

Теория кодирования и сжатия информации

Лабораторная работа №3

Гущин Андрей, 431 группа, 1 подгруппа

2022 г.

1 Задача

Разработать программу осуществляющую архивацию и разархивацию текстового файла используя алгоритм Шеннона. 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 Шеннона заключается в создании нового кода для каждого встречающегося в тексте символа на основе частоты встречи этого символа.

Алгоритм состоит из следующих шагов:

1. Вычислить частоты p_i всех встретившихся символов;
2. Отсортировать полученный список по невозрастанию;
3. Вычислить префиксные суммы b_i для списка частот;
4. Перевести префиксные суммы из десятичной системы счисления в двоичную;
5. В качестве кода для символа i необходимо использовать первые $\lceil -\log_2 p_i \rceil$ двоичных цифр после запятой из b_i .

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.

```

$ ./lab3
error: The following required arguments were not provided:
  -i <INPUT_FILE>
  -o <OUTPUT_FILE>

Usage: lab3 -i <INPUT_FILE> -o <OUTPUT_FILE>

For more information try '--help'
$ ./lab3 -i test1.txt -o test1.gs3 --compress
$ ./lab3 -i test1.gs3 -o test1.decomp.txt --decompress
$ ls -l
total 1832
-rwxr-xr-x  1 aguschin  staff   924466 Dec 15 06:22 lab3
-rw-r--r--  1 aguschin  staff         2 Dec 15 06:23 test1.decomp.txt
-rw-r--r--  1 aguschin  staff        13 Dec 15 06:23 test1.gs3
-rw-----@ 1 aguschin  staff         2 Dec 15 06:23 test1.txt
$ diff --report-identical-files test1.txt test1.decomp.txt
Files test1.txt and test1.decomp.txt are identical
$
```

Рис. 1: Сжатие текста Тест_1.txt

```

$ ./lab3
error: The following required arguments were not provided:
  -i <INPUT_FILE>
  -o <OUTPUT_FILE>

Usage: lab3 -i <INPUT_FILE> -o <OUTPUT_FILE>

For more information try '--help'
$ ./lab3 -i test6.txt -o test6.gs3 --compress
$ ./lab3 -i test6.gs3 -o test6.decomp.txt --decompress
$ ls -l
total 1856
-rwxr-xr-x  1 aguschin  staff   924466 Dec 15 06:22 lab3
-rw-r--r--  1 aguschin  staff       7958 Dec 15 06:24 test6.decomp.txt
-rw-r--r--  1 aguschin  staff       6342 Dec 15 06:24 test6.gs3
-rw-----@ 1 aguschin  staff       7958 Dec 15 06:24 test6.txt
$ diff --report-identical-files test6.decomp.txt test6.txt
Files test6.decomp.txt and test6.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	13	0.15385
Тест_2.txt	33	192	0.17188
Тест_3.txt	2739	2222	1.23267
Тест_4.txt	330	54	6.11111
Тест_5.txt	59	342	0.17251
Тест_6.txt	7958	6342	1.25481
Тест_7.txt	138245	91392	1.51266
Тест_8.txt	574426	363679	1.57949
Тест_9.txt	2752	352	7.81818
Тест_10.txt	2814	416	6.76442

Таблица 1: результаты тестирования

4.2 Характеристика 2 (Скорость сжатия)

Для тестирования скорости сжатия использовался произвольный двоичный файл размера 76450438 байт (≈ 73 мегабайта). В результате пяти последовательных запусков, среднее время запаковки файла составило 3.45 секунд, среднее время распаковки составило 180 секунд.

Таким образом, средняя скорость сжатия составила 21.13299 Мбайт в секунду, а средняя скорость разжатия составила 0.40505 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 shannon.rs

```
1 use std::cmp::Ordering;
2 use std::collections::HashMap;
3
4 #[derive(Debug, Clone)]
5 pub struct Weighted<T> {
6     pub weight: f32,
7     pub value: T,
8 }
9
10 #[derive(Clone, Copy, Debug, PartialEq)]
11 enum Bit {
12     Zero,
```

