МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Отчет по выполнению практического задания № 8.1

Тема:

«ОТДЕЛЬНЫЕ ВОПРОСЫ АЛГОРИТМИЗАЦИИ»
«Алгоритмы кодирования и сжатия данных»

Дисциплина: «Структуры и алгоритмы обработки данных»

Выполнил студент: Жаворонкова А.А.

Группа: ИКБО-43-23

Москва - 2024

СОДЕРЖАНИЕ

1 ЦЕЛЬ

Освоить приёмы алгоритмы кодирования и сжатия данных

2 ЗАДАНИЕ №1

2.1 Постановка задачи

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

Закодировать фразу методами Шеннона– Фано:

Кот пошёл за молоком,

А котята кувырком. Кот

пришёл без молока,

А котята ха-ха-ха.

Сжатие данных по методу Лемпеля– Зива LZ77 Используя двухсимвольный алфавит (0, 1) закодировать следующую фразу: 10101001101100111010

Закодировать следующую фразу, используя код LZ78: bigbonebigborebigbo

2.2 Кодировка методом Шеннона-Фано

Закодировать фразу «Кот пошёл за молоком,

А котята кувырком. Кот

пришёл без молока,

А котята ха-ха-ха.», используя метод Шеннона–Фено.

Результат представлен в таблице 1.

Таблица 1 - Кодировка Шеннона-Фано

Символ	Кол-во	1-я цифра	2-я цифра	3-я цифра	4-я цифра	5-я цифра	6-я цифра	Код	Кол-во бит
о	11	0	0	0				000	33,00
пробел	10	0	0	1				001	30,00
а	7	0	1	0	0			0100	28,00
т	6	0	1	0	1			0101	24,00
к	6	0	1	1				011	18,00
л	4	1	0	0	0	0		10000	20,00
м	4	1	0	0	0	1		10001	20,00
х	3	1	0	0	1			1001	12,00
К	2	1	0	1	0	0	0	101000	12,00
п	2	1	0	1	0	0	1	101001	12,00
ш	2	1	0	1	0	1		10101	10,00
ё	2	1	0	1	1	0		10110	10,00
з	2	1	0	1	1	1		10111	10,00
,	2	1	1	0	0	0	0	110000	12,00
А	2	1	1	0	0	0	1	110001	12,00
я	2	1	1	0	0	1		11001	10,00
р	2	1	1	0	1	0		11010	10,00
.	2	1	1	0	1	1		11011	10,00
-	2	1	1	1	0	0		11100	10,00
у	1	1	1	1	0	1	0	111010	6,00
в	1	1	1	1	0	1	1	111011	6,00
ы	1	1	1	1	1	0	0	111100	6,00
и	1	1	1	1	1	0	1	111101	6,00
б	1	1	1	1	1	1	0	111110	6,00
е	1	1	1	1	1	1	1	111111	6,00
									339,00

Незакодированная фраза – 79*8 бит = 632 бит.

Закодированная фраза = 399 бит

Результат сжатия: 46.3608%

2.2 Кодировка методом Лемпеля– Зива LZ77

Сжатие данных по методу Лемпеля– Зива LZ77 Используя двухсимвольный алфавит (0, 1) закодировать следующую фразу: 10101001101100111010. Результат сжатия будет представлен в таблице 2.

Таблица 2 - LZ77 для двоичного кода

Исходный текст	10101001101100111010
LZ-код	001 000 240 311 340 421 420
R	4
Выводимые коды	1010 1001 1011 0011 1010

Результат сжатия: -5%

В данном случае сжатие данных методом Лемпеля-Зива является неэффективным в связи с тем, что алгоритм LZ77 эффективен для длинных строк с большим количеством повторяющихся подстрок. Также присутствует чередование 0 и 1, что создает меньше длинных повторяющихся последовательностей.

2.3 Кодировка методом LZ78

Закодировать следующую фразу, используя код LZ78: bigbonebigborebigbo. Результат сжатия будет представлен в таблице 3.

