



МИНОБРНАУКИ РОССИИ
*Федеральное государственное бюджетное образовательное учреждение высшего
образования*
«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Отчет по выполнению практического задания №8.1

Тема:

Алгоритмы кодирования и сжатия данных

Дисциплина: «Структуры и алгоритмы обработки данных»

Выполнил студент: Васильев Б.А.

Группа: ИКБО-20-23

Москва 2024

СОДЕРЖАНИЕ

ЦЕЛЬ РАБОТЫ	3
ХОД РАБОТЫ	3
Задание 1	3
Формулировка задачи	3
Метод Шеннона-Фано	4
Метод Лемпеля-Зива LZ78	5
Задание 2	7
МЕТОД ШЕННОНА-ФАНО.....	7
Формулировка задачи	7
Описание подхода к решению	7
Код программы.....	8
Результаты тестирования	11
МЕТОД ХАФФМАНА	12
Формулировка задачи	12
Описание подхода к решению	12
Код программы.....	13
Результаты тестирования	16
ВЫВОД.....	17
СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ	17

ЦЕЛЬ РАБОТЫ

Познакомиться с алгоритмами кодирования и сжатия данных.

ХОД РАБОТЫ

Задание 1

Формулировка задачи

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

Рисунок 1 – Индивидуальный вариант задания

Выполнить каждую задачу варианта, представив алгоритм решения в виде таблицы и указав результат сжатия.

Метод Шеннона-Фано

Ана, дэус, рики, паки,
Дормы кормы
констунтаки,
Дэус дэус канадэус – бац!

Рисунок 2 – Текст для кодирования

Символ	Количество вхождений	1-я цифра	2-я цифра	3-я цифра	4-я цифра	5-я цифра	6-я цифра	7-я цифра	Код	Сумма длин кодов
пробел	8	0	1	1					011	24
а	6	0	1	0					010	18
к	6	0	0	1	1				0011	24
,	5	0	0	1	0				0010	20
у	5	0	0	0	1				0001	20
с	5	0	0	0	0				0000	20
н	4	1	1	1	1				1111	16
э	4	1	1	1	0				1110	16
и	4	1	1	0	1				1101	16
д	3	1	1	0	0	1			11001	15
р	3	1	1	0	0	0			11000	15
\n	3	1	0	1	1	1			10111	15
о	3	1	0	1	1	0			10110	15
Д	2	1	0	1	0	1			10101	10
м	2	1	0	1	0	0			10100	10
ы	2	1	0	0	1	1			10011	10
т	2	1	0	0	1	0	1		100101	12
А	1	1	0	0	1	0	0		100100	6
п	1	1	0	0	0	1	1		100011	6
–	1	1	0	0	0	1	0		100010	6
б	1	1	0	0	0	0	1		100001	6
ц	1	1	0	0	0	0	0	1	1000001	7
!	1	1	0	0	0	0	0	0	1000000	7
	73									314

Рисунок 3 – Индивидуальный вариант задания

Незакодированная фраза – $73 \cdot 8$ бит = 584 бит

Закодированная фраза – 314 бит

Метод Лемпеля-Зива LZ77

0100101010010000101

Рисунок 4 – Последовательность для кодирования

Таблица 1 – Последовательное кодирование подпоследовательностей

Содержимое окна (сжимаемый текст)	Содержимое упреждающего буфера	Код назначенный последовательности
	0100101010010000101	
0	100101010010000101	0
10	0101010010000101	10
01	01010010000101	11
010	10010000101	100
100	10000101	101
1000	0101	110
0101		111

Таблица 2 — Итоги кодирования

Исходный текст	0100101010010000101 0.10.01.010.100.1000.0101
LZ-код	0.10.11.110.100.1010.1001
R	2 3
Вводимые коды	- 10 11 100 101 110 111