```

13     One,
14 }
15 impl Bit {
16     fn from_u8(x: u8) -> Bit {
17         if x == 0 {
18             Bit::Zero
19         } else {
20             Bit::One
21         }
22     }
23
24     fn to_u8(x: &Bit) -> u8 {
25         match *x {
26             Bit::Zero => 0,
27             Bit::One  => 1,
28         }
29     }
30 }
31
32 fn get_probabilities(data: &Vec<u8>) -> Vec<Weighted<u8>> {
33     let mut counts = HashMap::new();
34     data.iter().for_each(|byte| {
35         *counts.entry(*byte).or_insert(0) += 1;
36     });
37     let mut counts: Vec<(u8, u32)> = counts.drain().collect();
38     counts.sort();
39
40     let total = data.len() as f32;
41     let mut weights = Vec::new();
42     counts.iter().for_each(|(byte, count)| {
43         weights.push(Weighted {
44             value: *byte,
45             weight: *count as f32 / total,
46         });
47     });
48     return weights;
49 }
50
51 fn prefix_sum(weights: &Vec<Weighted<u8>>) -> Vec<f32> {
52     let mut pf = Vec::new();
53     for i in 0..weights.len() {
54         if i == 0 {
55             pf.push(0f32);
56         } else {
57             pf.push(weights[i - 1].weight + pf[i - 1]);
58         }
59     }
60     return pf;
61 }
62
63 fn get_code(probabilities: &Vec<Weighted<u8>>) -> HashMap<u8, Vec<Bit>> {
64     let mut code = HashMap::new();
65
66     let mut ps = probabilities.clone();

```

```

67     ps.sort_by(|a, b|
↪     a.weight.partial_cmp(&b.weight).unwrap_or(Ordering::Equal));
68     ps.reverse();
69     let prefix = prefix_sum(&ps);
70
71     for i in 0..ps.len() {
72         let l = (-ps[i].weight.log2()).ceil() as i32;
73         if l == 0 {
74             code.insert(ps[i].value, vec![Bit::Zero]);
75         } else {
76             let bits = get_bits(prefix[i], l);
77             code.insert(ps[i].value, bits);
78         }
79     }
80
81     return code;
82 }
83
84 fn get_bits(x: f32, count: i32) -> Vec<Bit> {
85     let mut bits = Vec::new();
86
87     let mut tmp = x;
88     for _ in 0..count {
89         tmp *= 2f32;
90         if tmp >= 1f32 {
91             bits.push(Bit::One);
92             tmp -= 1f32;
93         } else {
94             bits.push(Bit::Zero);
95         }
96     }
97
98     return bits;
99 }
100
101 #[derive(Debug)]
102 struct Metadata {
103     probabilities: Vec<Weighted<u8>>,
104     code: HashMap<u8, Vec<Bit>>,
105     remainder: u8,
106 }
107
108 impl Metadata {
109     fn compute(data: &Vec<u8>) -> Self {
110         let probs = get_probabilities(&data);
111         let code = get_code(&probs);
112
113         return Self {
114             probabilities: probs,
115             code,
116             remainder: 0,
117         };
118     }
119 }

```

```

120     fn load(data: &Vec<u8>) -> Self {
121         let remainder = data[0];
122         let dict_len = data[1] as usize + 1;
123         let mut probabilities = Vec::new();
124         for i in 0..dict_len {
125             let pstart = 2 + 5 * i + 1;
126             let prob = [
127                 data[pstart + 0],
128                 data[pstart + 1],
129                 data[pstart + 2],
130                 data[pstart + 3],
131             ];
132             let prob: f32 = unsafe { std::mem::transmute(prob) };
133             probabilities.push(Weighted {
134                 value: data[2 + 5 * i],
135                 weight: prob,
136             });
137         }
138         let code = get_code(&probabilities);
139
140         return Self {
141             probabilities,
142             code,
143             remainder,
144         };
145     }
146
147     fn dump(&self) -> Vec<u8> {
148         let mut result = Vec::new();
149
150         result.push(self.remainder);
151         result.push((self.probabilities.len() - 1) as u8);
152         for p in &self.probabilities {
153             result.push(p.value);
154             let bweight: [u8; 4] = unsafe { std::mem::transmute(p.weight) }
155             ↪ ;
156             for b in bweight {
157                 result.push(b);
158             }
159
160             return result;
161         }
162     }
163
164     struct BitWriter {
165         buffer: Vec<Bit>,
166         remainder: u8,
167         result: Vec<u8>,
168     }
169
170     impl BitWriter {
171         fn new() -> Self {
172             Self {

```

