

# Теория кодирования и сжатия информации

## Лабораторная работа №4

Гущин Андрей, 431 группа, 1 подгруппа

2022 г.

### 1 Задача

Разработать программу осуществляющую архивацию и разархивацию текстового файла используя алгоритм адаптивного кода Хаффмана. 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 адаптивного кодирования Хаффмана заключается в обновлении дерева кодирования для каждого нового введенного символа таким образом, чтобы чаще встречающиеся символы в потоки получали наиболее эффективный код.

Основные свойства дерева, которые предстоит поддерживать при кодировании для получения корректного кода — это неявная нумерация и инвариантность. Неявная нумерация — все узлы пронумерованы по возрастанию по уровню, а также слева направо. Инвариантность — для любого веса  $w$  все листья веса  $w$  предшествуют всем внутренним узлам веса  $w$ .

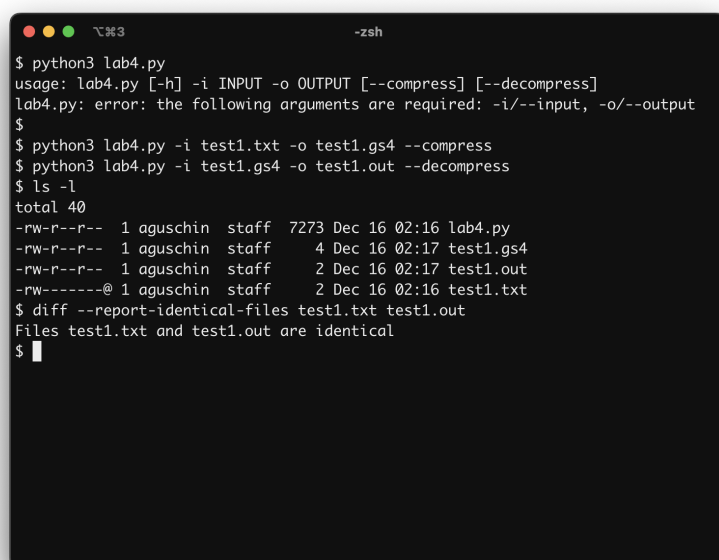
Алгоритм состоит из следующих шагов:

1. Создать дерево из единственного узла NYT (Not Yet Transferred, Ещё не передано). Этот узел всегда будет иметь вес равный нулю. В выходной поток передадим один бит 0, который в данный момент будет означать начало кодирования;
2. После получения нового символа из потока, закодируем передадим его в неизменном виде, после чего добавим в дерево, сохраняя свойства дерева;
3. Если следующий символ ещё не встречался до этого, то в выходной поток передадим сначала код, соответствующий NYT, а затем сам символ в неизменном виде. Далее добавим этот символ в дерево, сохраняя свойства дерева;
4. Если символ уже встречался, то увеличим его вес, а также веса всех его предков, сохраняя при этом свойства дерева.

Перед увеличением веса любого из узлов, необходимо сначала поменять его местами с лидером блока. Это действие не нужно производить только в том случае, если лидером является прямой предок узла. Блок — узлы одного веса. Лидер — узел блока с самым большим неявным номером.

### 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.



```
$ python3 lab4.py
usage: lab4.py [-h] -i INPUT -o OUTPUT [--compress] [--decompress]
lab4.py: error: the following arguments are required: -i/--input, -o/--output
$
$ python3 lab4.py -i test1.txt -o test1.gs4 --compress
$ python3 lab4.py -i test1.gs4 -o test1.out --decompress
$ ls -l
total 40
-rw-r--r--  1 aguschin  staff  7273 Dec 16 02:16 lab4.py
-rw-r--r--  1 aguschin  staff    4 Dec 16 02:17 test1.gs4
-rw-r--r--  1 aguschin  staff    2 Dec 16 02:17 test1.out
-rw-----@ 1 aguschin  staff    2 Dec 16 02:16 test1.txt
$ diff --report-identical-files test1.txt test1.out
Files test1.txt and test1.out are identical
$
```

