

Теория кодирования и сжатия информации

Лабораторная работа №11

Гущин Андрей, 431 группа, 1 подгруппа

2022 г.

1 Задача

Разработать программу осуществляющую архивацию и разархивацию цифрового изображения используя алгоритм Лемпеля — Зива — Велча (LZW). Программа архивации и разархивации должны быть представлены отдельно и работать независимо друг от друга. Определить для данного шифра характеристику 1, 2 и 3. К работе необходимо прикрепить отчет и программный проект.

2 Алгоритм

Алгоритм RLE заключается в кодировании стоящих подряд одинаковых значений в виде пар (n, byte), где n — количество повторений, byte — повторяющийся элемент.

3 Тестирование

Для проверки программы были использованы тестовые изображения 4.1.04.tiff (рис. 1, 2), 4.2.01.tiff (рис. 3, 4) и ruler.512.tiff (рис. 5, 6). Можно заметить, что после распаковки архива полученный файл совпадает с исходным.

```

$ ./lab11
error: The following required arguments were not provided:
  -i <INPUT_FILE>
  -o <OUTPUT_FILE>

Usage: lab11 -i <INPUT_FILE> -o <OUTPUT_FILE>

For more information try '--help'
$
$ ./lab11 -i 4.1.04.tiff -o 4.1.04.gs11
$ ./lab11 -i 4.1.04.gs11 -o 4.1.04.out.tiff --decompress
$
$ ls -l 4.1.04.*
-rw-r--r--  1 aguschin  staff   244041 Dec 16 14:48 4.1.04.gs11
-rw-r--r--  1 aguschin  staff   196806 Dec 16 14:48 4.1.04.out.tiff
-rw-----@ 1 aguschin  staff   196748 Sep 16  2020 4.1.04.tiff
$
```

Рис. 1: Сжатие файла 4.1.04.tiff

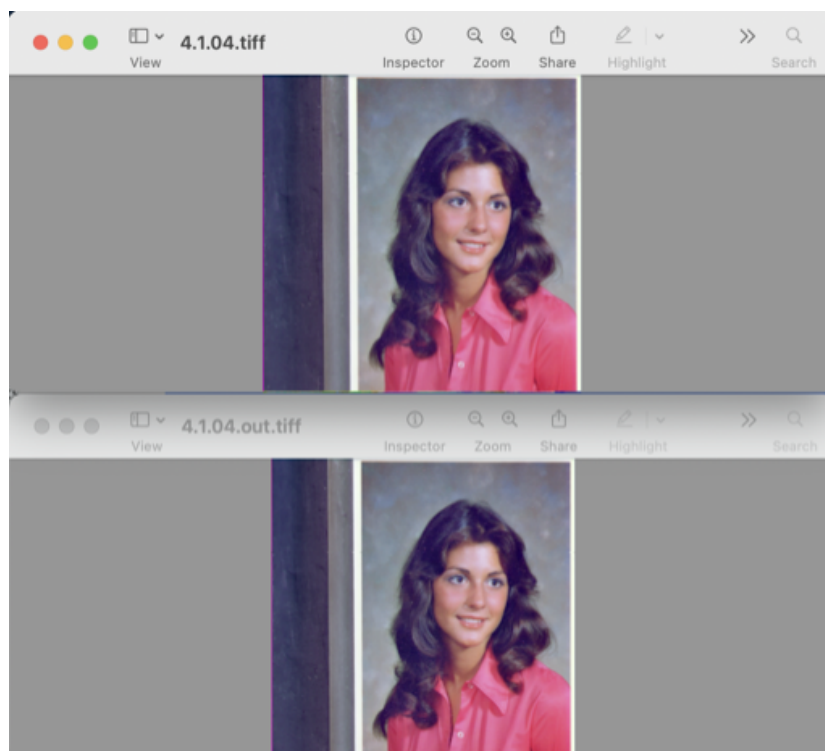


Рис. 2: Сравнение разжатого изображения с 4.1.04.tiff

```
-zsh
$ ./lab11
error: The following required arguments were not provided:
  -i <INPUT_FILE>
  -o <OUTPUT_FILE>

Usage: lab11 -i <INPUT_FILE> -o <OUTPUT_FILE>

For more information try '--help'
$
$ ./lab11 -i 4.2.01.tiff -o 4.2.01.gs11
$ ./lab11 -i 4.2.01.gs11 -o 4.2.01.out.tiff --decompress
$
$ ls -l 4.2.01.*
-rw-r--r--  1 aguschin  staff   1305870 Dec 16 14:49 4.2.01.gs11
-rw-r--r--  1 aguschin  staff    786630 Dec 16 14:50 4.2.01.out.tiff
-rw-----@ 1 aguschin  staff    786572 Sep 16 2020 4.2.01.tiff
$
```