Таблица 3 - Кодировка LZ78

Словарь	Считываемое содержимого	Код
	b	<0,b>
b = 1	i	<0,i>
b = 1 i = 2	g	<0, g>
b = 1 g = 3 i = 2	bo	<1,o>
b = 1 bo = 4 g = 3 i = 2	n	<0, n>
b = 1 bo = 4 g = 3 i = 2 n = 5	e	<0, e>
b = 1 bo = 4 e = 6 g = 3 i = 2 n = 5	bi	<1,i>
b = 1 bi = 7 bo = 4 e = 6 g = 3 i = 2 n = 5	gb	<3,b>
b = 1 bi = 7 bo = 4 e = 6 g = 3 gb = 8 i = 2 n = 5	o	<0,o>
b = 1 bi = 7 bo = 4 e = 6 g = 3 gb = 8 i = 2 n = 5 o = 9	r	<6,b>
b=1 bi=7 bo=4 e=6 g=3 gb= 8 i = 2 n = 5 o = 9 r = 10	eb	<2,g>
b=1 bi=7 bo=4 e=6 eb=11 g=3 gb=8 i=2 ig=12 n=5 o=9 r=10	ig	<1,o>
b = 1 bi = 7 bo = 4 e = 6 eb = 11 g = 3 gb = 8 i = 2 ig = 12 n = 5 o = 9 r = 10	bo	

Оригинальный размер: 19

Сжатый размер: 13

Процент сжатия: 31.5789%

3 ЗАДАНИЕ №2

3.1 Постановка задачи

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 с 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.

Требования к выполнению задания 2

1. 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.

2. 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.

3.2 Математическая модель решения Шеннона-Фано

Метод Шеннона-Фано является одним из алгоритмов для создания префиксного кодирования, который широко используется для сжатия данных. Он основывается на частотном анализе символов и позволяет построить эффективные двоичные коды для символов, встречающихся с разной вероятностью. Основная

цель — создание префиксного кода, где ни один код символа не является началом кода другого символа, что позволяет декодировать данные однозначно.

Алгоритм формирования префиксного дерева по методу Шеннона-Фано

Сортировка символов по вероятности появления:

На первом этапе подсчитываются частоты (или вероятности) появления каждого символа в сообщении, а затем символы упорядочиваются по убыванию их вероятности.

Рекурсивное разделение множества символов:

Множество символов делится на две группы с примерно одинаковыми суммарными вероятностями.

Каждой группе присваивается бит: например, для первой группы — 0, а для второй — 1.

Рекурсивное деление подмножеств:

Для каждой полученной группы алгоритм повторяется, пока не останется подмножества, состоящие из одного символа.

На каждом шаге добавляется новый бит к коду символа в зависимости от его группы (0 или 1).

В результате получится уникальный двоичный код для каждого символа.

Получение префиксного дерева:

На основе деления можно построить дерево, где каждый путь от корня до листа будет соответствовать коду символа, а листами будут сами символы.

Итогом этого алгоритма является дерево Шеннона-Фано, в котором каждый символ имеет уникальный префиксный код.

Алгоритм кодирования по дереву Шеннона-Фано

Создание кодовой таблицы:

На основе построенного дерева записываются коды для каждого символа. Например, если символ A был закодирован как 00, B — как 01, то в таблице будут указаны пары (символ, код).

Кодирование сообщения:

Для каждого символа в сообщении находится соответствующий код в таблице и последовательно записывается в закодированное сообщение.

В итоге, исходное сообщение преобразуется в последовательность битов, представляющих коды символов.

Алгоритм декодирования по дереву Шеннона-Фано

Построение дерева:

Дерево или кодовая таблица должна быть известна для правильного декодирования. Она может быть либо известна изначально, либо передана вместе с закодированным сообщением.

Поиск по дереву:

Декодирование выполняется путем последовательного чтения битов из закодированного сообщения.

Для каждого бита выбирается соответствующая ветвь в дереве: 0 — идем влево, 1 — вправо.

Когда доходим до листа дерева, находим символ, соответствующий этому пути.

Преобразование битов в символы:

После нахождения символа его добавляют к декодированному сообщению, и процесс повторяется с начала для оставшихся битов.

Процесс повторяется до тех пор, пока не будут обработаны все биты закодированного сообщения.

В результате получается восстановленное исходное сообщение.

3.3 Код программы Шеннона-Фано с комментариями

Продемонстрируем код программы на листинге 1, 2, 3.