Рис. 1: Сжатие текста Тест\_1.txt

```
$ python3 lab4.py
usage: lab4.py [-h] -i INPUT -o OUTPUT [--compress] [--decompress]
lab4.py: error: the following arguments are required: -i/--input, -o/--output
$
$ python3 lab4.py -i test6.txt -o test6.gs4 --compress
$ python3 lab4.py -i test6.gs4 -o test6.out --decompress
$ ls -l
total 64
-rw-r--r-- 1 aguschin staff 7273 Dec 16 02:16 lab4.py
-rw-r--r-- 1 aguschin staff 5654 Dec 16 02:18 test6.gs4
-rw-r--r-- 1 aguschin staff 7958 Dec 16 02:18 test6.out
-rw-----@ 1 aguschin staff 7958 Dec 16 02:17 test6.txt
$ diff --report-identical-files test6.txt test6.out
Files test6.txt and test6.out 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	4	0.5
Тест_2.txt	33	55	0.6
Тест_3.txt	2739	1910	1.43403
Тест_4.txt	330	65	5.07692
Тест_5.txt	59	105	0.5619
Тест_6.txt	7958	5654	1.4075
Тест_7.txt	138245	85621	1.61462
Тест_8.txt	574426	339607	1.69144
Тест_9.txt	2752	346	7.95376
Тест_10.txt	2814	368	7.64674

Таблица 1: результаты тестирования

### 4.2 Характеристика 2 (Скорость сжатия)

Для тестирования скорости сжатия использовался тестовый файл Тест\_7.txt файл размера 138245 байт ( $\approx 0.13$  мегабайта). В результате пяти последовательных запусков, среднее время запаковки файла составило 19.48

секунд, среднее время распаковки составило 9.46 секунд.

Таким образом, средняя скорость сжатия составила 0.00677 Мбайт в секунду, а средняя скорость разжатия составила 0.01389 Мбайт в секунду.

## 5 Реализация

Программа реализована на языке программирования Python с использованием стандартной библиотеки `argparse` для чтения параметров командной строки.

### 5.1 Содержимое файла `lab4.py`

```
1  import argparse
2
3
4  def to_bin(x):
5      res = []
6      for _ in range(8):
7          res.append(x % 2)
8          x //= 2
9      return res
10
11
12 def from_bits(bits):
13     res = 0
14     for i, bit in enumerate(bits):
15         res |= bit * (1 << i)
16     return res
17
18
19 class Leaf:
20     def __init__(self, value, weight=0):
21         self.weight = weight
22         self.value = value
23         self.parent = None
24
25     def __repr__(self):
26         return f"Leaf({self.value}, w={self.weight})"
27         return f"Leaf({self.value})"
28     __str__ = __repr__
29
30
31 class Node:
32     def __init__(self, left, right, weight=0):
33         self.weight = weight
34         self.left = left
35         self.right = right
36
37     def make_swap(self):
38         self.left, self.right = self.right, self.left
39
40     def replace(self, n1, n2):
41         if self.left == n1:
```

```

42         self.left = n2
43     elif self.right == n1:
44         self.right = n2
45
46     def __repr__(self):
47         return f"Node(w={self.weight}, {self.left}, {self.right})"
48         return f"Node({self.left}, {self.right})"
49     __str__ = __repr__
50
51
52 def find_leader(tree, fweight):
53     if tree.weight == fweight:
54         return tree
55
56     if isinstance(tree, Node):
57         if tree.right.weight == fweight:
58             return tree.right
59         if tree.left.weight == fweight:
60             return tree.left
61
62     leader = find_leader(tree.right, fweight)
63     if leader is not None:
64         return leader
65     return find_leader(tree.left, fweight)
66 elif isinstance(tree, Leaf):
67     if tree.weight == fweight:
68         return tree
69     else:
70         return None
71
72
73 def swap_nodes(node1, node2):
74     if node1.parent == node2.parent:
75         node1.parent.make_swap()
76     else:
77         node1.parent.replace(node1, node2)
78         node2.parent.replace(node2, node1)
79         node1.parent, node2.parent = node2.parent, node1.parent
80
81
82 def dfs(tree, code=None, fvalue=None, fweight=None):
83     if code is None:
84         code = []
85
86     if isinstance(tree, Node):
87         lf = l, lp = dfs(tree.left, code + [0], fvalue, fweight)
88         rf = r, rp = dfs(tree.right, code + [1], fvalue, fweight)
89         if lp:
90             return lf
91         elif rp:
92             return rf
93     elif isinstance(tree, Leaf):
94         res = True
95         if fvalue is not None:

```

