

Теория кодирования и сжатия информации

Лабораторная работа №7

Гущин Андрей, 431 группа, 1 подгруппа

2022 г.

1 Задача

Разработать программу осуществляющую архивацию и разархивацию текстового файла используя алгоритм Лемпеля-Зива (LZ77). Программы архивации и разархивации должны быть представлены отдельно и работать независимо друг от друга. Определить для данного шифра характеристики 1 (коэффициент сжатия) и 2 (скорость сжатия). К работе необходимо прикрепить отчет и программный проект.

2 Алгоритм

Алгоритм LZ77 заключается в создании нового кода для всех встречающихся последовательностей символов. Алгоритм использует идею «скользящего окна» для создания кодов.

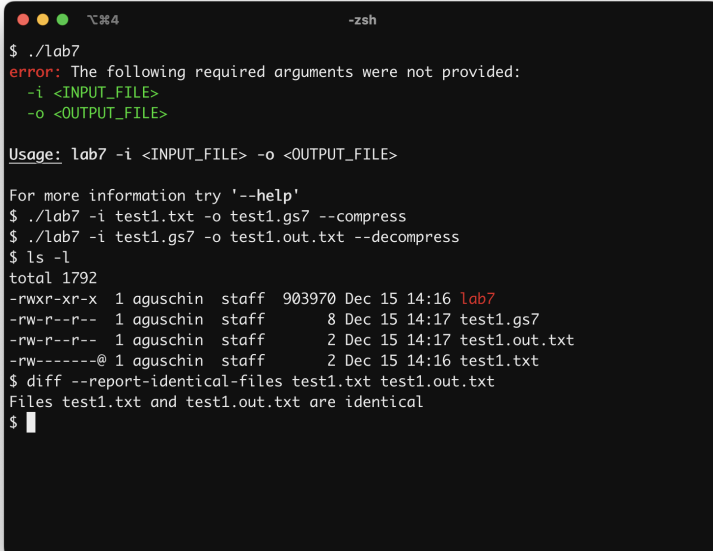
Алгоритм состоит из следующих шагов:

1. Прочитать первый символ. Так как он ни разу не встречался, закодировать его как $(0, 0, ?)$, где $(offset, l, char)$, $offset$ — сдвиг относительно текущей позиции, l — длина последовательности, $char$ — символ, стоящий после последовательности.
2. Далее необходимо найти наибольшую подстроку ps , начинающуюся в окне и при этом равную подстроке, начинающейся в текущей позиции.
3. Если строка не найдена, то просто добавить отдельный символ аналогично самому первому. Иначе закодировать с помощью сдвига и чтения подстроки длины l .
4. Если при чтении кода встретилась запись длины 0, то необходимо просто добавить указанный символ.
5. Иначе необходимо скопировать l символов, начиная с $current - offset$, а после них добавить один символ $char$.

3 Тестирование

Для проверки программы были использованы тестовые тексты 1 (рис. 1) и 6 (рис. 2). Можно заметить, что после распаковки архива полученный файл

совпадает с исходным (проверка с помощью утилиты diff). Также можно заметить, что для файлов малого размера архив увеличивает их размер за счёт метаданных.



```
$ ./lab7
error: The following required arguments were not provided:
  -i <INPUT_FILE>
  -o <OUTPUT_FILE>

Usage: lab7 -i <INPUT_FILE> -o <OUTPUT_FILE>

For more information try '--help'
$ ./lab7 -i test1.txt -o test1.gs7 --compress
$ ./lab7 -i test1.gs7 -o test1.out.txt --decompress
$ ls -l
total 1792
-rwxr-xr-x  1 aguschin  staff   903970 Dec 15 14:16 lab7
-rw-r--r--  1 aguschin  staff      8 Dec 15 14:17 test1.gs7
-rw-r--r--  1 aguschin  staff      2 Dec 15 14:17 test1.out.txt
-rw-----@ 1 aguschin  staff      2 Dec 15 14:16 test1.txt
$ diff --report-identical-files test1.txt test1.out.txt
Files test1.txt and test1.out.txt are identical
$
```