Код будет написан на языке программирования C++.

```
#include <iostream>
#include <vector>
#include <map>
#include <string>
#include <algorithm>
#include <fstream>
using namespace std;

// Структура для хранения символа и его частоты
  struct Symbol {
      char ch;
      int freq;
      string code;
  };

  // Сравнение для сортировки по частоте
  bool compareFreq(const Symbol &a, const Symbol &b) {
      return a.freq > b.freq;
  }

  // Функция рекурсивного кодирования
  void shannonFanoCoding(vector<Symbol>& symbols, int start, int
  end) {
      if (start >= end)
          return;

      // Находим середину для разделения на 2 группы
      int total_freq = 0;
      for (int i = start; i <= end; ++i)
          total_freq += symbols[i].freq;

      int half_freq = 0;
      int mid = start;
      for (int i = start; i <= end; ++i) {
          half_freq += symbols[i].freq;
          if (half_freq >= total_freq / 2) {
              mid = i;
              break;
          }
      }

      // Присваиваем код "0" левой части и "1" правой части
      for (int i = start; i <= mid; ++i)
          symbols[i].code += "0";
      for (int i = mid + 1; i <= end; ++i)
          symbols[i].code += "1";

      // Рекурсивно кодируем обе части
      shannonFanoCoding(symbols, start, mid);
      shannonFanoCoding(symbols, mid + 1, end);
  }
```

Листинг 1 - Код программы

```
51.  // Функция для сжатия текста
52.  string compress(const string& text, map<char, string>&
  encodingTable) {
53.      string compressed = "";
54.      for (char ch : text) {
55.          compressed += encodingTable[ch];
56.      }
57.      return compressed;
58.  }
59.
60.  // Функция для декодирования сжатого текста
61.  string decompress(const string& compressed, const map<string,
  char>& decodingTable) {
62.      string decoded = "";
63.      string buffer = "";
64.      for (char bit : compressed) {
65.          buffer += bit;
66.          if (decodingTable.find(buffer) != decodingTable.end())
  {
67.              decoded += decodingTable.at(buffer);
68.              buffer = "";
69.          }
70.      }
71.      return decoded;
72.  }
73.
74.  // Функция для вычисления процента сжатия
75.  double compressionRate(const string& original, const string&
  compressed) {
76.      return 100.0 * (1.0 - (double(compressed.size()) /
  (original.size() * 8)));
77.  }
78.
79.  int main() {
80.      // Открытие исходного текстового файла
81.      ifstream inputFile("input.txt");
82.      string text((istreambuf_iterator<char>(inputFile)),
  istreambuf_iterator<char>());
83.      inputFile.close();
84.
85.      // Подсчёт частоты символов
86.      map<char, int> freqMap;
87.      for (char ch : text) {
88.          freqMap[ch]++;
89.      }
90.
91.      // Заполнение вектора символов
92.      vector<Symbol> symbols;
93.      for (auto& pair : freqMap) {
94.          symbols.push_back({pair.first, pair.second, ""});
95.      }
```

Листинг 2 - Продолжение кода программы 1

```
96.      // Сортировка символов по частоте
97.      sort(symbols.begin(), symbols.end(), compareFreq);
98.
99.      // Выполнение кодирования Шеннона-Фано
100.     shannonFanoCoding(symbols, 0, symbols.size() - 1);
101.
102.     // Построение таблиц кодирования и декодирования
103.     map<char, string> encodingTable;
104.     map<string, char> decodingTable;
105.     for (Symbol& sym : symbols) {
106.         encodingTable[sym.ch] = sym.code;
107.         decodingTable[sym.code] = sym.ch;
108.         cout << "Символ: " << sym.ch
109.              << " встречается " << sym.freq << " раз,
  закодировано как "
110.             << sym.code << ", количество битов: " <<
  sym.code.length() << endl;
111.     }
112.
113.     // Сжатие текста
114.     string compressedText = compress(text, encodingTable);
115.     ofstream compressedFile("compressed.txt");
116.     compressedFile << compressedText;
117.     compressedFile.close();
118.
119.     // Декодирование сжатого текста
120.     string decompressedText = decompress(compressedText,
  decodingTable);
121.     ofstream outputFile("decompressed.txt");
122.     outputFile << decompressedText;
123.     outputFile.close();
124.
125.     // Вычисление процента сжатия
126.     double compressionPercentage = compressionRate(text,
  compressedText);
127.     cout << "Процент сжатия: " << compressionPercentage << "%"
  << endl;
128.
129.     return 0;
130. }
```

Листинг 3 - Продолжение кода программы

3.4 Результаты тестирования

Стоит задача протестировать программу. Результаты тестирования будут продемонстрированы на рисунке 2.

