



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»

ОТЧЕТ

по лабораторной работе №5
по курсу «Защита информации»
на тему: «Сжатие данных»
Вариант № 1 (LZW)

Студент ИУ7-72Б
(Группа)

(Подпись, дата)

Е. О. Карпова
(И. О. Фамилия)

Преподаватель

(Подпись, дата)

И. С. Чиж
(И. О. Фамилия)

2023 г.

ВВЕДЕНИЕ

Сжатие данных - это процесс уменьшения размера данных с целью уменьшения объема памяти, занимаемого этими данными, или увеличения скорости передачи данных. Существуют различные методы сжатия данных, такие как алгоритмы с потерями и без потерь, которые позволяют сжимать различные типы данных, включая текст, изображения, аудио и видео.

Цель данной работы — разработка алгоритма сжатия информации.

Для достижения поставленной цели необходимо выполнить следующие задачи.

- 1) Изучить алгоритм LZW.
- 2) Спроектировать алгоритм LZW.
- 3) Реализовать алгоритм LZW.
- 4) Протестировать реализацию алгоритма.

1 Теоретический раздел

1.1 LZW

Алгоритм LZW (Lempel-Ziv-Welch) - это универсальный алгоритм сжатия данных, который был разработан в 1984 году. Он широко используется для сжатия текстовых данных и изображений.

Принцип работы алгоритма LZW заключается в замене повторяющихся последовательностей символов на коды, что позволяет сократить объем данных. Алгоритм основан на словаре, который содержит все возможные комбинации символов, которые могут встретиться в исходных данных.

Шаги алгоритма LZW:

1. Инициализация словаря: в начале работы алгоритма создается словарь, который содержит все возможные символы (например, буквы алфавита) и коды для них.
2. Чтение исходных данных: алгоритм читает исходные данные и ищет самую длинную подстроку, которая уже есть в словаре.
3. Добавление новой последовательности в словарь: если найденная подстрока отсутствует в словаре, то она добавляется в словарь с новым кодом.
4. Замена последовательности на код: найденная подстрока заменяется на соответствующий ей код из словаря.
5. Повторение шагов 2-4: процесс поиска повторяющихся последовательностей и их замены на коды продолжается до тех пор, пока все исходные данные не будут обработаны.
6. Вывод закодированных данных: после завершения работы алгоритма получается закодированный поток данных, который занимает меньше места, чем исходные данные.

Алгоритм LZW имеет ряд преимуществ, таких как высокая степень сжатия, относительно простая реализация и возможность адаптации к различным типам данных. Однако он также имеет некоторые ограничения, например, он может потребовать больше памяти для хранения словаря при работе с большими данными.

2 Конструкторский раздел

2.1 Разработка алгоритма

На рисунках 2.1 и 2.2 представлены схемы реализации сжатия и разжатия данных алгоритмом LZW соответственно.

