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 №5 по курсу «Защита информации»

Тема Реализация алгоритма сжатия LZW

Студент Калашков П. А.

Группа ИУ7-76Б

Оценка (баллы)

Преподаватели Чиж И. С.

Москва — 2023 г.

Введение

В XXI веке количество информации, с которым встречается человек в течении дня, намного больше чем в другие периоды существования человечества. Информация хранится на физических носителях, передаётся по проводам и по беспровдной связи. Часто возникают ситуации, когда объём передаваемых или хранимых данных слишком велик, и возникает необходимость его уменьшить.

Сжатие данных — обратимое преобразование данных, производиме с целью уменьшения занимаемого ими объёма с целью уменьшения объёма данных, который требуется для хранения или передачи информации.

LZW — алгоритм сжатия данных, разработанный в 1984 году и опубликованнный Терри Велчем в качестве улучшенной реализации разработаннг в 1978 алгоритма. Он предназначен для сжатия (и последующего декодирования) информации, причём в 1984 году этот алгоритм получил широкое распространение из-за простоты реализации.

Целью данной работы является реализация в виде программы алгоритма сжатия данных LZW, обеспечить сжатие и разжатие произвольного файла с использованием разработанной программы, расчитывать коэффициент сжатия. Предусмотреть работу с пустым, однобайтовым файлами.

Для достижения поставленной цели необходимо выполнить следующие задачи:

1) изучить алгоритм сжатия LZW;

2) реализовать алгоритм сжатия LZW в виде программы, обеспечив возможности сжатия и разжатия произвольнго файла и расчёт коэффициента сжатия;

3) протестировать разработанную программу, показать, что удаётся сжимать и разжимать файлы разных форматов;

4) описать и обосновать полученные результаты в отчёте о выполненной лабораторной работе.

1 Аналитическая часть

В этом разделе будет рассмотрен алгоритм сжатия LZW, шаги для сжатия и разжатия, а также его преимущества, недостатки и возможные способы решения возникающих проблем.

1.1 Алгоритм LZW

Алгоритм сжатия LZW (аббревиатура от фамилий *Lempel*, *Ziv* и *Welch*) — универсальный алгоритм сжатия данных без потерь, в основе которого лежит использование словаря фраз.

Под словарём имеется в виду структура данных, предназначенная для объединения взаимосвязанной информации, доступ к элементам словаря предоставляется по ключу. Ключом может быть любой неизменяемый объект, число, строка, кортеж.

При сжатии создаётся словарь фраз,в котором ключами являютсся определённые фразы (т. е. строки или, в общем случае, последовательности бит), а значениями являются группы битоов фиксирваннй длины (например, 12-битные). Изначально словарь инициализируется всеми возможными 1-символьными фразами (или, в общем случае, всеми фразами минимальной длины, например, всеми 8-битными фразами).

По мере кодирования алгооритм просматривает сообщение (или последовательность бит) слева направо, и при чтении находится строка W максимальной длины, совпадающая с какой-то фразой из словаря. Она выбирается жадно, и, поскольку словарь был инициализован, она всегда найдётся. Затем код этой фразы подаётся на выход, а строка WK, где K — следующий за W символ сообщения (или, в общем случае, последовательность бит установленной длины), вносится в словарь в качестве новой фразы, ей присваивается новый код.

Более формально алгоритм LZW можно описать следующими шагами:

1) инициализируется словарь фраз всеми возможными последовательностями бит минимальной длин;

2) если дошли до конца сообщения, то выдача код для W и завершение алгоритма;

3) считывается очередную последовательность бит K установленной длины (такой, что все возможные последовательности бит данной длины содержатся в словаре);

4) если фраза WK уже есть в словоаре, то входной фразе W присваивается значение WK, осуществляется переход к шагу 2;

5) код для фразы W записывается в результат, фраза WK добавляется в словарь, а входной фразе W присваивается значение K, осуществляется переход к шагу 2.

