из мешка.

Второе взвешивание получится таким:

1) Тяжела чашка с двумя подозрительными монетами. По индуктивному правилу кладем их по одной на чашку, а монету с другой чашки — в сторону. Перевес означает фальшивость перевесившей тяжелой монеты. Равновесие — фальшивость отложенной легкой.

2) Тяжела чашка с одной подозрительной монетой. Взвешиваем легкие — по одной на чашку. Перевес — фальшива более легкая. Равновесие — отложена тяжелая фальшивая.

3) Равновесие. Теперь фальшива отложенная монетка — одноатомная первозданная молекула. Но у нас уже есть эталонные хорошие. Второе взвешивание расскажет нам, чем была нехороша отложенная монета.

d) Ну, а теперь мы можем, во-первых, отложить 2 монетки, как в случае b), а во-вторых, на чашки сунуть 3 подозрительные монетки (2 : 1) плюс одна добавочная из мешка.

Переход от $k = 1$ к $k = 2$ носил иллюстративный характер.

Но один подводный камень мы миновали по-иезуитски. Заметьте, что мы все время говорим *о максимальном числе* монеток, из которого... и т. д. А вовсе не о таком числе, что из *любого* не большего набора монет можно... и т. д. Все это оттого, что из 2 монет *никаким* числом взвешиваний фальшивой не найдешь, если мешок пуст. Трудность, с которой при программировании придется очень и очень считаться.

§ 7. Вернемся к программе

Мы остановились в § 5 на том, что решили посмотреть, как решает задачу человек. Конечно, может статься, что программа найдет и другой путь. Но хотелось бы, чтобы она по крайней мере *имела возможность* найти путь человека. Потому что — а вдруг другого пути нет?

Перед этим мы отметили одно несколько грустное обстоятельство: решая неразрешимую задачу, программа может работать бесконечно долго, если неверная гипотеза имеет буквенный характер.

А как с этим справляется человек? Да очень просто — наряду с попытками *доказать* гипотезу он всегда пробует еще и *опровергнуть* ее.

Что ж, пусть и программа научится так делать. Т. е. включим в нашу систему наряду с перебором возможностей для *доказательства* некоторой гипотезы еще и перебор возможностей для *построения противоречащего примера*. Причем опять введем ранги символов, как и при доказательстве. А чтобы собрать все в одну последовательность, поступим ровно так, как когда доказывают, скажем, счетность суммы двух счетных множеств: начнем с попыток доказательства, использующего символы ранга 1, затем попробуем построить противоречащий пример из символов ранга 1, потом — доказательство с участием символов ранга 2, противоречащий пример с символами ранга 2 и т. д. [4)]

С несколько грустным обстоятельством мы вроде бы справились — правда, слегка на словах о деле (программе), а не на самом деле — в реальной программе. Ну, и это кое-что.

А вот будет ли нам так же легко с другим, как мы уже говорили, гораздо более грустным обстоятельством?

Ведь как нам удалось решить задачу с точки зрения человека? Мы для этого ввели некоторую функцию СМВО и фактически *провели двойную индукцию*: один раз — по числу взвешиваний k, а другой раз — по величине СМВО.

Так вот, более грустное обстоятельство состоит в том, как бедная программа должна догадаться, что а) индукция двойная и что b) ее, индукцию, нужно вести не только по числу взвешиваний k, что естественно, а еще и по величине СМВО, которую и взять-то неизвестно откуда! Что ж теперь, по всем возможным величинам вести индукцию? Так, что ли?

[4)] Фактически, вероятно, можно и полезно ввести некоторое неравноправие: скажем, на два-три ранга в пользу доказательства давать лишь один ранг на противоречащий пример и т. п.

§ 8. Да, именно так

И, более того, при построении противоречащих примеров — тоже. И на всех этажах — тоже.

Идея о таком неограниченном применении индукции в машинных доказательствах возникла в беседах на эти темы между М. Я. Шилтере, А. Я. Подрабиновичем и автором. Может быть, где-нибудь она высказывалась и раньше. Тогда мы не претендуем на приоритет.

Так или иначе, теперь вырисовываются контуры программы. Ну, конечно, в эдаком маниловском духе. Как ее мыслит, что ли, математик, которому ее не писать.

Так вот, став да минутку на такую «чистую» точку зрения, получим вот какой прожект.

С самого начала решим, чтó у нас будет информация о молекулах. Это вообще для любых задач о взвешивании по существу. И информация про мешок хороших монет. Тоже по существу.

А вот то, что молекул бывает всего 3 — легкая, тяжелая и первозданная — уже особенность избранной задачи. Так что есть две возможности:

a) Не фиксировать в программе число и характер молекул — пусть сама вырабатывает их хоть миллион. А программа должна быть так здорово устроена, чтобы в *данной задаче* по самим ее условиям не допустить химических излишеств.

По общности — привлекательно. По трудности — не слишком (если делать самому).

b) Фиксировать покуда то, что молекулы бывают легкие и тяжелые. Облегчить себе жизнь. Но зато ограничить программу в возможностях — она решит теперь и без того решенную человеком задачу. Что слегка обидно.

Пусть мы сперва пойдем по второму, более легкому пути. Посмотрим, с какими неприятностями нам придется иметь дело.

Отвлечемся — мы ведь сейчас стоим на позициях чистого математика! — от кардинальнейшего вопроса, как именно будет сделано все дальнейшее, т. е. в каком виде будет задаваться информация и как она, следовательно, должна программой перерабатываться.

Останемся при несколько гуманитарном «математически-точном» описании алгоритма. Это еще очень далеко от «программно-точного» описания. Последнее называют описанием просто.

Итак — гуманитарное описание.

Для $k = 1$ непосредственным перебором возможностей мы можем решить задачи a)–d) со с. 187. Фактически, вероятно, их также хорошо

решить для $k = 2$ и, скажем, $k = 3$. Это понадобится при *опровержении* неверных гипотез. А быть может, решение задач a)–d) для конкретных малых k должно происходить тоже «вплетенным» в общий перебор способом, — по мере того как усложняются гипотезы, увеличивается и материал для «примерки» каждой из них, — чтобы быстро обрезать ветки на дереве игры. Теперь по ходу перебора вместе с ростом ранга рассматриваемых гипотез будет расти и величина n — число разрешенных взвешиваний, для которых мы уже знаем ответы на задачи a)–d). Может статься, что это — довольно общий прием.

Отсюда первый вывод — необходимость иметь программу «старопереборного типа» — для решения таких легких задач ($k = 1, 2, 3, \ldots$). Придется считать, что это очень простое дело. Вероятно, при реальном написании главной программы так оно и будет выглядеть — наши оценки трудности всегда относительны.

Ну, а как же дальше? Дальше поначалу мы должны сделать гипотезу о зависимости $N(k)$ от k. Между прочим, для задач a)–d) величины $N(k)$ записываются нехитрыми формулами:

$$\text{a)}\ N_{\text{a}}(k) = \frac{3^k - 1}{2} - 1;$$

$$\text{b)}\ N_{\text{b}}(k) = \frac{3^k - 1}{2};$$

$$\text{c)}\ N_{\text{c}}(k) = \frac{3^k - 1}{2};$$

$$\text{d)}\ N_{\text{d}}(k) = \frac{3^k - 1}{2} + 1.$$

Первое, что приходит в голову, — включить в разрешенный набор *операций для конструирования символов разных рангов* действия арифметики плюс действие exp. Тогда мы, и правда, на некотором, даже не очень далеком, шагу выдвинем верные гипотезы про каждый из ответов.

И тем не менее мне очень сильно кажется, что такой путь ложен. Потому что я совершенно не представляю себе, как может идти доказательство сформулированной в таком виде гипотезы.

Ведь трудно надеяться на *неиндуктивное* доказательство. А в формулах в написанном виде *отсутствует явный переход* от $k = n - 1$ к $k = n$. Правда, индуктивно, вероятно, придется задавать определение действия exp. То есть, точнее, свойство (аксиому, если exp — изначальная операция):

$$a^{n+1} = a^n + a^n + \ldots + a^n \text{ (индукция по } n).$$

А такой перенос трудности вглубь — в определения — должен, по-моему, привести к нарастанию трудностей.

Вероятней, формулы должны выглядеть как-нибудь так:

$$\text{a)}\ N_a(k+1) = 3N_a(k) + 3; \quad N_a(1) = 0$$
$$\text{b)}\ N_b(k+1) = 3N_b(k) + 1; \quad N_b(1) = 1$$
$$\text{c)}\ N_c(k+1) = 3N_c(k) + 1; \quad N_c(1) = 1$$
$$\text{d)}\ N_d(k+1) = 3N_d(k) - 1; \quad N_d(1) = 2$$

Но и в этом случае, хотя мы уже исключили из разрешенных операций столь неприятную экспоненту, осталась все еще очень и очень сомнительная операция — умножение. Мне представляется и это совершенно негодным для наших целей. Задача о взвешивании — «линейная» задача. И действия для формирования формул-гипотез на всяком шагу должны быть линейными. Тогда хоть как-то видно, как они — если повезет — смогут быть доказаны. В переписанном виде будет так:

$$\text{a)}\ N_a(k) = N_a(k-1) + N_a(k-1) + N_a(k-1) + 3; \quad N_a(1) = 0$$
$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots$$
$$\text{d)}\ N_d(k) = N_d(k-1) + N_d(k-1) + N_d(k-1) - 1; \quad N_d(1) = 2$$

И вот это — уже форма приемлемая.

Ну, а как же пойдет индуктивное доказательство?

Довольно быстро мы додумаемся, например, в задаче а) до такого первого взвешивания (ход А):

$N_a(k-1) + 1 : N_a(k-1) + 1$ при $N_a(k-1) + 1$ откинутых монетках.

Теперь наш противник-природа (игрок В) имеет 3 возможности ответа: перевес левой чашки, перевес правой и равновесие.

Результат-равновесие приводит нас к задаче с) для числа взвешиваний $k-1$. Если индукция идет параллельно для всех четырех задач (!), то это — уже конец хода В — «равновесие».

Теперь пусть получилось неравновесие. Мы пришли к сформированию легкой и тяжелой молекул по $N_a(k-1) + 1$ атомов в каждой. Как быть теперь?

Про необходимость параллельного решения задач а)–d) мы еще ведь тоже не решили — программа должна, что ли, додуматься до задачи с), если ей *задана* задача а) (соответственно от b) к d)), или мы ей это подскажем?

Ну, хорошо, а как быть с постановкой задачи для ситуации: уже есть тяжелая и легкая молекулы и еще мешок хороших, причем все три по одинаковому числу элементов; за сколько действий выделяется хорошая? Если опять подсказывать, то ведь так и вообще мы скатимся к программе *проверки точности нашего собственного человеческого доказательства.*

Значит, вроде бы сама программа должна «прийти к идее» проверить, как обстоят дела для новополучившейся задачи. Но заметьте, что если мы введем даже этот еще один этаж (и довольно-таки тяжелый этаж — ни мало, ни много, выработка дочерних задач!), так даже при таком разрешении мы имеем вовсе не однозначно поставленную задачу. Пусть наша *информационная система позволяет* сформулировать эту дочернюю задачу: даны легкая и тяжелая молекулы и мешок хороших. Ведь для нее нельзя начать «сверху» — от случая $k-1$ взвешиваний, к которому мы пришли в *нашем основном переборе возможностей первого взвешивания, подчиненном в свою очередь перебору гипотез о связи* $N(k)$ с $N(k-1)$. Так что мы имеем сейчас «перебор 3-го ранга» — перебор подчиненных задач. Ведь и в самом деле — на что́ смотреть программе в получившейся задаче?! Мы пришли к ситуации: равноатомные молекулы плюс такой же мешок хороших. И притом число атомов v в каждой молекуле (и мешке) есть $N(k-1)+1$.

По числу s взвешиваний программа и начнет пробовать решать новую задачу — выделить фальшивую монетку из v-атомной тяжелой и v-атомной легкой молекул при мешке с v хорошими монетами.

При $s=1$ она, конечно, быстро убедится, что $v(1)=1$: взвесив легкую монету с хорошей, мы получаем результат, а если положить $v(1)=2$, — «старый перебор» покажет несостоятельность такой гипотезы.

Но ведь так мы пойдем по ложному следу! Потому что уже при $s=2$ получается, что допустимо $v=4$. Достаточно для $s=2$ при первом взвешивании положить на одну чашку, скажем, 2 «легкие» и одну «тяжелую» монетки; на другую — одну «легкую», одну «тяжелую» и одну хорошую. А 2 тяжелые и одну легкую — оставить вне весов.

Но теперь при втором взвешивании мы опять пришли к новой задаче — в легкой и тяжелой молекулах *разное* число атомов. Итак — снова новая задача!

Дело ведь в действительности состоит в том, что важны не числа монеток в легкой и тяжелой молекулах по отдельности, а их сумма. Но бедная программа этого не знает. Вот мы и пришли еще к одному перебору — перебору возможных постановок задачи...

§ 9. Удастся ли все-таки создать программу?

Надеюсь, что да. Хотя в предшествующих параграфах я и постарался представить дело в его истинном свете. Что говорить, легким это дело не назовешь! Если, конечно, его предстоит делать тебе, а не подчиненным программистам (см. § 5 последней беседы). Заметим, что сама «этажность» перебора не в смысле глубины ветвей, а в смысле: перебор индуктивных гипотез по k — перебор символов по рангам при заданной

гипотезе — перебор *постановок* дочерних задач — перебор индуктивных гипотез дочерних задач по k дочернему — перебор символов по рангам для очередной гипотезы про дочернюю задачу — перебор постановок «внучатных задач»... и т. д. — что-то не слишком фиксирована.

То есть придется нормальную «старопереборную» схему зациклить на себя уже не по глубине ветвей, а «крупноблочно» — по тому, что вообще-то делается на данном этаже.

И все же трудности не кажутся непроходимыми — когда-то так же примерно выглядела милая домашняя «старопереборная» задача. А я — оптимист. Только вот эту задачу делать не умею. Хотя и думаю про нее уже с год (это к моменту написания «Бесед», т. е. к ноябрю 1964 года).

Зачем же я толкую о том, чего не умею делать? Ну, не хотите — не читайте. А можете рассматривать, товарищи собеседники, эту беседу как вызов на соревнование. Это — дело вкуса.

§ 10. Зачем все это нужно? Нужно ли это вообще? Не лучше ли заняться чем-нибудь другим?

Буду отвечать в неприятном, категорическом тоне. Да, нужно. Это еще, конечно, не настоящая «думательная» программа. Но задача «в буквах» — новый тип задач, которые пока программы не решают. Пробиться в этом месте — значит отвоевать новый кусок территории у неизвестного. Вот зачем это нужно.