Тестовые данные:

Кот пошёл за молоком,

А котята кувырком. Кот

пришёл без молока,

А котята ха-ха-ха.

```
Символ: о встречается 11 раз, закодировано как 000, количество битов: 3
Символ:   встречаетс 10 раз, закодировано как 001, количество битов: 3
Символ: а встречается 7 раз, закодировано как 0100, количеств битов: 4
Символ: т встречается 6 раз, закодировано ка 0101, количество битов: 4
Символ: к встречается 6 раз, з кодировано как 011, количество битов: 3
Символ: м встречается 4 раз, закодировано как 10000, количество битов: 5
Символ: л встречается 4 раз, закодировано как 10001, количество битов: 5
Символ: х встречается 3 раз, закодиро но как 1001, количество битов: 4
Символ: я встречается 2 раз, закодировано как 10100, количество битов: 5
Симво ш встречается 2 раз, закодировано как 10101, количеств битов: 5
Символ: з встречается 2 раз, закодировано ка 10110, количество битов: 5
Символ: р встречается 2 раз, закодировано как 10111, количество битов: 5
Символ: п встречается 2 раз, закодировано как 110000, количество бито 6
Символ: ё встречается 2 раз, закодировано как 110001, личество битов: 6
Символ: К встречается 2 раз, закодировано как 11001, количество битов: 5
Символ: А встреча ся 2 раз, закодировано как 11010, количество битов: 5
Си вол: . встречается 2 раз, закодировано как 11011, количе тво битов: 5
Символ: - встречается 2 раз, закодировано как 11100, количество битов: 5
Символ: , встречается 2 раз, закодировано как 111010, количество битов: 6
Символ: н встречается 1 раз, закодировано как 111011, количество битов: 6
Символ: е встречается 1 раз, закодировано как 111100, количество битов: 6
Символ: б встречается 1 раз, за одировано как 111101, количество битов: 6
Символ: й встречается 1 раз, закодировано как 111110, количество битов: 6
Символ: В встречается 1 раз, закодировано как 1111110, количество битов: 7
Символ: у встречается 1 раз, закоди ровано как 1111111, количество битов: 7
Процент сжатия: 46.3608%
```

Рисунок 1 - Вывод на экран при выполнении программы

main.cpp | compressed.txt | decompressed.txt | input.txt

```
1 Кот пошёл за молоком,
2 А котята кувырком. Кот
3 пришёл без молока,
4 А котята ха-ха-ха.
```

Рисунок 2 - Входные данные для кода

main.cpp | compressed.txt | decompressed.txt | input.txt

```
1 00111001001000001010010110011011000100000101111010010100111000100000101111010010100111000100000101111010010100111
```

ZIP имеет процент сжатия равный 64%, что является более хорошим показателем, чем сжатие Шеннона-Фано, который имеет 46,3608%.

3.5 Математическая модель решения Хаффмана

Построим таблицу 4 частот встречаемости символов в исходной строке символов для чего сформируем алфавит исходной строки и посчитаем количество вхождений символов и их вероятности появления. Для этого используем ФИО Zhavoronkova Aleksandra Alekseevna.

Таблица 4 - Таблица частот

Алфавит	d	h	A	a	n	Z	пробел
Кол. вх.	1	1	2	5	3	1	2
Вероятность	0.029	0.029	0.059	0.147	0.088	0.029	0.059

Алфавит	v	o	r	e	k	l	s
Кол. вх.	3	3	2	4	3	2	2
Вероятность	0.088	0.088	0.059	0.118	0.088	0.059	0.059

Отсортируем алфавит в порядке убывания частот появления символов в таблице 5.

Таблица 5 - Таблица отсортированных частот

Алфавит	a	e	v	o	n	k	A
Кол. вх.	5	4	3	3	3	3	2
Вероятность	0.147	0.118	0.088	0.088	0.088	0.088	0.059

Алфавит	l	s	r	пробел	Z	d	h
Кол. вх.	2	2	2	2	1	1	1
Вероятность	0.059	0.059	0.059	0.059	0.029	0.029	0.029

Построим дерево кодирования Хаффмана и предоставим его на рисунке 5.