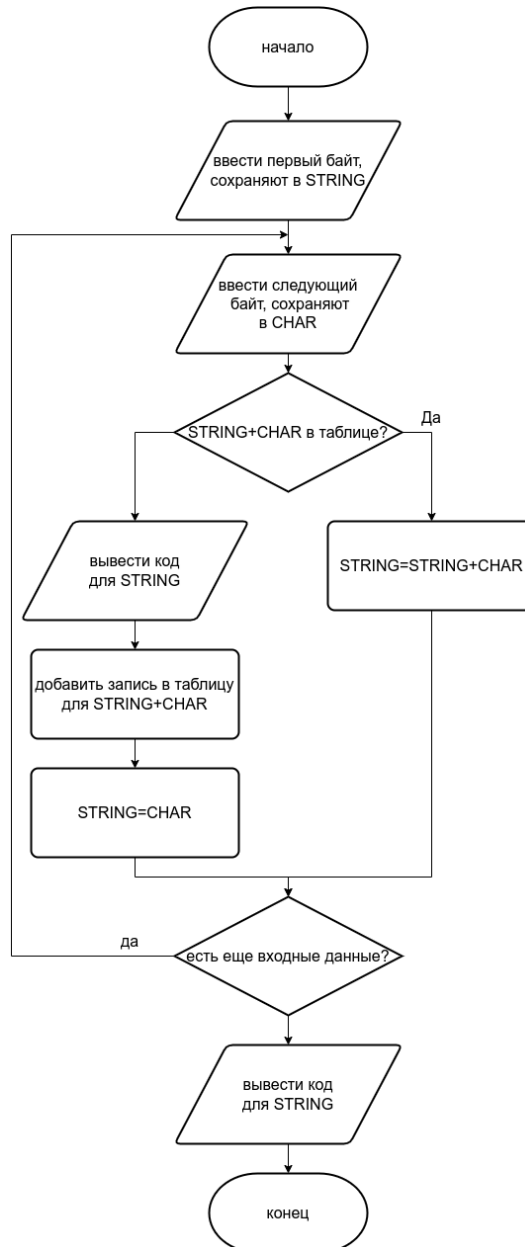


Рисунок 2.1 – Схемы для реализации сжатия данных алгоритмом LZW

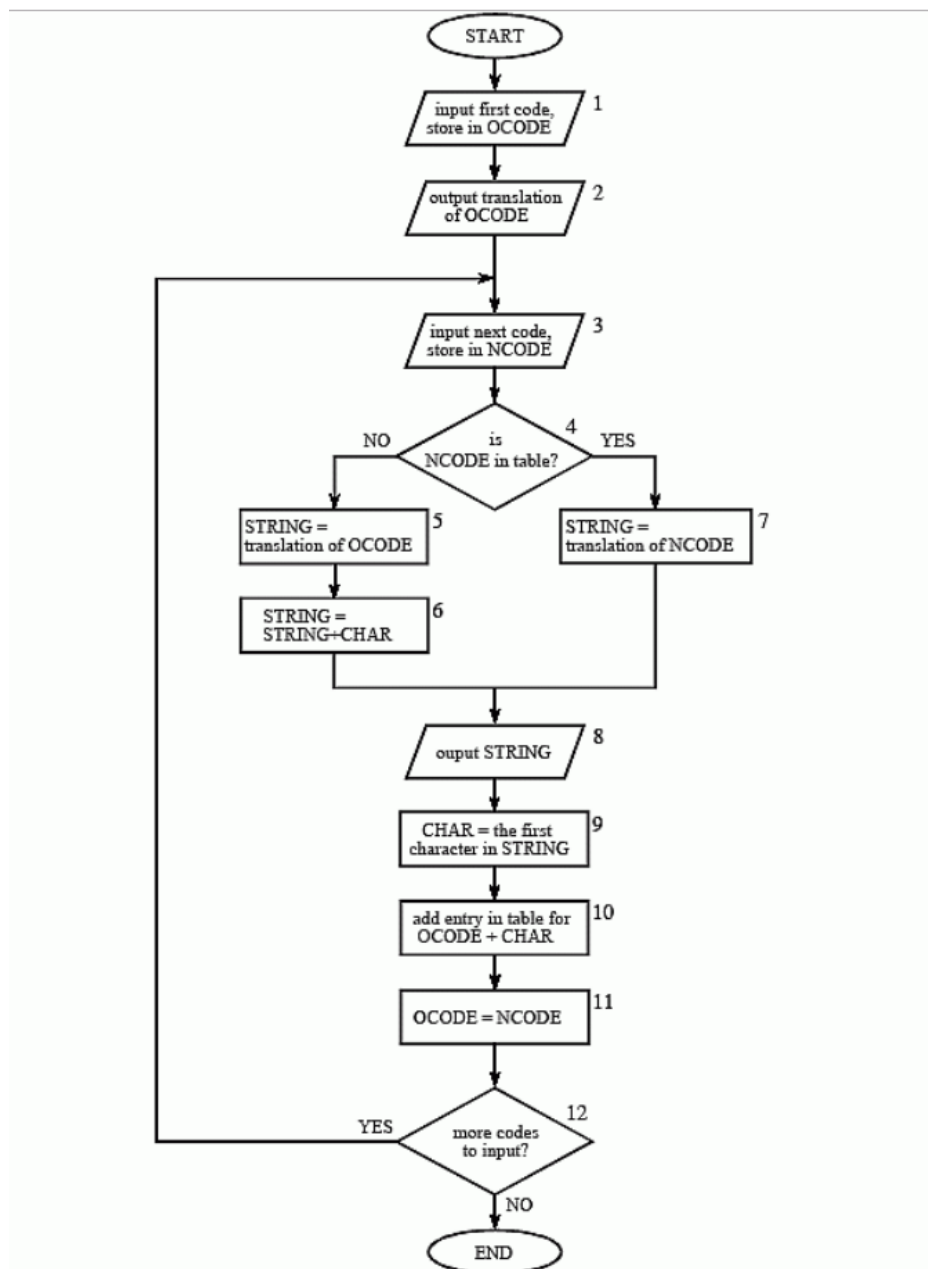


Рисунок 2.2 – Схемы для реализации разжатия данных алгоритмом LZW

3 Практический раздел

3.1 Листинг алгоритма MD5

Листинг 3.1 – Реализация алгоритма LZW на сжатие

```
1 void encode_lzw_data(FILE *f_in, FILE *f_out, dict_t *d) {
2     uint8_t C = '\0';
3     bytes_t P = {0};
4
5     // Read data from file and create table dynamically.
6     if (fread(&C, 1, 1, f_in)) {
7         append_byte(&P, C);
8         while (!feof(f_in)) {
9             if (fread(&C, 1, 1, f_in)) {
10                bytes_t temp = {0};
11                append_bytes(&temp, P);
12                append_byte(&temp, C);
13                if (is_in_dict(d, temp)) {
14                    copy_bytes(&P, temp);
15                } else {
16                    // Output the data for P.
17                    short code = get_index(d, P);
18                    write_byte_data(f_out, code, 0);
19                    add_to_dict(d, temp);
20                    copy_data(&P, C);
21                }
22
23                free_bytes(&temp);
24            }
25        }
26        int code = get_index(d, P);
27        write_byte_data(f_out, code, 0);
28        // Write remaining data if any.
29        write_byte_data(f_out, 0, 1);
30    }
31    free_bytes(&P);
32 }
```

Листинг 3.2 – Реализация алгоритма LZW на распаковку

```
1 void decode_lzw_data(FILE *f_in, FILE *f_out, dict_t *d) {
2     short new;
3
4     short old = get_lzw_code(f_in);
