АВТОМАТИЗАЦИЯ ГЕНЕРАЦИИ СЛУЧАЙНЫХ ТЕСТОВ ДЛЯ ОЛИМПИАДНЫХ ЗАДАЧ ПО ПРОГРАММИРОВАНИЮ

В.Е. Аксенов, М.В. Буздалов
Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики
E-mail: aksenov@rain.ifmo.ru

В наше время продолжает набирать популярность спортивное программирование [1]. За последние несколько лет было создано множество проектов для упрощения проведения соревнований, такие как система проведения соревнований *PCMS2* и система подготовки задач *Polygon* [2]. Однако, большинство подобных проектов лишь помогает проводить соревнования и немного структурирует разработку задачи, в то время как сам процесс разработки задачи все еще проводится вручную.

В процессе разработки олимпиадной задачи, помимо условия и примера решения, как правило, требуется подготовить набор тестов, программы для проверки корректности тестов и ответов. Среди перечисленных задач подготовка тестов считается одной из самых сложных.

В большинстве случаев, набор тестов создается при помощи программы – генератора тестов. На реализацию такой программы часто уходит много времени. В данной работе описывается утилита, разработанная для облегчения и ускорения реализации генераторов тестов. Данная утилита обеспечивает генерацию множества структур данных, широко использующихся в тестах для олимпиадных задач, а также учитывает ограничения в условиях задачи.

Мотивация

Существует несколько типов тестов, используемых в олимпиадных задачах:

- Тесты, проверяющие крайние случаи. Из них следует особо отметить «минимальные» тесты, которые обычно реализуются вручную.
- Тесты производительности. Как правило, это тесты, размер и сложность которых приближается к максимально возможным для данной задачи.
- «Случайные» тесты. Их цель – с некоторой вероятностью найти такие ошибки в решениях, возможность которых не была предусмотрена жюри. К ним относятся, в частности, различные ошибки реализации.

Многие из описанных тестов генерируются по заданным шаблонам, при этом определенные параметры этих тестов задаются случайным образом. Для реализации генератора тестов часто требуется писать процедуры генерации определенных структур данных с использованием генератора случайных чисел. В различных генераторах для этого зачастую используются схожие, если не одинаковые, фрагменты кода. По этой причине создание утилиты, берущей на себя рутинную работу такого рода, является актуальным.

Формат описания

Для генерации одного или нескольких тестов разработанная утилита использует файл, в котором описывается вид генерируемых тестов.

Утилита имеет ряд встроенных структур данных, которые чаще всего встречаются в тестах для олимпиадных задач:

- целые числа (Integer);
- массивы (Array);
- символы (Char);
- строки (String);
- многоугольники (Polygon);
- графы (Graph).

Также поддерживается создание своих структур данных на основе уже имеющихся.

Рассмотрим подробнее имеющиеся конструкторы структур данных:

- *Integer*([L_1..R_1], …, [L_N..R_N]) – генерируется случайное целое число, принадлежащее хотя бы одному из данных диапазонов.
- *Array*(*length*, *Type*[, *options*]*) – генерируется массив случайных элементов типа *Type* длины *length*. На данный момент поддерживаются следующие опции: *sort*, означающая, что массив будет отсортирован, и *splitter*=”<text>”, означающая, что при выводе в файл элементы массива будут разделены данной строкой. По умолчанию, разделителем элементов массива является пробел. Опции возможно комбинировать, перечисляя их через запятую.
- *Character*([L_1..R_1], …, [L_N..R_N]) – генерируется случайный символ, принадлежащий хотя бы одному из данных диапазонов.
- *String*(*length*, (*characters*)[, *options*]*) – генерируется строка длины *length*, содержащая случайные символы, описываемые (по аналогии с конструктором *Character*) выражением *characters*. Данный конструктор поддерживает одну опцию *palindrom*, позволяющую генерировать строку-палиндром.
- *Polygon*(*N*, X_L, Y_L, X_R, Y_R [, *options*]*) – генерируется многоугольник из *N* вершин, содержащийся в прямоугольнике [(X_L, Y_L), (X_R, Y_R)]. Опция *convex* позволяет генерировать выпуклый многоугольник.