Метод Лемпеля-Зива LZ78

кукуркурекурекун

Рисунок 5 – Индивидуальный вариант задания

Таблица 3 – Кодирование с помощью метода LZ78

Словарь	Считываемое содержимое	Код
	к	<0, к>
к = 1	у	<0, у>
к = 1, у = 2	ку	<1, у>
к = 1, у = 2, ку = 3	р	<0, р>
к = 1, у = 2, ку = 3, р = 4	кук	<3, к>
к = 1, у = 2, ку = 3, р = 4, кук = 5	ур	<2, р>
к = 1, у = 2, ку = 3, р = 4, кук = 5, ур = 6	е	<0, е>
к = 1, у = 2, ку = 3, р = 4, кук = 5, ур = 6, е = 7	кур	<3, р>
к = 1, у = 2, ку = 3, р = 4, кук = 5, ур = 6, е = 7, кур = 8	ек	<7, к>
к = 1, у = 2, ку = 3, р = 4, кук = 5, ур = 6, е = 7, кур = 8, ек = 9	ун	<2, н>
к = 1, у = 2, ку = 3, р = 4, кук = 5, ур = 6, е = 7, кур = 8, ек = 9, ун = 10	EOF	<0, EOF>

Результат:

<0, к><0, у><1, у><0, р><3, к><2, р><0, е><3, р><7, к><2, н><0, EOF>

Задание 2

МЕТОД ШЕННОНА-ФАНО

Формулировка задачи

Разработать алгоритм и реализовать программу сжатия текста алгоритмом Шеннона-Фано. Разработать алгоритм и программу восстановления сжатого текста. Выполнить тестирование программы на текстовом файле. Определить процент сжатия.

Описание подхода к решению

Метод Шеннона-Фано представляет собой алгоритм для построения префиксного кода, направленного на минимизацию средней длины кодов для кодируемых сообщений. В основе метода лежит принцип деления символов на две группы, примерно равные по суммарной вероятности или частоте появления. Этот процесс начинается с сортировки всех символов в порядке убывания частот, после чего множество символов делится на две группы так, чтобы суммы частот в этих группах были как можно ближе друг к другу.

Далее каждой группе присваивается свой префикс: обычно группе с более высокой суммарной частотой назначается префикс "0", а группе с меньшей — "1". Затем процесс повторяется для каждой из полученных групп: каждая группа снова делится на две подгруппы с равными суммами частот, и каждой подгруппе назначается следующий бит в коде. Этот рекурсивный процесс продолжается до тех пор, пока не будут достигнуты отдельные символы, для которых таким образом и будет определен уникальный префиксный код.

Метод Шеннона-Фано позволяет получить эффективное представление символов за счет того, что более частым символам назначаются более короткие коды, а менее частым — более длинные. Однако, в отличие от метода Хаффмана, метод Шеннона-Фано не всегда гарантирует минимально возможную длину кода для всех случаев, так как деление на группы по частотам может быть неточным, что иногда приводит к незначительно большему объему данных.

Код программы

Реализуем код приложения на языке программирования C++ (рис. 6-8).

```
1  #include <string>
2  #include <vector>
3  #include <map>
4  #include <iostream>
5  #include <algorithm>
6  #include <fstream>
7  #include <bitset>
8
9  using namespace std;
10
11 struct Symbol {
12     char character;
13     double probability;
14     string code;
15
16     Symbol(char ch, double prob, const string& c) : character(ch), probability(prob), code(c) {}
17 };
18
19 bool compareByProbability(const Symbol& a, const Symbol& b) {
20     return a.probability > b.probability;
21 }
22
23 void shannonFanoCode(vector<Symbol>& symbols, int start, int end) {
24     if (start >= end) return;
25
26     int splitIndex = start;
27     double totalProbLeft = 0, totalProbRight = 0;
28
29     for (int i = start; i <= end; i++) {
30         totalProbRight += symbols[i].probability;
31     }
32
33     double diff = totalProbRight;
34     for (int i = start; i <= end; i++) {
35         totalProbLeft += symbols[i].probability;
36         totalProbRight -= symbols[i].probability;
37
38         if (abs(totalProbLeft - totalProbRight) < diff) {
39             diff = abs(totalProbLeft - totalProbRight);
40             splitIndex = i;
41         }
42     }
43
44     for (int i = start; i <= splitIndex; i++) {
45         symbols[i].code += "0";
46     }
47     for (int i = splitIndex + 1; i <= end; i++) {
48         symbols[i].code += "1";
49     }
50
51     shannonFanoCode(symbols, start, splitIndex);
52     shannonFanoCode(symbols, splitIndex + 1, end);
53 }
54
55 string encodeText(string& text, map<char, string>& encodingMap) {
56     string encodedText;
57     for (char ch : text) {
58         encodedText += encodingMap.at(ch);
59     }
60     return encodedText;
61 }
```

