Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчёт по лабораторной работе №4 по курсу «Защита информации»

Тема Программа для создания и проверки электронной цифровой подписи

Студент Волков Г.В.

Группа ИУ7-71Б

Оценка (баллы)

Преподаватели Чиж И. С.

Москва — 2023 г.

Введение

Сжатие данных — алгоритмическое преобразование данных, производимое с целью уменьшения занимаемого ими объёма. Применяется для более рационального использования устройств хранения и передачи данных. Синонимы — упаковка данных, компрессия, сжимающее кодирование, кодирование источника. Обратная процедура называется восстановлением данных (распаковкой, декомпрессией). Сжатие основано на устранении избыточности, содержащейся в исходных данных. Простейшим примером избыточности является повторение в тексте фрагментов (например, слов естественного или машинного языка). Подобная избыточность обычно устраняется заменой повторяющейся последовательности ссылкой на уже закодированный фрагмент с указанием его длины. Другой вид избыточности связан с тем, что некоторые значения в сжимаемых данных встречаются чаще других. Сокращение объёма данных достигается за счёт замены часто встречающихся данных короткими кодовыми словами, а редких — длинными (энтропийное кодирование). Сжатие данных, не обладающих свойством избыточности (например, случайный сигнал или белый шум, зашифрованные сообщения), принципиально невозможно без потерь. Сжатие без потерь позволяет полностью восстановить исходное сообщение, так как не уменьшает в нем количество информации, несмотря на уменьшение длины. Такая возможность возникает только если распределение вероятностей на множестве сообщений не равномерное, например часть теоретически возможных в прежней кодировке сообщений на практике не встречается.

Цель — Разработка алгоритма сжатия информации LZW.

Для достижения поставленной цели необходимо выполнить следующие задачи:

— изучить алгоритм работы LZW;

— реализовать в виде программы алгоритм LZW;

— обеспечить сжатие и разжатие произвольного файла с использованием разработанной программы, рассчитывать коэффициент сжатия;

— предусмотреть работу программы с пустым, однобайтовым файлом и с файлами архива (rar, zip или др.).

1 Аналитическая часть

1.1 Алгоритм LZW

Алгоритм Лемпеля – Зива – Уэлча (Lempel-Ziv-Welch, LZW) — это универсальный алгоритм сжатия данных без потерь, созданный Авраамом Лемпелем, Яаковом Зивом и Терри Велчем. Он был опубликован Велчем в 1984 году в качестве улучшенной реализации алгоритма LZ78, опубликованного Лемпелем и Зивом в 1978 году. Алгоритм разработан так, чтобы его было достаточно просто реализовать как программно, так и аппаратно. Акроним «LZW» указывает на фамилии изобретателей алгоритма: Лемпель, Зив и Велч.

Опубликование алгоритма LZW произвело большое впечатление на всех специалистов по сжатию информации. За этим последовало большое количество программ и приложений с различными вариантами этого метода.

Этот метод позволяет достичь одну из наилучших степеней сжатия среди других существующих методов сжатия графических данных, при полном отсутствии потерь или искажений в исходных файлах. В настоящее время используется в файлах формата TIFF, PDF, GIF, PostScript и других, а также отчасти во многих популярных программах сжатия данных (ZIP, ARJ, LHA).

Процесс сжатия выглядит следующим образом: последовательно считываются символы входного потока и происходит проверка, существует ли в созданной таблице строк такая строка. Если такая строка существует, считывается следующий символ, а если строка не существует, в поток заносится код для предыдущей найденной строки, строка заносится в таблицу, а поиск начинается снова.

Алгоритм кодирования:

1. Все возможные символы заносятся в словарь. Во входную фразу X заносится первый символ сообщения;

2. Считать очередной символ Y из сообщения;

3. Если Y — это символ конца сообщения, то выдать код для X, иначе: если фраза XY уже имеется в словаре, то присвоить входной фразе значение XY и перейти к Шагу 2, иначе выдать код для входной фразы

X, добавить XY в словарь и присвоить входной фразе значение Y и перейти к Шагу 2

Программы кодирования и декодирования должны начинаться с одного и того же начального словаря. Для декодирования на вход подается только закодированный текст, поскольку алгоритм LZW может воссоздать соответствующую таблицу преобразования непосредственно по закодированному тексту. Декодер LZW сначала считывает индекс (целое число), ищет этот индекс в словаре и выводит подстроку, связанную с этим индексом. Первый символ этой подстроки конкатенируется с текущей рабочей строкой. Эта новая конкатенация добавляется в словарь (подобно тому, как подстроки были добавлены во время сжатия). Затем декодированная строка становится текущей рабочей строкой (текущий индекс, т.е. подстрока, запоминается), и процесс повторяется.

Алгоритм декодирования:

1. Все возможные символы заносятся в словарь. Во входную фразу X заносится первый символ сообщения;

2. Считать очередной код Y из сообщения;

3. Если Y — это конец сообщения, то выдать символ, соответствующий коду X, иначе: если фразы под кодом XY нет в словаре, вывести фразу, соответствующую коду X, а фразу с кодом XY занести в словарь, Иначе присвоить входной фразе код XY и перейти к Шагу 2

2 Конструкторская часть

В этом разделе будут представлены сведения о модулях программы.