```

96         res = res and fvalue == tree.value
97         if fweight is not None:
98             res = res and fweight == tree.weight
99         return (tree, code), res
100     return (tree, []), False
101
102
103 def get_root(tree):
104     cur = tree
105     while cur.parent is not None:
106         cur = cur.parent
107     return cur
108
109
110 def increase_weights(node):
111     parent = node.parent
112     if parent is not None:
113         sibling = parent.right
114         leader = find_leader(get_root(node), node.weight)
115         if leader != parent and leader != node:
116             swap_nodes(leader, node)
117             parent = node.parent
118             node.weight += 1
119             increase_weights(parent)
120     else:
121         node.weight += 1
122
123
124 def insert_char(tree, char):
125     (nyt, _), _ = dfs(tree, fweight=0)
126     parent = Node(None, None)
127     new = Leaf(char, 0)
128
129     if nyt.parent is not None:
130         nyt.parent.replace(nyt, parent)
131     parent.parent = nyt.parent
132     nyt.parent = parent
133     new.parent = parent
134     parent.left = nyt
135     parent.right = new
136
137     increase_weights(new)
138     return get_root(parent)
139
140
141 def find_char(tree, char):
142     (node, code), res = dfs(tree, fvalue=char)
143     return res, node, code
144
145
146 class BitWriter:
147     def __init__(self):
148         self._buffer = []
149         self._run = []

```

```

150
151     def _get_byte(self):
152         byte = 0
153         for power, bit in enumerate(self._run):
154             byte |= bit << power
155         return byte
156
157     def write_bit(self, bit):
158         if len(self._run) == 8:
159             byte = self._get_byte()
160             self._buffer.append(byte)
161             self._run.clear()
162             self._run.append(bit)
163
164     def write_bits(self, bits):
165         for bit in bits:
166             self.write_bit(bit)
167
168     def get_buffer(self):
169         if len(self._run) > 0:
170             byte = self._get_byte()
171             r = len(self._run)
172             return (self._buffer.copy() + [byte], r)
173         else:
174             return (self._buffer.copy(), 8)
175
176
177 class BitReader:
178     def __init__(self, data, r):
179         self._buffer = []
180         self._data = data
181         self._ptr = 0
182         self._r = r
183
184     def _next_byte(self):
185         run = []
186         if self._ptr < len(self._data):
187             byte = self._data[self._ptr]
188             while byte != 0:
189                 run.append(byte % 2)
190                 byte //= 2
191             while len(run) != 8:
192                 run.append(0)
193             if self._ptr == len(self._data) - 1:
194                 run = run[:self._r]
195             self._buffer.extend(run[::-1])
196             self._ptr += 1
197             return len(run) != 0
198         return False
199
200     def read_bit(self):
201         if len(self._buffer) == 0:
202             if not self._next_byte():
203                 return None

```

```

204         return self._buffer.pop()
205
206
207     def compress(data):
208         root = Leaf(0, 0)
209
210         writer = BitWriter()
211         writer.write_bit(0)
212         for char in data:
213             inside, node, code = find_char(root, char)
214             if not inside:
215                 (_, nyt_code), _ = dfs(root, fweight=0)
216                 writer.write_bits(nyt_code)
217                 writer.write_bits(to_bin(char))
218                 root = insert_char(root, char)
219             else:
220                 writer.write_bits(code)
221                 increase_weights(node)
222                 root = get_root(node)
223         compressed, r = writer.get_buffer()
224         return [r] + compressed
225
226
227     def decompress(archive):
228         r = archive[0]
229         data = archive[1:]
230         root = Leaf(0, 0)
231
232         reader = BitReader(data, r)
233         cur_node = root
234         result = []
235         while True:
236             bit = reader.read_bit()
237             if bit is None:
238                 break
239
240             if isinstance(cur_node, Node):
241                 if bit == 0:
242                     cur_node = cur_node.left
243                 elif bit == 1:
244                     cur_node = cur_node.right
245
246             if isinstance(cur_node, Leaf):
247                 # Был прочитан NYT
248                 if cur_node.weight == 0:
249                     bits = []
250                     for _ in range(8):
251                         bits.append(reader.read_bit())
252                     char = from_bits(bits)
253                     root = insert_char(root, char)
254                 else:
255                     char = cur_node.value
256                     increase_weights(cur_node)
257                     root = get_root(cur_node)

```



```

258         result.append(char)
259         cur_node = root
260     return result
261
262
263 def main():
264     parser = argparse.ArgumentParser()
265     parser.add_argument('-i', '--input', required=True)
266     parser.add_argument('-o', '--output', required=True)
267     parser.add_argument('--compress', default=True, action='store_true')
268     parser.add_argument('--decompress', default=False,
269         ↪ action='store_true')
270     args = parser.parse_args()
271
272     with open(args.input, "rb") as f:
273         data = f.read()
274
275     if args.decompress:
276         data = list(data)
277         decompressed = decompress(data)
278         with open(args.output, "wb") as f:
279             f.write(bytearray(decompressed))
280     elif args.compress:
281         archive = compress(data)
282         with open(args.output, "wb") as f:
283             f.write(bytearray(archive))
284
285 if __name__ == "__main__":
286     main()
287

```