Рисунок 6 – код программы (часть 1)


```

63 string decodeText(const string& encodedText, map<string, char>& decodingMap) {
64     string decodedText;
65     string temp;
66     for (char bit : encodedText) {
67         temp += bit;
68         if (decodingMap.find(temp) != decodingMap.end()) {
69             decodedText += decodingMap.at(temp);
70             temp.clear();
71         }
72     }
73     return decodedText;
74 }
75
76 void writeBinaryFile(const string& filename, const string& bitString) {
77     ofstream outputFile(filename, ios::binary);
78     if (!outputFile) {
79         cerr << "Error opening output file." << endl;
80         return;
81     }
82
83     for (size_t i = 0; i < bitString.size(); i += 8) {
84         bitset<8> byte(bitString.substr(i, 8));
85         outputFile.put(static_cast<char>(byte.to_ulong()));
86     }
87     outputFile.close();
88 }
89
90 string readBinaryFile(const string& filename) {
91     ifstream inputFile(filename, ios::binary);
92     if (!inputFile) {
93         cerr << "Error opening input file." << endl;
94         return "";
95     }
96
97     string bitString;
98     char byte;
99     while (inputFile.get(byte)) {
100         bitset<8> bits(byte);
101         bitString += bits.to_string();
102     }
103
104     inputFile.close();
105     return bitString;
106 }
107
108 size_t getFileSize(const string& filename) {
109     ifstream file(filename, ios::binary | ios::ate);
110     return file.tellg();
111 }
112
113 double calculateCompressionRatio(const string& inputFile, const string& encodedFilename) {
114     return static_cast<double>(getFileSize(inputFile)) / getFileSize(encodedFilename);
115 }
116
117 int main() {
118     setlocale(LC_ALL, "ru_RU.UTF-8");
119     string inputText;
120     ifstream inputFile("input.txt");
121
122     if (!inputFile.is_open()) {
123         cout << "Error opening file." << endl;
124         return 1;