Рис. 3: Сжатие файла 4.2.01.tiff



Рис. 4: Сравнение разжатого изображения с 4.2.01.tiff

```
-zsh
$ ./lab11
error: The following required arguments were not provided:
  -i <INPUT_FILE>
  -o <OUTPUT_FILE>

Usage: lab11 -i <INPUT_FILE> -o <OUTPUT_FILE>

For more information try '--help'
$
$ ./lab11 -i ruler.512.tiff -o ruler.512.gs11
$ ./lab11 -i ruler.512.gs11 -o ruler.512.out.tiff --decompress
$
$ ls -l ruler.512.*
-rw-r--r--  1 aguschin  staff   13423 Dec 16 14:50 ruler.512.gs11
-rw-r--r--  1 aguschin  staff  262330 Dec 16 14:51 ruler.512.out.tiff
-rw-----@ 1 aguschin  staff  262278 Sep 16  2020 ruler.512.tiff
$
```

Рис. 5: Сжатие файла ruler.512.tiff

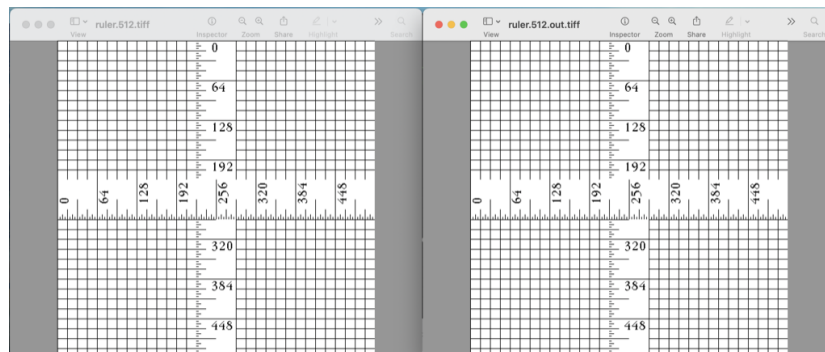


Рис. 6: Сравнение разжатого изображения с ruler.512.tiff

4 Вычисленные характеристики

4.1 Характеристика 1 (Коэффициент сжатия)

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

Название	Исходный размер, байт	Сжатый размер, байт	Коэффициент
4.1.04.tiff	196748	244041	0.80621
4.1.05.tiff	196748	239220	0.82246
4.1.06.tiff	196748	242346	0.81185
4.1.08.tiff	196748	260436	0.75546
4.2.01.tiff	786572	1305870	0.60234
4.2.03.tiff	786572	1165509	0.67487
4.2.05.tiff	786572	1039467	0.75671
4.2.07.tiff	786572	980445	0.80226
5.1.09.tiff	65670	80995	0.81079
5.1.11.tiff	65670	127582	0.51473
5.1.13.tiff	65670	13879	4.73161
5.1.14.tiff	65670	82720	0.79388
5.2.10.tiff	262278	273973	0.95731
5.3.01.tiff	1048710	1347127	0.77848
5.3.02.tiff	1048710	1243720	0.8432
boat.512.tiff	262278	366754	0.71513
gray21.512.tiff	262278	12961	20.2359
house.tiff	786572	1007154	0.78098
ruler.512.tiff	262278	13423	19.5394

Таблица 1: результаты тестирования

4.2 Характеристика 2 (Скорость сжатия)

Для тестирования скорости сжатия использовался произвольный графический файл размера 4808956 байт (≈ 4.6 мегабайта). В результате пяти последовательных запусков, среднее время запаковки файла составило 13.25 секунды, среднее время распаковки составило 0.04 секунд.

Таким образом, средняя скорость сжатия составила 354.43367 Кбайт в секунду, а средняя скорость разжатия составила 117406.15234 Кбайт в секунду (114.65445 Мбайт в секунду).

4.3 Характеристика 3 (Качество сжатия)

Качество изображения не изменилось после сжатия, так как этот алгоритм является алгоритмом сжатия без потерь.

5 Реализация

Программа реализована на языке программирования Rust с использованием библиотеки `clap` для чтения параметров командной строки, а также библиотеки `tiff` для чтения и записи `tiff` файлов. Сборка производится с помощью программы `cargo`, поставляющейся вместе с языком.

