Теория кодирования и сжатия информации
Лабораторная работа №8

Гущин Андрей, 431 группа, 1 подгруппа

2022 г.

1 Задача

Разработать программу осуществляющую архивацию и разархивацию текстового файла используя алгоритм Лемпеля-Зива (LZ78). 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 LZ78 заключается в создании нового кода для всех встречающихся последовательностей символов. Алгоритм использует динамический словарь.

Алгоритм состоит из следующих шагов:

1. Прочитать первый символ. Так как он ни разу не встречался, добавить его в словарь.
2. Далее необходимо найти наибольшую подстроку ps, уже находящуюся в словаре.
3. Если строка не найдена, то просто добавить отдельный символ аналогично самому первому. Иначе записать найденную подстроку и добавить в словарь конкатенацию найденной записи и следующего символа.

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.

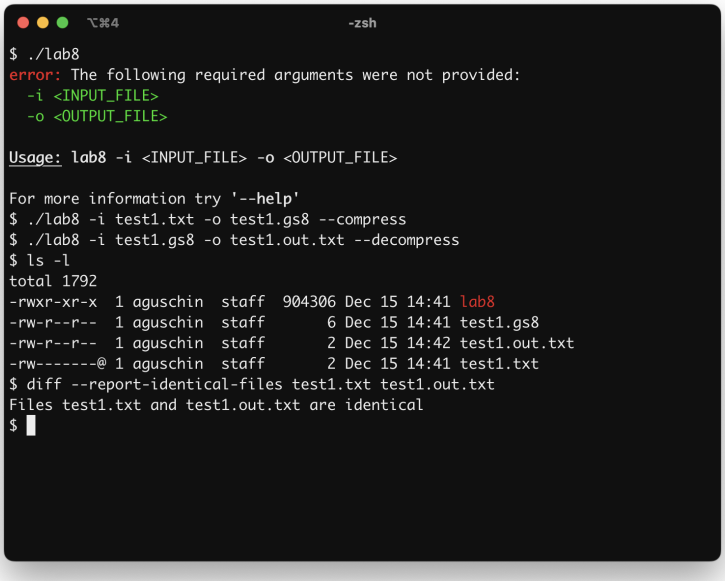

Рис. 1: Сжатие текста Тест_1.txt

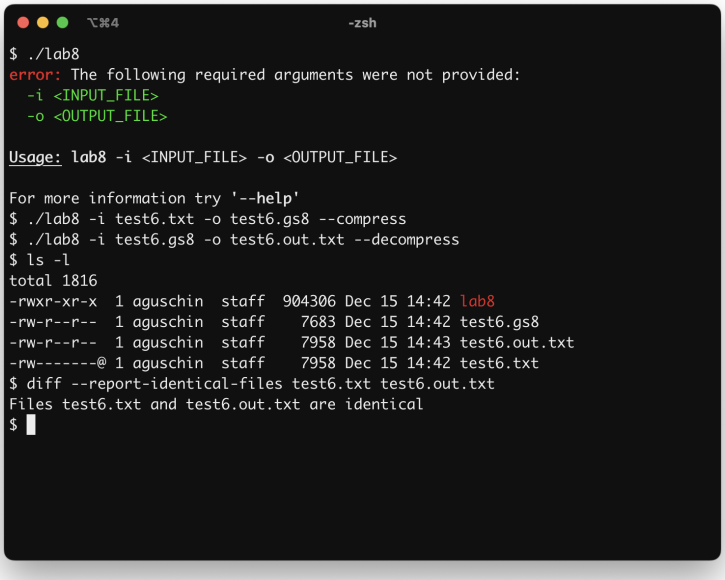

Рис. 2: Сжатие текста Тест_6.txt

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	6	0.33333
Тест_2.txt	33	99	0.33333
Тест_3.txt	2739	1227	2.23227
Тест_4.txt	330	108	3.05556
Тест_5.txt	59	177	0.33333
Тест_6.txt	7958	7683	1.03579
Тест_7.txt	138245	105015	1.31643
Тест_8.txt	574426	405459	1.41673
Тест_9.txt	2752	222	12.3964
Тест_10.txt	2814	351	8.01709

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 11.28 секунды, среднее время распаковки составило 0.02 секунд.