```

Рисунок 7 – код программы (часть 2)

```

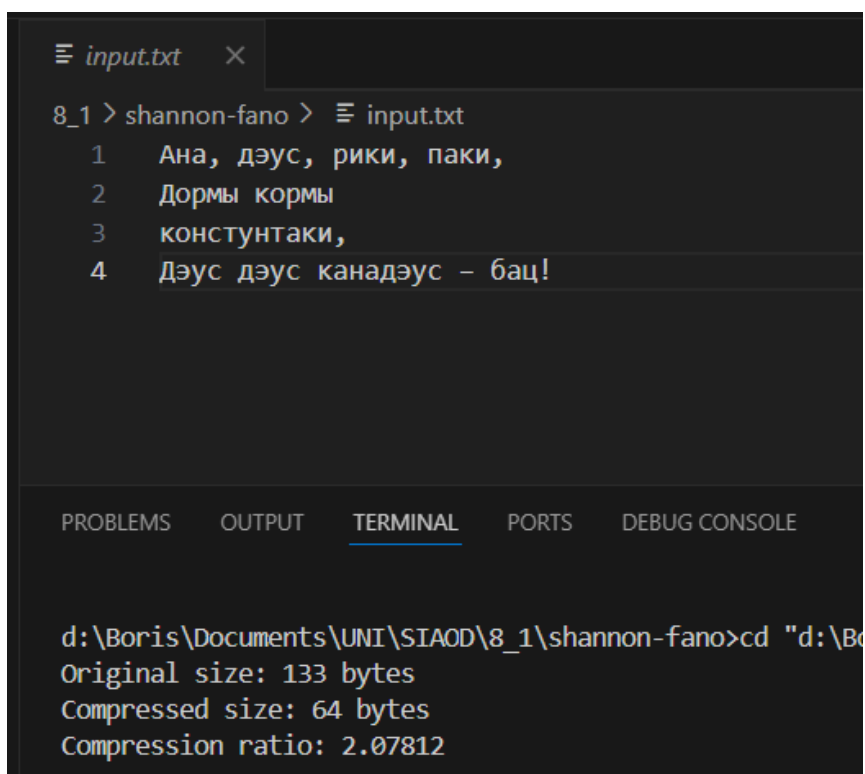
125     }
126     getline(inputFile, inputText, '\0');
127     inputFile.close();
128
129     map<char, int> frequencyMap;
130     for (char ch : inputText) {
131         frequencyMap[ch]++;
132     }
133
134     vector<Symbol> symbols;
135     for (const auto& pair : frequencyMap) {
136         symbols.emplace_back(pair.first, static_cast<double>(pair.second) / inputText.size(), "");
137     }
138
139     sort(symbols.begin(), symbols.end(), compareByProbability);
140
141     shannonFanoCode(symbols, 0, symbols.size() - 1);
142
143     map<char, string> encodingMap;
144     map<string, char> decodingMap;
145
146
147     for (const auto& symbol : symbols) {
148         encodingMap[symbol.character] = symbol.code;
149         decodingMap[symbol.code] = symbol.character;
150     }
151
152     string encodedText = encodeText(inputText, encodingMap);
153
154     writeBinaryFile("encoded.bin", encodedText);
155     ofstream encodedFile("encoded.txt");
156     encodedFile << encodedText;
157     encodedFile.close();
158
159     string readEncodedText = readBinaryFile("encoded.bin");
160     string decodedText = decodeText(encodedText, decodingMap);
161     ofstream decodedFile("decoded.txt");
162     decodedFile << decodedText;
163     decodedFile.close();
164
165     double compressionRatio = calculateCompressionRatio("input.txt", "encoded.bin");
166     cout << "Compression ratio: " << compressionRatio << endl;
167     return 0;
168 }

```

Рисунок 8 – код программы (часть 3)

Результаты тестирования

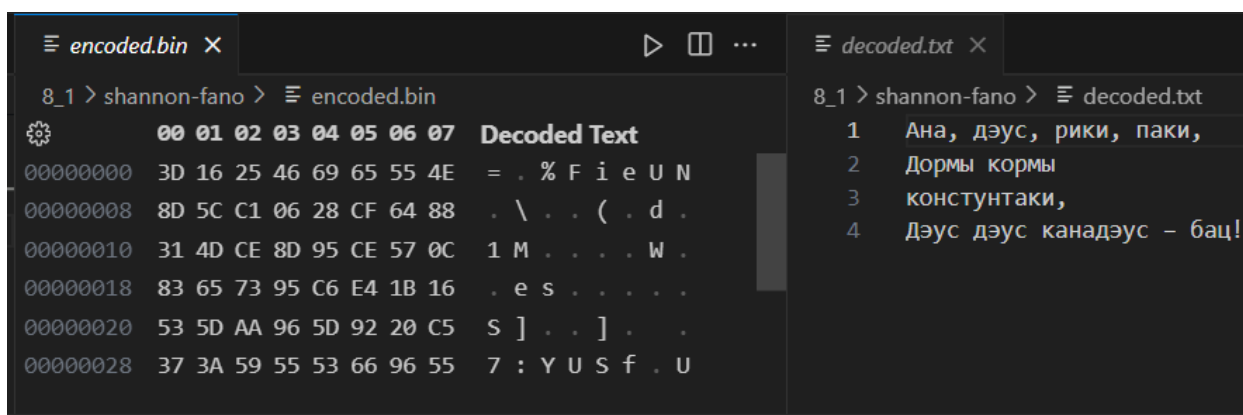
Выполним тестирование программы (рис. 9-10).



```
8_1 > shannon-fano > input.txt
1 Ана, дэус, рики, паки,
2 Дормы кормы
3 констунтаки,
4 Дэус дэус канадэус – бац!

d:\Boris\Documents\UNI\SIAOD\8_1\shannon-fano>cd "d:\Boris\Documents\UNI\SIAOD\8_1\shannon-fano"
Original size: 133 bytes
Compressed size: 64 bytes
Compression ratio: 2.07812
```