Нужно ли все это вообще? Да, нужно. В общем смысле про это говорилось в одиннадцатой беседе. А от ожидания чуда чудо не произойдет. Вероятно, сперва, еще задолго до «самодумающих» программ, появятся такие «программы-помощники думания». И предлагаемая задача может оказаться одним из первых случаев возникновения «помощника думанию». А что нам придется ей так много подсказывать — что ж, дети — опора старости, а сколько их приходится учить и растить! А пока что наш будущий «помощник» еще даже и не новорожденный — программа еще когда-то будет. Но будет!

Ну, и ответ на последний вопрос. Конечно, для *меня* он отрицателен. Иначе бы я, понятно, этим другим, чем лучше заняться, и занялся бы.

Другим действительно занимаются. Например, в Ленинграде. Там делают очень интересную работу — программу для получения доказательств теорем логики.

И, главное, — это программа действует! Пусть она доказывает пока совсем простые теоремы — кто сам такое делал, знает,

Что строчки с кровью — убивают.
Нахлынут горлом и убьют!

А строчки подобного типа программ и правда с кровью... Ну, а чья дорога окажется плодотворней — покажет жизнь.

Мне казалось: важнее добраться поскорее до рассуждений в сравнительно трудных задачах при некотором пренебрежении к тому, откуда мы взяли исходные положения (подсказка). В надежде прорваться с помощью программы к решению хотя бы одной задачи, не решенной программистом заранее. Я думаю, что тут может хлынуть золотой дождь удачи. Когда, конечно, удастся прорваться.

А ленинградцы думают, что наоборот — стоит пренебречь хлесткими результатами, а строить здание начиная с фундамента. Не поднимается у меня язык ругать их, ибо «тяжек их труд и горек их хлеб...».

Во всяком случае не сладок!

Беседа четырнадцатая

Вычислительные задачи с точки зрения невычислительных

А ищи всех паче
Разума в задаче.

Магницкий

§ 1. Издержки автоматизации

Перед эрой ЭВМ люди научились очень недурно считать на электрических арифмометрах. Конечно, скорость «Мерседеса», даже в руках наилучшей лаборантки, не сравнишь со скоростями ЭВМ. А вот число действий, нужное для решения той или иной реальной задачи, постепенно научались делать все меньшим и меньшим. Быстро развивались *вычислительные алгоритмы*. Но это был период первоначального накопления вычислительной математики.

Мастерство математика-вычислителя было похоже на мастерство виртуозов-умельцев. Ну, вроде лесковского Левши. Делились друг с другом рецептами мастерства — совершенно профессиональными (кустарными): в таком-то случае делай так-то. Ну, скажем:

«...решая задачу о критразмерах реактора, не ставь точек на край, а ставь в центр клетки...».

Или:

«...будешь считать задачу теплопроводности — пиши балансные соотношения. Из них получится конечно-разностная схема куда экономнее, чем из уравнения в частных производных».

Большая часть таких «рецептов мастерства» носила интуитивный характер. Действовали согласно эпиграфу. В книгах вы ничего этого

не найдете. Кроме, пожалуй, книги А. Н. Крылова [10]. Но Крылов и был белой вороной — он сам считал и сам написал книгу. А обычно вместо И приходится ставить ИЛИ. Отчего ценность соответствующих руководств почему-то понижается. Не помогает здесь разделение труда!

Так вот, когда стали считать на ЭВМ, весь этот начинавший было накапливаться капитал пришлось обесценить.

Тому было 4 причины:

У ПЕРВЫХ ЭВМ БЫЛА МАЛЕНЬКАЯ ПАМЯТЬ. Приходилось вжимать программу в объем памяти. И черт с ними, с разными ускорениями. Потому что не уложишься — не пойдет задача никак. И будешь себе считать на руках.

НА ЭВМ ПОНАЧАЛУ СЧИТАЛИ ЛЕГКИЕ ЗАДАЧИ. Просто по инерции. Понадобилось некоторое время, чтобы у заказчиков (физиков, техников, даже самих математиков) разгорелся аппетит. А у программистов — смелость. Сперва же решали примерно те задачи, что шли на руках. И только беспрерывно дивились на скорость ЭВМ. Дивиться, и правда, было на что. А легкие задачи было все равно как решать. Ну, полсекунды, ну пять минут. Какая разница? И правда никакой.

ЭВМ ТРУДНО ПОРУЧИТЬ СЛОЖНЫЙ АЛГОРИТМ. Так как его нужно задавать программой. А это иной раз требует больших усилий человека. На руках — что было делать. Не считать же 2 года вместо 2 месяцев! Да и лаборантки взбунтуются. А на ЭВМ — два дня вместо двух часов? Отчего же и нет. ЭВМ-то не взбунтуется... Да и проигрыш, глядишь, окупится быстрой отладкой.

МЫ НЕ УМЕЛИ СФОРМУЛИРОВАТЬ СВОИХ ПРИЕМОВ. И это, может быть, самое существенное. В ручном счете руководитель следит (по крайней мере, может следить) за ходом счета на любом этапе.

Все перед глазами. И нет нужды выговаривать всевозможные ответвления счета заранее. Посмотрели — кое-что сменили в расчетной схеме. Потом кое-где удвоили число точек. А то и удесятерили. И притом не всюду, а где нужно. При случае сделали замену переменной. А то и вовсе выбросили целый кусок вычислений и его (и только его) заменили другим. И все это по ходу дела. Конечно, никто заранее всех случаев не предусматривал и не оговаривал. И не очень заранее заботился, как в точности будет действовать. А действовали по обстоятельствам. И делали это сознательно. Но боюсь, очень боюсь, что не без помощи подсознания ...что-то уж больно медленно выговариваются все нужные слова, когда все-таки захотелось уместить наши приемы в программу. Лаборантками-то мы командовали куда как быстрее. Вот откуда произошли издержки автоматизации.

§ 2. Почему вдруг понадобилось экономить?

Во-первых, не вдруг, а постепенно. Во-вторых, у современных ЭВМ память больше не маленькая. И скоро станет совсем большой. Так что все времянки, рассчитанные на компенсацию любой ценой (по времени) недостатка памяти, отомрут. В частности, отомрут библиотечные системы типа КИС. По этой причине мы их ни с какой, даже критической, стороны не рассматривали.

Итак, отпадают ограничения на память. А с другой стороны, за десяток лет заказчики вошли во вкус. И появились задачи, требующие машинного времени уже не секунды, а часы и дни. Иногда — недели и месяцы, т. е. столько, сколько раньше шел ручной счет.

Ну, а за это время и мы, программисты, подучились маленько. Правда, не слишком уж сильно — могли бы и побольше. Но все же подучились. И накопили некоторый багаж в виде стандартных и нестандартных программ. И, пожалуй, некоторые интуитивные сведения о том, как это делается (подсознание?).

Первый и последний абзацы говорят, почему *можно* приступить к наведению экономии времени в сложных счетных программах. А средний — почему это *понадобилось*. А все вместе — что такая задача *стала на повестку дня*.

§ 3. Как выглядела раньше программа СОВЕРШЕННО СТАНДАРТНЫЙ ИНТЕГРАЛ и как выглядит она теперь

При работе с заданной относительной (абсолютной) точностью программа поступала так.

Отрезок интегрирования дробился последовательно на $1, 2, 4 \ldots$ равных отрезочка. По каждому отрезочку бралась гауссова квадратура пятого [1] порядка и результаты складывались. Получались ответы S_1, S_2, S_4 и т. д. Теперь, когда, скажем, S_8 и S_{16} различаются меньше, чем предельно допустимо по заданию, величина S_{16} и принималась за ответ.

Казалось бы, все очень складно. Но выяснились две неприятности, пожиравшие при случае уйму времени.

ИНТЕГРИРОВАНИЕ ВБЛИЗИ ОСОБЕННОСТИ. Надо бы в этом районе мелко дробить отрезки. Даже если особенность не у функции, а у ее производной. Но по описанному алгоритму одновременно

[1] Просто факт; можно взять вместо 5, например, 4 или 7.

раздробится столь же мелко и весь интервал интегрирования. Станет это очень недешево — потеря десятки в скорости в этом случае не редкость.

ДВОЙНОЙ ИНТЕГРАЛ ВИДА $\int_0^{30} dx\, e^{-x} \int_0^{1} \sin^2(x^2 y)\, dy$. Как вы полагаете, нужно в этой задаче при вычислении $\int_0^{1} \sin^2(20^2 \cdot y)\, dy$ заботиться о той же относительной или даже абсолютной точности, что и при вычислении $\int_0^{1} \sin^2(0{,}5^2 y)\, dy$?

Конечно, собственно программа интегрирования здесь как бы и ни при чем. Ну, зададим для больших x меньшие точности, чем для малых — и дело с концом. Но ведь как это осуществить фактически? Ведь точность-то придется во внутреннем интеграле поставить в зависимость от его вклада во внешний. А про это придется догадываться во внешнем интеграле. Да еще не прямо по значению множителя e^{-x} при данном x, а по вкладу всего произведения $e^{-x} \int_0^{1} \sin^2(x^2 y)\, dy$ во внешний интеграл. А то ведь иной раз может получиться, что падение множителя e^{-x} компенсируется ростом множителя-интеграла!

Неприятности, подобные этой, покуда в ИНТЕГРАЛЕ не ликвидированы.

А с целью избавиться от первой поступили так.

Прежде всего деталь чисто техническая. Для контроля точности гауссовой квадратуры мы применяли разбиение отрезка пополам. Итого — 10 добавочных точек. Можно обойтись 6 гауссовыми точками; будет всего одиннадцать — пять основных плюс шесть контрольных. Но тогда уж хочется распорядиться этими добавочными 6 точками как можно лучше. Для этого служат уточняющие квадратуры имени меня. Кто хочет подробно об этом — см. [11]. А для понимания дела можно считать 6 добавленных точек узлами гауссовой квадратуры 6-го порядка, а никакими не узлами имени меня. Без разницы (кроме времени).

Ну-с, так, программа начинает работу с того, что считает интеграл на всем отрезке по 5 и 11 точкам. Сравнивает результаты по заданному эталону точности. Если хорошо — более точная квадратура идет в ответ. Если плохо — дробим отрезок пополам и занимаемся левой *половиной*. Если опять плохо, от рассматриваемого отрезка берется левая *четверть*. Затем, коли плохо, левая *восьмая* от последнего рассмотренного отрезка и т. д. Ускорение дробления связано с фактом «внематематическим» — практически часто неприятности бывают на концах интервала интегрирования.

Наконец, стало хорошо. Проявим оптимизм — попробуем, двигаясь вправо, удвоить отрезок интегрирования. Если удвоение прошло — удвоим и оптимизм и увеличим в следующий раз отрезок уже не вдвое, а вчетверо и т. д.

Hv, конечно, следя за тем, чтобы не перехватить через правый конец!

Как только стало снова плохо — вернемся к пессимистической тактике: вместо удвоения, учетверения и т. д. будем брать половинки, четвертушки и т. п. Так здесь сделано фактически — большого смысла в серединке интервала интегрирования такое не имеет. Скорей даже замедляет дело; было бы лучше ограничить пессимизм половиной — никак ведь не угадаешь, где именно плохо. А, наоборот, не сделано, а может быть, имело бы смысл — при движении направо в случае «плохо» отступать от правого края поначалу не сильно. Или даже начать движение справа налево, чтобы правый конец был обслужен так же, как перед тем левый. Может быть, правда, это мелочи.

А пессимистическая и оптимистическая тактики сменяют друг друга по ходу результатов. Только если после ряда «плохо» вышло, наконец, «хорошо», то оптимизм включается не сразу, а через раз. А один раз мы идем по отрезку той же длины (кроме случая левого края). Такая программа носит почти игровой характер. В нее введена *тактика*, да еще *переменная* (правила смены оптимизма и пессимизма, темп нарастания оптимизма). Все это пришло, конечно, из невычислительных задач.

Остается сказать два слова о критериях ХОРОШО—ПЛОХО.

Пусть нам задана относительная точность ε.

Тогда прежде всего по первому же интегрированию оценим значение интеграла (по 11 точкам). Пусть это будет S_0. Примем за нужную абсолютную точность величину $\varepsilon \cdot |S_0|$. Пусть еще L — длина всего интервала интегрирования.

Теперь для оценки результатов интегрирования M_5 и M_{11} по 5 и 11 точкам на маленьком отрезке длины l рассмотрим величины:

$$1)\ \left|\frac{M_{11}-M_5}{M_{11}}\right|;$$

$$2)\ \left|\frac{M_{11}-M_5}{l}\right| : \left|\frac{S_0}{L}\right| = \left|\frac{M_{11}-M_5}{S_0}\right| : \left|\frac{l}{L}\right|.$$

Если хоть одна из них $< \varepsilon$, оценим положение на этом отрезке как ХОРОШО. Первая величина — просто относительная точность. Со второй хитрее. Пусть мы интегрируем e^{-x} на $[0; 6]$. На отрезке, скажем, $[5; 6]$ отношение $\left|\frac{M_{11}-M_5}{M_{11}}\right|$ огромно. А между тем *вклад*

отрезка [5; 6] в весь интеграл столь ничтожен, что заботиться об увеличении точности незачем. Для борьбы с излишествами типа дробления [5; 6] в нашем примере и введена вторая величина. Она трактует об относительной ошибке разности на маленьком отрезке, но не по отношению к величине интеграла на нем, а по отношению к «доле» всего интеграла, пропорциональной длине отрезка. Теперь места, где подынтегральная функция мала, будут автоматически проходиться с более низкой точностью. Иначе это требование можно сформулировать так:

$$|M_{11} - M_5| < \varepsilon \cdot |S_0| \cdot \frac{l}{L}.$$

Если на каждом отрезке это неравенство выполнено, то

$$\sum |M_{11} - M_5| < \varepsilon \cdot |S_0|,$$

где сумма распространена на все отрезочки.

И — последнее. Пусть мы интегрируем, скажем, $\sqrt{x}$ на [0; b]. Легко видеть, что изменение длины отрезка [0; b] не изменяет относительной точности, получающейся при пользовании как гауссовой, так и уточняющей квадратурами. Это приведет к тому, что отрезок около нуля будет дробиться все мельче и мельче, пока не вступит в силу второе ограничение. Но мы можем позволить себе оборвать этот процесс не только по критерию

$$|M_{11} - M_5| < \varepsilon \cdot |S_0| \cdot \frac{l}{L},$$

а и гораздо раньше. Если такая неприятность бывает только в одном месте, можно, скажем, удовлетворяться тем, что на данном отрезочке имеет место

$$|M_{11} - M_5| < \frac{1}{10} \cdot \varepsilon \cdot |S_0|.$$

А если думать, что такое может выйти несколько раз, то к критериям ХОРОШО на данном отрезочке присоединим еще

$$|M_{11} - M_5| < \frac{1}{64} \cdot \varepsilon \cdot |S_0|,$$

и это будет и правда хорошо.