Таким образом, средняя скорость сжатия составила 0.26378 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 lz78.rs

```rust
const DICT_SIZE: usize = 4096;

pub fn lz_match(data: &Vec<u8>, pos1: usize, pos2: usize, length: usize)
    -> bool {
    for i in 0..length {
        if data[pos1 + i] != data[pos2 + i] {
            return false;
        }
    }
    return true;
}

```

```rust
fn lz78_encode(data: &Vec<u8>) -> Vec<(u32, u8)> {
    let mut encoded = Vec::new();
    let mut dict = Vec::new();
    let mut count = 1;

    let mut ptr = 0;
    while ptr < data.len() {
        let mut saved = None;
        let mut l = 1;
        while ptr + l < data.len() && l < 256 {
            let mut found = false;
            for i in (0..dict.len()).rev() {
                let (pos1, l1, val) = dict[i];
                if l != l1 {
                    continue;
                }
                if lz_match(data, pos1, ptr, l) {
                    found = true;
                    saved = Some((pos1, l, val));
                    l += 1;
                    break;
                }
            }
            if !found {
                break;
            }
        }
        if let Some((_, l, val)) = saved {
            let t = (val, data[ptr + l]);
            encoded.push(t);
            if dict.len() < DICT_SIZE {
                dict.push((ptr, l + 1, count));
                count += 1;
            }
            ptr += l + 1;
        } else {
            encoded.push((0, data[ptr]));
            if dict.len() < DICT_SIZE {
                dict.push((ptr, 1, count));
                count += 1;
            }
            ptr += 1;
        }
    }

    return encoded;
}

pub fn compress(data: &Vec<u8>) -> Vec<u8> {
    let mut result = Vec::new();
    let encoded = lz78_encode(data);

    for (value, byte) in &encoded {
```

```
65          let value_bytes: [u8; 4] = unsafe { std::mem::transmute(*value)
    ↪  };
66          result.push(value_bytes[0]);
67          result.push(value_bytes[1]);
68          result.push(*byte);
69      }
70
71      return result;
72  }
73
74  pub fn decompress(data: &Vec<u8>) -> Vec<u8> {
75      let mut decoded = Vec::new();
76      let mut dict = Vec::new();
77      let mut count = 1;
78      let mut ptr = 0;
79
80      let mut caret = 0;
81      while caret < data.len() {
82          let value_bytes: [u8; 4] = [data[caret + 0], data[caret + 1], 0,
    ↪  0];
83          let value: u32 = unsafe { std::mem::transmute(value_bytes) };
84          let byte = data[caret + 2];
85          caret += 3;
86
87          if value == 0 {
88              if dict.len() < DICT_SIZE {
89                  dict.push((ptr, 1, count));
90                  count += 1;
91              }
92              decoded.push(byte);
93              ptr += 1;
94          } else {
95              let (pos1, l, _) = dict[value as usize - 1];
96              for i in 0..l {
97                  decoded.push(decoded[pos1 + i]);
98              }
99              decoded.push(byte);
100             if dict.len() < DICT_SIZE {
101                 dict.push((ptr, l + 1, count));
102                 count += 1;
103             }
104             ptr += l + 1;
105         }
106     }
107
108     return decoded;
109 }
```

5.2 Содержимое файла main.rs

```
1  mod lz78;
2  use clap::Parser;
3  use std::fs::File;
4  use std::io::{Error, ErrorKind, Read, Write};
```

```rust
use std::path::PathBuf;

#[derive(Parser)]
struct Cli {
    #[arg(short)]
    input_file: PathBuf,

    #[arg(short)]
    output_file: PathBuf,

    #[arg(long, default_value_t = true)]
    compress: bool,

    #[arg(long, default_value_t = false)]
    decompress: bool,
}

fn run_decompressor(cli: &Cli) -> Result<(), Error> {
    let mut input_f = File::open(cli.input_file.to_str().unwrap())?;
    let mut archive = Vec::new();
    input_f.read_to_end(&mut archive)?;
    let data = lz78::decompress(&archive);

    let mut output_f = File::create(cli.output_file.to_str().unwrap())?;
    output_f.write_all(&data)?;

    return Ok(());
}

fn run_compressor(cli: &Cli) -> Result<(), Error> {
    let mut input_f = File::open(cli.input_file.to_str().unwrap())?;
    let mut data = Vec::new();
    input_f.read_to_end(&mut data)?;
    let archive = lz78::compress(&data);

    let mut output_f = File::create(cli.output_file.to_str().unwrap())?;
    output_f.write_all(&archive)?;

    return Ok(());
}

fn main() -> std::io::Result<()> {
    let cli = Cli::parse();

    let result = if cli.decompress {
        run_decompressor(&cli)
    } else {
        run_compressor(&cli)
    };

    if let Err(error) = result {
        match error.kind() {
            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 = "lab8"
3  version = "0.1.0"
4  edition = "2021"
5
6  [dependencies]
7  clap = { version = "4.0.17", features = ["derive"] }
```