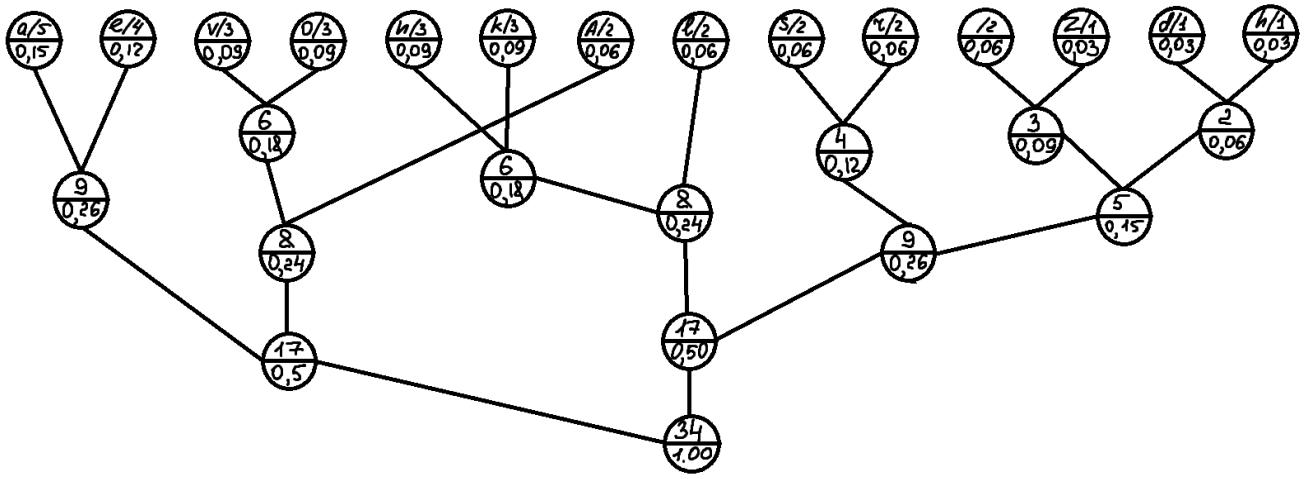

Рисунок 5 - Дерево кодирования Хаффмана

Упорядочим дерево кодирования и присвоим коды (рис. 6)