5     bytes_t tmp = get_from_dict(d, old);
6     bytes_t C = {0};
7     bytes_t S = {0};
8
9     fwrite(tmp.data, 1, tmp.len, f_out);
10    copy_data(&C, tmp.data[0]);
11
12    while ((new = get_lzw_code(f_in)) != -1) {
13        if (d->data[new].len == 0) {
14            S = get_from_dict(d, old);
15            append_bytes(&S, C);
16        } else {
17            S = get_from_dict(d, new);
18        }
19
20        fwrite(S.data, 1, S.len, f_out);
21        copy_data(&C, S.data[0]);
22        tmp = get_from_dict(d, old);
23        append_bytes(&tmp, C);
24        add_to_dict(d, tmp);
25        old = new;
26    }
27 }
```


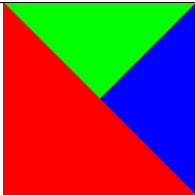
3.2 Тестирование

Корректность алгоритма проверялось путем применения дешифрации на шифрованное сообщение.

Тестирование было проведено на файлах с типами: текстовый (txt), графический (jpeg, png), архив (zip), несуществующий (ubc). Также, был проведен тест с повреждением зашифрованного файла.

В таблице 3.1 представлены тестовые данные.

Таблица 3.1 – Тестовые данные

Номер теста	Тип файла	Содержимое файла
1	txt (1142.8572%)	aaaa...
2	ubc (0%)	∅
3	zip (68.7722%)	Файлы с тестов 1, 2, 4
4	png (68.7669%)	
5	bmp (5023.1802%)	

ЗАКЛЮЧЕНИЕ

В результате выполнения данной лабораторной работы поставленная цель достигнута: реализована программа сжатия данных алгоритмом LZW.

В ходе выполнения лабораторной работы были выполнены все задачи.

- 1) Изучен алгоритм LZW.
- 2) Спроектирован алгоритм LZW.
- 3) Реализован алгоритм LZW.
- 4) Протестирована реализация алгоритма.