Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

Кафедра «Компьютерные технологии»

В. Ю. Лоторейчик

Сжатие текстов по методу Зива-Лемпеля (LZ) на основе конечного автомата

Программирование с явным выделением состояний
Проектная документация

Санкт-Петербург
2005

Оглавление:

1. Введение…….3
2. Описание алгоритма……..4
3. Особенности реализации……..7
4. Эффективность метода……..8
5. Автомат, соответствующий программе……...9
6. Заключение……10
7. Источники……..11
8. Приложение. Исходный код программы…….12

Введение

Первоначально возможность сжатия текстов была интересна только специалистам по телеграфной и радиосвязи (включая очень рано появившиеся работы по экономной передаче телевизионных изображений). С появлением персональных компьютеров программы сжатия текстов вошли в обиход практически всех потребителей вычислительной техники, а в последних версиях операционных систем, например в MS DOS, начиная с версии 6.0, сжимающие программы становятся уже системной частью. Некоторые из схем сжатия информации запатентованы, и выпускаются реализующие их электронные устройства.

У истоков кодирования текстов стояли алгоритмы алфавитного кодирования (Шеннона-Фано, Хаффмена) и словарного кодирования (Зива-Лемпеля).

В данной работе реализована демонстрационная программа сжатия текстов по методу Зива-Лемпеля. Ядро алгоритма построено по **Switch-технологии**, разработанной и внедряемой профессором А.А. Шалыто.

Описание алгоритма

Рассмотрим для начала следующий способ кодирования.

1. Исходное сообщение по некоторому алгоритму разбивается на последовательности символов, называемые словами (слово может иметь одно или несколько вхождений в исходный текст сообщения).
2. Полученное множество слов считается буквами нового алфавита. Для этого алфавита строится разделимая схема алфавитного кодирования (равномерного кодирования или оптимального кодирования, если для каждого слова подсчитать число вхождений в текст). Полученная схема обычно называется словарем, так как она сопоставляет слову код.
3. Далее код сообщения строится как пара – код словаря и последовательность кодов слов из данного словаря.
4. При декодировании исходное сообщение восстанавливается путем замены кодов слов на слова из словаря.

Указанный способ допускает усовершенствования.

На практике используется следующая идея, которая известна также как адаптивное сжатие. За один проход по тексту одновременно динамически строится словарь и кодируется текст. При этом словарь не хранится – за счет того, что при декодировании используется тот же самый алгоритм построения словаря, словарь динамически восстанавливается.

Здесь приведена простейшая реализация этой идеи, известная как алгоритма Зива-Лемпеля. Вначале словарь D: **array[int] of string** содержит пустое слово**,** имеющее код 0. Далее в тексте последовательно выделяются слова. Выделяемое слово – это максимально длинное слово из уже имеющихся слов в словаре плюс еще один символ. В сжатое представление записывается найденный код слова и расширяющая буква, а словарь пополняется расширенной комбинацией.

Алгоритм 1. Упаковка по методу Зива-Лемпеля

Вход: исходный текст, заданный массивом кодов символов f : **array** [1..n] **of char.**

Выход: сжатый текст, представленный последовательностью пар <p, q>, где p – номер слова в словаре, q – код дополняющей буквы.

D[0] := “”; d := 0 { начальное состояние словаря }

k := 1 { номер текущей буквы в исходном тексте }

while k ≤ n **do**

p := FD(k) { p – индекс найденного слова в словаре}

l := Length(D[p]) { l – длина найденного слова в словаре}

yield <p, f[k + *l*]> { код найденного слова и еще одна буква }

d := d + 1; D[d] := D[p] ∪ f[k + *l*] { пополнение словаря, здесь ∪ – это конкатенация }

k := k + *l* + 1 { продвижение вперед по исходному тексту}

end while

Слово в словаре ищется с помощью несложной функции FD.

Вход: k – номер символа в исходном тексте, начиная с которого нужно искать в тексте слова из словаря.

