



МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования

«МИРЭА - Российский технологический университет»

РТУ МИРЭА

Отчет по выполнению практического задания №6

Тема: Кодирование и сжатие данных методами без потерь

Дисциплина Структуры и алгоритмы обработки данных

Выполнил студент

Анисимов Д.Н.

группа

ИКБО-24-20

Москва 2021

Содержание

Содержание	2
Цель работы	3
Отчет по заданию.	3
Задание 1 Исследование алгоритмов сжатия на примерах	3
Индивидуальный вариант:	3
Алгоритм решения:	4
Задание 2 Разработать программы сжатия и восстановления текста методами Хаффмана и Шеннона – Фано.	7
Кодирование строки “Фамилия Имя Отчество” по алгоритму Хаффмана... 8	
Описание алгоритма:	10
Код программы:	11
Результат тестирования:	18
Общий вывод	25
Список информационных источников	25

Цель работы

Получить знания алгоритмов кодирования и сжатия данных методами без потерь и навыки их применения.

Отчет по заданию.

Задание 1 Исследование алгоритмов сжатия на примерах

- 1) Выполнить каждую задачу варианта, представив алгоритм решения в виде таблицы и указав результат сжатия. Примеры оформления решения представлены в Приложении 1 этого документа.
- 2) Описать процесс восстановления сжатого текста.
- 3) Сформировать отчет, включив задание, вариант задания, результаты выполнения задания варианта.

Индивидуальный вариант:

Вариант	Закодировать фразу методами Шеннона– Фано	Сжатие данных по методу Лемпеля– Зива LZ77 Используя двухсимвольный алфавит (0, 1) закодировать следующую фразу:	Закодировать следующую фразу, используя код LZ78
5	Прибавь к ослиной голове еще одну, получишь две. Но сколько б ни было ослов, они и двух не свяжут слов.	10100010010101000 1011	какатанекатанекатата

Алгоритм решения:

1. Для метода Шеннона-Фано:

Закодировать фразу «Прибавь к ослиной голове ещё одну, получишь две. Но сколько б ни было ослов, они и двух не свяжут слов.», используя метод Шеннона–Фано.

Таблица 1.

Символ	Кол- во	1-я цифра	2-я цифра	3-я цифра	4-я цифра	5-я цифра	6-я цифра	7-я цифра	Код	Кол- во бит
Пробел	19	0	0	0					000	57
о	14	0	0	1					001	42
в	7	0	1	0					010	21
л	7	0	1	1	0				0110	28
и	6	0	1	1	1				0111	24
н	6	1	0	0	0				1000	24
с	5	1	0	0	1				1001	20
е	4	1	0	1	0	0			10100	20
у	4	1	0	1	0	1			10101	20
б	3	1	0	1	1	0			10110	15
ь	3	1	0	1	1	1			10111	15
к	3	1	1	0	0	0			11000	15
д	3	1	1	0	0	1			11001	15
п	2	1	1	0	1	0			11010	10
,	2	1	1	0	1	1	0		110110	12
.	2	1	1	0	1	1	1		110111	12
р	1	1	1	1	0	0	0		111000	6
а	1	1	1	1	0	0	1	0	1110010	7
й	1	1	1	1	0	0	1	1	1110011	7
г	1	1	1	1	0	1	0		111010	6
щ	1	1	1	1	0	1	1	0	1110110	7
ё	1	1	1	1	0	1	1	1	1110111	7
ч	1	1	1	1	1	0	0		111100	6

ш	1	1	1	1	1	0	1	0	1111010	7
ы	1	1	1	1	1	0	1	1	1111011	7
х	1	1	1	1	1	1	0	0	1111100	7
я	1	1	1	1	1	1	0	1	1111101	7
ж	1	1	1	1	1	1	1	0	1111110	7
т	1	1	1	1	1	1	1	1	1111111	7
										438

Незакодированная фраза – $103 \cdot 8$ бит = 824 бит.

Закодированная фраза – 438 бит.

Каждый код символа уникален, значит раскодировать можно, зная коды символов и заменяя их обратно.

2. Для метода Лемпеля – Зива LZ77

Для сжатия двоичного кода:

101000100101010001011

Таблица 2.

Содержимое окна (сжимаемый текст)	Содержимое буфера	Код
101000100101010001011	1	<0,0,1>
101000100101010001011	0	<0,0,0>
101000100101010001011	100	<2,2,0>
101000100101010001011	01001	<4,4,1>
101000100101010001011	010100	<2,5,0>
101000100101010001011	01011	<6,4,1>

Результат сжатия: (<0,0,1>, <0,0,0>, <2,2,0>, <4,4,1>, <2,5,0>, <6,4,1>)

Чтобы восстановить строку, нужно последовательно пройти по коду (<символов назад, сколько пройти, символ в конце прохода>).

3. Для метода Лемпеля – Зива LZ78

Для фразы: какатанекатанекатата

Таблица 3.

Содержимое словаря	Содержимое считаной строки	Код
	к	<0, к>
к	а	<0, а>
к, а	ка	<1, а>
к, а, ка	т	<0, т>
к, а, ка, т	ан	<2, н>
к, а, ка, т, ан	е	<0, е>
к, а, ка, т, ан, е	кат	<3, т>
к, а, ка, т, ан, е, кат	ане	<5, е>
к, а, ка, т, ан, е, кат, ане	ката	<6, а>
к, а, ка, т, ан, е, кат, ане, ката	та	<4, а>

Результат сжатия: (<0, к>, <0, а>, <1, а>, <0, т>, <2, н>, <0, е>, <3, т>, <5, е>, <6, а>, <4, а>)

Чтобы восстановить строку, проходимся по коду, строя словарь и беря из него значения.