Рисунок 9 – Пример текстового файла



```
8_1 > shannon-fano > encoded.bin
00000000 3D 16 25 46 69 65 55 4E = . % F i e U N
00000008 8D 5C C1 06 28 CF 64 88 . \ . . ( . d .
00000010 31 4D CE 8D 95 CE 57 0C 1 M . . . . W .
00000018 83 65 73 95 C6 E4 1B 16 . e s . . . . .
00000020 53 5D AA 96 5D 92 20 C5 S ] . . ] . .
00000028 37 3A 59 55 53 66 96 55 7 : Y U S f . U

1 Ана, дэус, рики, паки,
2 Дормы кормы
3 констунтаки,
4 Дэус дэус канадэус – бац!
```

Рисунок 10 – Пример текстового файла

Тестирование показало, что программа работает корректно.

МЕТОД ХАФФМАНА

Формулировка задачи

Провести кодирование(сжатие) исходной строки символов «Фамилия Имя Отчество» с использованием алгоритма Хаффмана. Исходная строка символов, таким образом, определяет индивидуальный вариант задания для каждого студента.

Описание подхода к решению

Алгоритм кодирования методом Хаффмана представляет собой жадный алгоритм для построения префиксного кода, минимизирующего среднюю длину кодируемых сообщений. Алгоритм начинается с анализа исходных данных и вычисления частоты каждого символа в тексте. На основании этих частот строится бинарное дерево, где каждый узел представляет символ и его частоту, а каждый путь от корня к листу определяет уникальный префиксный код для символа.

Для построения дерева Хаффмана все символы добавляются в приоритетную очередь в виде узлов с весами, соответствующими их частотам. Затем из очереди последовательно извлекаются два узла с наименьшими частотами, и для них создается новый родительский узел с частотой, равной сумме частот этих узлов. Новый узел возвращается в очередь, и процесс продолжается до тех пор, пока в очереди не останется один узел, который становится корнем дерева. В результате получается дерево, в котором более часто встречающиеся символы находятся ближе к корню, а редкие — дальше от корня, что позволяет получить короткие коды для часто встречающихся символов.

После построения дерева каждому символу присваивается код путем прохождения от корня к листьям, добавляя "0" при переходе в левое поддереву и "1" — в правое. Эти коды и используются для сжатия исходного текста. Кодировка Хаффмана обеспечивает компактное представление данных, так как более короткие коды назначаются более частым символам, а редкие символы получают более длинные коды, что позволяет эффективно сжимать данные.

Код программы

Реализуем код приложения на языке программирования C++ (рис. 11-13).

```
1  #include <iostream>
2  #include <queue>
3  #include <map>
4  #include <vector>
5  #include <fstream>
6  #include <locale>
7
8  using namespace std;
9
10 struct Node {
11     wchar_t character;
12     int frequency;
13     Node *left, *right;
14
15     Node(wchar_t character, int frequency) {
16         left = right = nullptr;
17         this->character = character;
18         this->frequency = frequency;
19     }
20 };
21
22 struct Compare {
23     bool operator()(Node* left, Node* right) {
24         return left->frequency > right->frequency;
25     }
26 };
27
28 void generateHuffmanCodes(Node* root, wstring code, map<wchar_t, wstring>& huffmanCodes) {
29     if (!root) return;
30
31     if (!root->left && !root->right) {
32         huffmanCodes[root->character] = code;
33     }
34
35     generateHuffmanCodes(root->left, code + L"0", huffmanCodes);
36     generateHuffmanCodes(root->right, code + L"1", huffmanCodes);
37 }
38
39 map<wchar_t, wstring> buildHuffmanTree(const wstring& text) {
40
41     map<wchar_t, int> frequencyMap;
42     for (wchar_t ch : text) {
43         frequencyMap[ch]++;
44     }
45
46     priority_queue<Node*, vector<Node*>, Compare> minHeap;
47
48     for (auto pair : frequencyMap) {
49         minHeap.push(new Node(pair.first, pair.second));
50     }
51
52     while (minHeap.size() != 1) {
53         Node* left = minHeap.top();
54         minHeap.pop();
55         Node* right = minHeap.top();
56         minHeap.pop();
57
58         int sum = left->frequency + right->frequency;
59         Node* newNode = new Node(L'\0', sum);
60         newNode->left = left;
61         newNode->right = right;
62         minHeap.push(newNode);
63     }
```