И да не смутит вас, читатель, нематематичность этого утверждения. Ибо, например, для многочлена 22-й степени

$$P(x) = [L_5(x) \cdot L_6(x)]^2,$$

где $L_n(x)$ — полином Лежандра порядка n, отнюдь не верно, что

$$\int_{-1}^{1} P(x)\,dx = 0,$$

а именно такой ответ мы получим, интегрируя $P(x)$ на $[-1; 1]$ как по Γ_5 — гауссовой квадратуре 5-го порядка, так и при контроле с помощью Γ_6 — гауссовой квадратуры 6-го порядка.

И тем не менее счет по Γ_5 с контролем по Γ_6 — это хорошо. А многочлен $P(x)$ следует считать исключением. В вычислительных делах это бывает.

И еще одно замечание, чтобы покончить с нынешней совершенно стандартной программой интегрирования — ее в Б-61 называют ИНТЕГРАЛ ОСОБЕННЫЙ; а сделан ИНТЕГРАЛ ОСОБЕННЫЙ М. З. Розенфельд (в обсуждении участвовали также В. Арлазаров и я). Так вот, пусть программа дошла до конца счета. Теперь пусть S — полученный ответ. Если $|S| \geqslant |S_0|$, то мы на каждом отрезочке завысили требования к абсолютной точности по сравнению с заданными, т. е. все, и правда, хорошо. Но может быть, скажем, $|S| = 0{,}01 \cdot |S_0|$. Тогда, может статься, заданная относительная точность не выдержана — мы рассчитывали по прикидочному просчету, что абсолютная величина интеграла $\approx S_0$ и, соответственно, абсолютной ошибке разрешили достигать $\varepsilon \cdot |S_0|$, а теперь выяснилось, что нельзя было уходить за $0{,}01 \cdot \varepsilon \times |S_0|$. В таком случае надо повторить расчеты с самого начала, заменив для допустимой абсолютной ошибки $\varepsilon \cdot |S_0|$ на $\varepsilon \cdot |S|$. В программе это предусмотрено. Фактически такая ситуация случается не часто, но все же пару раз случилась.

§ 4. Как программа ИНТЕГРАЛ будет выглядеть завтра

Теперь этой программой занимается Реет Александровна Пукк. Ее программа должна по идее сэкономить раза в 2–2,5 время против программы М. З. Розенфельд. Ну, понятно, «в среднем» — почти как средняя температура больных в больнице. А время сравнивается, естественно, *по числу обращений к подынтегральной функции*, потребовавшихся при взятии одного и того же интеграла при той же заданной точности в каждой из программ.

На чем Реет экономит время?

В общем-то, по мелочам. Но нельзя забывать, что, кроме однократных интегралов, существуют еще и 2-, 3- и 4-кратные. А тогда двойка экономии обернется четверкой, восьмеркой и шестнадцатью соответственно. И притом, чем труднее задача — больше кратность, — тем больше экономия. Двойка — и правда, не стоит мараться. А вот 16?

Итак, МЕЛОЧЬ ПЕРВАЯ. Пусть на некотором отрезочке сосчитан интеграл по Γ_5. У нас нет пока критериев для оценки точности. Казалось бы, пока не проведена $У_{11}$ — уточняющая квадратура по 11 точкам,

нельзя решить, придется ли нам дробить отрезок. Однако можно сделать и по-другому. Возьмем нарочито *грубую* квадратуру, использующую значения подынтегральной функции в *точках, где $f(x)$ уже сосчитана*. Конкретно Реет берет такую квадратуру: на левой половине она считает по трем левым узлам с весами, дающими верный результат для квадратных трехчленов. И так же — на правой половине. Полученное значение (2×3) сравнивается с результатами по Γ_5.

Если ошибка меньше заданной для ХОРОШО — можно не переходить к квадратуре $У_{11}$, а считать результатом Γ_5. Дальше, если разница между (2×3) и Γ_5 невелика, но больше заданной, перейдем к $У_{11}$. Это значит СРЕДНЕ. Наконец, если расхождение (2×3) и Γ_5 велико, то мало надежд на $У_{11}$. Это значит ПЛОХО. Теперь можно сразу дробить отрезок, не тратя времени на счет $f(x)$ в 6 дополнительных точках. Кое-что экономится.

МЕЛОЧЬ ВТОРАЯ. На отрезке уже сосчитано $У_{11}$ и одиннадцать значений $f(x)$ сохранены. Пусть $У_{11}$ не сильно, но больше, чем было задумано, разошлась с Γ_5. Обязательно ли дробиться? Есть еще соломинка. Чтобы за нее схватиться, Реет вводит еще одну квадратуру (2×6). Она сделана по аналогии с (2×3), только веса на каждой из половинок должны обеспечить верное интегрирование многочленов уже не второй, а *пятой степени*. Тоже в хозяйстве сгодится.

А ТРЕТЬЕ УЖЕ И НЕ СОВСЕМ МЕЛОЧЬ. Пусть все-таки и Γ_5 с $У_{11}$ и $У_{11}$ с (2×6) разошлись порядочно. Но ведь не вовсе же сильно — иначе бы мы (кроме исключительных случаев) и вообще бы не считали $У_{11}$ — дробились бы прямо.

Как поступил бы в старые добрые времена ручного счета руководитель работы? Ну, для $У_{11}$ может и плюнул бы и раздробился. А вот если бы интегрирование шло по Симпсону? Ну, тогда, если он не вовсе дурак, — вставил бы точки. А если вовсе дурак, — раздробившись вдвое, велел бы считать заново. А уж лаборантки, независимо от умственных качеств, сами заметили бы, что в четных точках они просто повторяют расчет. И использовали бы эти старые, уже сосчитанные значения. Это — к вопросу о воспитании интеллекта на базе личного труда.

Вот ровно так и заставила Реет поступить программу. К 11 узлам $У_{11}$ она добавила еще 12 «вставных». В ИТЭФ, где Р. А. Пукк была аспиранткой, существовали подходящие программы для вычисления нужных ей узлов и весов. И получилась квадратура дальнейшего уточнения $ДУ_{23}$. Теперь и программа, подобно упомянутой лаборантке, не выбрасывала незадавшиеся 11 точек — узлов $У_{11}$, а «сдваивала» точки. Дальше шло по шаблону — сравнили результаты $У_{11}$ и $ДУ_{23}$. И опять за соломинку приняли квадратуру (2×12). Какова она — контрольный вопрос к собеседнику.

О ГРАНИЦАХ ОПТИМИЗМА. Это — четвертая мелочь. На некотором отрезке получилось ХОРОШО. Теперь мы пойдем вправо. Оптимизм учит попробовать увеличить шаг. Но во сколько раз?

Главное психологическое состояние в любой азартной игре таково:

ТРУСОСТЬ БОРЕТСЯ С ЖАДНОСТЬЮ

А ведь программа-то у Реет игровая!

Трусость повелевает: увеличивайся полегоньку, а то потратишь зря силы — придется сокращаться обратно.

А жадность подталкивает на игру ва-банк. Вдруг придет карта (в виде гладкого участка подынтегральной функции).

Но теперь у Реет появились научно обоснованные предпосылки прогнозирования разрешенного азарта.

В самом деле, отчего на нашем отрезке было ХОРОШО? Если мы еле-еле дошли с помощью $ДУ_{23}$, то и увеличивать отрезок незачем — пойдем тем же шагом. Но трусость теперь подстрахована: если отрезок мы взяли маленьким, то зато и обойдется это дешево: ХОРОШО пройдет уже по сравнению, скажем, $Г_5$ с (2×3). Если же на отрезке случилось ХОРОШО по умеренным ценам (с участием $У_{11}$, но без $ДУ_{23}$), то можно, пожалуй, и удвоиться.

Теперь наш умеренный оптимизм страхуется с обеих сторон: пожадничали — сэкономим на новом отрезке точки, обойдясь без $У_{11}$. Сазартничали — у нас в запасе $ДУ_{23}$...

А вот если было ХОРОШО — ОТЛИЧНО, т. е. $Г_5$ и (2×3) обошлись без $У_{11}$ вовсе, то нечего ограничивать порыв энтузиазма двойкой. Тут и четверка к месту — ведь в запасе еще и $У_{11}$ и $ДУ_{23}$!

На остальных деталях научного вычисления границ оптимизма я не останавливаюсь, чтобы не отбивать хлеб у автора [12–14]. Только замечу, что двойка и четверка поставлены условно — их в действительности Реет заменила другими числами. Полученными из экспериментов над стандартным набором функций-тестов (об этом тоже см. [12–14] [2]).

§ 5. Про внешнее интегрирование

Это — про кратные интегралы. Опять-таки, как поступит человек, скажем, при вычислении двойного интеграла

$$\int_0^1 dx \int_{a(x)}^{b(x)} F(x, y)\, dy$$

[2] За время, прошедшее до выхода этой книги, Реет сильно усовершенствовала свои алгоритмы. Какими они стали, можно прочесть в ее статьях.

вручную? Наверное, прежде всего он посчитает $F(x, y)$ по грубой сетке. Чтобы представить себе примерно, как разбивать дальше отрезок $[0; 1]$ изменения переменной x. Ровно так Реет и сделала. Она заменила функцию

$$\varphi(x) = \int_{a(x)}^{b(x)} F(x, y)\, dy$$

на «грубую» функцию $\varphi_{\text{грубое}}(x)$, взяв для каждого x интеграл по 2–5 гауссовым точкам. А теперь можно дешево найти разбиение, обеспечивающее заданную ε-точность для интеграла

$$\int_{\text{грубый}} = \int_0^1 \varphi_{\text{грубое}}(x)\, dx.$$

Дешево в том смысле, чти интеграл однократный, а цена каждого значения $\varphi_{\text{грубого}}(x)$ — от двух до пяти обращений к $F(x, y)$.

Зато теперь мы сможем для $\int_{\text{грубого}}$ сетку по x взять очень и очень экономную в смысле числа узлов — можно позволить себе потратить для этого много обращений к $\varphi_{\text{грубому}}(x)$ — функция-то эта дешева.

Получилась довольно неожиданная задача: для *конкретной* функции $\varphi(x)$ найти оптимальную в смысле числа узлов квадратуру, обеспечивающую заданную точность. Но ведь по теореме о среднем достаточно одной точки (!). Конечно, но ведь квадратуру-то мы ищем в *классе разрешенных квадратур*, получающихся дроблением интервала интегрирования на отрезки, где применяется уже некий стандартный набор квадратур. И теорема о среднем больше не опасна. Как эта задача делается, см. в [12].

После того как для $\int_{\text{грубого}}$ Реет нашла сетку узлов $\{x_s\}$, она поступает по-разному.

Во-первых, можно поверить, что эта же сетка узлов $\{x_s\}$ будет хороша и для интегрирования функции

$$\varphi_{\text{точное}}(x) = \int_{a(x)}^{b(x)} F(x, y)\, dy.$$

Для вычисления $\varphi_{\text{точного}}(x_s)$ интеграл по y надо брать с некоторой относительной точностью. Но ее мы можем теперь задать не только постоянной. Ведь $\int_{\text{грубый}}$ у нас уже сосчитан. И веса w_s, соответствующие

узлам x_s, у нас тоже есть. И

$$\int_{\text{грубый}} = \sum_s w_s \cdot \varphi_{\text{грубое}}(x_s).$$

Оценим ошибку выражения

$$\int_{\text{точный}} = \sum_s w_s \cdot \varphi_{\text{точное}}(x_s).$$

как

$$\text{ОШ}_{\text{общ}} = \sum_s w_s \cdot \text{ОШ}_s,$$

а для того, чтобы $\text{ОШ}_{\text{общ}}$ была $\leqslant \varepsilon \cdot \left|\int_{\text{точный}}\right|$, нам достаточно потребовать, чтобы при каждом s было

$$w_s \cdot \text{ОШ}_s \leqslant \frac{w_s}{\sum\limits_k w_k} \cdot \varepsilon \cdot \left|\int_{\text{точный}}\right|.$$

Теперь, веря в то, что

$$\varphi_{\text{грубое}}(x) \approx \varphi_{\text{точное}}(x)$$

и соответственно

$$\int_{\text{грубый}} \approx \int_{\text{точный}},$$

мы при интегрировании по y для каждого x_s можем задать допустимую абсолютную ошибку ДАО_s так:

$$\text{ДАО}_s = \frac{w_s}{\sum\limits_k w_k} \cdot \varepsilon \cdot \left|\int_{\text{грубый}}\right|.$$

Этим мы избавимся от потери времени, которая случалась оттого, что в

$$\int_0^{30} dx\, e^{-x} \int_0^1 \sin^2(x^2 y)\, dy$$

мы вычисляли при больших x излишне точно внутренний интеграл.

Только, конечно, в этом случае надо принять за $F(x, y)$ *все произведение* $e^{-x}\sin^2(x^2y)$ [3].

Подумаем, какая информация должна храниться для того, чтобы можно было осуществить описанный *первый метод* внешнего интегрирования Р. А. Пукк.

Очевидно, это — узлы x_s, веса w_s и еще допустимые ошибки ДАО$_s$, т. е. всего $3n$ величин, если сетка $\{x_s\}$ содержит n узлов. И еще, конечно, величина $\int_{\text{грубый}}$.

Для внешнего интегрирования это не так все страшно, но несколько противоречит совершенной стандартности программы ИНТЕГРАЛ.

Следующая трудность — похуже. Как нам проверить, насколько разумна наша гипотеза, что сетка x_s, выбранная по $\varphi_{\text{грубому}}$, удовлетворительна для $\varphi_{\text{точного}}$?

А то ведь мы и вовсе потеряем представление о *действительной точности*.

Проверку можно бы осуществить, введя, например, наряду с квадратурами Γ_5 квадратуры (2×3). И поверив, что если на данном отрезке [4]

$$\left|\left(\frac{M_5 - M_{(2\times3)}}{M_5}\right)_{\text{грубое}}\right| \geqslant \left|\left(\frac{M_5 - M_{(2\times3)}}{M_5}\right)_{\text{точное}}\right|$$

то функция $\varphi_{\text{точное}}$ интегрируется по нашей сетке не хуже, чем $\varphi_{\text{грубое}}$. Конечно, в этой оценке можно еще поставить некоторый либеральный множитель (0,8–0,5) перед правой частью неравенства. Но ведь это — именно проверка. Для спокойствия души. Ну, а как окажется, что наш критерий сказал, что дело плохо? Ведь проверяем-то мы точность, чтобы в случае чего *исправить дело*, а не просто констатировать факт неудачи.