2.1 Сведения о модулях программы

Программа состоит из двух модулей:

1) *main.c* — файл, содержащий точку входа и алгоритм LZW;

3 Технологическая часть

В данном разделе будут рассмотрены средства реализации, представлены листинги реализаций алгоритма шифрования LZW и результаты тестирования.

3.1 Средства реализации

В данной работе для реализации был выбран язык программирования C. Данный язык удовлетворяет поставленным критериям по средствам реализации.

3.2 Реализация алгоритма

В листинге 3.1 представлена реализация алгоритма шифрования RSA.

Листинг 3.1 – RSA

```
1  int encode(int in_fd, int out_fd) {
2      auto dict = get_dict();
3      uint16_t code = NEW_CODE;
4
5      int rc = 0;
6      char ch;
7      if ((rc = read(in_fd, &ch, sizeof(char))) < 0) {
8          perror("read␣char␣error");
9          return -1;
10     }
11     if(rc == 0) return 0;
12     string p(1, ch);
13
14     while ((rc = read(in_fd, &ch, sizeof(char))) > 0) {
15         if (dict.contains(p + ch)) p += ch;
16         else {
17             if (write(out_fd, &dict[p], sizeof(uint16_t)) < 0) {
18                 perror("write␣code␣error");
19                 return -1;
```

```
20              }
21              dict[p + ch] = code++;
22              p = ch;
23          }
24          if(code == INT16_MAX){
25              if (!p.empty()) {
26                  if (write(out_fd, &dict[p], sizeof(uint16_t)) < 0) {
27                      perror("write code error");
28                      return -1;
29                  }
30              }
31
32              dict = get_dict();
33              code = NEW_CODE;
34
35              if ((rc = read(in_fd, &ch, sizeof(char))) < 0) {
36                  perror("read char error");
37                  return -1;
38              }
39              if(rc == 0) return 0;
40              p = ch;
41
42              uint16_t tmp = CLR_CODE;
43              if (write(out_fd, &tmp, sizeof(uint16_t)) < 0) {
44                  perror("write clr code error");
45                  return -1;
46              }
47          }
48      }
49      if (rc < 0) {
50          perror("read char error");
51          return -1;
52      }
53      if (!p.empty()) {
54          if (write(out_fd, &dict[p], sizeof(uint16_t)) < 0) {
55              perror("write code error");
56              return -1;
57          }
58      }
59
60      return 0;
```

```
61 }
62
63 int decode(int in_fd, int out_fd) {
64     auto dict = get_inv_dict();
65     uint16_t code = NEW_CODE;
66
67     string s, entry;
68     uint16_t k;
69     int rc = 0;
70     if ((rc = read(in_fd, &k, sizeof(uint16_t))) < 0){
71         perror("read␣code␣error");
72         return -1;
73     }
74     if(rc == 0) return 0;
75     s = dict[k];
76     if(write_str(out_fd, s) < 0){
77         perror("write␣str␣error");
78         return -1;
79     }
80
81     while ((rc = read(in_fd, &k, sizeof(uint16_t))) > 0) {
82         if(k == CLR_CODE){
83             dict = get_inv_dict();
84             code = NEW_CODE;
85
86             if ((rc = read(in_fd, &k, sizeof(uint16_t))) < 0){
87                 perror("read␣code␣error");
88                 return -1;
89             }
90             if(rc == 0) return 0;
91             s = dict[k];
92             if(write_str(out_fd, s) < 0){
93                 perror("write␣str␣error");
94                 return -1;
95             }
96             continue;
97         }
98
99         if (dict.contains(k))
100        entry = dict[k];
101        else if (k == code)
```

```
102             entry = s + s[0];
103             else
104             throw exception();
105 
106             if(write_str(out_fd, entry) < 0){
107                 perror("write str error");
108                 return -1;
109             }
110             dict[code++] = s + entry[0];
111             s = entry;
112         }
113         if (rc < 0) {
114             perror("read code error");
115             return -1;
116         }
117 
118         return 0;
119 }
```

3.3 Тестирование

Для тестирование написанной программы сжимались и разжимались файлы и проверялся коэффициент сжатия.

Таблица 3.1 – Функциональные тесты ЭЦП

Входной файл	Ожидаемый результат	Результат
Текстовый файл	Файлы совпали КС < 1	Файлы совпали КС $= 0.5$
1 байтовый файл	Файлы совпали КС < 1	Файлы совпали КС $= 0.5$
Пустой файл	Файлы совпали КС = -//-	Файлы совпали КС = -//-
Архив	Файлы совпали КС < 1	Файлы совпали КС $= 0.56$
PNG изображение	Файлы совпали КС < 1	Файлы совпали КС $= 0.69$
Текст из множества символов «a»	Файлы совпали КС > 1	Файлы совпали КС $= 11$
Однотонная картинка	Файлы совпали КС > 1	Файлы совпали КС $= 194$

Заключение

В результате лабораторной работы были изучены принципы работы алгоритма LZW, была реализована программа, сжимающая и разжимающая произвольные файлы.

Были решены следующие задачи:

1) изучен алгоритм работы LZW;
2) реализован в виде программы алгоритм LZW;
3) обеспечено сжатие и разжатие произвольного файла с использованием разработанной программы, рассчитывается коэффициент сжатия;
4) предусмотрена работа программы с пустым, однобайтовым файлом и с файлами архива (rar, zip или др.).