Задание 2 Разработать программы сжатия и восстановления текста методами Хаффмана и Шеннона – Фано.

- 1) Реализовать и отладить программы.
- 2) Сформировать отчет по разработке каждой программы в соответствии с требованиями.

- По методу Шеннона-Фано привести: постановку задачи, описать алгоритм формирования префиксного дерева и алгоритм кодирования, декодирования, код и результаты тестирования. Рассчитать коэффициент сжатия. Сравнить результат сжатия вашим алгоритмом с результатом любого архиватора.

- по методу Хаффмана выполнить и отобразить результаты выполнения всех требований, предъявленных в задании и оформить разработку программы: постановка, подход к решению, код, результаты тестирования.

Для выполнения работы необходимо выполнить следующие действия:

- 2.1 Построить таблицу частот встречаемости символов в исходной строке символов для чего сформировать алфавит исходной строки и посчитать количество вхождений (частот) символов и их вероятности появления.
- 2.2 Отсортировать алфавит в порядке убывания частот появления символов.
- 2.3 Построить дерево кодирования Хаффмана.
- 2.4 Присвоить ветвям коды.
- 2.5 Определить коды символов.
- 2.6 Провести кодирование исходной строки.
- 2.7 Рассчитать коэффициенты сжатия относительно кодировки ASCII и относительно равномерного кода.
- 2.8 Рассчитать среднюю длину полученного кода и его дисперсию.

Кодирование строки “Фамилия Имя Отчество” по алгоритму Хаффмана.

2.1, 2.2 Таблица частот встречаемости символов.

Таблица 4.

Алфавит	и	а	в	л	н	о	< >
Кол. вх.	5	2	2	2	2	2	2
Вероятн.	0.232	0.077	0.077	0.077	0.077	0.077	0.077

Таблица 5.

Алфавит	А	Д	Н	е	к	м	с	ч
Кол. вх.	1	1	1	1	1	1	1	1
Вероятн.	0.038	0.038	0.038	0.038	0.038	0.038	0.038	0.038

2.3, 2.4 Дерево кодирования Хаффмана с кодами, присвоенными ветвям.

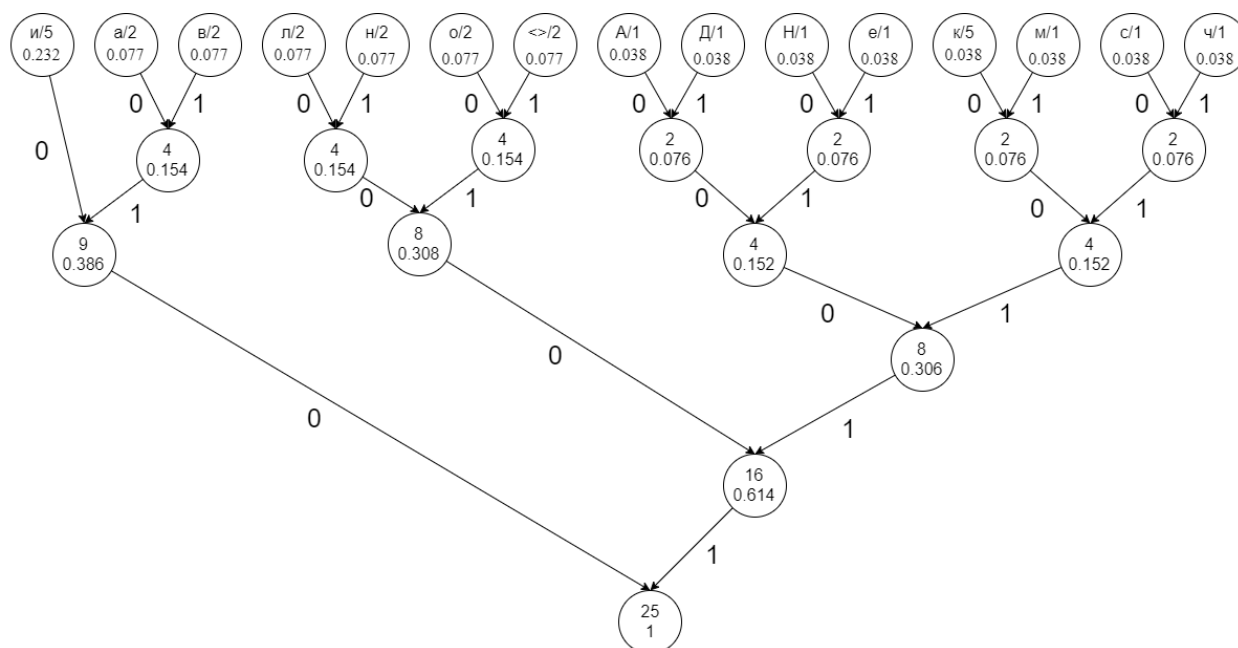


Рис. 1. Дерево Хаффмана для ФИО.

2.5 Коды символов:

и – 00, а – 010, в – 011, л – 1000, н – 1001, о – 1010, < > - 1011, А – 11000, Д – 11001, Н – 11010, е – 11011, к – 11100, м – 11101, с – 11110, ч – 11111.