Что ж, если плохо, можно на данном отрезочке по x перейти к У$_{11}$ и дальше к ДУ$_{23}$. Да еще по дороге использовать для контроля квадратуры (2×6) и (2×12), при этом работая, понятно, уже прямо с $\varphi_{\text{точным}}$.

А если и это не даст результата, то раздробить рассматриваемый отрезок...

В общем: действовать поближе к нормальному человеческому способу — по обстоятельствам.

---

[3] К потере времени при вычислениях это не приведет: e^{-x}, конечно, вычисляется в этом случае во внешнем цикле и хранится в отведенной для него ячейке как множитель.

[4] Обозначения, как в предыдущем параграфе: M_5, $M_{(2\times3)}$ и т. п. — соответствующие квадратуры.

Еще одна иллюстрация к тому, как невычислительные задачи применяются в вычислительных. Вернее — точка зрения, возникшая при работе с логическими (игровыми) задачами. Впрочем, автор не настаивает; если читателя раздражает такая точка зрения, он может просто использовать экономные алгоритмы Реет, считая, что невычислительные задачи тут и вовсе ни при чем.

Раздражение пройдет. Экономия — останется[5)].

И еще одно, последнее, замечание к первому способу. Об объеме хранимой информации. Мы можем хранить фактически вместо узлов и весов лишь величины отрезочков, на которые разбит интервал интегрирования, а еще лучше — точки разбиения. Если, конечно, на каждом отрезочке применяется именно Γ_5. Объем хранимой информации уменьшится в 10 раз. С $ДАО_s$ — хуже. Чтобы навести и здесь экономию, можно пожертвовать тонким различением ДАО для разных узлов одного и того же отрезка, а хранить некие величины ДАО отрезочков. Теперь все же хоть и не в 10, а 7,5 раз мы по объему выиграли.

Это — про первый способ.

А второй отличается от первого тем, что он позволяет не хранить ни $3n$ величин (узлы, веса, ДАО), полученных при работе с $\varphi_{\text{грубым}}$, ни даже $\frac{2}{15}$ от этих $3n$.

В самом деле, пусть нам удалось получить квазиоптимальное разбиение интервала интегрирования на отрезочки, исходя из работы с $\varphi_{\text{грубым}}$. Тогда, считая, что работа с $\varphi_{\text{грубым}}$ *дешева* по сравнению с вычислениями $\varphi_{\text{точного}}$, мы можем *при выборе каждого следующего отрезочка разбиения* повторить, например, весь алгоритм нахождения этого отрезочка с помощью $\varphi_{\text{грубого}}$.

Но до этого по большей части дело даже и не доходит: как правило, алгоритмы, во всяком случае у Реет, работают по системе «отрезок за отрезком». И потерь на повторение *всего* алгоритма не происходит.

Вот этим-то и можно воспользоваться; хранения информации нефиксированной *длины* мы избежим.

Теперь схема работы, например, такова:

Сперва получается по $\varphi_{\text{грубому}}$ приближенная величина $\left|\int_{\text{грубый}}\right|$, считанная, скажем, по $У_{11}$ или даже по $ДУ_{23}$.

Затем по одному из алгоритмов Реет (см. [14]) выбирается для $\varphi_{\text{грубого}}$ самый левый отрезочек. На нем вычисляются величины $M_{5_{\text{грубое}}}$ и $M_{(2\times3)_{\text{грубое}}}$. Получаем для каждого из 5 узлов $\{x_s\}$ *данного* отрезочка величины $ДАО_s$, как сказано выше. Мы пользуемся при этом $\left|\int_{\text{грубый}}\right|$ и заданным ε.

[5)] См. примечание к с. 204

С заданной абсолютной точностью ДАО$_s$ считаем теперь величины $\varphi_{\text{точное}}(x_s)$. Теперь составим $M_{5_{\text{точное}}}$ и $M_{(2\times3)_{\text{точное}}}$ и сравним

$$L \cdot \left| \left(\frac{M_5 - M_{(2\times3)}}{M_5} \right)_{\text{точное}} \right| \quad \text{с} \quad \left| \left(\frac{M_5 - M_{(2\times3)}}{M_5} \right)_{\text{грубое}} \right|,$$

где L — наш либеральный множитель — ну, например, 0,5. Заметим только, что все же не следует брать L столь уж либеральным, если, скажем, $\left| \left(\frac{M_5 - M_{(2\times3)}}{M_5} \right)_{\text{грубое}} \right| = 0{,}7$. Вообще, если отношение $\left| \left(\frac{M_5 - M_{(2\times3)}}{M_5} \right)_{\text{грубое}} \right|$ очень велико, — а это могло случиться, если отрезочек выбран, скажем, по совпадению M_{11} с M_5, а не M_5 с $M_{(2\times3)}$, — то такой критерий вовсе плох. Боюсь, что в этом случае придется контроль вести прямо-таки по M_5 и $M_{11_{\text{точным}}}$. Но если даже выбор отрезочка происходил для $\varphi_{\text{грубого}}$ из сравнения M_5 с M_{11}, а величины M_5 и $M_{(2\times3)}$ разнятся *не слишком* сильно, то вдвое более дешевое сравнение $M_{5_{\text{точное}}}$ с $M_{(2\times3)_{\text{точным}}}$ вполне приемлемо.

Итак, мы сравнили оценки ошибок для $\varphi_{\text{грубого}}$ и $\varphi_{\text{точного}}$.

Если у $\varphi_{\text{точного}}$ выявилась тенденция интегрироваться *не хуже*, чем $\varphi_{\text{грубое}}$, — все хорошо, и мы переходим к следующему отрезку.

А если сравнение сказало ПЛОХО, то на отрезочке мы переходим теперь, забыв про $\varphi_{\text{грубое}}$, к вычислению $\int \varphi_{\text{точное}}(x)\,dx$ по всей науке, описанной в § 4. И лишь покончив с этим отрезочком — двинемся дальше.

Теперь нам предстоит выбрать очередной отрезок. На нем мы дальше поступим в точности так, как на самом левом. А вот каким его взять?

Во-первых, мы можем получить его, используя алгоритм выбора оптимальной сетки для $\varphi_{\text{грубого}}$. Но, кроме того, можно учесть также и данные о наших трудах с $\varphi_{\text{точным}}$ на предыдущем отрезке.

Например, если $M_{5_{\text{точное}}}$ с $M_{(2\times3)_{\text{точным}}}$ совпали много лучше, чем $M_{5_{\text{грубое}}}$ с $M_{(2\times3)_{\text{грубым}}}$, прогноз по $\varphi_{\text{точному}}$ гласит: следующий отрезок можно оптимистически попробовать взять в 4 раза большим. Сравним это с прогнозом по $\varphi_{\text{грубому}}$. Теперь дело в характере (программиста). Если программист — оптимист, то можно прямо пробовать более азартный вариант — больший из отрезков. Пессимисту следует побояться и выбрать все-таки тоже больший из отрезков. Потому что пессимизм — пессимизмом, а экономия — экономией.

Но вот если по $\varphi_{\text{грубому}}$ отрезок был угадан с большим завышением и нам пришлось разбиваться и разбиваться, то теперь нужно на следую-

щем шагу проявлять осторожность. И коль скоро на предыдущем шагу сработало, скажем, для $\varphi_{\text{точного}}$ лишь сравнение M_{23} с $M_{(2\times12)}$, а теперь $\varphi_{\text{грубое}}$ предсказывает *уменьшение* отрезка, пессимист может взять реванш: стоит послушаться осторожного совета и так-таки отрезок и взять небольшим.

Ну, и так далее. Не буду отбивать хлеб у Реет, тем более что все ее алгоритмы все равно не опишешь.

§ 6. А должна ли вообще программа ИНТЕГРАЛ быть совершенно стандартной?

В шестой беседе мы ответили на этот вопрос положительно. А в этой беседе вроде бы напрашивается отрицательный ответ?

Нет, не напрашивается. Посмотрите на эпиграф. Ну, с какой стати вычислять двойной интеграл, применяя всю тяжелую артиллерию предыдущих параграфов, в задаче, где, скажем, этот интеграл — вещь эпизодическая и его вычисление берет 2 % времени от всей работы! А память, наоборот, перегружена, и приходится пользоваться барабанами. Но вот если время берет именно интегрирование — пустимся во все тяжкие!

И опять же: пусть теперь самый внутренний интеграл берется, скажем, по z от

$$f(x, y, z) = \frac{\sin(x+y+z)}{x+y+z} L_{17}(z),$$

где $L_{17}(z)$ — полином Лежандра 17-го порядка.

Тогда, учитывая большую гладкость множителя $\frac{\sin(x+y+z)}{x+y+z}$, нужно лишь позаботиться о достаточной точности квадратуры при вычислении $L_{17}(z)$. Многочлен L_{17} — 17-й степени. Значит Γ_5 возьмет его неверно. И хорошо применить Γ_9 — гауссову квадратуру 9-го порядка, обеспечивающую точное интегрирование многочленов до 17-й степени включительно.

И больше не требовать никакого контроля точности во внутреннем интеграле!

Вот так и устроен мир — довольно-таки разнообразно.

И, наконец, еще об одной задаче.

Очень часто приходится вычислять интегралы с параметром. А параметр p принимает много значений, и каждое следующее значение, скажем, несколько отличается от предыдущего. Как теперь использовать это обстоятельство? По-видимому, сетка, полученная для вычисле-

ния интеграла (пусть, для простоты — однократного) при некотором значении параметра, более или менее сгодится и для следующего значения параметра, близкого к рассмотренному. Теперь выгодно эту сетку хранить. А при изменении параметра от нее и отправляться. Только заметим, что наряду с контролем точности при переходе к новому значению параметра[6)], приводящему при неудаче к *дроблению* сетки, надо позаботиться и о возможном *укрупнении* сетки. Иначе для функции, имеющей, например, излом в точке, двигающейся в зависимости от параметра от левого конца интервала интегрирования к правому, мы постепенно раздробим в мелкую дребезгу весь интервал интегрирования.

Разумный оптимизм и необходимый пессимизм программируются в этом случае примерно по образу того, как это делалось в § 5. При этом функция $f(x, p_{\text{пред}})$ играет роль $\varphi_{\text{грубого}}$, а $f(x, p_{\text{текущ}})$ — роль $\varphi_{\text{точного}}$. Конечно, то же происходит при вычислении в двойном интеграле внутренних интегралов. Роль параметра в этом случае играет величина внешней переменной.

Еще более сложная ситуация возникает при вычислении двойных интегралов с параметром. Теперь хочется во внутреннем интегрировании по y при данном x пользоваться опытом, полученным при вычислениях интеграла по y при предыдущем значении x. Но, кроме того, ведь хочется еще и учесть, что у нас случалось для сетки по x для предыдущего значения параметра. А может захотеться для экономного выбора сетки по y при разных x учесть опыт работы с предыдущим параметром более тонко. Можно даже представить себе программу, следящую, скажем, за поведением участка, требующего мелкого дробления, прогнозирующую его перемещение с ходом параметра на основе предшествующих вычислений и использующую этот прогноз для быстрого выбора подходящих отрезков. Только не следует увлекаться и упускать из виду главное — экономию времени. А то ведь неудачная всеобщность может привести к прогнозированию везде и всюду. Это не даст существенного выигрыша в сетке, а времени на изыскания начнет занимать все больше и больше. Особенно, если все это заавтоматизировать!

В общем, еще раз вспомните об эпиграфе к этой главе.

[6)] Этот контроль можно осуществить сравнением, скажем, величин:

$$\left|\left(\frac{M_5 - M_{(2\times3)}}{M_5}\right)_{\text{предыдущее}}\right| \quad \text{и} \quad \left|\left(\frac{M_5 - M_{(2\times3)}}{M_5}\right)_{\text{текущее}}\right| \quad \text{и т. д.}$$

§ 7. Решение системы двух уравнений с двумя неизвестными

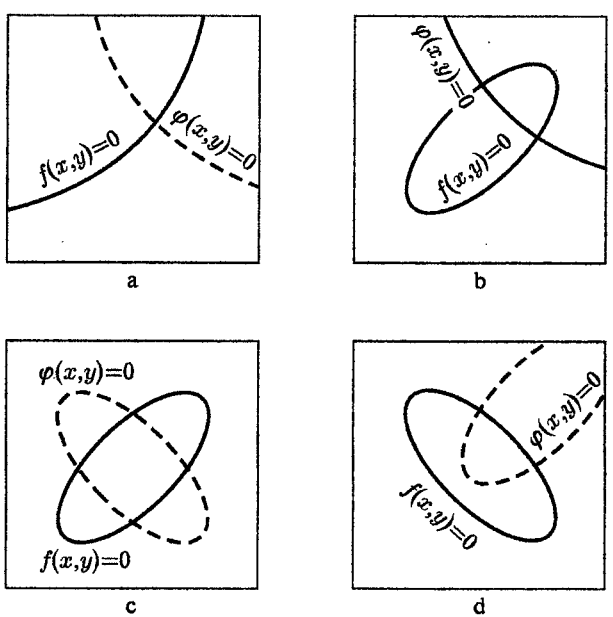
Пусть нам надо решить систему:

$$\begin{cases} f(x, y) = 0, \\ \varphi(x, y) = 0. \end{cases}$$

Рассмотрим самый простой случай. Пусть разрешенная область изменения для x и y — единичный квадрат

$$0 \leqslant x \leqslant 1,$$
$$0 \leqslant y \leqslant 1.$$

и пусть еще для простоты в этой области каждое из уравнений системы имеет в качестве решения гладкий гомеоморф окружности или гладкую простую дугу с концами на краях области задания (см. рисунки).



Отвлечемся также от неприятного случая касания кривых, задаваемых нашими уравнениями по отдельности. Если в нашем распоряжении есть программа КОРЕНЬ ФУНКЦИИ, мы для каждого фиксированного

значения x можем найти, например, минимальный по y корень $f(x, y)$ (или убедиться, что его нет). Пусть

$$y = y(x)$$

и есть этот минимальный корень.