- *Graph*(*N*, [*tree*|*tournament*]) – генерируется случайный граф из *N* вершин определенного вида – дерево или турнир.
- *Graph*(*N*, *M*, [, *options*]*) – генерируется случайный граф, содержащий *N* вершин и *M* ребер. Дополнительные опции: *directed* – генерируется ориентированный граф, *connected* – генерируется связный (неориентированный) граф, *strongly-connected* – генерируется сильно связный (ориентированный) граф, *forest* – генерируется набор деревьев («лес»). Опция *matrix* определяет то, что граф будет выводиться в тестовый файл в виде матрицы смежности. По умолчанию, граф выводится в виде списка ребер.
- Описание новых структур данных имеет следующий вид:

```
Type <name> {
      <описание полей>
      print:
          <описание строкового представления>
}
```

В описании теста возможно использование переменных, причем значения переменных могут быть использованы дальше по тексту описания в параметрах конструкторов.

Вывод структуры данных осуществляется оператором *print* с одним аргументом – переменной, содержащей требуемую структуру. Все встроенные структуры, кроме графов, имеют единственный формат вывода, для графов этот формат регулируется описанными выше опциями.

Файл, описывающий процесс генерации теста, состоит из одного или более объявления переменных, одного или более операторов *print* и, опционально, описаний структур. Описания могут следовать в произвольном порядке, однако, не разрешается использовать переменные и структуры данных, не определенные ранее.

Пример применения

Пример описания тестов приведен в лист. 1. Предположим, что это описание сохранено в файле с именем `test`. Следующая команда сгенерирует 10 случайных тестов с именами 01, 02, ..., 10 по данному описанию:

```
java -jar generator.jar test 1 10
```

Пример сгенерированного теста приведен в лист. 2.

Функциональность была проверена на задачах соревнования NEERC-2011 [3]. При этом сокращался объем кода, написанного для генерации тестов.

Заключение

В данной работе описана утилита, позволяющая достаточно быстро создавать случайные тесты для олимпиадных задач. Помимо генерации тестов, входящих в набор тестов для задачи, эта утилита также может быть использована при стресс-тестировании различных решений этой задачи на большом объеме тестовых данных.

Дальнейшая работа в этом направлении состоит в дальнейшем совершенствовании языка описания формата тестовых данных. При этом предполагается использовать предметно-ориентированную библиотеку на языке *Scala* [4], что позволит при схожей краткости описания использовать синтаксические структуры языка *Scala* и существующие библиотеки языков *Scala* и *Java*.

На данный момент, написанная утилита генерирует случайные тесты, не заботясь об их качестве. Для решения этой проблемы планируется использование эволюционных алгоритмов для улучшения тестов, генерируемых программой [5, 6]. При этом представление тестов и генетические операторы над ними возможно строить автоматически по тому же структурному описанию, что и сами тесты.

Листинг 1. Пример файла описания тестов

```
Type Pair {
    a = Integer([1..10])
    b = Integer([1..10])
    print:
        "This is the pair " + a + " and " + b
}
a = Integer([1..10],[20..30])
b = Integer([a..100])
c = Integer([a..b])
print a
print b
print c
array1 = Array(c, Integer([1..49]))
print array1
array2 = Array(c, Integer([1..49]), sort)
print array2
```

```
d = Pair()
print d
string = String(c, ([a..z]))
print string
string = String(c, (A,G,T,C))
print string
polygon = Polygon(c, 0, 100, 0, 100, convex)
print polygon
tree = Graph(10, tree)
print tree
graph = Graph(10, 15, connected)
print graph
```

Листинг 2. Пример сгенерированного файла.

```
2
88
7
41 24 27 24 11 44 34
12 16 20 41 44 47 48
This is the pair 7 and 9
nxgnuhd
GGTACTT
7
5 87
97 13
95 10
89 3
20 6
11 15
3 33

10
3 1
9 1
1 6
2 6
4 7
6 5
6 10
8 7
10 8

10 15
2 1
7 2
2 6
2 10
6 3
3 5
8 3
4 6
7 5
8 6
9 6
9 7
8 10
8 9
9 10
```

Литература

1. Шалыто А. А., Царев Ф. Н. Небывалая победа российской школы программирования // Компьютерные инструменты в образовании. 2009. № 2, с.3–5.
2. Система подготовки задач Polygon. http://codecenter.sgu.ru/polygon/
3. North-Eastern European Regional Contest. http://neerc.ifmo.ru/
4. The Scala Programming Language. http://www.scala-lang.org/
5. Буздалов М.В. Генерация тестов для олимпиадных задач по программированию с использованием генетических алгоритмов //Научно-технический вестник Санкт-Петербургского государственного университета информационных технологий, механики и оптики. 2011, №2(72), с. 72–77.
6. Буздалов М.В. Генерация тестов для олимпиадных задач по теории графов с использованием эволюционных стратегий //Научно-технический вестник Санкт-Петербургского государственного университета информационных технологий, механики и оптики. 2011, №6(76), с. 123–127.