Выход: p – индекс самого длинного слова в словаре, совпадающего с символами f[k]..f[k + *l*]. Если такого слова в словаре нет, то p = 0.

l := 0; p := 0 { начальное состояние }

for i **from** 1 **to** d **do**

if D[i] = f[k..k + Length(D[i]) - 1] & Length(D[i]) > *l* **then**

p := i; *l* := Length(D[i]) { нашли более подходящее слово }

end if

end for

return p

Распаковка осуществляется следующим алгоритмом.

Алгоритм 2. Распаковка.

Вход: сжатый текст, представленный массивом пар g : **array** [1..m] **of record** p: **int;** q**: char; endrecord,** где p – номер слова в словаре, q – код дополняющей буквы.

Выход: исходный текст, заданный последовательностью строк и символов.

D[0] := “”; d := 0 { начальное состояние словаря }

for k ≤ n **do**

p := g[k].p { p – индекс слова в словаре }

q := g[k].q { q – дополнительная буква }

yield p ∪ q { вывод слова и еще одной буквы }

d := d + 1; D[d] := D[p] ∪ q { пополнение словаря, здесь ∪ – это конкатенация}

end for

На практике применяют различные усовершенствования этой схемы.

Особенности реализации

В данной конкретной реализации алгоритма сжатия текстов по методу Зива-Лемпеля введены некоторые усовершенствования по сравнению с тем, что написано в описании алгоритма (см. выше).

1. Словарь реализован в виде «бора». Это позволяет выполнять поиск слов значительно быстрее. В классической реализации, которая описана выше, поиск осуществляется за время `O(n)`, где `n` – количество слов в словаре на текущий момент. В реализации с помощью «бора» поиск осуществляется за врем `O(d)`, где `d` – длина искомого слова. Есть разница, и еще какая. Вкратце о том что же такое «бор» (иногда говорят нагруженное дерево или луч). В англоязычной литературе trie от re*trie*val. «Бор» представляет собой дерево, где каждому ребру сопоставлен символ. Говорят, слово принадлежит «бору», если существует путь из корня такой, что на первом ребре написана первая буква слова, на втором вторая и т. д., а из вершины, в которую придем, ведет ребро с написанном на нем специальным символом, нигде не встречающимся в тексте.
2. Весь алгоритм реализован по **Switch – технологии.** Демонстрационная программа работает с явным выделением состояний. Это позволило написать всю алгоритмическую часть внутри одного оператора `switch`, вложенного в цикл `while`.

Эффективность метода

Будем считать, что максимальный размер словаря 65536 слов. Это соответствует двум байтам для кодирования слов из словаря. Это ограничение вполне естественно, так как если взять три байта на кодирование слов из словаря, то это явно избыточно и приведет к тому, что метод сжатия будет наоборот увеличивать размер исходного текста. Аналогично никакого положительного эффекта не добиться, если взять один байт для кодирования слов из словаря.

Для тестирования программы возьмем следующий текст:

ехали медведи на велосипеде а за ними кот задом на перед а за ним комарики на воздушном шарике
а за ними раки на хромой собаке волки на кобыле львы в автомобиле зайчики в трамвайчике жаба на
метле едут и смеются пряники жуют
ехали медведи на велосипеде а за ними кот задом на перед а за ним комарики на воздушном шарике
а за ними раки на хромой собаке волки на кобыле львы в автомобиле зайчики в трамвайчике жаба на
метле едут и смеются пряники жуют
ехали медведи на велосипеде а за ними кот задом на перед а за ним комарики на воздушном шарике
а за ними раки на хромой собаке волки на кобыле львы в автомобиле зайчики в трамвайчике жаба на
метле едут и смеются пряники жуют
ехали медведи на велосипеде а за ними кот задом на перед а за ним комарики на воздушном шарике
а за ними раки на хромой собаке волки на кобыле львы в автомобиле зайчики в трамвайчике жаба на
метле едут и смеются пряники жуют
ехали медведи на велосипеде а за ними кот задом на перед а за ним комарики на воздушном шарике
а за ними раки на хромой собаке волки на кобыле львы в автомобиле зайчики в трамвайчике жаба на
метле едут и смеются пряники жуют~