Рисунок 11 – код программы (часть 1)

```

62     while (minHeap.size() > 1) {
63     }
64
65     map<wchar_t, wstring> huffmanCodes;
66     generateHuffmanCodes(minHeap.top(), L"", huffmanCodes);
67
68     return huffmanCodes;
69 }
70
71 wstring encodeText(const wstring& text, map<wchar_t, wstring>& huffmanCodes) {
72     wstring encodedText = L"";
73     for (wchar_t ch : text) {
74         encodedText += huffmanCodes[ch];
75     }
76     return encodedText;
77 }
78
79 wstring decodeText(const wstring& encodedText, map<wstring, wchar_t>& reverseHuffmanCodes) {
80     wstring decodedText = L"";
81     wstring codeBuffer = L"";
82     for (wchar_t bit : encodedText) {
83         codeBuffer += bit;
84         if (reverseHuffmanCodes.find(codeBuffer) != reverseHuffmanCodes.end()) {
85             decodedText += reverseHuffmanCodes[codeBuffer];
86             codeBuffer.clear();
87         }
88     }
89     return decodedText;
90 }
91
92 int main() {
93     setlocale(LC_ALL, "ru_RU.UTF-8");
94
95     wstring inputText;
96
97     wifstream inputFile(L"input.txt");
98     if (!inputFile.is_open()) {
99         wcerr << L"Error opening file." << endl;
100         return 1;
101     }
102
103     wstring line;
104     getline(inputFile, inputText, L'\0');
105     inputFile.close();
106
107     map<wchar_t, wstring> huffmanCodes = buildHuffmanTree(inputText);
108
109     wcout << L"Huffman codes for each symbol:\n";
110     for (auto pair : huffmanCodes) {
111         wstring characterRepresentation;
112
113         if (pair.first == L'\n') {
114             characterRepresentation = L"\\n";
115         } else {
116             characterRepresentation = wstring(1, pair.first);
117         }
118
119         // Выводим символ и его код
120         wcout << L"\" << characterRepresentation << L"': " << pair.second << endl;
121     }
122 }