Для декодирования требуется только закодированный текст, поскольку словарь фраз может быть воссоздан непосредственно по декодируемому коду. Таким образом, результат сжатия является однозначно декодируемым: каждый раз, коогда генерируется новый код, строка добавляется в словарь фраз. Таким образом, каждая строка будет храниться в единственном экземпляре и иметь свой уникальный номер (последовательность бит).

При декодировании во время получения нового кода генерируется новая строка, и при получении известного кода строка извлекается из словаря.

1.1.1 Преимущества и недостатки алгоритма LZW

Преимущества

Простота реализации является одной из причин широкого распространения данного алгоритма. В основе алгоритма лежит использование словаря, или же префиксного дерева.

Хорошая степень сжатия, которую обеспечивает алгоритм LZW для различных типов данных, также внесла свой вклад в распространение алгоритма. Особенно высоким будет коэффициент сжатия для текстовых файлов с повторяющимися фрагментами: чем больше в сжимаемом сообщении повторяющихся фраз, тем больше места получится сэкономить при замене этих фраз на битовые последовательности из словаря.

Универсальность алгоритма LZW — ещё одно преимущество, ведь алгоритм может быть использован для сжатия любых типов данных, включая текст, изображения, звук и видео.

Недостатки

Эффективность метода LZW зависит от размера словаря и (косвенно) от типа данных. Больший словарь может обеспечить лучшую степень сжатия (т.к. вместит в себя больше фраз), однако требует больше памяти для хранения.

Зависимость от типа данных проявляется следующим образом: если содержимое файла содержит небольшое число повторяющихся последовательностей, эффективность сжатия может быть отрицательной: так, при сжатии уже сжатого файла, в котором повторяющиеся фрагменты отсутствуют или минимальны, в результате сжатия объём данных лишь увеличится, поскольку коротким фразам-ключам в словаре будут соответствовать превышающие их по длине значения.

Зависимость от размера словаря можно частично решить изменением максимального возможного размера словаря с числом роста фраз. Длины кодов, на которые заменяются фразы сообщения, будут расти (например, от 9 до 16 бит).

Ещё одной проблемой, которая возникает при использовании алгоритма LZW — ограничение по количеству фраз, которые может вместить в себя словарь. Так, при фиксированной длине в 12 бит словарь будет способен хранить всего 4096 фраз. Возможным решением данной проблемы будет введение специальной последовательности, встречая которую при разжатии, алгоритм будет очищать свой словарь и начинать заполнять его заново.

Вывод

В данном разделе был рассмотрен алгоритм сжатия LZW, шаги для сжатия и разжатия, а также его преимущества, недостатки и возможные способы решения возникающих проблем.

2 Конструкторская часть

В этом разделе представлены сведения о модулях программы, а также схемы алгоритмов, которые нужно реализовать: алгоритмы сжатия и разжатия LZW.

2.1 Сведения о модулях программы

Программа состоит из трёх модулей:

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

2) *menu.c* — файл, содержащий код меню программы;

3) *lzw.c* — файл, содержайший реализацию алгоритма сжатия LZW.