5.1 Содержимое файла `lzw.rs`

```

1  const DICT_SIZE: usize = 4096;
2
3  pub fn lz_match(data: &Vec<u8>, pos1: usize, pos2: usize, length: usize)
   ↪ -> bool {

```

```

4     for i in 0..length {
5         if data[pos1 + i] != data[pos2 + i] {
6             return false;
7         }
8     }
9     return true;
10 }
11
12 fn lzw_encode(data: &Vec<u8>) -> Vec<(u32, u8)> {
13     let mut encoded = Vec::new();
14     let mut dict = Vec::new();
15     let mut count = 1;
16
17     let mut ptr = 0;
18     while ptr < data.len() {
19         let mut saved = None;
20         let mut l = 1;
21         while ptr + l < data.len() && l < 256 {
22             let mut found = false;
23             for i in (0..dict.len()).rev() {
24                 let (pos1, l1, val) = dict[i];
25                 if l != l1 {
26                     continue;
27                 }
28                 if lz_match(data, pos1, ptr, l) {
29                     found = true;
30                     saved = Some((pos1, l, val));
31                     l += 1;
32                     break;
33                 }
34             }
35             if !found {
36                 break;
37             }
38         }
39         if let Some((_, l, val)) = saved {
40             let t = (val, data[ptr + l]);
41             encoded.push(t);
42             if dict.len() < DICT_SIZE {
43                 dict.push((ptr, l + 1, count));
44                 count += 1;
45             }
46             ptr += l + 1;
47         } else {
48             encoded.push((0, data[ptr]));
49             if dict.len() < DICT_SIZE {
50                 dict.push((ptr, 1, count));
51                 count += 1;
52             }
53             ptr += 1;
54         }
55     }
56
57     return encoded;

```

```

58 }
59
60 pub fn lzw_compress(data: &Vec<u8>) -> Vec<u8> {
61     let mut result = Vec::new();
62     let encoded = lzw_encode(data);
63
64     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 lzw_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 }

```



```

110
111 fn split_data(data: &Vec<u8>) -> (Vec<u8>, Vec<u8>, Vec<u8>) {
112     let mut r = Vec::new();
113     let mut g = Vec::new();
114     let mut b = Vec::new();
115
116     for i in 0..data.len() / 3 {
117         r.push(data[i * 3 + 0]);
118         g.push(data[i * 3 + 1]);
119         b.push(data[i * 3 + 2]);
120     }
121
122     return (r, g, b);
123 }
124
125 fn dump_u32(data: &mut Vec<u8>, val: u32) {
126     let val_bytes: [u8; 4] = unsafe { std::mem::transmute(val) };
127     data.push(val_bytes[0]);
128     data.push(val_bytes[1]);
129     data.push(val_bytes[2]);
130     data.push(val_bytes[3]);
131 }
132
133 fn read_u32(data: &Vec<u8>, offset: usize) -> u32 {
134     let val_bytes = [
135         data[offset + 0],
136         data[offset + 1],
137         data[offset + 2],
138         data[offset + 3],
139     ];
140     return unsafe { std::mem::transmute(val_bytes) };
141 }
142
143 pub fn compress_rgb(data: &Vec<u8>, dim: (u32, u32)) -> Vec<u8> {
144     let mut result = Vec::new();
145     let (r, g, b) = split_data(data);
146     let r_enc = lzw_compress(&r);
147     let g_enc = lzw_compress(&g);
148     let b_enc = lzw_compress(&b);
149
150     result.push(1);
151     dump_u32(&mut result, dim.0);
152     dump_u32(&mut result, dim.1);
153
154     dump_u32(&mut result, r_enc.len() as u32);
155     for elem in r_enc {
156         result.push(elem as u8);
157     }
158
159     dump_u32(&mut result, g_enc.len() as u32);
160     for elem in g_enc {
161         result.push(elem as u8);
162     }
163