```

173         buffer: Vec::new(),
174         remainder: 0,
175         result: Vec::new(),
176     }
177 }
178
179 fn dump_byte(&mut self) {
180     let mut byte = 0_u8;
181     self.buffer
182         .iter()
183         .map(Bit::to_u8)
184         .enumerate()
185         .for_each(|(i, bit)| byte |= bit << i);
186     self.result.push(byte);
187     self.buffer.clear();
188 }
189
190 fn write_bit(&mut self, bit: Bit) {
191     self.buffer.push(bit);
192     if self.buffer.len() == 8 {
193         self.dump_byte();
194     }
195 }
196
197 fn write_bits(&mut self, bits: &Vec<Bit>) {
198     bits.iter().for_each(|bit| self.write_bit(*bit));
199 }
200
201 fn finish(&mut self) {
202     let remainder = 8 - self.buffer.len() as u8 % 8;
203     if remainder != 0 {
204         self.dump_byte();
205     }
206     self.remainder = remainder;
207 }
208 }
209
210 pub fn compress(data: &Vec<u8>) -> Vec<u8> {
211     let mut result = Vec::new();
212     let mut metadata = Metadata::compute(data);
213
214     let mut writer = BitWriter::new();
215     data.iter().for_each(|byte| {
216         let bits = metadata.code.get(byte).unwrap();
217         writer.write_bits(&bits);
218     });
219     writer.finish();
220     metadata.remainder = writer.remainder;
221
222     let md_dump = metadata.dump();
223     md_dump.iter().for_each(|byte| result.push(*byte));
224     writer.result.iter().for_each(|byte| result.push(*byte));
225
226     return result;

```

```

227 }
228
229 struct BitReader<'a> {
230     data: &'a Vec<u8>,
231     metadata: &'a Metadata,
232     buffer: Vec<Bit>,
233     ptr: usize,
234 }
235
236 impl<'a> BitReader<'a> {
237     fn new(data: &'a Vec<u8>, metadata: &'a Metadata) -> Self {
238         Self {
239             data,
240             metadata,
241             buffer: Vec::new(),
242             ptr: 0,
243         }
244     }
245
246     fn read_byte(&mut self) {
247         if let Some(byte) = self.data.get(self.ptr) {
248             for i in 0..=7 {
249                 let bit = byte & (1 << i);
250                 self.buffer.push(Bit::from_u8(bit));
251             }
252             if self.ptr == self.data.len() - 1 {
253                 for _ in 0..self.metadata.remainder {
254                     self.buffer.pop();
255                 }
256             }
257             self.buffer.reverse();
258             self.ptr += 1;
259         }
260     }
261
262     fn read_bit(&mut self) -> Option<Bit> {
263         if self.buffer.len() == 0 {
264             self.read_byte();
265         }
266         self.buffer.pop()
267     }
268 }
269
270 pub fn decompress(archive: &Vec<u8>) -> Vec<u8> {
271     let mut result = Vec::new();
272     let metadata = Metadata::load(archive);
273
274     let data = archive[2 + metadata.probabilities.len() * 5..].to_vec();
275     let mut reader = BitReader::new(&data, &metadata);
276
277     let mut run = Vec::new();
278     while let Some(bit) = reader.read_bit() {
279         run.push(bit);
280         'checker: for p in &metadata.code {

```



```

281         if p.1.len() != run.len() {
282             continue;
283         }
284         for i in 0..run.len() {
285             if p.1[i] != run[i] {
286                 continue 'checker';
287             }
288         }
289         result.push(*p.0);
290         run.clear();
291         break;
292     }
293 }
294
295 return result;
296 }

```

5.2 Содержимое файла main.rs

```

1  mod shannon;
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 = shannon::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 = shannon::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 = "lab3"
3  version = "0.1.0"
4  edition = "2021"
5
6  [dependencies]
7  clap = { version = "4.0.17", features = ["derive"] }

```