Теперь применим ту же программу КОРЕНЬ ФУНКЦИИ[7] к нахождению корня функции

$$\psi(x) = \varphi[x, y(x)].$$

Если даже отвлечься от неприятного для такой методы случая d (см. рисунки), все равно наш алгоритм расточителен — мы имеем повторное применение программы корня, что, грубо говоря, означает «в среднем» квадрат числа действий против числа действий для нахождения однократного корня.

А как поступает человек? Ну, конечно, просто рисует грубо сосчитанные кривые, а потом уже уточняет найденное примерно место пересечения.

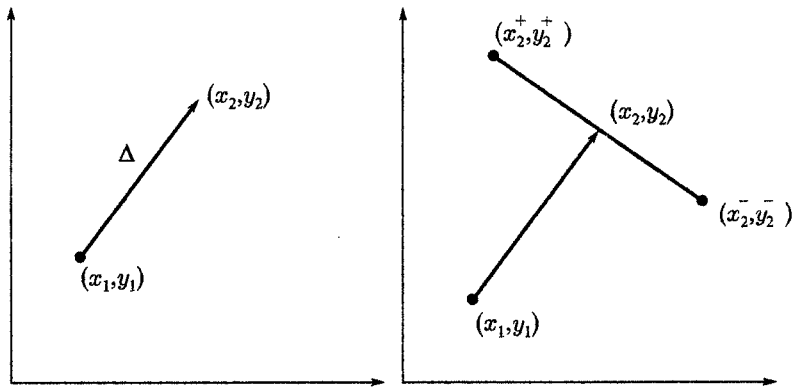
А как же сделать это программе?

Ну, прямо ровно так я не умею. А вот примерно так — можно. Прежде всего найдем такое значение x, при котором корень $f(x, y)$ уже существует. И аккуратно вычислим точку (x_0, y_0) на кривой $f(x, y) = 0$. Теперь мы можем идти *вдоль* этой кривой, следя за поведением $\varphi(x, y)$.

Но так мы «в высшем смысле» ничего не выиграли: как искали на каждом шагу корень $f(x, y)$, так и ищем. Правда, не совсем так: уже зная предыдущие точки кривой, мы сможем недурно прогнозировать новые точки. Время от этого сильно выиграется — поиск будет идти на маленьком отрезке. Только лучше отказаться от выделения переменной x как независимой. Перейдем от нее к оси, расположенной вдоль касательной к кривой в данной точке. Нам сразу станет легче на поворотах (т. е. где $\frac{dy}{dx} = +\infty$). И мы сможем обойти всю кривую.

Но нам дальше нет нужды и в точном нахождении корня $f(x, y) = 0$ на каждом шагу. Пусть мы находимся в точке (x_1, y_1) этой кривой и знаем направление касательной. Пройдем по нему отрезочек длины Δ. Мы пришли в точку (x_2, y_2). Теперь возьмем отрезочек, перпендикулярный к направлению движения и с серединой в (x_2, y_2). Пусть концы его суть $(x_2^{\pm}, y_2^{\pm})$ (см. рисунок на с. 214). Вычислим функцию $f(x, y)$ во всех этих трех точках: (x_2, y_2) и $(x_2^{\pm}, y_2^{\pm})$.

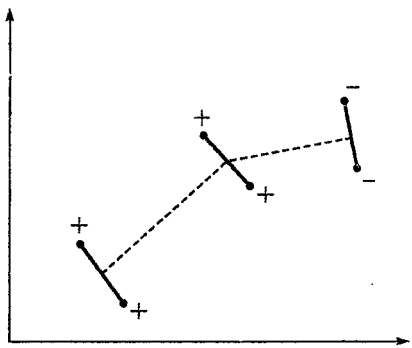
[7] Если она совершенно стандартна. В противном случае — ее двойник.



Расстояние $(x_2^{\pm}, y_2^{\pm})$ от (x_2, y_2) можно взять много меньшим Δ. Если $f(x, y)$ в точках (x_2^+, y_2^+) и (x_2^-, y_2^-) *разного* знака, мы можем поручиться, что на отрезке (x_2^+, y_2^+), (x_2^-, y_2^-) есть корень $f(x, y)$, т. е. точки (x_2^+, y_2^+) и (x_2^-, y_2^-) образуют ВОРОТА ПО $f(x, y)$. Теперь мы можем двигаться дальше, *не зная точного положения корня* $f(x, y)$ в воротах. Некоторое затруднение будет в выборе направления касательной. Но ведь нам и его-то знать абсолютно точно ни к чему! А по трем точкам (x_2^+, y_2^+); (x_2, y_2) и (x_2^-, y_2^-) и значениям функции $f(x, y)$ в этих точках мы недурно вычислим приблизительное положение корня (x_2^0, y_2^0). И теперь, скажем, пойдем дальше по направлению от (x_1, y_1) к (x_2^0, y_2^0). Конечно, сама точка (x_1, y_1) тоже не обязана была быть точным корнем $f(x, y)$. Она могла играть роль (x_2^0, y_2^0) в своих воротах, т. е. это как бы точка (x_1^0, y_1^0). Понятно, что если нам не повезло и точки $(x_2^{\pm}, y_2^{\pm})$ не дают ворот по $f(x, y)$, следует расширить отрезок $[(x_2^+, y_2^+), (x_2^-, y_2^-)]$ для поиска ворот. А может случиться, что мы даже и вовсе сошли с кривой. Тогда придется уменьшить шаг Δ.

И еще, конечно, мы можем применить при поиске точки (x_3, y_3) не линейную экстраполяцию (по x_1^0, y_1^0 и x_2^0, y_2^0), а параболическую по трем предыдущим точкам. Выше второй степени идти не советую (из опыта).

Итак с помощью ВОРОТ ПО $f(x, y)$ мы обходим всю кривую-корень $f(x, y)$. А где же все-таки лежит общий корень нашей системы? Будем следить за $\varphi(x, y)$ на краях ВОРОТ ПО $f(x, y)$. Вернее — за знаками значений $\varphi(x, y)$ на краях. Пусть до некоторого шага был знак плюс на обоих краях. А на этом шагу сменился сразу на обоих краях на минус (см. рисунок).



У нас все основания полагать, что здесь-то и лежит корень. Теперь, очевидно, надо дробить Δ вдвое и затем уточняться на последнем участке. Подробно всей дальнейшей процедуры я описывать не стану. Здесь применимы и, более того, уместны все соображения, аналогичные тем, какие применены в обычной программе КОРЕНЬ ФУНКЦИИ. Отмечу еще только, что коль скоро знаки у $\varphi(x, y)$ на краях очередных ворот вышли *разными*, следует *сузить* ВОРОТА ПО $f(x, y)$ до той степени, чтобы у $\varphi(x, y)$ снова стали *одинаковые знаки* на краях суженных ворот. Если опять эти одинаковые знаки те же, что и раньше, — двинемся вперед. А если сменились — начнем уточнение положения корня.

Описанный алгоритм интересен автору не только как автору (алгоритма), а главным образом — как еще один пример приложения «игровых» соображений в вычислительных задачах. А то не подумал бы собеседник, что ИНТЕГРАЛ — единственная область приложения невычислительных задач к вычислительным.

Строго говоря, в этом параграфе собственно игровые соображения не выговорены. Дана лишь вычислительная канва алгоритма. Но внимательный собеседник-программист увидит, где именно в этом алгоритме появятся игровые элементы (тактика, оптимизм — пессимизм и т. д.), оставленные в беседе за сценой. А на такого именно собеседника автор и рассчитывает.

Беседа пятнадцатая и последняя

Кибернетика или математика? Кто такие программисты?

Меня милый не целует,
Не садится близко:
Я, мол, чистый математик,
А ты программистка.

Фольклор второй половины
XX века

§ 1. Что такое наука кибернетика?

Слово «кибернетика» произошло от переложения на русский язык английского слова «cybernetics». А оно — от переложения на английский греческого κιβερνοσ, что имеет смысл — управлять, управление. А непосредственно с греческого на русский слово κιβερνοσ тоже перелагалось. Отсюда произошли: губернатор, губерния, гувернантка. Так что «Кибернетический сборник» в позапрошлом веке назывался «Губернские ведомости».

Из самого названия ясно следует, что наука-кибернетика есть наука об управлении. Кем, чем управлении — из названия не следует. Обратимся к авторитетам.

Отцом науки-кибернетики является ныне покойный американский математик Норберт Винер. Точнее — крестным отцом. Так как это он придумал имя «наука-кибернетика».

После этого Норберт Винер, автор отличных работ по математике (винеровские кольца, метод Винера—Хопфа), написал несколько книжек про науку-кибернетику. Из них мы и узнали про ее, науки-кибернетики, всеобъемлющий характер. Получилось так. По ведомству науки-кибернетики проходит все, где есть управление:

Математические машины управляются?	...	Математическая кибернетика.
Живой организм управляется?	...	Биологическая кибернетика.
Человеческое общество управляется?	...	Социальная кибернетика.

И по более мелким подразделениям:

Язык грамматикой управляется?	...	Лингвистическая кибернетика.
Организм с болезнями управляется?	...	Медицинская кибернетика.
Автомобиль (рулем) управляется?	...	Автокибернетика.

Ну, тут, положим, я уж заврался. Нет... автокибернетики, нет... А остальное — есть.

Некоторые кибернетики отложились от метрополии. И, превратившись в республики, сменили наименование. Так случилась бионика (до независимости — биологическая кибернетика).

Все ученые сразу и очень охотно согласились, что раз повсюду управление, то должны быть и аналогии.

§ 2. Еще одна аналогия

Религиозно образованный читатель знает схему христианского вероучения. Антирелигиозно образованный — тоже. Для ни туда ни сюда не образованных рекомендуется литература [15–16].

Итак — аналогия. Пусть наш мир — это электронная вычислительная машина. А Бог — коллектив ее сотрудников, рабочих-изготовителей и бригада наладчиков.

Во-первых, мы имеем модель сотворения мира. Во-вторых, коллективизм — тоже аналогия. Раскрывающая тайну божественного триединства. В качестве Бога-отца выступает, конечно, конструктор. Бог-святой дух — явно наладчики. О Боге-сыне отдельно. А единство в том, что ни конструктор без наладчиков, ни они без конструктора действующую машину не сотворят. И если будут действовать вразброд — также. А лица-то это все разные. Вот вам и единый Бог в трех лицах. На модели получается очень наглядно.

Ну-с, а кто же в нашем мире-модели исполняет роль человека? Понятно кто — программа. Ибо именно она живет и действует в оперативной памяти, плодясь и множась (формирование команд) и наполняя

землю (роспись массивов). А кроме того, у нее есть Высшая цель — правильно достичь своего естественного конца (стоп в главной собиралке). Выполнив при этом Божественное предначертание Программиста — решив заданную задачу.

И здесь естественно появляются понятия греха, искупления, наказания, жизни вечной и т. д.

В самом деле, что считать грехом? Ясно что — отклонение от Божественного предначертания — ошибку в программе. А что за это будет? Тут есть две возможности.

Грех бывает смертный (очень грубая ошибка). И за это программу просто стирают (общий сброс). И нет ей Жизни Вечной.

А бывает и грех простительный. И тогда происходит Искупление. С помощью ПОПРАВКИ С ПУЛЬТА. И, заметьте, эту муку берет на себя Программист. Вот он — Бог-сын [1]. А вслед за Искуплением наступает Обновление [2] и новая, уже безгрешная, жизнь — до очередной ошибки. Долготерпелив Программист — много ошибок он берет на себя (ПОПРАВКА С ПУЛЬТА), и только смертный грех заставит его стереть программу начисто.

Ну, а как с Жизнью Вечной? Она дается только праведникам. И осуществляется на перфокартах. И Воскресение (в ферритной памяти) может произойти в любой день и час — было бы по расписанию машинное время! А неисправимым грешникам такого не дано — какой же дурак будет хранить неверную колоду?

Это — вариант христианский.

А для буддистов есть свое. Как осуществляется переселение душ? Ну ясно как — с магнитного барабана. Или с магнитной ленты. Из Магнитной Нирваны — в активную ферритную жизнь. А по завершении пути — снова в Нирвану. Так что достичь Нирваны — явная цель программы в жизни ферритной. И ТУДА—ОБРАТНО много раз.

А переселение (на другие адреса) осуществляет ВЫЗОВ СО СДВИГОМ. И, конечно, в своем новом воплощении программа никак не помнит, на какие адреса вызывали ее раньше. Аналогия полная, глубокая и очень плодотворная.

Дальнейшее ее развитие я оставляю тем из читателей, кто всерьез посвятит себя этим проблемам. Сам же удовлетворюсь сознанием, что моими трудами ко всем существовавшим до этого кибернетикам добавилась еще

ДЕОКИБЕРНЕТИКА

---

[1] Программисты, как правило, молоды..

[2] А. Л. Брудно предлагал так и называть восстановление возобновлением или обновлением.

Как сказал поэт:

...с меня довольно
Сего сознанья...

§ 3. Как должна развиваться наука-кибернетика?

Плодотворно. Для этого только надо:

1°. Чтобы программированием занимались программисты.

2°. Механизмами в живом организме занимались физиологи, биофизики, биохимики и т. д.

3°. Специалисты по социальным дисциплинам изучали законы, управляющие обществом.

4°. Лингвисты и филологи занимались своим делом.

5°. Врачи лечили и с этой конечной целью изучали болезни и больных, а также и здоровых.

6°. Шоферы водили автомобили (автобусы, самосвалы и т. д.) и

7°. Никто без нужды не болтал попусту.

А когда врачам нужны теория вероятностей, математическая статистика, дифференциальные уравнения или работа на электронных машинах — им необходимо всемерно помогать в этом деле. И иногда — учить и даже подталкивать, потому что человек часто просто не знает о возможностях чужой науки.

И делать это надо профессионально.

А наука-кибернетика здесь ни при чем.

И автор даже подозревает, что она и ни в каком деле ни при чем. Потому что он лично не слышал ни о каком достижении этой науки-кибернетики.

И полагает, что, кроме названия, у науки-кибернетики больше ничего и нет.

Как у поручика КИЖЕ. Впрочем — это личное мнение автора.

§ 4. Кто такие программисты

Определение 3[3]. Математик, владеющий программированием, называется *программистом*.

Определение 4. Остальные математики называются *настоящими*, или *чистыми*, *математиками*.

[3] Первые два определения см. в третьей беседе.

Прежде всего одно замечание к определению 3. Почему именно математик? А если программированием владеет нематематик, как тогда? Тогда я покаюсь и сменю определение. Но если понимать под программированием то, что в этой книге понимается, — т. е. решение серьезных вычислительных и невычислительных задач средствами машинной техники, — то я ни разу не видел программиста-нематематика. Ну, конечно, в широком смысле.