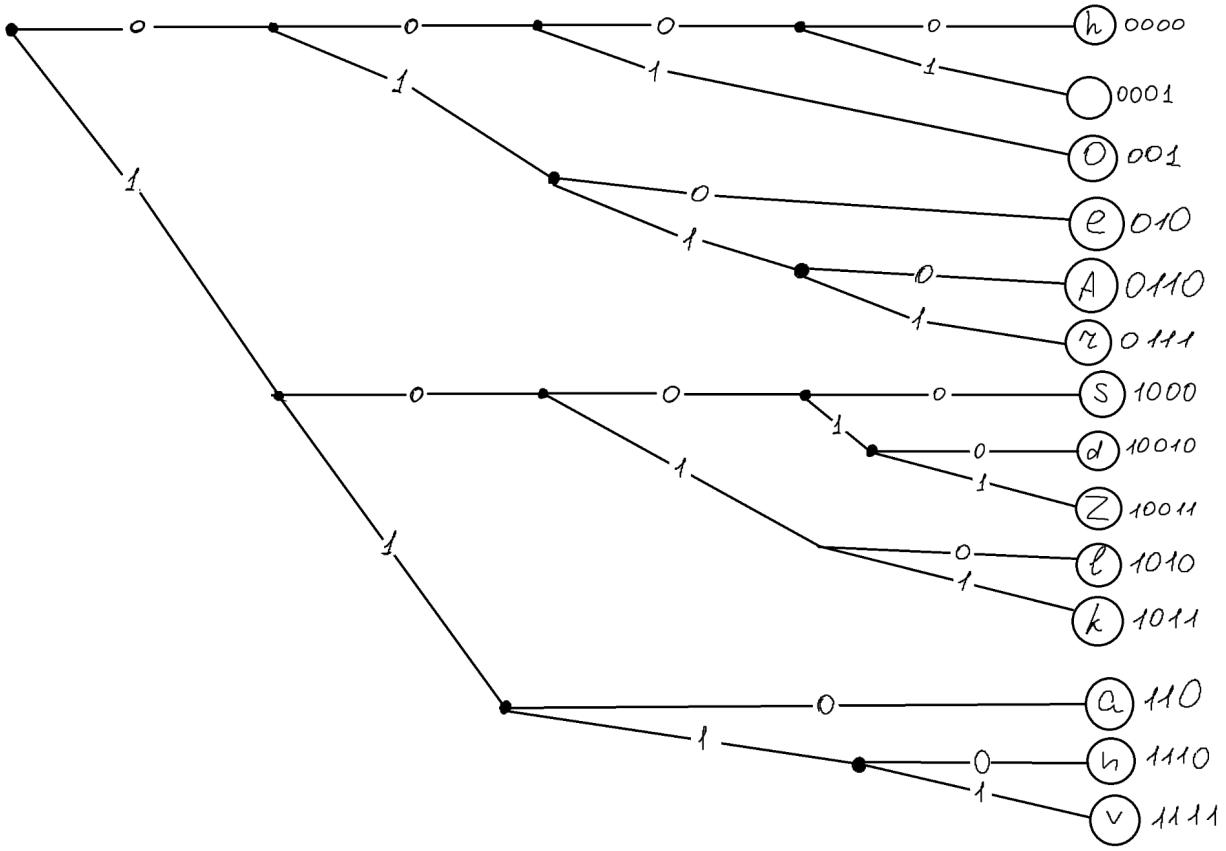

Рисунок 6 - Упорядоченное дерево кодирования Хаффмана

Проведем кодирование исходной строки.

Zhavoronkova Aleksandra Alekseevna= 10011 0000 110 1111 001 0111 001 1110 1011 001 1111 110 0001 0110 1010 010 1011 1000 110 1110 10010 0111 110 0001 0110 1010 010 1011 1000 010 010 1111 1110 110

Исходный размер (ASCII): 272 бит

Сжатый размер: 126 бит

Коэффициент сжатия относительно ASCII: 46.3235%

Размер равномерного кода: 136 бит

Коэффициент сжатия относительно равномерного кода: 92.6471%

Средняя длина кода: 3.70588

Дисперсия длины кода: 0.32526

3.6 Код программы Хаффмана с комментариями

Продемонстрируем код программы на листинге 4, 5, 6 и 7.

Код будет написан на языке программирования C++.

```
1. #include <iostream>
2. #include <fstream>
3. #include <vector>
4. #include <queue>
5. #include <unordered_map>
6. #include <string>
7. #include <cmath>
8.
9. using namespace std;
10.
11.  // Узел дерева Хаффмана
12.  struct Node {
13.      char ch;
14.      int freq;
15.      Node* left;
16.      Node* right;
17.
18.      // Конструктор для листовых узлов
19.      Node(char c, int f) : ch(c), freq(f), left(nullptr),
   right(nullptr) {}
20.
21.      // Конструктор для внутренних узлов
22.      Node(char c, int f, Node* l, Node* r) : ch(c), freq(f),
   left(l), right(r) {}
23.  };
```

Листинг 4 - Код программы

```
24.  // Компаратор для приоритетной очереди
25.  struct Compare {
26.      bool operator()(Node* l, Node* r) {
27.          return l->freq > r->freq;
28.      }
29.  };
30.
31.  // Построение кодов Хаффмана
32.  void buildHuffmanCodes(Node* root, unordered_map<char,
  string>& huffmanCode, string str) {
33.      if (!root) return;
34.
35.      // Если это лист
36.      if (!root->left && !root->right) {
37.          huffmanCode[root->ch] = str;
38.      }
39.
40.      buildHuffmanCodes(root->left, huffmanCode, str + "0");
41.      buildHuffmanCodes(root->right, huffmanCode, str + "1");
42.  }
43.
44.  // Удаление дерева Хаффмана
45.  void deleteTree(Node* root) {
46.      if (!root) return;
47.      deleteTree(root->left);
48.      deleteTree(root->right);
49.      delete root;
50.  }
51.
52.  // Кодирование строки
53.  void huffmanCoding(const string& text) {
54.      // Подсчет частот символов
55.      unordered_map<char, int> freq;
56.      for (char ch : text) {
57.          freq[ch]++;
58.      }
59.
60.      // Подсчет вероятностей символов
61.      int totalCharacters = text.length();
62.      unordered_map<char, double> probabilities;
63.      for (auto pair : freq) {
64.          probabilities[pair.first] = (double)pair.second /
  totalCharacters;
65.      }
66.
67.      // Вывод таблицы частот и вероятностей
68.      cout << "Частота и вероятность встречаемости символов:\n";
69.      for (auto pair : freq) {
70.          cout << pair.first << ": частота = " << pair.second <<
  ", вероятность = " << probabilities[pair.first] << "\n";
71.      }
```