2.2 Разработка алгоритмов

На рисунках 2.1 и 2.2 представлены схемы алгоритмов сжатия и разжатия LZW

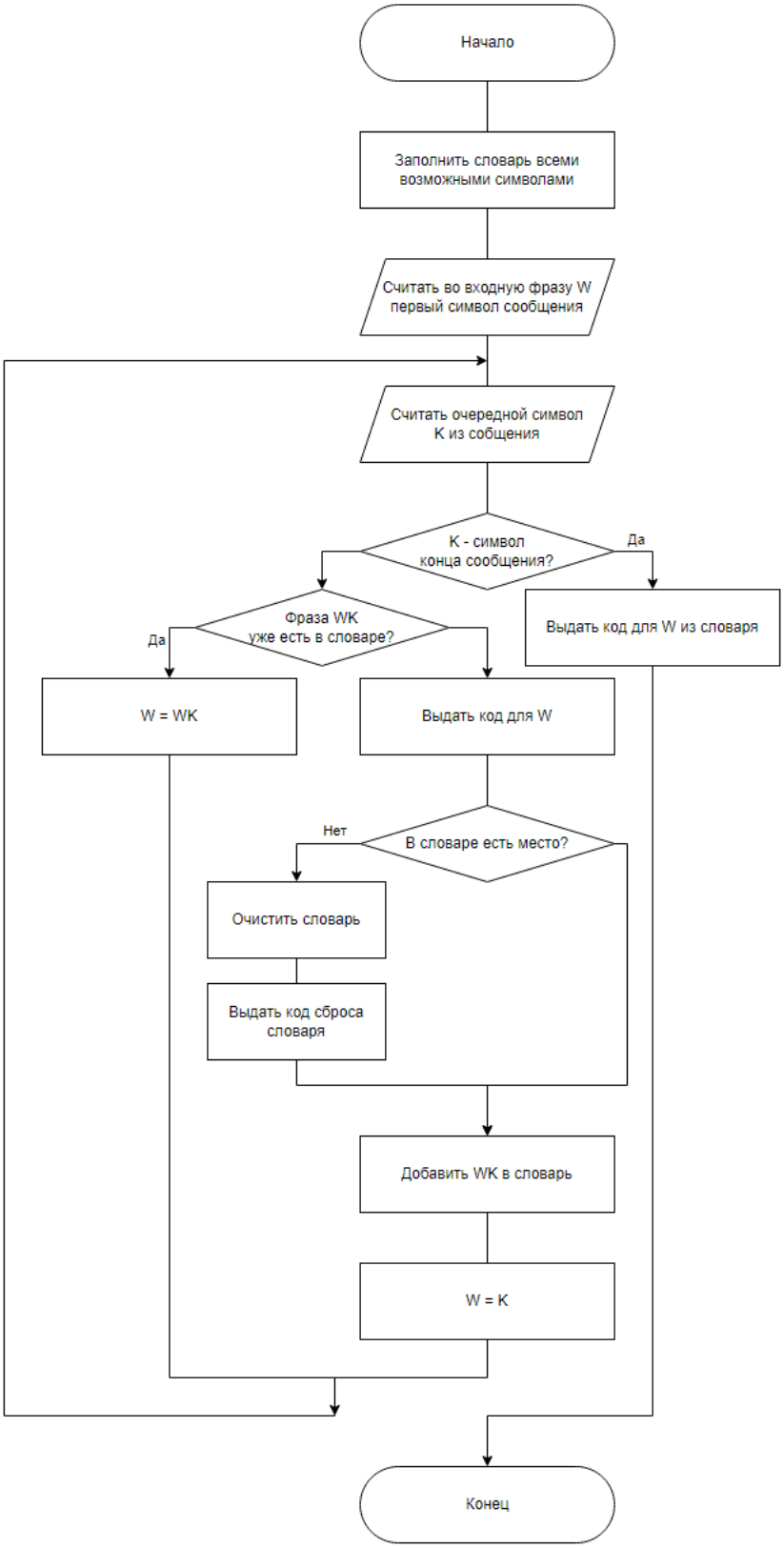

Рисунок 2.1 – Схема алгоритма сжатия LZW

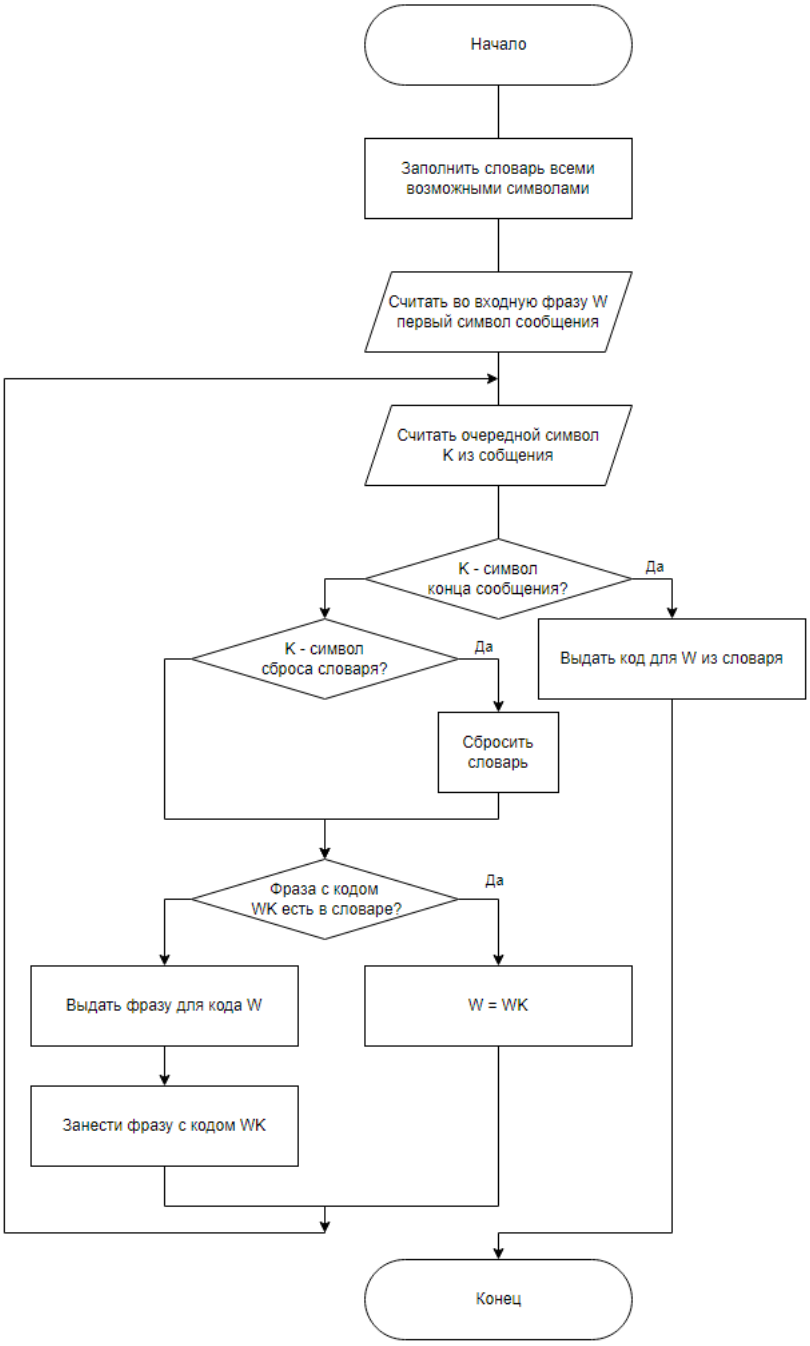

Рисунок 2.2 – Схема алгоритма разсжатия LZW

Вывод

В данном разделе были представлены сведения о модулях программы, а также схемы алгоритмов, которые нужно реализовать: алгоритмы сжатия и разжатия LZW.

3 Технологическая часть

В данном разделе рассмотрены средства реализации, реализации алгоирмтов сжатиия и разжатия LZW, а также произведено тестирование.

3.1 Средства реализации

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

3.2 Реализация алгоритма

В листингах 3.1–3.3 представлена реализация алгоритма сжатия LZW, а на листингах 3.4–3.7 — реализация алгоритма разжатия LZW.