```

Рисунок 12 – код программы (часть 2)

```

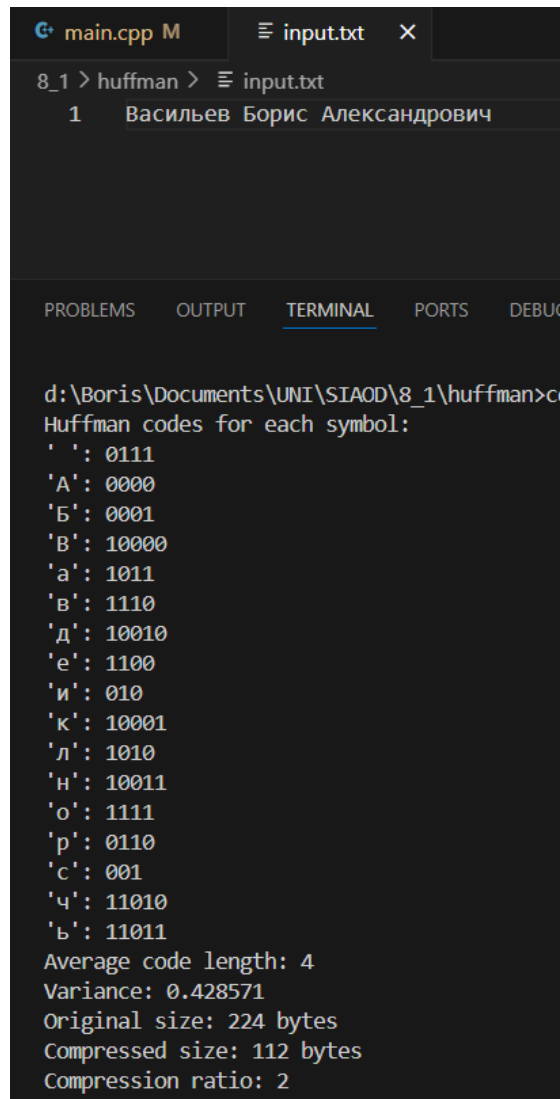
123     wstring encodedText = encodeText(inputText, huffmanCodes);
124
125     wofstream encodedFile("encoded.txt");
126     encodedFile << encodedText;
127     encodedFile.close();
128
129
130     map<wstring, wchar_t> reverseHuffmanCodes;
131     for (const auto& pair: huffmanCodes) {
132         reverseHuffmanCodes[pair.second] = pair.first;
133     }
134
135     wstring decodedText = decodeText(encodedText, reverseHuffmanCodes);
136
137     wofstream decodedFile("decoded.txt");
138     decodedFile << decodedText;
139     decodedFile.close();
140 }

```

Рисунок 13 – код программы (часть 3)

Результаты тестирования

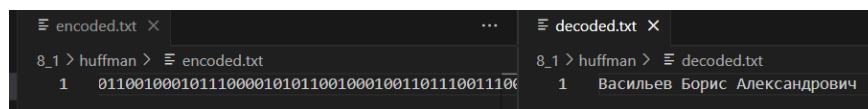
Выполним тестирование программы на различных значениях n. (рис. 14-15)



The screenshot shows a code editor with two tabs: 'main.cpp' and 'input.txt'. The 'input.txt' tab is active, showing the text '1 Васильев Борис Александрович'. Below the editor, the 'TERMINAL' tab is active, displaying the output of the program. The output shows the Huffman codes for each symbol, the average code length, variance, original size, compressed size, and compression ratio.

```
d:\Boris\Documents\UNI\SIAOD\8_1\huffman>c
Huffman codes for each symbol:
' ': 0111
'A': 0000
'Б': 0001
'В': 10000
'a': 1011
'в': 1110
'д': 10010
'e': 1100
'и': 010
'к': 10001
'л': 1010
'н': 10011
'o': 1111
'р': 0110
'с': 001
'ч': 11010
'ь': 11011
Average code length: 4
Variance: 0.428571
Original size: 224 bytes
Compressed size: 112 bytes
Compression ratio: 2
```

Рисунок 14 – исходный файл и запуск программы



The screenshot shows a code editor with two tabs: 'encoded.txt' and 'decoded.txt'. The 'encoded.txt' tab is active, showing the binary representation of the input text: '1 0110010001011100001010110010001001101110011100'. The 'decoded.txt' tab is also active, showing the original text: '1 Васильев Борис Александрович'.

Рисунок 15 – закодированный и декодированный файл

Тестирование показало, что программа работает корректно.

ВЫВОД

В результате выполнения работы были освоены различные приёмы кодирования и сжатия данных.

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Рысин, М. Л. Введение в структуры и алгоритмы обработки данных : учебное пособие / М. Л. Рысин, М. В. Сартаков, М. Б. Туманова. — Москва : РТУ МИРЭА, 2022 — Часть 2 : Поиск в тексте. Нелинейные структуры данных. Кодирование информации. Алгоритмические стратегии — 2022. — 111 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/book/310826> (дата обращения: 02.11.2024).
2. Документация по языку C++ [Электронный ресурс]. URL: <https://docs.microsoft.com/ruru/cpp/cpp/> (дата обращения 07.11.2024).