```

```

164     dump_u32(&mut result, b_enc.len() as u32);
165     for elem in b_enc {
166         result.push(elem as u8);
167     }
168
169     return result;
170 }
171
172 pub fn compress_gray(data: &Vec<u8>, dim: (u32, u32)) -> Vec<u8> {
173     let mut result = Vec::new();
174     let enc = lzw_compress(data);
175
176     result.push(0);
177     dump_u32(&mut result, dim.0);
178     dump_u32(&mut result, dim.1);
179
180     dump_u32(&mut result, enc.len() as u32);
181     for elem in enc {
182         result.push(elem as u8);
183     }
184
185     return result;
186 }
187
188 pub fn decompress(data: &Vec<u8>) -> (Vec<u8>, (u32, u32)) {
189     let mut result = Vec::new();
190     let is_gray = data[0] == 0;
191     let width = read_u32(&data, 1);
192     let height = read_u32(&data, 5);
193
194     if is_gray {
195         let archive = Vec::from(&data[13..]);
196         let mut decoded = lzw_decompress(&archive);
197         result.append(&mut decoded);
198     } else {
199         let mut shift = 9;
200         let rsize = read_u32(&data, shift) as usize;
201         shift += 4;
202         let r_archive = Vec::from(&data[shift..shift + rsize]);
203         shift += rsize;
204
205         let gsize = read_u32(&data, shift) as usize;
206         shift += 4;
207         let g_archive = Vec::from(&data[shift..shift + gsize]);
208         shift += gsize;
209
210         let bsize = read_u32(&data, shift) as usize;
211         shift += 4;
212         let b_archive = Vec::from(&data[shift..shift + bsize]);
213
214         let r_decode = lzw_decompress(&r_archive);
215         let g_decode = lzw_decompress(&g_archive);
216         let b_decode = lzw_decompress(&b_archive);
217

```

```

218         for i in 0..r_decode.len() {
219             result.push(r_decode[i]);
220             result.push(g_decode[i]);
221             result.push(b_decode[i]);
222         }
223     }
224
225     return (result, (width, height));
226 }

```

5.2 Содержимое файла main.rs

```

1  mod lzw;
2  use clap::Parser;
3  use std::fs::File;
4  use std::io::{Error, ErrorKind, Read, Write};
5  use std::path::PathBuf;
6  use tiff;
7  use tiff::decoder::DecodingResult;
8  use tiff::encoder::colortype;
9  use tiff::ColorType;
10
11  #[derive(Parser)]
12  struct Cli {
13      #[arg(short)]
14      input_file: PathBuf,
15
16      #[arg(short)]
17      output_file: PathBuf,
18
19      #[arg(long, default_value_t = true)]
20      compress: bool,
21
22      #[arg(long, default_value_t = false)]
23      decompress: bool,
24  }
25
26  fn run_decompressor(cli: &Cli) -> Result<(), Error> {
27      let mut input_f = File::open(cli.input_file.to_str().unwrap())?;
28      let mut archive = Vec::new();
29      input_f.read_to_end(&mut archive)?;
30      let (data, dim) = lzw::decompress(&archive);
31
32      let output_f = File::create(cli.output_file.to_str().unwrap())?;
33      let mut encoder = tiff::encoder::TiffEncoder::new(output_f).unwrap();
34      if archive[0] == 0 {
35          encoder
36              .write_image:::<colortype::Gray8>(dim.0, dim.1, &data)
37              .unwrap();
38      } else {
39          encoder
40              .write_image:::<colortype::RGB8>(dim.0, dim.1, &data)
41              .unwrap();
42      }

```

```

43     return Ok(());
44 }
45
46
47 fn run_compressor(cli: &Cli) -> Result<(), Error> {
48     let input_f = File::open(cli.input_file.to_str().unwrap())?;
49     let mut decoder = tiff::decoder::Decoder::new(&input_f).unwrap();
50     let img_coded = decoder.read_image();
51     let dim = decoder.dimensions().unwrap();
52     match img_coded {
53         Ok(DecodingResult::U8(data)) => {
54             let archive = match decoder.colortype() {
55                 Ok(ColorType::RGB(_)) => lzw::compress_rgb(&data, dim),
56                 Ok(ColorType::Gray(_)) => lzw::compress_gray(&data, dim),
57                 _ => panic!("unsupported colortype"),
58             };
59             let mut output_f =
↳ File::create(cli.output_file.to_str().unwrap())?;
60             output_f.write_all(&archive)?;
61         }
62         _ => {
63             panic!("something went wrong");
64         }
65     }
66
67     return Ok(());
68 }
69
70 fn main() -> std::io::Result<()> {
71     let cli = Cli::parse();
72
73     let result = if cli.decompress {
74         run_decompressor(&cli)
75     } else {
76         run_compressor(&cli)
77     };
78
79     if let Err(error) = result {
80         match error.kind() {
81             ErrorKind::NotFound => println!("Указанный файл не найден"),
82             ErrorKind::AlreadyExists => println!("Указанный файл уже
↳ существует"),
83             _ => println!("Произошла непредвиденная ошибка"),
84         };
85     }
86
87     Ok(())
88 }

```

5.3 Содержимое файла Cargo.toml

```

1 [package]
2 name = "lab11"
3 version = "0.1.0"

```

```
4 edition = "2021"  
5  
6 [dependencies]  
7 clap = { version = "4.0.17", features = ["derive"] }  
8 tiff = "0.8.1"
```