А теперь подумаем: какое из состояний — программист или чистый математик — выше?

Очевидно, более высоким нужно считать состояние человека, дающее ему больше возможностей. Следовательно, безусловное предпочтение надо отдать чистому математику. Потому что он может стать программистом, а программист чистым математиком — не может (см. опр. 3 и 4).

И зря девочка в эпиграфе жаловалась на судьбу. Целовать ее с его стороны было бы чистым мезальянсом. Так что она, дура, сама виновата. И так ей и надо.

§ 5. Нужно ли каждому математику самому программировать?

Конечно, нет. Даже если он работает в вычислительном центре или институте прикладных вычислений. Конечно, если этот математик — профессор. Или — больше. Тогда ему следует иметь при себе некоторый штат программистов, которым он будет давать ценные и руководящие указания (ЦУ и РУ соответственно). А они — осуществлять эти указания. Докладывать о получившихся результатах. Получать новые ЦУ и РУ. Снова докладывать. И т. д. Такое разделение труда очень эффективно: руководитель не тратит сил по пустякам, и вся его интеллектуальная мощь обрушивается на преодоление идейных трудностей задачи.

А программисты тоже работают более плодотворно: не обремененные заботами об идейных трудностях, они все освободившиеся силы приложат к составлению очень хороших программ. И дело будет двигаться быстро. Это называется

РАЗДЕЛЕНИЕ ТРУДА В НАУКЕ.

Нужно сказать, что такое разделение труда — не новость. Вероятно, Ньютон не открыл бы ни своих законов физики, ни интегрального и дифференциального исчислений, не будь у него штата помощников. Так, из истории физики известно, что один из помощников дежурил в саду. Он наблюдал за яблонями. И докладывал результаты наблюдений. А другой помощник-теоретик был астрономом. И тоже докладывал — но уже вопросы теоретические: законы Кеплера и пр. Третий

занимался Луной. А Ньютон обобщал все их доклады. И ему оставалось только, сведя все вместе, открыть свой всемирный закон. И так же с интегральным и дифференциальным исчислениями (см. «Историю математики» М. Я. Выгодского).

То же было и с Эйнштейном. У него ассистенты подразделялись на оптиков-экспериментаторов, оптиков-теоретиков, теоретиков-механиков и — математиков (тех из них, кто дифференцировал, называли дифференциаторами, а интегрировавших — интеграторами, как теперь программирующих — программистами; однако термины эти быстро устарели, и интеграторами стали называть не людей, а один вид аналоговых машин). Эйнштейн же давал им задания, проверял результаты и думал — как назвать эти результаты. И назвал их теорией относительности (частной). Как вы думаете, мог ли бы он сам, без научных сотрудников придумать так быстро такую трудную теорию? А так у Эйнштейна было даже свободное время для работы в патентном бюро (по 8 часов в день). И в дальнейшем, чтобы подчеркнуть коллективный характер творчества, он очередную теорию так и назвал: *общая* теория относительности.

Если я что-нибудь написал не так — прошу извинения, я все-таки не являюсь настоящим специалистом-профессионалом в области истории физики.

А в программировании — являюсь. И потому заявляю совершенно категорически: самому программировать математику совершенно незачем. Так же как и интегрировать. И — дифференцировать. Труд следует разделять.

§ 6. Взаимоотношение программирования и математики

Строго говоря, программирование само по себе не требует знания математики. Кроме, конечно задач, по самой своей формулировке математических, — интегрирование, решение уравнений в частных производных и т. д. А как раз наиболее трудные задачи программирования — невычислительные. Так что *формально* здесь хватило бы знаний неполной средней школы.

По существу дело обстоит как раз наоборот. Действительно хорошие программисты все без исключения — люди с профессиональным математическим образованием. И более того — сильные математики в классическом, «старом» смысле, т. е. каждый из них способен придумывать и доказывать трудные теоремы.

Я думаю, что дело в большой логической напряженности работы программиста. Ближе всего в этом смысле к программированию как раз и подходит нормальная «чистая» математика. Более того, эмпирически

наблюдается часто такое развитие интересов: задачи на построение — теория функций действительного переменного — программирование. Но это уже из области домыслов при малой статистике.

Итак, математическое образование в допрограммистском смысле является, по мнению автора, лучшей и, вероятно, пока единственной школой для будущего программиста. Быть может, с накоплением весомого собственного багажа программирование и не будет требовать латынь чистой математики. Пока же этот багаж мал.

Это — одна сторона дела. А другая состоит в том, нужно ли программирование чистому математику. Думаю, что, как правило, — нет. Так как оно не поможет при доказательстве теорем чистой математики. Но так обстоит дело сегодня и именно сегодня.

А в дальнейшем, рано или поздно, будет достигнут такой уровень, что машины (программы) станут способными решать «думательные» задачи. Быть может, этот процесс окажется сильно растянутым: одни задачи станут доступными, скажем, через 10–15, а другие — через 40–50 лет. Пока издали различие между ними кажется мелким по сравнению с расстоянием до ближайших из «думательных» задач. Правда, по мере нашего приближения к этим ближним вершинам дальние могут отступить.

Но так или иначе наступит время, когда серьезная мыслительная работа будет делаться всегда с помощью машин. По крайней мере математиками. Так же, как сегодня никто не станет всерьез поднимать большие тяжести без помощи подъемного крана. Разве что на соревнованиях.

Ну, что ж, если не я, то более юные читатели этой книги доживут, быть может, до такого времени, когда будут проводиться *спортивные соревнования по думанию*. На них будет запрещено пользоваться математическими машинами. Нужно будет думать, так сказать, «голыми мозгами». Как сегодня тяжелоатлеты поднимают штангу «голыми руками» — без лебедок и подъемников. Только спортивные соревнования с работой никто не перепутает.

§ 7. Наука для науки?

Если это про чистую математику, то что ж. В заметной степени, пожалуй, что и да. Во всяком случае стоит очень и очень задуматься — для чего нужны новые теоремы чистой математики. И, главное — стоит ли на них направлять усилия молодежи. Правда, «направлять усилия» творчески одаренного и активного человека удается не так-то легко, но сколько-то направлять их все же можно. Представим себе на минутку, что создание новых математических теорий и доказательство трудных теорем является нашей конечной целью. Что же в этом случае надо делать? Прибегнем снова к сравнению.

Пусть нам нужно выкопать большой канал. С чего стоит начать? Первое, что приходит в голову, — взять лопаты и копать. И, если не думать, что можно построить экскаватор, то чем скорее возьмешь лопату и чем энергичнее станешь ею действовать, — тем быстрее и глубже будет твоя канава.

Ну, а если поверить, что экскаватор построить все-таки можно? Тогда, пожалуй, стоит повременить с лопатой. И как можно энергичнее заняться экскаватором. Особенно, если копать лопатой и строить экскаватор должны одни и те же люди. И, конечно, если канава не нужна немедленно и непосредственно, во имя спасения жизни. С основной частью чистой математики этого вроде бы нет.

§ 8. Наука. Прикладная Наука. Наука для Науки

Обойдемся без определений. Подумаем только о приложениях. Наибольшие приложения дает наука просто. В макрокосме — это теория относительности. В микрокосме — программа для игры в шахматы.

Прикладная Наука дает для практики меньше. А иногда даже и не быстрей, чем наука просто. Может быть, оттого, что, поставив перед собой задачу своей работой во что бы то ни стало помочь практике, исследователь связывает себя лишними обременительными для исследования условиями. Если повесить себе на спину рюкзак, в олимпийском беге не победишь. В научном исследовании — тоже.

Это если цель — исследование. А вот если цель — донести рюкзак, то тогда все выглядит как раз наоборот. Но только с этого давайте и начинать. И получатся — в меру таланта несущего — изобретения и открытия, хорошие изобретения и открытия, отличные, великолепные, делающие революцию изобретения и открытия... Это если цель — практика.

А смешивают два эти ремесла очень часто в Теории Техники. И в Методах Вычислений. Вы никогда не думали, почему этой последней Наукой занимаются одни люди, а вычисляют другие? И между собой они как-то совсем не взаимодействуют...

Ну, и, наконец, Наука для Науки. Зачем, все-таки нужна была Царица Математики — Теория Чисел?

Да, да, удивительное число сот лет Лучшие Умы и т. д. и т. д. Но ведь это можно рассмотреть и с обратной точки зрения: ведь вот уже сколь удивительное число сот лет Лучшие Умы растрачивают свои силы, а все без толку...

Сразу оговорюсь — кроме наслаждения эстетического. И значения педагогического.

§ 9. О критериях

Легко отличить науку от хорошей практики (т. е. того, что непосредственно нужно людям и что в действительности занимает место Прикладной Науки; само же название Прикладная Наука сильно скомпрометировано профессионалами этой деятельности). Да, собственно, и отличать не нужно — делает каждый свое дело и обычно вполне уважительно относится к деятельности другой стороны. Как мастера одного ремесла — к мастерам другого. И перепутаем — тоже не беда. Не все ли равно, по какой епархии считать, скажем, открытие лазеров или пенициллина или создание ЭВМ?

Прикладную Науку отличить тоже легко — по жухлым ее плодам.

А вот науку от Науки для Науки? Тем более, что одна (первая) легко переходит со временем в другую (вторую). Хотя обратного процесса не наблюдается. Ведь и признаки делимости — вещь небесполезная. А вот простые числа-близнецы?...

По этому поводу я могу только пересказать одну восточную легенду, слышанную мной от В. А. Рохлина. Ею мы и закончим.

«Калифу Багдадскому наскучили жены в его гареме. Велел Калиф Главному Евнуху найти новую жену, с которой он, Калиф, не соскучился бы. Срок — месяц. Награда — два мешка золота. Не получится — голову с плеч!

Главный Евнух очень испугался и посоветовался со знакомым Купцом. Обещал Купец помочь. Ровно через месяц привел он к Евнуху закутанную в шаль женщину. Срок истекал, и Евнух, не глядя, отвел ее к Калифу. Наутро, ничего не соображая от страха, Главный Евнух явился к Калифу Багдадскому — получить золото или сложить голову. Получил золото. Очухавшись, побежал он к знакомому Купцу и отдал ему один мешок. А потом совсем развеселился и стал расспрашивать Купца, как тот догадался, что Калиф с этой женщиной не соскучится. И ответил знакомый Купец Главному Евнуху:

— А для этого надо не быть евнухом...».

§ 10. Предлагает ли автор запретить чистую математику?

— Нет, не предлагает. И — будь его воля запрещать и разрешать — разрешил бы. Потому что лучше с рвением ковырять землю лопатой, чем из-под палки конструировать экскаватор. Тем более, что конструировать из-под палки нельзя вовсе. А тем более — программировать невычислительные задачи.

И еще не предлагает запрещать потому, что ни разу не видел толку от запрещения той или иной науки или теории. Как-то так получалось, что запрещенная наука (теория) и оказывалась как раз особенно плодотворной. Так уж не везло во все времена с запретами.

Но, с другой стороны, автор не видит и причин для запрета *обсуждения* пользы и целей чистой математики. Потому что ведь экскаватор-то построят!

И субъективно я воспринимаю научную важность адептов чистой математики как все более напоминающую важность аристократов Сен-Жерменского предместья. В эпоху, ну, скажем, конца прошлого столетия. Когда титулы еще оставались, но значения им уже никто, кроме их владельцев, не придавал. И все вокруг ветшало, ветшало...

§ 11. Не исключить ли зато программирование из математики?

Машины математики восприняли в массе очень уж неприязненно. Кто не согласен — оглянитесь кругом. Ну, скажем, на вузы и втузы, где есть ЭВМ. Во многих ли втузах на математических машинах работают профессора, доценты и т. д. кафедр математики? Не парадоксально ли это?

Один весьма видный математик в 1951 году в откровенной беседе (дело происходило на праздновании докторской защиты одного моего знакомого — его ученика) утверждал, что математические машины — это мода, которая скоро пройдет, как проходили многие моды. Вокруг было несколько способных студентов, и разгорелся яростный спор. А я удивился, как может случиться, чтобы математик настоящего калибра не смог отличить моду от революции. Теперь, правда, уже отличил.

Главный аргумент чистых математиков против программирования как части математики — отсутствие в программировании истинной теории: определений, теорем. Вместо этого — приближенное вычисление на машине интегралов, которые иной раз и берутся-то в элементарных функциях. Переборные схемы. Алгоритмы, про которые *ничего* не доказывается. В общем, какая уж там математика...

И вот по этому поводу приходит мне в голову еще одна аналогия. На этот раз уже последняя. Было уже такое. Двести с лишком лет назад. Когда на смену элегантным и точным трактатам о циссоидах и конхоидах пришло варварское исчисление бесконечно малых. Сперва даже не на смену. А просто стало рядом. Не было в нем тогда ни точных доказательств, ни теорем. И изящества тоже не было. Шел период первоначального накопления.

А было в этом исчислении одно — грубая сила.

И никто никому ни циссоид, ни конхоид не запрещал. Даже наоборот. Создатели новой математики стеснялись своего детища. И нет-нет старались обойтись старыми методами. А потом просто дожило свой век поколение, воспитанное на циссоидах и конхоидах. И ушло. А с ним вместе ушли из живой науки красота и изящество доньютоновской математики. Растворились, как клубы тумана летним утром. Даже дымки не осталось к полудню. И только историки математики бережно хранят воспоминания о классической элегантности циссоид и конхоид. А все место под солнцем заняло новое и сильное.

Вот так будет и теперь.

Послесловие

Дорогие собеседники! Наши с вами беседы окончены. И автору хочется сказать еще несколько слов, если угодно — в свое оправдание.

Прежде всего — извиниться за то, что был я временами груб. Начиная с самой первой страницы, где исключил непрофессионалов из числа собеседников. Но что же поделаешь — ведь эти беседы и правда бесполезны как легкое чтение. И мне очень бы не хотелось, чтобы бесформульная «литература» вроде 11-й и 15-й бесед послужила материалом интеллигентных умствований для *околонаучных* мальчиков и девочек, вроде того как последнее трудное стихотворение модного поэта служит материалом художественных умствований для мальчиков и девочек *окололитературных*. Для тех, кому неинтересны беседы 3-я и 14-я, бесполезны 11-я и 15-я.

Затем — о стиле изложения. Я был почти всюду предельно категоричен. Это — всего лишь литературный прием, сокращающий длину рассуждения. Причем дело даже не в экономии бумаги — так отчетливее формулируется мысль автора. И собеседник быстрее может решить, хочет он согласиться или не согласиться с этой мыслью.