Листинг 5 - Продолжение кода программы

```
72.      // Построение дерева
73.      priority_queue<Node*, vector<Node*>, Compare> pq;
74.      for (auto pair : freq) {
75.          pq.push(new Node(pair.first, pair.second));
76.      }
77.
78.      while (pq.size() != 1) {
79.          Node* left = pq.top(); pq.pop();
80.          Node* right = pq.top(); pq.pop();
81.
82.          int sum = left->freq + right->freq;
83.          pq.push(new Node('\0', sum, left, right));
84.      }
85.
86.      Node* root = pq.top();
87.
88.      // Построение кодов
89.      unordered_map<char, string> huffmanCode;
90.      buildHuffmanCodes(root, huffmanCode, "");
91.
92.      // Вывод кодов
93.      cout << "\nКоды Хаффмана для символов:\n";
94.      for (auto pair : huffmanCode) {
95.          cout << pair.first << ": " << pair.second << "\n";
96.      }
97.
98.      // Кодирование строки
99.      string encodedString = "";
100.     for (char ch : text) {
101.         encodedString += huffmanCode[ch];
102.     }
103.
104.     cout << "\nЗакодированная строка: " << encodedString <<
  "\n";
105.
106.     // Расчет коэффициентов сжатия
107.     int originalSize = text.length() * 8; // исходная строка в
  битах (8 бит на символ в ASCII)
108.     int compressedSize = encodedString.length(); // сжатая
  строка в битах
109.     double compressionRatioASCII = (double)compressedSize /
  originalSize;
110.
111.     int uniformCodeSize = ceil(log2(freq.size())); // биты на
  символ для равномерного кода
112.     int totalUniformSize = uniformCodeSize * text.length(); //
  исходный размер в равномерном коде
113.     double compressionRatioUniform = (double)compressedSize /
  totalUniformSize;
```

Листинг 6 - Продолжение кода программы

```
114.     cout << "\nИсходный размер (ASCII): " << originalSize << "
  бит\n";
115.     cout << "Сжатый размер: " << compressedSize << " бит\n";
116.     cout << "Коэффициент сжатия относительно ASCII: " <<
  compressionRatioASCII * 100 << "%\n";
117.
118.     cout << "\nРазмер равномерного кода: " << totalUniformSize
  << " бит\n";
119.     cout << "Коэффициент сжатия относительно равномерного
  кода: " << compressionRatioUniform * 100 << "%\n";
120.
121.     // Расчет средней длины кода
122.     double averageLength = 0;
123.     for (auto pair : probabilities) {
124.         averageLength += pair.second *
  huffmanCode[pair.first].length();
125.     }
126.     cout << "\nСредняя длина кода: " << averageLength << "\n";
127.
128.     // Расчет дисперсии длины кода
129.     double variance = 0;
130.     for (auto pair : probabilities) {
131.         double diff = huffmanCode[pair.first].length() -
  averageLength;
132.         variance += pair.second * diff * diff;
133.     }
134.     cout << "Дисперсия длины кода: " << variance << "\n";
135.
136.     // Освобождение памяти
137.     deleteTree(root);
138. }
139.
140. int main() {
141.     // Чтение строки из файла
142.     ifstream inputFile("input.txt");
143.     if (!inputFile.is_open()) {
144.         cerr << "Не удалось открыть файл!" << endl;
145.         return 1;
146.     }
147.
148.     string text;
149.     getline(inputFile, text); // Чтение строки из файла
150.     inputFile.close();
151.
152.     cout << "Исходный текст: " << text << endl;
153.     huffmanCoding(text);
154.     return 0;
155. }
```

Листинг 7 - Продолжение кода программы

3.7 Результаты тестирования

Стоит задача протестировать программу. Результаты тестирования будут продемонстрированы на рисунке 7.