2.6 Кодирование исходной строки:

11000 1001 00 11110 00 11101 1010 011 1011 11001 010 1001 00 00 1000 1011
11010 00 11100 1010 1000 010 11011 011 00 11111

2.7 Рассчитаем коэффициент сжатия относительно использования кодировки ASCII (8 бит/символ).

$$L_{ASCII} = 8 * 26 = 208 \text{ бит.}$$

$$L_{Huff} = 5 * 2 + 2 * 2 * 3 + 2 * 4 * 4 + 5 * 8 = 94 \text{ бит.}$$

Следовательно, коэффициент сжатия будет равен

$$K_{сж} = \frac{L_{ASCII}}{L_{Huff}} \approx 2,213$$

Коэффициент сжатия относительно равномерного кода (5 бит/символ, т. к. у нас всего 26 символов) будет равен

$$K_{сж} = \frac{5 * 26}{L_{Huff}} = \frac{130}{94} \approx 1,383$$

2.8 Рассчитаем среднюю длину полученного кода по формуле

$l_{cp} = \sum_s p_s * l_s$, где s — множество символов алфавита; p_s — вероятность появления символа; l_s — количество бит в коде символа.

Для полученного кода средняя длина будет равна

$$l_{cp} = 0,232 * 2 + 0,077 * 3 * 2 + 0,077 * 4 * 4 + 0,038 * 5 * 8 = 3,678 \text{ бит/символ.}$$

Дисперсия рассчитывается по формуле

$$\delta = \sum_s p_s (l_s - l_{cp})^2$$

Для полученного кода дисперсия будет равна

$$\delta = 0,232 * (2 - 3,678)^2 + 0,077 * (3 - 3,678)^2 * 2 + 0,077 * (4 - 3,678)^2 * 4 + 0,038 * (5 - 3,678)^2 * 8 = 1,287261.$$

Описание алгоритма:

Создание словарей частоты и вероятности символов:

Присваиваем каждому символу количество его вхождений в тексте для частоты и частоту, разделённую на количество символов в тексте для вероятности.

Создание префиксного дерева Шеннона-Фано:

Имеем узел-корень со словарём, отсортированным по убыванию с повторяющимися ключами: вероятность и значением равным соответствующему ей символу.

Разделяем словарь по вероятностям на две группы, так чтобы разделение было максимально ровным.

Создаём новые 2 узла и перемещаем словарь левой группы в первый узел и словарь правой группы во второй узел.

Создаём ссылки у узла, из которого мы брали словарь на созданные узлы, первый новый узел становится левым сыном, а второй новый узел становится правым сыном.

Повторяем разделение и создание ссылок у новых узлов, если в их словарях больше 1 элемента.

Создание префиксного дерева Хаффмана:

Присваиваем по отдельности элементы словаря ключей - вероятность и значений – символ, новым узлам и кладём эти узлы в приоритетную очередь, в которой всегда в начале узел с наименьшей суммой вероятностей элементов его словаря.

Пока не останется только один узел в очереди, который станет корнем:

- Вытаскиваем из очереди 2 узла.
- Объединяем их словари в один, создаём новый узел и присваиваем ему этот словарь, создаём ссылки у нового узла – левый сын — это первый вытасканный из очереди узел, а правый сын – второй вытасканный из очереди.
- Кладём этот новый узел в очередь.

Кодирование сообщения:

Получаем строку и корень дерева Шеннона-Фано или Хаффмана.

Рекурсивно проходим по дереву сверху вниз, переходя по ссылкам на сыновей и если мы прошли на ссылку левого сына, то добавляем в передаваемый код 1, а если на ссылку правого сына, то добавляем в передаваемый код 0.

Когда у узла не будет сыновей, то мы берём из словаря этого узла символ и тот код который получился в итоге рекурсивного обхода и заменяем все вхождения необходимого символа в исходной строке на этот код.

Декодирование сообщения:

Получаем закодированную строку и корень дерева Шеннона-Фано или Хаффмана.

Создаём словарь символ: код, и заполняем его рекурсивно обходя дерево сверху вниз, добавляя в передаваемый код 1, если мы прошли влево и добавляя в передаваемый код 0, если мы прошли вправо, когда мы доберёмся до узла без ссылок на сыновей, то создадим новый элемент в словаре – рекурсивно полученный код и символ из словаря узла.

Используя полученный словарь, декодируем сообщение, постепенно проходя по строке, создавая подстроки и сравнивая их с кодами в словаре и если подстрока равна коду из словаря, то заменяем её на нужный символ.

Код программы:

Листинг 1. Структура узла деревьев Шеннона-Фано и Хаффмана.

```
1 struct node
2 {
3     multimap<float, char, greater<float>> probabilities; // Словарь с повтором ключей
4     (вероятность : символ), всегда отсортирован по убыванию
5     node* left = NULL; // Левое поддерево
6     node* right = NULL; // Правое поддерево
7 };
```

Листинг 2. Создание словаря Символ:Частота.

```
8 map<char, int> build_frequency(string input) {
9     map<char, int> m; // Словарь (символ : кол-во вхождений в тексте)
10    for (auto symbol: input) // по всем символам строки
11    {
12        if (m.find(symbol) == m.end()) // если символа нет в словаре
13        {
14            m[symbol] = 0; // создаём элемент
15        }
16        m[symbol] += 1; // увеличиваем счётчик вхождений символа
17    }
18    return m;
19 }
```

Листинг 3. Создание словаря Вероятность:Символ, с возможностью повторения ключей и сортируемого по убыванию значений ключей.

```
20 multimap<float, char, greater<float>> build_probability(map<char, int> m, int length) {
21     multimap<float, char, greater<float>> p; // Словарь с повтором ключей (вероятность
22     : символ), всегда отсортирован по убыванию
```

```

23     for (auto item: m) // по всем элементам словаря вхождений символа в тексте
24     {
25         p.insert(make_pair((float)(item.second) / length), item.first)); // ввод
26     элемента словаря вероятностей
27     }
28     return p;
29 }