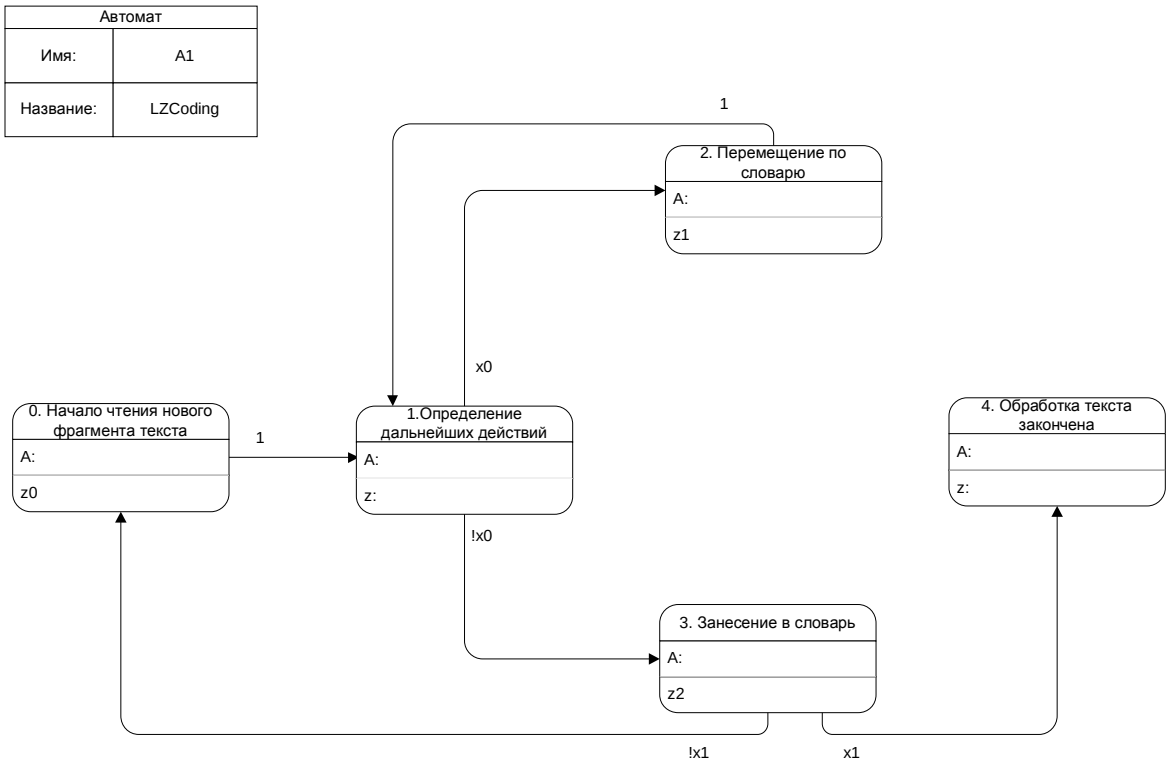
В тесте один и тот же фрагмент текста повторяется много раз. При прогоне этого текста через разработанную программу получаем коэффициент сжатия ***0.9920***. Это уже дает положительный эффект. Повторив тот же самый текст еще раз, получим коэффициент сжатия ***0.8093***.

Дальше интересней. Возьмем повесть Николая Васильевича Гоголя «Записки Сумасшедшего», прогоним её через программу, получим коэффициент сжатия ***0.7294.*** Однако на небольших текстах можно получить коэффициент сжатия больший единицы – текст при сжатии увеличился.

Таким образом, область применения разработанной программы – большие тексты. Однако стоит заметить, что программа чисто демонстрационная и сам сжатый файл она не создает.

Автомат, соответствующий программе

На рисунке приведен автомат, соответствующий программе.



z0	Переход в корень «бора»
z1	Переход к следующему символу текста и к потомку текущего элемента
z2	Занесение нового слова в словарь и вывод очередной пары (номер слова в словаре, символ)

x0	У текущего элемента есть потомок, соответствующий просматриваемому символу текста
x1	Конец текста

Заключение

В ходе работы были получены следующие результаты.