А вот нарочито ненаучный, «вольный» способ изложения проведен всюду совершенно сознательно. Я и правда думаю, что обычный, чуть высокопарный, способ изложения научных вопросов с соблюдением всех оговорок, со строгой индукцией и т. д. и т. п. не только *ничего не дает читателю*, но и *затрудняет* внутреннее, неформальное понимание вопроса. Один раз мой учитель Дмитрий Евгеньевич Меньшов сказал: «Написание и чтение математических статей состоит в том, что автор упаковывает, а читатель распаковывает мысли...».

Эти слова Д. Е. Меньшова кажутся автору глубоко проникающими в самую суть дела. И вольный, неточный способ изложения — это именно попытка хоть немножко избавить собеседника от труда по распаковке.

Книга эта написана «единым духом». Но мысли, в ней изложенные, накопились у автора за почти 20 лет работы, сперва в области вычислительной математики, а потом — в области машин и программирования. И если я позволял себе иной раз гротеск, то не ошибитесь: прав я или не прав, но все, что сказано, сказано в здравом уме и твердой памяти. За всякое слово и всякое утверждение я отвечаю целиком и полностью. И решалось все это не днями, а годами.

Ну, и последнее. О людях, упоминающихся в этой книге — с фамилиями и без.

За долгие годы работы мне привелось встречаться, сотрудничать, а иногда и сталкиваться, спорить и воевать с большим числом очень разных людей нескольких специальностей.

Среди них встречались люди истинно талантливые, желавшие добра всякому доброму делу и помогавшие ему. Иногда это были люди с именами, известными всему научному миру. А иногда простые рабочие, как например механик Павел Ермилов с Московского завода САМ, техники Виктор Виноградов и Николай Авдеев, легендарный такелажник Серафим Герасимович, начальник отдела снабжения, ныне покойный Петр Борисович Хейфец или перфораторщица Шура Ганюшкина. Все они были чем-то очень похожи. Наверное, добрым (хотя и вовсе иной раз *не мягким*) своим отношением к нужному делу.

Но встречались мне и иные люди — нерадивые, неталантливые, подчас вовсе недобросовестные. Бывали среди них и простые лаборанты — от этих вред невелик. Но иногда такие люди занимали должности даже и довольно высокие. И тогда — беда. Потому что всех таких людей тоже объединяет нечто общее. И — летят десятки тысяч, сотни тысяч, а то и многие десятки миллионов, коту под хвост. И выпускается большая серия, скажем, негодных ЭВМ с фиксированной запятой.

И вот поэтому-то мне пришлось, именно пришлось, а не только захотелось, говорить и о Непосредственном Начальнике и других ему подобных.

Так как очень уж дорого приходится расплачиваться стране за их, этих людей, неталантливость и несоответствие занимаемой должности. А что до хороших, талантливых и работящих людей, то писал я о тех, с кем встречался. И поэтому не случайно такой большой процент из упомянутых добром в книге — это мои хорошие знакомые и товарищи по работе. Еще, кстати, и потому, что встретив интересного, яркого и добросовестного человека, поневоле стараешься свести с ним знакомство...

Но автор твердо верит, что и среди незнакомых ему людей много, очень много ходит Бессоновых и Шитиковых, Хейфецов и Ганюшкиных. И что их будет в нашем обществе становиться с каждым годом все больше и больше. А нерадивых лаборанток и, главное, Непосредственных Начальников — все меньше и меньше.

И эта вера в минуту жизни трудную очень помогает автору.

Литература

0. *Брудно А. Л.* Введение в программирование. Изд. «Наука», 1965.
1. Библиотека Б-61. Ч. 1. Описание и инструкции. Ч. II. Программы. Препринт Института теоретической и экспериментальной физики, 1965.
2. *Розенфельд М. З.* О совершенно стандартных программах. Доклады Академии наук СССР, т. 154, № 5, 1964.
3. *Адельсон-Вельский Г. М., Брудно А. Л., Кронрод А. С. и Резниковский П. Т.* О системе команд для трехадресной машины без регистра адреса. Доклады Академии наук СССР, т. 154, № 3, 1964.
4. *Кронрод В. А.* Крестики-нолики. «Проблемы кибернетики», вып. 19 (в печати).
5. *Брудно А. Л. и Ландау И. Я.* Одномастка (программирование игровой задачи). «Проблемы кибернетики», вып. 13, 1965.
6. *Адельсон-Вельский Г. М. и Ландис Е. М.* Один алгоритм организации информации. Доклады Академии наук СССР, т. 146, № 2, 1962.
7. Итоги всесоюзной переписи населения 1959 года. Госстатиздат, 1962.
8. *Newell A., Shaw J. C. and Simon H. A.* Report on a general problem-solving programming. «Proc. Intern. Conf. on Information Processing» UNESKO, Paris, 1960. См. также: *Ньюэлл А., Шоу Д., Саймон Г.* Разновидность интеллектуального обучения «вычислителя для решения задач общего типа». Сб.: «Самоорганизующиеся системы», М., Изд. «Мир», 1964.
9. *Адельсон-Вельский Г. М., Арлазаров В. Л. и Усков А. В.* Шахматная программа. Reviews of Geophysics, 1965.
10. *Крылов А. Н.* Лекции о приближенных вычислениях. Гостехиздат, 1959.
11. *Кронрод А. С.* Узлы и веса квадратурных формул. Изд. «Наука», 1965.
12. *Пукк Р. А.* Исследование алгоритмов оптимизации числа узлов квадратурных формул при заданной точности квадратуры. См.[20], т. 5, 1965.
13. *Пукк Р. А.* Алгоритм интегрирования функции одного переменного с заданной точностью, экономящий число обращений к подынтегральной функции. Послано в [20].
14. *Пукк Р. А.* Алгоритмы кратного интегрирования. Там же.
15. Ветхий и новый завет. Издание Московской патриархии, 1963.
16. *Ярославский Е.* Занимательная библия. Госполитиздат, 1963.
17. *Брудно А. Л.* Грани и оценки. «Проблемы кибернетики», вып. 10, 1963.
18. *Кронрод В. А.* Библиотека Б-65 для машины БЭСМ-6. Препринт Института теоретической и экспериментальной физики, 1965.
19. *Арлазаров В. Л., Кронрод А. С., Кронрод В. А.* О новом типе вычислительных машин. Доклады Академии наук СССР (в печати).
20. Журнал вычислительной математики и математической физики. АН СССР. Изд. «Наука».
21. *Адельсон-Вельский Г. М. и Филлер Ф. М.* Программа вычисления сетевых графиков. См. [20], т. 5, № 1.

Адельсон-Вельский Г. М., Арлазаров В. Л.,
Леферов Е. В., Резниковский П. Т., Хабарова С. А.

Приложение 1

Программа *А*

Существующая на машине М-20 восьмеричная печать очень неудобна для выдачи команд, ибо все восьмеричные цифры располагаются подряд, без интервалов между частями команды. Интервалы можно образовать, используя для выдачи команд десятичную печать. Однако для этого приходится программным путем превращать триады командной ячейки в тетрады и вставлять дополнительные тетрады, дающие интервалы между частями команд.

Программа *А* предназначена для подготовки к десятичной печати двух младших адресов командной ячейки, предварительно пересланной в ячейку $\alpha_{\text{печ}}$.

В программе *А* адреса разбиваются на триады, перед каждой триадой ставится одноразрядный нуль; кроме того, между тетрадами, соответствующими среднему и правому адресам, вставляется дополнительная тетрада, дающая интервал при десятичной печати; такая же тетрада ставится в коде. Результат помещается в ячейку $\gamma_{\text{печ}}$, выход из программы осуществляется через стандартную ячейку Ω.

Идея создания программы *А* и ее первоначальный вариант (длиной в 13 ячеек) принадлежат П. Т. Резниковскому. Позже Г. М. Адельсон-Вельский и В. Л. Арлазаров составили программу длиной в 9 ячеек. Наконец, Е. В. Леферову удалось написать программу *А* длиной в 8 команд. С. А. Хабарова написала программу *А* длиной в 9 ячеек. В отличие от предыдущих, эта программа реализует основную идею очень наглядно и не использует регистра адреса, что позволяет перенести ее на другие трехадресные машины.

Кроме того, ее программа имеет самый короткий цикл и идет в полтора раза быстрее других вариантов программы *А*.

Все четыре варианта программы приводятся ниже в содержательных обозначениях, а первый вариант и в закодированном виде.

Во всех вариантах триады обрабатываются, начиная с самой правой, подряд, но алгоритмы этой обработки разные.

ВАРИАНТ I

(1)	$0 \quad = \gamma_{\text{печ}}$	1000	075	0	0	0026
(2)	$(0,0,7) \quad = R1$	01	075	0	1014	0011
(3)	$\text{PA}1 \quad 0^* \quad 0 \quad \Omega - 1$	02	452	0	0	0006
(4)	$\alpha_{\text{печ}} \wedge R1 \quad = R2$	03	055	0025	0011	0012
(5)	$\overline{100}^* \rightarrow, \ R2 \quad = R2$	04	414	0100	0012	0012
(6)	$\gamma_{\text{печ}} \vee R2 \quad = \gamma_{\text{печ}}$	05	075	0026	0012	0026
(7)	$\overline{103} \rightarrow, \ R1 \quad = R1$	06	014	0103	0011	0011
(8)	$\text{PA} < \bar{3} \quad (4) \quad \bar{1}^*$	07	112	0003	1003	0001
(9)	$\text{PA} < \overline{10} \quad (8) \quad 0^*$	1010	112	0010	1007	0
(10)	$(\omega = 0) \quad (6) \quad (4) \quad \Omega - 2$	11	076	1005	1003	0005
(11)	$(12) \quad \Omega - 2 \quad R2$	12	056	1013	0005	0012
(12)	017; 0, 0360, 0	13	017	0	0360	0
(13)	(0, 0, 7)	14	0	0	0	0007

При помощи первых трех команд восстанавливаются начальные состояния $\gamma_{\text{печ}}$, высекателя $R1$, РА и запоминается старое значение РА. Затем последовательно из ячейки $\alpha_{\text{печ}}$ высекаются триады, сдвигаются на $0, 1, 2 \ldots$ разрядов влево и складываются (команды $(4) \div (8)$). Увеличение РА до 0010 (команды (8), (9)) приводит к дополнительному сдвигу всех триад среднего адреса на 4 разряда, т. е. к образованию нулевой триады между средним и правым адресами.

После выхода из цикла (команда (10)) постоянная команда (11) и формируемая в $\Omega - 2$ команда обеспечивают занесение в $\gamma_{\text{печ}}$ тетрад, дающих при десятичной печати нужные интервалы.

ВАРИАНТ II

(1)	$\alpha_{\text{печ}} = \gamma_{\text{печ}}$
(2)	$\overline{103}^{*} \rightarrow,\ (F, F, F) = R1$
(3)	PA1 0^{*} $\overline{7775}$ $\Omega - 1$
(4)	$\gamma_{\text{печ}} \wedge R1 = R2$
(5)	$R2+, \gamma_{\text{печ}} = \gamma_{\text{печ}}$
(6)	PA $< \overline{16}$ (5) $\overline{1}^{*}$
(7)	$\overline{104} \rightarrow,\ R1 = R1$
(8)	$(\omega = 0)$ (5) (4) $\Omega - 2$
(9)	$\overline{0340}$ $\Omega - 2$ $R2$

Первые три команды программы играют ту же роль, что и в варианте I (через (F, F, F) обозначена библиотечная константа, имеющая во всех разрядах мантиссы единицы). Основой алгоритма перевода триад в тетрады является высечение старшей части ячейки $\gamma_{\text{печ}}$ и удвоение ее; при этом после очередного удвоения высекатель сдвигается на 4 разряда влево (команды (4), (5), (7)). Внутренний цикл из команд (5), (6) приводит, вместо удвоения старших разрядов, к увеличению в 16 раз (к сдвигу на 4 разряда); так образуется нулевая триада между адресами. Команды (8) и (9) обеспечивают после выхода из цикла запись тетрад, дающих при десятичной печати интервал.

ВАРИАНТ III

(1)	РА1 0^* $\overline{3166}$ $\Omega - 1$
(2)	РА $\geqslant \overline{6200}$ (6) $\overline{1400}^*$
(3)	РА $< \overline{2000}$ $\Omega - 1$ $\overline{7775}^*$
(4)	$0^* \rightarrow \alpha_{\text{печ}}$ $= R1$
(5)	$\overline{072} \rightarrow R1$ $= R1$
(6)	$\overline{074} \rightarrow \gamma_{\text{печ}}$ $= \gamma_{\text{печ}}$
(7)	$\gamma_{\text{печ}}, +R1$ $= \gamma_{\text{печ}}$
(8)	0 (2) $R1$

Рабочий цикл программы состоит из команд (4)÷(8). Очередная триада двумя сдвигами (команды (4), (5)) помещается в младшие разряды КОПа рабочей ячейки $R1$, причем в остальных разрядах КОПа оказываются нули. Ячейка $\gamma_{\text{печ}}$ сдвигается вправо на 4 разряда, освобождая место для очередной тетрады, затем ей сложением КОПов присваивается КОП $R1$ (команды (6), (7)). При этом содержание мантиссы $R1$ несущественно.

Через каждые 4 раза часть рабочего цикла, формирующая КОП из очередной триады, обходится (команда (2)). При этом в $R1$ лежит (016; 0, 0, 0), что обеспечивает при печати интервал в нужном месте. И указанный обход, и выбор нужной триады осуществлены по РА. Это оказалось возможным по той причине, что в М-20 сдвиг по адресу управляется лишь семью младшими разрядами левого исполнительного адреса, а изменение РА, управляющее обходом (команда (2)), не затрагивает эти разряды, так как приращение РА равно 1 400. Выход осуществлен по РА, управляющему обходом (команда (3)), а начальное значение РА, управляющее сдвигом $\alpha_{\text{печ}}$ (команда (1)), выбрано так, чтобы в первых трех циклах команда (6) чистила КОП $\gamma_{\text{печ}}$.

ВАРИАНТ IV

(1)	$\overline{103} \rightarrow (F, F, F) = R2$
(2)	$\alpha_{\text{печ}} \wedge (0, F, 0) \quad = \gamma_{\text{печ}}$
(3)	$\overline{103} \rightarrow \gamma_{\text{печ}} \quad = \gamma_{\text{печ}}$
(4)	$\alpha_{\text{печ}} \wedge (0, 0, F) \quad = R1$
(5)	$R1+, \gamma_{\text{печ}} \quad = \gamma_{\text{печ}}$
(6)	$R2 \wedge \gamma_{\text{печ}} \quad = R1$
(7)	$\overline{104} \rightarrow R2 \quad = R2$
(8)	$(\omega = 0) \quad (5) \quad (5) \quad \Omega - 1$
(9)	$\overline{0340} \quad \Omega - 1 \quad R1$

Команда (1) восстанавливает высекатель $R2$. Команды (2)÷(4) сдвигают 2-й адрес ячейки $\alpha_{\text{печ}}$ на 3 разряда влево и одновременно восстанавливают начальное состояние ячейки $\gamma_{\text{печ}}$.