```

Листинг 4. Создание нижних узлов корня дерева Шеннона-Фано.

```

30 void shannon_leaf_creator(node* parent) {
31     float prob1 = 0, prob2, prob1buf = 0, prob2buf = 0, diversitybuf = 1; //
32     переменные для расчёта групп вероятностей
33     for (auto item : parent->probabilities) // по элементам словаря вероятностей узла
34     {
35         prob2buf += item.first; // увеличиваем вероятность правой группы
36     }
37     prob2 = prob2buf;
38     for (auto item : parent->probabilities) // по элементам словаря вероятностей узла
39     {
40         prob1 += item.first; // увеличиваем группу вероятностей слева
41         prob2 -= item.first; // уменьшаем группу вероятностей справа
42         if (diversitybuf < abs(prob2 - prob1)) // если разделение групп
43     вероятностей
44     {
45         // было
46         более ровным одну итерацию раньше
47         prob2 = prob2buf; // вернём разделение
48         обратно
49         prob1 = prob1buf; //
50         break; // и завершим
51     }
52     else
53     {
54         prob1buf = prob1; // запоминаем
55     разделение
56         prob2buf = prob2; //
57         diversitybuf = abs(prob2buf - prob1buf); // и насколько оно было
58     ровным
59     }
60 }
61 multimap<float, char, greater<float>> l; // словарь вероятностей символов с
62 повтором ключей для левого сына
63 multimap<float, char, greater<float>> r; // словарь вероятностей символов с
64 повтором ключей для правого сына
65 float p1 = 0; // переменная для проверки что мы разделили словарь правильно
66 for (auto item : parent->probabilities) // по элементам словаря вероятностей узла
67 {
68     if (p1 != prob1) // пока у левого сына словарь не станет быть как левая
69     группа вероятностей
70     {
71         l.insert(make_pair(item.first, item.second)); // добавляем элементы
72     левой группы в словарь левого сына
73         p1 += item.first; // увеличиваем вероятность для проверки разделения
74     }
75     else
76     {
77         r.insert(make_pair(item.first, item.second)); // добавляем элементы
78     правой группы в словарь правого сына
79     }
80 }
81 node* left_node = new node;
82 node* right_node = new node;
83 left_node->probabilities = l;
84 right_node->probabilities = r;

```

```

85     parent->left = left_node;
86     parent->right = right_node;
87     parent->probabilities.clear(); // очищаем словарь узла-родителя
88     if (l.size() > 1) // пока словарь для левого сына имеет больше 2 элементов
89     {
90         shannon_leaf_creator(parent->left); // разделяем ещё дальше словарь у
91     левого сына
92     }
93     if (r.size() > 1) // пока словарь для правого сына имеет больше 2 элементов
94     {
95         shannon_leaf_creator(parent->right); // разделяем ещё дальше словарь у
96     правого сына
97     }
98 }

```

Листинг 5. Создание дерева Шеннона-Фано.

```

99 node* shannon_tree_creator(multimap<float, char, greater<float>> p) {
100     node* tree = new node; // корень дерева
101     tree->probabilities = p; // словарь вероятностей корня
102     shannon_leaf_creator(tree); // строим поддеревья
103     return tree;
104 }

```

Листинг 6. Компаратор для приоритетной очереди.

```

105 class Comparator
106 {
107 public:
108     bool operator()(node* l, node* r) { //Сравниваем общую частоту в словарях узлов
109         float probs1 = 0, probs2 = 0;
110         for (auto item : l->probabilities)
111         {
112             probs1 += item.first;
113         }
114         for (auto item : r->probabilities)
115         {
116             probs2 += item.first;
117         }
118         return probs1 > probs2; // Узлы с наименьшей общей вероятностью в вершине
119     очереди
120 }
121 };

```

Листинг 7. Создание дерева Хаффмана.

```

122 node* huffman_tree_creator(multimap<float, char, greater<float>> p) {
123     priority_queue<node*, vector<node*>, Comparator> pq; // приоритетная очередь узлов
124     с сортировкой по возрастанию вероятности
125     for (auto item : p) // наполняем очередь узлами символов с вероятностью из словаря
126     {
127         node* current_node = new node;
128         current_node->probabilities.insert(make_pair(item.first, item.second));
129         pq.push(current_node);
130     }
131     while (pq.size() != 1) // пока не останется корень
132     {
133         node* leftNode = pq.top(); // элемент из начала очереди с минимальной общей
134     вероятностью
135         pq.pop(); // удаляем его из очереди
136         node* rightNode = pq.top(); // первый элемент из начала очереди с
137     минимальной общей вероятностью
138         pq.pop(); // удаляем его из очереди
139         multimap<float, char, greater<float>> sum_of_probs; //объединяем словари
140     меньших узлов в один

```

```

141         for (auto item : leftNode->probabilities)
142         {
143             sum_of_probs.insert(make_pair(item.first, item.second));
144         }
145         for (auto item : rightNode->probabilities)
146         {
147             sum_of_probs.insert(make_pair(item.first, item.second));
148         }
149         node* newNode = new node; // новый узел - родитель объединённых
150         newNode->probabilities = sum_of_probs;
151         newNode->left = leftNode; // меньший из 2 объединённых узлов - левый сын у
152     родителя
153         newNode->right = rightNode; // больший из 2 объединённых узлов - правый сын
154     у родителя
155         pq.push(newNode); // кладем родителя в очередь
156     }
157     node* tree = pq.top(); // верхний узел в очереди - корень
158     return tree;
159 }

```

Листинг 8. Замена вхождения строки на другую строку.

```

160 static void ReplaceAll(string& str, const string& from, const string& to) { // для замены
161     подстроки на строку
162     size_t start_pos = 0;
163     while ((start_pos = str.find(from, start_pos)) != string::npos) {
164         str.replace(start_pos, from.length(), to);
165         start_pos += to.length();
166     }
167 }