1. Написана программа, на базе которой можно писать программы компрессии и декомпрессии текстов с эффективностью сжатия порядка 70% (для русского языка). Причем эти программы будут сжимать тексты с достаточно большой скоростью (несколько минут на роман «Войну и мир»).
2. Продемонстрировано применение **Switch-**технологии для описания алгоритма сжатия.

Программа написана на *C++ .NET*, поэтому для её запуска необходимы соответствующие инструментарии.

Источники

1. *Шалыто А. А.* Switch-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998.
2. *Вельдер С. Э., Бедный Ю. Д.* Универсальный инфракрасный пульт для бытовой техники. СПб.: СПбГУ ИТМО. 2005. http://is.ifmo.ru/projects/irrc/
3. *Головешин А.* Использование конвертора Visio2Switch. 2002. http://is.ifmo.ru/progeny/visio2switch/
4. *Романовский И. В.* Дискретный анализ. СПб.: Невский Диалект, 2000.
5. *Новиков Ф. А.* Дискретная математика для программистов. СПб.: Питер, 2002.

Приложение. Исходный код программы

```
// Vladimir J. Lotoreichik 2005

#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

#using <mscorlib.dll>

#define MAX 65535
#define MAXLENGTH 1000000

using namespace System;


struct leave //Структура данных "Бор"
{
      leave* parent;
      leave* son[256];
      unsigned int num;
};

unsigned int gnum = 0;
int i = 0, n = 0;           //вспомогательные переменные
int length = 0;             //Длина текста
char ch = 0;                //Буфер для чтения символа
unsigned char s[MAXLENGTH]; //Строка с текстом
int state = 0;              //Номер состояния
leave* p = 0;               //указатель на текущий элемент Бора
leave* root;                //указатель на корень Бора

void z0(void)
{
   p = root;
}
void z1(void)
{
      p = p->son[s[i]];
      i++;
}

void z2(void)
{
      fprintf(f, " %d %d%c\n", gnum + 1, p->num, s[i]);
      printf(" %d %d%c\n", gnum + 1,  p->num, s[i]);
      if (gnum < MAX)
      {
            p->son[s[i]] = new leave;
            p->son[s[i]]->parent = p;
            for (int j = 0; j < 256; j++) p->son[s[i]]->son[j] = 0;
            p = p->son[s[i]];
            gnum++;
            p->num = gnum;
      }
      i++;
}

bool x0(void)
{
      return (p->son[s[i]] != 0);
}
```

```
bool x1(void)
{
	return (i > n);
}

int _tmain()
{
	//Инициалиазация "Бора"
	root = new leave;
	root->parent = 0;
	for (i = 0; i < 256; i++)
		root->son[i] = 0;
	root->num = 0;

	//Чтение из файла ------------------
	FILE* f = fopen("in.txt", "rt");

	i = 0;
	while (ch != '~')
	{
	  fscanf(f,"%c", &ch);
	  if ((ch != '~') && (ch != 13))
	  {
		if (ch == ' ') ch = '_';
		printf("%c", ch);
		s[i] = ch;
		i++;
		length++;
	  }


	}
	s[i] = 0;
	n = i;
	printf("\n");
	fclose(f);
	//----------------------------

	i = 0;

	int state = 0;
	leave* p = 0;
	int prev = 0;

	f = fopen("out.txt", "wt");

	fprintf(f, "DYNAMIC DICTIONARY \n");

	//Основной модуль

	while (state != 4)
	{
		switch (state)
		{

		case 0: z0(); state = 1;  break;

		case 1: if (x0()) state = 2;
				else state = 3;

				break;

		case 2:
```

```
                        z1();
                        state = 1;

                        break;

            case 3: {
                      z2();

                        if ((x1()) state = 4; else state = 0;
                        }
                        break;
            }
      }

      //Вывод коэффициента сжатия
      fprintf(f, "COMPRESSION RATIO = %0.4f",  3 * (double) gnum / (double)
length);

      fclose(f);
      getch();
      return 0;
      }
```