Рис. 1: Сжатие текста Тест_1.txt

```

$ ./lab7
error: The following required arguments were not provided:
  -i <INPUT_FILE>
  -o <OUTPUT_FILE>

Usage: lab7 -i <INPUT_FILE> -o <OUTPUT_FILE>

For more information try '--help'
$ ./lab7 -i test6.txt -o test6.gs7 --compress
$ ./lab7 -i test6.gs7 -o test6.out.txt --decompress
$ ls -l
total 1816
-rwxr-xr-x  1 aguschin  staff  903970 Dec 15 14:16 lab7
-rw-r--r--  1 aguschin  staff    7424 Dec 15 14:18 test6.gs7
-rw-r--r--  1 aguschin  staff    7958 Dec 15 14:18 test6.out.txt
-rw-r--r--  1 aguschin  staff    7958 Dec 15 14:18 test6.txt
$ diff --report-identical-files test6.txt test6.out.txt
Files test6.txt and test6.out.txt are identical
$
```

Рис. 2: Сжатие текста Тест_6.txt

4 Вычисленные характеристики

4.1 Характеристика 1 (Коэффициент сжатия)

Результаты применения программы к каждому из тестовых текстовых файлов занесены в таблицу 1.

Название	Исходный размер, байт	Сжатый размер, байт	Коэффициент
Тест_1.txt	2	8	0.25
Тест_2.txt	33	132	0.25
Тест_3.txt	2739	176	15.5625
Тест_4.txt	330	16	20.625
Тест_5.txt	59	236	0.25
Тест_6.txt	7958	7424	1.68459
Тест_7.txt	138245	104980	1.31687
Тест_8.txt	574426	415612	1.38212
Тест_9.txt	2752	48	57.3333
Тест_10.txt	2814	56	50.25

Таблица 1: результаты тестирования

4.2 Характеристика 2 (Скорость сжатия)

Для тестирования скорости сжатия использовался произвольный двоичный файл размера 3120002 байт (≈ 3 мегабайта). В результате пяти последовательных запусков, среднее время запаковки файла составило 17.23 секунды,

среднее время распаковки составило 0.02 секунд.

Таким образом, средняя скорость сжатия составила 0.17269 Мбайт в секунду, а средняя скорость разжатия составила 148.77329 Мбайт в секунду.

5 Реализация

Программа реализована на языке программирования Rust с использованием библиотеки clap для чтения параметров командной строки. Сборка производится с помощью программы cargo, поставляющейся вместе с языком.