```

Листинг 9. Рекурсивное кодирование исходной строки по дереву Шеннона-Фано или Хаффмана.

```

168 void unified_encode_part(string& input, node* node, string code) {
169     if (node->left == NULL && node->right == NULL)
170     {
171         string to_replace(1, node->probabilities.begin()->second); // из символа в
172     строку
173         ReplaceAll(input, to_replace, code); // заменяем все символы в строке на их
174     код
175         cout << "Замена '" << to_replace << "' на " << code << "\n";
176     }
177     else if (node->left != NULL && node->right != NULL)
178     {
179         unified_encode_part(input, node->left, code + '1'); // кодируем символы
180     левого поддерева, добавляя 1 в код
181         unified_encode_part(input, node->right, code + '0'); // кодируем символы
182     правого поддерева, добавляя 0 в код
183     }
184     else if (node->left != NULL && node->right == NULL)
185     {
186         unified_encode_part(input, node->left, code + '1');
187     }
188     else
189     {
190         unified_encode_part(input, node->right, code + '0');
191     }
192 }

```

Листинг 10. Запуск кодирования.

```

193 string unified_encoder(string input, node* tree) {
194     string result = input;

```

```

195 unified_encode_part(result, tree, ""); // кодируем строку, читая узлы сверху вниз
196 return result;
197 }

```

Листинг 11. Создание словаря для декодирования по дереву Шеннона-Фано или Хаффмана.

```

198 void create_dict_part(map<string, string>& dict, node* node, string code) {
199     if (node->left == NULL && node->right == NULL)
200     {
201         string to_replace(1, node->probabilities.begin()->second); // из символа в
202 строку
203         dict[code] = to_replace; // присваиваем коду значение символа
204     }
205     else if (node->left != NULL && node->right != NULL)
206     {
207         create_dict_part(dict, node->right, code + '0'); // в код для правых
208 поддеревьев дописывается 0
209         create_dict_part(dict, node->left, code + '1'); // в код для левых
210 поддеревьев дописывается 1
211     }
212     else if (node->right != NULL && node->left == NULL)
213     {
214         create_dict_part(dict, node->right, code + '0');
215     }
216     else
217     {
218         create_dict_part(dict, node->left, code + '1');
219     }
220 }

```

Листинг 12. Декодирование по дереву Шеннона-Фано или Хаффмана.

```

221 string unified_decoder(string input, node* tree) {
222     string result = input;
223     map<string, string> dict; // словарь (код : символ, который мы закодировали)
224     create_dict_part(dict, tree, ""); // строим словарь
225     for (size_t i = 0; i < input.length(); i++) // двигаем подстроку чтобы
226     { //
227 раскодировать сообщение правильно
228         for (size_t j = 1; j < (input.length() / 2) - 1; j++)
229         {
230             if (result[i] == NULL) // конец строки
231             {
232                 return result; // выход из циклов
233             }
234             string sub = result.substr(i, j); // подстрока
235             if (dict.find(sub) != dict.end()) // подстрока есть как код в словаре
236             {
237                 result.replace(result.find(sub), sub.size(), dict.find(sub)-
238 >second); // заменяем код на символ
239                 //cout << "Замена " << sub << " на " << dict.find(sub)->second
240 << "\n";
241                 break;
242             }
243         }
244     }
245     return result;
246 }

```

Листинг 13. Чтение файла в одну строку.

```

247 string fileReader(string fileName) { // читает файл
248     ifstream file(fileName); // в кодировке ANSI
249     stringstream ss;
250     ss << file.rdbuf();

```

```

251         return ss.str();
    }

```

Листинг 14. Запись строки в файл.

```

252 void fileWriter(string fileName, string info) {
253     ofstream file; // записывает в файл
254     file.open(fileName); // в кодировке UTF-8
255     file << info;
256     file.close();
257 }

```

Листинг 15. Функция main.

```

258 int main() {
259     SetConsoleCP(1251);
260     SetConsoleOutputCP(1251);
261     int option = 3, choice = 3;
262     string str;
263     while (choice != 0)
264     {
265         cout << "Что кодировать?\n" << "1. ФИО\n" << "2. input.txt\n";
266         cin >> option;
267         switch (option)
268         {
269             case 1:
270             {
271                 str = "Анисимов Даниил Николаевич";
272                 break;
273             }
274             case 2:
275             {
276                 str = fileReader("input.txt"); // читаем исходный файл
277                 break;
278             }
279             default:
280                 str = fileReader("input.txt"); // читаем исходный файл
281                 break;
282             }
283             map<char, int> f = build_frequency(str); // строим словарь частоты
284             упоминаний символов
285             multimap<float, char, greater<float>> p = build_probability(f,
286             str.length()); // строим словарь вероятностей символов
287             cout << "Содержимое исходной строки:\n\n" << str << "\n\n";
288             cout << "Вероятность : символ" << "\n";
289             for (auto item : p)
290             {
291                 cout << item.first << " : " << item.second << "\n";
292             }
293
294             cout << "По какому методу кодировать и декодировать?" << "\n";
295             cout << "1. Шеннон-фано\n" << "2. Хаффман\n" << "0. Выход\n";
296             cin >> choice;
297             switch (choice)
298             {
299                 case 1:
300                 {
301                     node* tree = shannon_tree_creator(p); // строим префиксное дерево
302                     шеннона-фано для кодирования
303                     cout << "\n" << "Какие символы на какие коды заменяем:" << "\n";
304
305                     string encoded = unified_encoder(str, tree);
306                     cout << "\n" << "Закодированная информация:" << "\n" << encoded <<
307                     "\n";