Рабочий цикл состоит из команд (5)÷(8). Как необработанные триады, так и готовые тетрады лежат подряд в ячейке $\gamma_{\text{печ}}$.

Команда (5) сдвигает триады на разряд влево, оставляя тетрады на месте. При этом образуется очередная тетрада. Команда (6) обеспечивает работу команды (5) в следующем цикле, высекая из $\gamma_{\text{печ}}$ еще не обработанные триады.

Команда (7) сдвигает высекатель на 4 разряда влево, причем когда он становится нулем, зацикливающая команда (8) передает управление на выход из цикла. Кроме того, команда (8) засылает команду (5) в стандартный предконец $\Omega - 1$.

Команда (9) совмещает выход из блока на ячейку $\Omega - 1$ с засылкой в ячейку $R1$ слова (016; 0, 0340, 0). Команда (5), стоящая в ячейке $\Omega - 1$, обеспечивает в $\gamma_{\text{печ}}$ пропуски в коде и между адресами.

Программа A явилась существенной составной частью библиотечной программы «Печать программы», которая организует выдачу

из оперативной памяти на печать массива командных ячеек, причем каждая команда выдается парой строк. В начале первой строки пары ставится последний восьмеричный разряд адреса команды, в перед каждым восьмеричным десятком команд выдается адрес начала этого десятка.

Приложение 2

Система команд ЭВМ М-20

Большая часть примеров программ, приведенных в этой книге, написана для ЭВМ М-20. Ниже представлена таблица, содержащая список команд машины М-20. Отметим некоторые особенности этих команд.

Все команды являются трехадресными. Каждая из них занимает 45-разрядную ячейку памяти. 36 младших разрядов ячейки отводится под три 12-разрядных адреса. Шесть разрядов отводится под код команды (КОП). Три старших разряда ячейки являются признаками модификации адресов. Если в каком-либо из этих разрядов находится 1, то соответствующий адрес модифицируется, то есть к нему добавляется по модулю 2^{12} содержимое специально выделенного 12-разрядного регистра адреса. Существуют команды, позволяющие изменять этот регистр адреса. Одна из них заносит в регистр адреса 2-й адрес команды, а другая — 2-й адрес содержимого ячейки, адрес которой указан во 2-м адресе команды. В книге эти команды обозначаются, соответственно, как РА1 и РА2.

Если при выполнении какой-либо команды М-20 происходит запись в некоторую ячейку, то адрес этой ячейки всегда находится в 3-м адресе команды.

Команды передачи управления всегда передают управление по 2-му адресу.

Операции над числами (p, q, r, – порядки чисел X, Y, Z по A_1, A_2, A_3)

КОП	Название операции	Модификации кодов: 60	6Н	60 и 6Н	ω = 1	АВОСТ	Время*
01	Сложение	21	41	61	z<0	z>177₈	28,5
02	Вычитание	22	42	62	z<0	z>177₈	28,5
03	Вычитание модулей	23	43	63	z<0	z>177₈	28,5
04	Деление	24	—	—	r>100₈	x≥2y, y=0, z>177₈	136,5
05	Умножение	25	45	65	r>100₈	z>177₈	69,5
44	Извлечение кв. корня	64	—	—	r>100₈	z>177₈, X<0	275
47	Вывод младших разрядов произведения					z>177₈	

Изменение порядка

КОП	Название операции		Результат	ω = 1	АВОСТ	Время*
06	Сложение или вычитание	адреса	$z = q + (A_1^{(r)} - 100)$	r>100₈	z>177₈	61,5
26		порядка	$z = q + (p - 100)$	r>100₈	z>177₈	24,0
46		адреса	$z = q - (A_1^{(r)} - 100)$	r>100₈	z>177₈	61,5
66		порядка	$z = q - (p - 100)$	r>100₈	z>177₈	24,0

Изменение [РА]

КОП	Код, заносимый в РА	Код, заносимый в A_2'	Время*
52	A_2'	052; 0000, A_1', 0000	28,5
72	2-й адрес кода по A_2'	052; 0000, A_1', 0000	28,5

Останов машины

КОП	Дополнительные действия при останове	Время*
17	На Р1 и 02 ПУ выдаются коды по A_1' и A_2', вместо кода по A_3'	24
37		24
57		24
77		24

Команды ввода

10	Ввод с остановом при несовпадении сумм	A_1' – № ячейки, с которой начинается запись вводимых кодов A_2' – адрес, по которому передается управление при несовпадении сумм A_3' – адрес суммы
30	Ввод без останова при несовпадении сумм	

Логические операции

КОП	Название операции	Результат ξ, η, ζ-значения кодов соб. поп. A_1', A_2'	ω=1	Время*
15	Сравнение	$\zeta_k = \lvert \xi_k - \eta_k \rvert$	$\zeta=0$	24
35	Сравнение с остановом	То же, что и 15, но останов при несовпадении кодов	$\zeta=0$	24
55	Логическое умножение	$\zeta_k = \xi_k \wedge \eta_k$ $(\xi_k \wedge 0 = 0,\ \xi_k \wedge 1 = \xi_k)$	$\zeta=0$	24
75	Логическое сложение	$\zeta_k = \xi_k \vee \eta_k$ $(\xi_k \vee 0 = \xi_k,\ \xi_k \vee 1 = 1)$	$\zeta=0$	24

Специальные операции над кодами

КОП	Название операции	Результаты	ω=1	Время*
07	Циклическое сложение	$\zeta' = (\xi' + \eta') \bmod (2^9-1)$ $\zeta'' = (\xi'' + \eta'') \bmod (2^{36}-1)$	$\zeta'' \geq 2^{35}$	24
27	Циклическое вычитание	$\zeta' = (\xi' - \eta') \bmod (2^9-1)$ $\zeta'' = (\xi'' - \eta'') \bmod (2^{36}-1)$	$\xi'' < \eta''$	24
13	Сложение мантисс	$\zeta' = \xi'$ $\zeta'' = (\xi'' + \eta'') \bmod 2^{36}$	$\zeta'' \geq 2^{36}$	24
33	Вычитание мантисс	$\zeta' = \xi''$ $\zeta'' = (\xi'' - \eta'') \bmod 2^{36}$	$\xi'' < \eta''$	24
53	Сложение КОП об	$\zeta_1' = (\xi' + \eta') \bmod 2^9$ $\zeta_1'' = \xi''$	$\zeta_1 \geq 2^9$	24
73	Вычитание КОП об	$\zeta_1' = (\xi' - \eta') \bmod 2^9$ $\zeta_1'' = \xi''$	$\xi' < \eta'$	24
00	Засылка кодов	$[A_1']$ засылается в A_3' A_2' - безразличен	—	24
20	Засылка кода с ОПУ	[РПУ] — A_3', A_1' задающих разряда определяют номер РПУ	—	24
40	Засылка нуля в A_3'	$0 \to A_3'$ A_1' и A_2' - безразличны	—	24
60				
67	Циклический сдвиг	$\zeta_i = \xi_{(i+24) \bmod 48}$ A_2' - безразличен	—	24

Операции сдвига

КОП	Название операции		Величина сдвига	ω=1	Время*
54	Сдвиг кода по	адресу	$\zeta_{k+S} = \eta_k$; $S = A_1^{(7)} - 100$ $A_1^{(7)}$ - семь младших разрядов A_1'	$\zeta=0$	61,5 + 1,5S
74		порядку	$\zeta_{k+S} = \eta_k$; $S = p - 100$	$\zeta=0$	24 + 1,5S
14	Сдвиг мантиссы кода по	адресу	$\zeta_{k+S} = \eta_k$; $S = A_1^{(7)} - 100$	$\zeta''=0$	61,5 + 1,5S
34		порядку	$\zeta_{k+S} = \eta_k$; $S = p - 100$	$\zeta''=0$	24 + 1,5S

Команды управления с изменением [РА]

КОП	Код в РА	Условие передачи управления в A'_2	Условие передачи управления дальше	* Время
11	A'_3	РА < A'_1 и Ω=1	РА ≥ A'_1 или Ω=0	24
31	A'_3	РА ≥ A'_1 и Ω=1	РА < A'_1 или Ω=0	24
51	A'_3	РА < A'_1 и Ω=0	РА ≥ A'_1 или Ω=1	24
71	A'_3	РА ≥ A'_1 и Ω=0	РА < A'_1 или Ω=1	24
12	A'_3	РА < A'_1	РА ≥ A'_1	24
32	A'_3	РА ≥ A'_1	РА < A'_1	24

Команды управления с занесением в A'_3

КОП	Код, заносимый в A'_3	Условие передачи упр. в A'_2	дальше	* Время
16	016; 0000, A'_1, 0000	Всегда	—	24
36	[A'_1]	Ω=1	Ω=0	24
56	[A'_1]	Всегда	—	24
76	[A'_1]	Ω=0	Ω=1	24

Команды обращения к МЗУ

Обозн. опер.	КОП	A'_1	A'_2	A'_3	* Время
М(а)	50	УЧ	А мзу	Ω мзу	24
М(б)	70	α мзу	АП	А Σ	24

Таблица значений разрядов УЧ

№ разряда	УЧ	Операции
1,2	0000 0003	номер МЛ или МБ
3	0004 0004	чтение или запись
4	0010	Барабан
5	0020	Лента
6	0040	разметка ленты
7	0100	Печать
8	0200	Перфорация
9	0400	Блокировка останова
10	1000	Обратное направление
11	2000	Блокировка контроля
12	4000	Блокировка МЗУ

* Время в мксек.

Январь 1960г.

Арлазаров В. Л.

Несколько слов об авторе

Александр Семенович Кронрод (1921–1986) — человек примечательный. Со студенческой скамьи он ушел добровольцем на фронт. Через три года, после тяжелого ранения, вернулся, сложил в стол награды и снова стал студентом. А еще через пять лет — он уже доктор наук, создатель теории функций двух переменных, лауреат Сталинской премии, руководитель (хочется сказать — предводитель) семинара молодых, честолюбивых, талантливых ученых.

И вдруг... (хотя ничего у него, конечно, не вдруг. Просто характер, талант и вера в себя), вдруг он кладет в стол уже готовые к публикации математические работы и полностью переключается на программирование на ЭВМ. Кстати, через двадцать лет А. С. Кронрод точно так же покончит с программированием и займется лечением рака. Стоит ли удивляться, но опять — с успехом.

Итак, вычислительные машины. Он знает, что за ними будущее. Он создает настоящий конвейер решения физических задач. Он придумывает, как сделать работу математика за пультом удобной, какие смены должны быть у операторов, как добиться безошибочной кодировки и набивки программ — тысяча мелочей, которые отнюдь не кажутся ему мелочами. Следы этого вы, наверное, заметили при чтении книги.

Но его настоящая цель — создание машин, решающих сложные, «человеческие» задачи. И Кронрод организует новый семинар, где занимаются программированием игр, задачами узнавания, придумываются новые вычислительные и невычислительные алгоритмы. С ним рядом друзья и соратники. Я не называю их, Кронрод никогда не забывает сделать это сам, обычно слегка притушевывая собственную роль. Однако именно благодаря А. С. Кронроду этот семинар вырос в целую школу программирования, и множество ученых успешно представляют ее и в Европе, и в Америке, и даже в Австралии.

Главной чертой школы Кронрода являются методы решения задачи, при которых на каждом этапе четко просматривается связь с ее физическим смыслом. Ибо тогда легче следить за тем, что выбранный метод ведет к цели, уверенно вести отладку, да и, вообще, эпиграф к каждой главе этой книги есть жизненное кредо автора.

В этом смысле «Беседы» и сегодня прекрасный учебник для знакомства с некоторыми идеями искусственного интеллекта и «интеллектуальными» подходами к вычислительным проблемам.

Эта книга написана почти сорок лет назад. Разумеется, сегодня что-то в ней покажется банальностью, а что-то глупостью. Автор не виноват: сорок лет назад банальностей не было. Зато есть и плюсы: за сорок лет глупостей стало заметно меньше.

Вы обратили внимание, с чего начинается книга? Конечно, сейчас все программируют «структурно». Но Кронрод излагает это как очевидный способ написания программ за много лет до «открытия» этого метода Дейкстрой. И ему ясно, что сделать программу понятной можно только на плоскости, рисуя не только команды и блоки, но и связи между ними. Ах, как не хватает ему сегодняшних цветных графических мониторов, хотя бы для того, чтобы рисовать стрелки! Он-то всегда программирует визуально, да вот только на листах бумаги. Отсюда и его неприятие АЛГОЛА, явно тяготеющего к тому, чтобы вытянуть программу в линейку.

Еще пример. Помните, зачем программисту барабан? Тривиальность? Но этот вопрос был предметом насмешек многих рецензентов. Да что говорить, еще в 1990 году очень серьезный специалист по трансляторам, ныне уважаемый профессор, всерьез доказывал, что отладочная информация не должна храниться в рабочее время, так как отнимает память. Понадобилась мощь и «наглость» Микрософта, чтобы все привыкли, что эта информация абсолютно необходима.

Впрочем, я думаю, что и еще через 40 лет можно будет наслаждаться и пониманием программирования и неординарностью мышления, блестками истинной мудрости щедро разбросанными по страницам этой книги.

Оглавление

Александр Семёнович Кронрод (1921–1986)

Эта книга была написана замечательным ученым-математиком А.С.Кронродом почти 40 лет тому назад, но публикуется впервые. Один из зачинателей программирования у нас в стране и основателей целой школы программирования, А.С.Кронрод в доступной и интересной форме обсуждает вопросы организации системного программирования, отладки программ и архитектуры ЭВМ. Значительная часть книги посвящена различным задачам искусственного интеллекта и использованию «интеллектуальных» подходов для решения вычислительных задач.

2211 ID 17761
9 785354 005659 >

ИЗДАТЕЛЬСТВО УРСС
НАУЧНОЙ И УЧЕБНОЙ ЛИТЕРАТУРЫ
E-mail: URSS@URSS.ru
Каталог изданий
в *Internet:* http://URSS.ru
Тел./факс: 7 (095) 135–44–23
Тел./факс: 7 (095) 135–42–46