Листинг 3.1 – Реализация алгоритма сжатия LZW (часть 1)

```
1  ssize_t lzw_compress(struct lzw_state *state, uint8_t *src, size_t
       slen, uint8_t *dest, size_t dlen)
2  {
3      if (state->was_init == false)
4      {
5          lzw_init(state);
6          lzw_output_code(state, CODE_CLEAR);
7      }
8      code_t code = CODE_EOF;
9      size_t prefix_end = 0;
10     state->wptr = 0;
11
12     while (state->rptr + prefix_end < slen)
13     {
14         if (state->wptr + (state->tree.code_width >> 3) + 1 + 2 + 2
               > dlen)
15             return state->wptr;
```

Листинг 3.2 – Реализация алгоритма сжатия LZW (часть 2)

```
1
2           ++prefix_end;
3           bool overlong = ((state->longest_prefix_allowed > 0) &&
              (prefix_end >= state->longest_prefix_allowed));
4           bool existing_code = lzw_string_table_lookup(state, src +
              state->rptr, prefix_end, &code);
5           if (!existing_code || overlong)
6           {
7               uint8_t symbol = src[state->rptr + prefix_end - 1];
8               code_t parent = code;
9               code_t parent_len = 1 +
                  lzw_node_prefix_len(state->tree.node[parent]);
10              lzw_output_code(state, parent);
11
12              if (state->tree.next_code == (1UL <<
                  state->tree.code_width)
13  #if LZW_MAX_CODE_WIDTH == 16
14                  || (state->tree.next_code == LZW_MAX_CODES - 1)
15  #endif
16              )
17              {
18                  if (state->tree.code_width < LZW_MAX_CODE_WIDTH)
19                  {
20                      ++state->tree.code_width;
21                  }
22                  else
23                  {
24                      lzw_flush_reservoir(state, dest, false);
25                      lzw_output_code(state, CODE_CLEAR);
26                      lzw_reset(state);
27                      lzw_flush_reservoir(state, dest, false);
28                      state->tree.next_code = CODE_EOF;
29                  }
30              }
31              state->tree.node[state->tree.next_code++] =
                  lzw_make_node(symbol, parent, parent_len);
32
33              if (parent_len > state->longest_prefix)
34                  state->longest_prefix = parent_len;
```

Листинг 3.3 – Реализация алгоритма сжатия LZW (часть 3)

```
            state->rptr += parent_len;
            prefix_end = 0;
            lzw_flush_reservoir(state, dest, false);
        }
    }
    if (prefix_end != 0)
    {
        lzw_output_code(state, code);
        lzw_flush_reservoir(state, dest, false);
        state->rptr += prefix_end;
        prefix_end = 0;
    }

    if ((state->rptr + prefix_end == slen && state->tree.prev_code
       != CODE_EOF)
        || (state->wptr == 0 && state->bitres_len > 0))
    {
        lzw_output_code(state, CODE_EOF);
        lzw_flush_reservoir(state, dest, true);
    }
    return state->wptr;
}
```

Листинг 3.4 – Реализация алгоритма разжатия LZW (часть 1)

```
ssize_t lzw_decompress(struct lzw_state *state, uint8_t *src,
   size_t slen, uint8_t *dest, size_t dlen)
{
    if (state->was_init == false)
        lzw_init(state);
    uint32_t bitres = state->bitres;
    uint32_t bitres_len = state->bitres_len;

    uint32_t code = 0;
    size_t wptr = 0;

    while (state->rptr < slen)
    {
```