```



```

308         cout << "\n" << "Коэффициент сжатия: " << (float(float(str.length() *
309 8) / encoded.length())) << "\n";
310         fileWriter("output.txt", encoded); // записываем закодированный файл
311         string encoded_get = fileReader("output.txt");
312
313         string decoded = unified_decoder(encoded_get, tree);
314         cout << "\n" << "Декодированная информация:" << "\n" << decoded <<
315 "\n\n";
316         break;
317     }
318     case 2:
319     {
320         node* tree = huffman_tree_creator(p); // строим префиксное дерево
321 хаффмана для кодирования
322         cout << "\n" << "Какие символы на какие коды заменяем:" << "\n";
323
324         string encoded = unified_encoder(str, tree);
325         cout << "\n" << "Закодированная информация:" << "\n" << encoded <<
326 "\n";
327         cout << "\n" << "Коэффициент сжатия: " << (float(float(str.length() *
328 8) / encoded.length())) << "\n";
329         fileWriter("output.txt", encoded); // записываем закодированный файл
330         string encoded_get = fileReader("output.txt");
331
332         string decoded = unified_decoder(encoded_get, tree);
333         cout << "\n" << "Декодированная информация:" << "\n" << decoded <<
334 "\n\n";
335         break;
336     }
337     case 0:
338         break;
339     default:
340         break;
341     }
342 }
343 return 0;
344 }

```

Результат тестирования:

```
Что кодировать?
1. ФИО
2. input.txt
2
Содержимое исходной строки:
Прибавь к ослиной голове ещё одну, получишь две. Но сколько б ни было ослов, они и двух не свяжут слов.
Вероятность : символ
0.184466 : 
0.135922 : о
0.0679612 : в
0.0679612 : л
0.0582524 : и
0.0485437 : н
0.0485437 : с
0.038835 : е
0.038835 : у
0.0291262 : б
0.0291262 : д
0.0291262 : к
0.0291262 : ь
0.0194175 : ,
0.0194175 : .
0.00970874 : ё
0.00970874 : Н
0.00970874 : П
0.00970874 : а
0.00970874 : г
0.00970874 : ж
0.00970874 : й
0.00970874 : п
0.00970874 : р
0.00970874 : т
0.00970874 : х
0.00970874 : ч
0.00970874 : ш
0.00970874 : щ
0.00970874 : ы
0.00970874 : я
По какому методу кодировать и раскодировать?
1. Шеннон-фано
2. Хаффман
0. Выход
```

Рис. 2. Вывод таблицы вероятностей символов для текста из входного файла
Все вероятности вычислены правильно.

```

По какому методу кодировать и раскодировать?
1. Шеннон-фано
2. Хаффман
0. Выход
1

Какие символы на какие коды заменяем:
Замена ' ' на 111
Замена 'о' на 110
Замена 'в' на 101
Замена 'л' на 1001
Замена 'и' на 1000
Замена 'н' на 0111
Замена 'с' на 0110
Замена 'е' на 01011
Замена 'у' на 01010
Замена 'б' на 01001
Замена 'д' на 01000
Замена 'к' на 00111
Замена 'ь' на 00110
Замена ',' на 001011
Замена '.' на 001010
Замена 'ё' на 0010011
Замена 'Н' на 0010010
Замена 'П' на 001000
Замена 'а' на 0001111
Замена 'г' на 0001110
Замена 'ж' на 0001101
Замена 'й' на 0001100
Замена 'п' на 0001011
Замена 'р' на 0001010
Замена 'т' на 000100
Замена 'х' на 0000111
Замена 'ч' на 0000110
Замена 'ш' на 000010
Замена 'щ' на 0000011
Замена 'ы' на 0000010
Замена 'я' на 000000

Закодированная информация:
00100000010101000010010001111101001101110011111110011010011000011111000011001110001110110100111010101011111010110000011
001001111111001000011101010001011111000101111010010101000001101000000010001101110100010101011001001011011101100
0111110100100110001111101110100111101111000111010010000010100111011110011010011101010010111111001111000111100011101000
10101010000011111101110101111101101010000000011010101000010011101101001110101001010

Коэффициент сжатия: 1.85586

Декодированная информация:
Прибавь к ослиной голове ещё одну, получишь две. Но сколько б ни было ослов, они и двух не свяжут слов.

```

Рис. 3. Результат кодирования по Шеннону-Фано, декодирования и вычисления коэффициента сжатия для текста из входного файла

Кодирование прошло успешно, декодирование тоже прошло успешно.