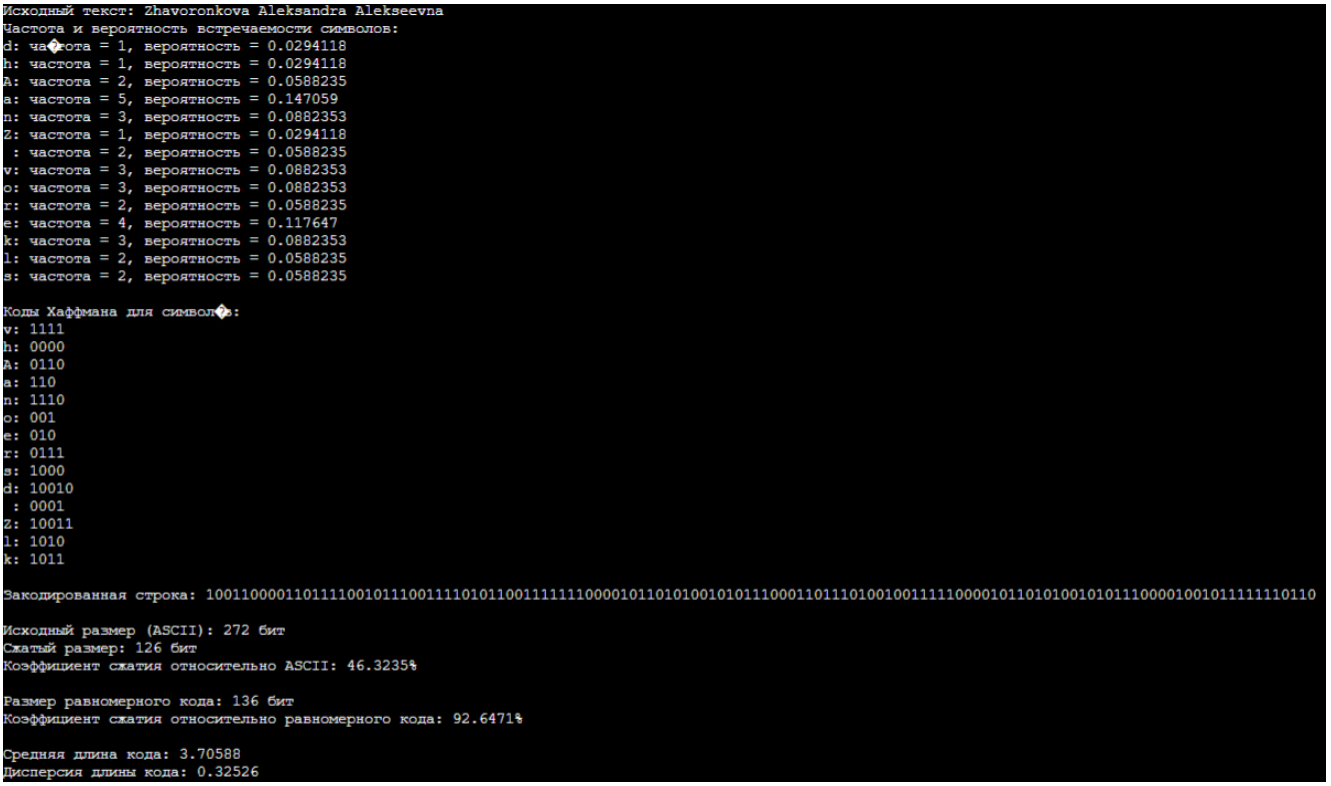

Рисунок 7 - Вывод на экран при выполнении программы

ZIP имеет процент сжатия равный 88%, что является более хорошим показателем, чем сжатие относительно ASCII, который имеет 46,3235%, но хуже, чем относительно равномерного кода, который имеет 92,6471.

4 ВЫВОДЫ

Все методы сжатия показали себя эффективными для текстов с неравномерным распределением символов, при этом алгоритм Хаффмана имеет преимущество перед Шенноном–Фано за счет своей способности обеспечивать более высокое сжатие за счет структуры дерева.

Алгоритм Хаффмана продемонстрировал более низкую среднюю длину кода и меньшую дисперсию, что делает его более предпочтительным для задачи минимизации длины закодированного текста.

Эти методы важны в ситуациях, когда требуется хранить текстовые данные в сжатом виде, а также для создания специализированных алгоритмов сжатия для систем с ограниченной памятью.