Листинг 3.5 – Реализация алгоритма разжатия LZW (часть 2)

```
1         while ((bitres_len < state->tree.code_width) &&
              (state->rptr < slen))
2         {
3             bitres |= src[state->rptr++] << bitres_len;
4             bitres_len += 8;
5         }
6
7         state->bitres = bitres;
8         state->bitres_len = bitres_len;
9
10        if (state->bitres_len < state->tree.code_width)
11        {
12            return LZW_INVALID_CODE_STREAM;
13        }
14
15        code = bitres & mask_from_width(state->tree.code_width);
16        bitres >>= state->tree.code_width;
17        bitres_len -= state->tree.code_width;
18
19        if (code == CODE_CLEAR)
20        {
21            if (state->tree.next_code != CODE_FIRST)
22                lzw_reset(state);
23            continue;
24        } else if (code == CODE_EOF)
25        {
26            break;
27        } else if (state->must_reset)
28        {
29            return LZW_STRING_TABLE_FULL;
30        }
31
32        if (code <= state->tree.next_code)
33        {
34            bool known_code = code < state->tree.next_code;
35            code_t tcode = known_code ? code :
                  state->tree.prev_code;
36            size_t prefix_len = 1 +
                  lzw_node_prefix_len(state->tree.node[tcode]);
```

Листинг 3.6 – Реализация алгоритма разжатия LZW (часть 3)

```
1             uint8_t symbol = 0;
2             if (!known_code && state->tree.prev_code == CODE_EOF)
3                 return LZW_INVALID_CODE_STREAM;
4
5             if (prefix_len > state->longest_prefix)
6                 state->longest_prefix = prefix_len;
7
8             if (prefix_len + (known_code ? 0 : 1) > dlen)
9                 return LZW_DESTINATION_TOO_SMALL;
10
11            if (wptr + prefix_len + (known_code ? 0 : 1) > dlen)
12                return wptr;
13
14            for (size_t i=0 ; i < prefix_len ; ++i)
15            {
16                symbol = lzw_node_symbol(state->tree.node[tcode]);
17                dest[wptr + prefix_len - 1 - i] = symbol;
18                tcode = lzw_node_parent(state->tree.node[tcode]);
19            }
20            wptr += prefix_len;
21
22            if (state->tree.prev_code != CODE_EOF)
23            {
24                if (!known_code) {
25                    dest[wptr++] = symbol;
26                }
27
28                state->tree.node[state->tree.next_code] =
                      lzw_make_node(symbol, state->tree.prev_code, 1 +
                      lzw_node_prefix_len(
29                     state->tree.node[state->tree.prev_code]
30                ));
31
32                if (state->tree.next_code >=
                      mask_from_width(state->tree.code_width)) {
33                    if (state->tree.code_width ==
                          LZW_MAX_CODE_WIDTH)
34                    {
35                        state->must_reset = true;
36                        state->tree.prev_code = code;
```

Листинг 3.7 – Реализация алгоритма разжатия LZW (часть 4)

```
                        continue;
                    }
                    ++state->tree.code_width;
                }
                state->tree.next_code++;
            }
            state->tree.prev_code = code;
        }
        else
            return LZW_INVALID_CODE_STREAM;
    }
    return wptr;
}
```

3.3 Тестирование

Тестирование разработанной программы производилось следующим образом: для файлов различных форматов производилось сжатие и разжатие, проверка совпадения исходного значения файла и получившегося, а также расчёт коэффициента сжатия для конкретного файла.

Таблица 3.1 – Функциональные тесты

Формат файла	Размер файла, байты	Коэффициента сжатия, %
TXT	376844	75.75
PNG	16760	-33.28
BMP	818058	4.90
MP3	733645	-31.21
JS	470167	68.65

Вывод

В данном разделе были рассмотрены средства реализации, реализации алгоирмтов сжатиия и разжатия LZW, а также произведено тестирование.

Заключение

В результате лабораторной работы был реализована программа, позволяющая сжимать и разжимать файлы при помощи алгоритма LZW и расчитывать коэффициент сжатия.

Был и выполнены следующие задачи:

1) изучен алгоритм сжатия LZW;
2) реализован алгоритм сжатия LZW в виде программы, позволяющей сжимать и разжимать произвольные файлы и расчитывать коэффициент сжатия;
3) протестирована разработанная программа;
4) описаны и обоснованы полученные результаты в отчёте о выполненной лабораторной работе.