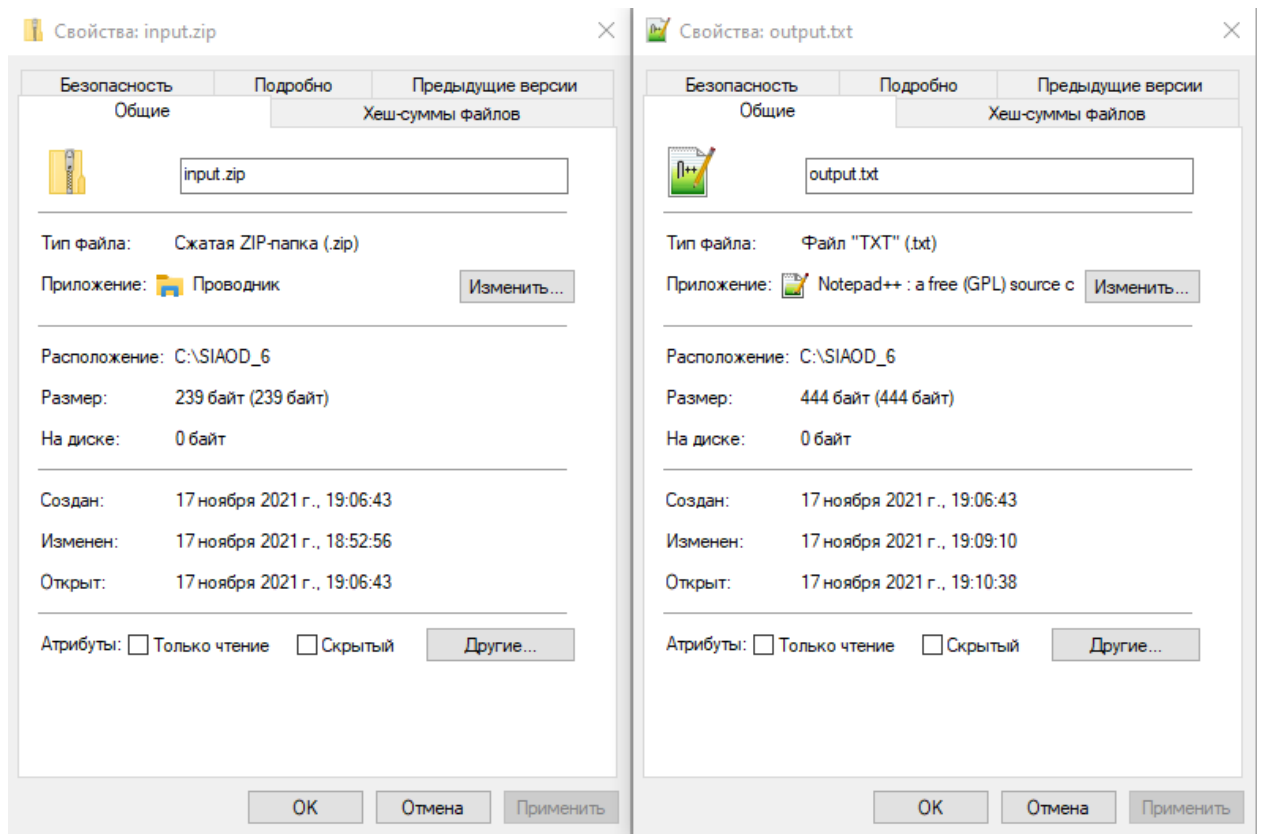


Рис. 4. Сравнение результата кодирования Шеннона-Фано с результатом архиватора по алгоритму Deflate

Сжатие архиватора по Deflate в 2 раза сильнее чем по алгоритму Шеннона-Фано.

```

По какому методу кодировать и раскодировать?
1. Шеннон-фано
2. Хаффман
0. Выход
2

Какие символы на какие коды заменяем:
Замена ' ' на 11
Замена 'н' на 1011
Замена 'и' на 1010
Замена 'ь' на 10011
Замена 'д' на 10010
Замена 'к' на 10001
Замена 'б' на 10000
Замена 'в' на 0111
Замена 'л' на 0110
Замена 'о' на 010
Замена 'е' на 00111
Замена ',' на 001101
Замена 'г' на 0011001
Замена 'й' на 0011000
Замена '.' на 001011
Замена 'т' на 0010101
Замена 'х' на 0010100
Замена 'я' на 0010011
Замена 'щ' на 0010010
Замена 'ч' на 0010001
Замена 'Н' на 0010000
Замена 'ё' на 0001111
Замена 'р' на 0001110
Замена 'ш' на 0001101
Замена 'ы' на 0001100
Замена 'у' на 00010
Замена 'п' на 0000111
Замена 'ж' на 0000110
Замена 'П' на 0000101
Замена 'а' на 0000100
Замена 'с' на 00000

Закодированная информация:
0000101000111010101000000010001111001111100011101000000011010101011010001100011001100100110011110011110010010
0001111110101001010110001000110111000011101001100001000100011010000110110011111001001110011110010011100100000101100000100
010100110100111000101011100001110111010111000000011000110010110100000001100100111001101101011101011101011100100111000
10001010011101100111110000001100100110000110000100010101110000001100100111001011

Коэффициент сжатия: 1.86848

Декодированная информация:
Прибавь к ослиной голове ещё одну, получишь две. Но сколько б ни было ослов, они и двух не свяжут слов.

```

Рис. 5. Результат кодирования по Хаффману, декодирования и вычисления коэффициента сжатия для текста из входного файла

Кодирование прошло успешно, декодирование тоже прошло успешно.