5.1 Содержимое файла lz77.rs

```
1 pub fn lz_match(data: &Vec<u8>, pos1: usize, pos2: usize, length: usize)
  ↪ -> bool {
2     for i in 0..length {
3         if data[pos1 + i] != data[pos2 + i] {
4             return false;
5         }
6     }
7     return true;
8 }
9
10 pub fn lz77_encode(data: &Vec<u8>, window_size: usize) -> Vec<(u32, u8,
  ↪ u8)> {
11     let mut ptr = 0;
12     let mut encoded = Vec::new();
13
14     while ptr < data.len() {
15         let mut saved = None;
16         let mut l = 1;
17         while ptr + l < data.len() && l < 256 {
18             let mut found = false;
19             for offset in 1..window_size {
20                 if (ptr as i32) - (offset as i32) < 0 {
21                     break;
22                 }
23                 if lz_match(data, ptr - offset as usize, ptr, l) {
24                     found = true;
25                     saved = Some((l, offset));
26                     l += 1;
27                     break;
28                 }
29             }
30             if !found {
31                 break;
32             }
33         }
34         if let Some((l, offset)) = saved {
35             encoded.push((offset as u32, l as u8, data[ptr + l]));
36             ptr += l + 1;
37         } else {
```

```

38         encoded.push((0, 0, data[ptr]));
39         ptr += 1;
40     }
41 }
42
43     return encoded;
44 }
45
46 pub fn compress(data: &Vec<u8>) -> Vec<u8> {
47     let mut result = Vec::new();
48     let encoded = lz77_encode(data, 8192);
49
50     for (offset, length, byte) in &encoded {
51         let offset_bytes: [u8; 4] = unsafe { std::mem::transmute(*offset) }
↪    };
52         result.push(offset_bytes[0]);
53         result.push(offset_bytes[1]);
54
55         result.push(*length);
56         result.push(*byte);
57     }
58
59     return result;
60 }
61
62 pub fn decompress(archive: &Vec<u8>) -> Vec<u8> {
63     let mut result = Vec::new();
64     let mut ptr: usize = 0;
65     let block_size = 4;
66     for i in 0..archive.len() / block_size {
67         let b_offset = i * block_size;
68
69         let offset_bytes: [u8; 4] = [archive[b_offset + 0],
↪    archive[b_offset + 1], 0, 0];
70         let offset: u32 = unsafe { std::mem::transmute(offset_bytes) };
71         let length = archive[b_offset + 2] as usize;
72         let byte = archive[b_offset + 3];
73
74         if length == 0 {
75             result.push(byte);
76             ptr += 1;
77         } else {
78             for j in 0..length as usize {
79                 result.push(result[ptr - offset as usize + j]);
80             }
81             result.push(byte);
82             ptr += length as usize + 1;
83         }
84     }
85     return result;
86 }

```

5.2 Содержимое файла main.rs

```
1 mod lz77;
2 use clap::Parser;
3 use std::fs::File;
4 use std::io::{Error, ErrorKind, Read, Write};
5 use std::path::PathBuf;
6
7 #[derive(Parser)]
8 struct Cli {
9     #[arg(short)]
10    input_file: PathBuf,
11
12    #[arg(short)]
13    output_file: PathBuf,
14
15    #[arg(long, default_value_t = true)]
16    compress: bool,
17
18    #[arg(long, default_value_t = false)]
19    decompress: bool,
20 }
21
22 fn run_decompressor(cli: &Cli) -> Result<(), Error> {
23     let mut input_f = File::open(cli.input_file.to_str().unwrap())?;
24     let mut archive = Vec::new();
25     input_f.read_to_end(&mut archive)?;
26     let data = lz77::decompress(&archive);
27
28     let mut output_f = File::create(cli.output_file.to_str().unwrap())?;
29     output_f.write_all(&data)?;
30
31     return Ok(());
32 }
33
34 fn run_compressor(cli: &Cli) -> Result<(), Error> {
35     let mut input_f = File::open(cli.input_file.to_str().unwrap())?;
36     let mut data = Vec::new();
37     input_f.read_to_end(&mut data)?;
38     let archive = lz77::compress(&data);
39
40     let mut output_f = File::create(cli.output_file.to_str().unwrap())?;
41     output_f.write_all(&archive)?;
42
43     return Ok(());
44 }
45
46 fn main() -> std::io::Result<()> {
47     let cli = Cli::parse();
48
49     let result = if cli.decompress {
50         run_decompressor(&cli)
51     } else {
52         run_compressor(&cli)
```

```

53     };
54
55     if let Err(error) = result {
56         match error.kind() {
57             ErrorKind::NotFound => println!("Указанный файл не найден"),
58             ErrorKind::AlreadyExists => println!("Указанный файл уже
↪ существует"),
59             _ => println!("Произошла непредвиденная ошибка"),
60         };
61     }
62
63     Ok(())
64 }

```

5.3 Содержимое файла Cargo.toml

```

1  [package]
2  name = "lab7"
3  version = "0.1.0"
4  edition = "2021"
5
6  [dependencies]
7  clap = { version = "4.0.17", features = ["derive"] }

```