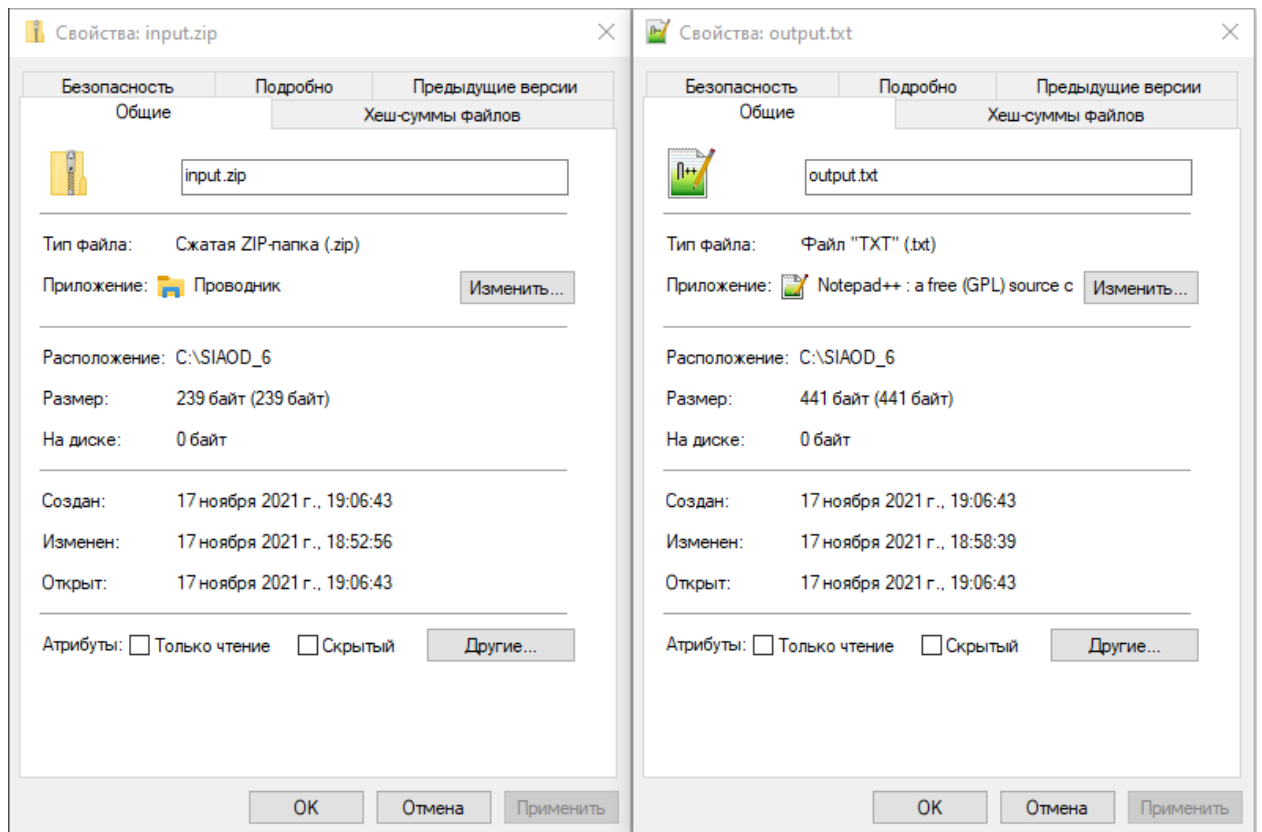


Рис. 6. Сравнение результата кодирования Хаффмана с результатом архиватора по алгоритму Deflate

Сжатие архиватора по Deflate опять в 2 раза сильнее чем по алгоритму Хаффмана.

```

Что кодировать?
1. ФИО
2. input.txt
1
Содержимое исходной строки:
Анисимов Даниил Николаевич

Вероятность : символ
0.230769 : и
0.0769231 : а
0.0769231 : в
0.0769231 : л
0.0769231 : н
0.0769231 : о
0.0769231 :
0.0384615 : А
0.0384615 : Д
0.0384615 : Н
0.0384615 : е
0.0384615 : к
0.0384615 : м
0.0384615 : с
0.0384615 : ч
По какому методу кодировать и раскодировать?
1. Шеннон-фано
2. Хаффман
0. Выход
1
Какие символы на какие коды заменяем:
Замена 'и' на 111
Замена 'а' на 110
Замена 'в' на 1011
Замена 'л' на 1010
Замена 'н' на 100
Замена 'о' на 011
Замена ' ' на 0101
Замена 'А' на 01001
Замена 'Д' на 01000
Замена 'Н' на 00111
Замена 'е' на 00110
Замена 'к' на 0010
Замена 'м' на 00011
Замена 'с' на 00010
Замена 'ч' на 0000

Закодированная информация:
0100110011110001011100011011101101010001101001111111010010100111111001001110101100011010111110000

Коэффициент сжатия: 2.12245

Декодированная информация:
Анисимов Даниил Николаевич

```

Рис. 7. Результат кодирования по Шеннону-Фано, декодирования и вычисления коэффициента сжатия для ФИО

Кодирование прошло успешно, декодирование тоже прошло успешно.

```

Что кодировать?
1. ФИО
2. input.txt
1
Содержимое исходной строки:
Анисимов Даниил Николаевич

Вероятность : символ
0.230769 : и
0.0769231 : а
0.0769231 : в
0.0769231 : л
0.0769231 : н
0.0769231 : о
0.0769231 :
0.0384615 : А
0.0384615 : Д
0.0384615 : Н
0.0384615 : е
0.0384615 : к
0.0384615 : м
0.0384615 : с
0.0384615 : ч
По какому методу кодировать и декодировать?
1. Шеннон-фано
2. Хаффман
0. Выход
2

Какие символы на какие коды заменяем:
Замена 'н' на 111
Замена 'в' на 110
Замена 'и' на 10
Замена 'о' на 0111
Замена 'а' на 0110
Замена 'л' на 0101
Замена 'с' на 01001
Замена 'ч' на 01000
Замена ' ' на 0011
Замена 'А' на 00101
Замена 'к' на 00100
Замена 'м' на 00011
Замена 'д' на 00010
Замена 'Н' на 00001
Замена 'е' на 00000

Закодированная информация:
001011111001001100001101111100011000100110111101001010011000011000100011101010110000001101001000

Коэффициент сжатия: 2.16667

Декодированная информация:
Анисимов Даниил Николаевич

```

Рис. 8. Результат кодирования по Хаффману, декодирования и вычисления коэффициента сжатия для ФИО

Кодирование прошло успешно, декодирование тоже прошло успешно.

Общий вывод

Я получил знания алгоритмов кодирования и сжатия данных методами без потерь и навыки их применения.

Список информационных источников

1. Кораблин Ю.П. Структуры и алгоритмы обработки данных. Часть 1 [Электронный ресурс]: учебно методическое пособие / Ю.П.Кораблин, В.П.Сыромятников, Л.А. Скворцова – М.: РТУ МИРЭА, 2020. – 1 электрон. опт. диск (CD-ROM)