****

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«МИРЭА – Российский технологический университет»**

**РТУ МИРЭА**

Отчет по выполнению практического задания №2.6

**Тема: кодирование и сжатие данных методами без потерь**

Дисциплина: структуры и алгоритмы обработки данных

Выполнил студент Кузнецов А.А.

группа ИКБО-01-21

Москва 2022

Оглавление

[Цель работы 3](#_Toc120837325)

[Постановка задачи 3](#_Toc120837326)

[Подход к решению 4](#_Toc120837327)

[Задание 1: 4](#_Toc120837328)

[Задание 2: 7](#_Toc120837329)

[Тестирование 15](#_Toc120837330)

[Вывод 16](#_Toc120837331)

# Цель работы

Получение знаний и практических навыков по выполнению операций и применению алгоритмов для работы с различными алгоритмами сжатия и кодирования.

# Постановка задачи

Задание 1. Исследование алгоритмов сжатия на примерах.

1) Выполнить задачи варианта (см. задание варианта в таблице 1), представив алгоритм решения в виде таблицы и указав результат сжатия.

2) Сформировать отчет, включив задание, вариант задания, результаты выполнения задания варианта.

Таблица 1 – Индивидуальный вариант

|  |  |  |  |
| --- | --- | --- | --- |
| **Вариант** | **Закодировать фразу методами Шеннона– Фано** | **Сжатие данных по методу Лемпеля-Зива LZ77** | **Закодировать следующую фразу, используя код LZ78** |
| 2 | One, two, Freddy's coming for you Three, four, better lock your door Five, six, grab a crucifix Seven, eight, gonna stay up late. | 0100100010010000101 | упупапекапекаупуп |

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.

3) 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.

4) По методу Хаффмана выполнить и отобразить результаты выполнения всех требований, предъявленных в задании и оформить разработку программы: постановка, подход к решению.

# Подход к решению

Задание 1:

Для метода Шеннона-Фано:

Закодировать методом Шеннона-Фано фразу: *One, two, Freddy's coming for you Three, four, better lock your door Five, six, grab a crucifix Seven, eight, gonna stay up late.*

Решение задания представлено в таблице 2.

Таблица 2 – Кодирование методом Шеннона-Фано

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Символ** | **Кол-во** | **1-ая цифра** | **2-ая цифра** | **3-ья цифра** | **4-ая цифра** | **5-ая цифра** | **6-ая цифра** | **7-ая цифра** | **Код** | **Кол-во бит** |
| Пробел | 22 | 0 | 0 | 0 |  |  |  |  | 000 | 66 |
| e | 11 | 0 | 0 | 1 |  |  |  |  | 001 | 33 |
| o | 10 | 0 | 1 | 0 | 0 |  |  |  | 0100 | 40 |
| r | 9 | 0 | 1 | 0 | 1 |  |  |  | 0101 | 36 |
| , | 8 | 0 | 1 | 1 | 0 |  |  |  | 0110 | 32 |
| i | 6 | 0 | 1 | 1 | 1 |  |  |  | 0111 | 24 |
| t | 6 | 1 | 0 | 0 | 0 |  |  |  | 1000 | 24 |
| n | 5 | 1 | 0 | 0 | 1 | 0 |  |  | 10010 | 25 |
| a | 5 | 1 | 0 | 0 | 1 | 1 |  |  | 10011 | 25 |
| u | 5 | 1 | 0 | 1 | 0 | 0 |  |  | 10100 | 25 |
| c | 4 | 1 | 0 | 1 | 0 | 1 |  |  | 10101 | 20 |
| g | 4 | 1 | 0 | 1 | 1 | 0 |  |  | 10110 | 20 |
| y | 4 | 1 | 0 | 1 | 1 | 1 |  |  | 10111 | 20 |
| s | 3 | 1 | 1 | 0 | 0 | 0 |  |  | 11000 | 15 |
| d | 3 | 1 | 1 | 0 | 0 | 1 | 0 |  | 110010 | 18 |
| f | 3 | 1 | 1 | 0 | 0 | 1 | 1 |  | 110011 | 18 |
| v | 2 | 1 | 1 | 0 | 1 | 0 |  |  | 11010 | 10 |
| x | 2 | 1 | 1 | 0 | 1 | 1 | 0 |  | 110110 | 12 |
| h | 2 | 1 | 1 | 0 | 1 | 1 | 1 |  | 110111 | 12 |
| l | 2 | 1 | 1 | 1 | 0 | 0 | 0 |  | 111000 | 12 |
| b | 2 | 1 | 1 | 1 | 0 | 0 | 1 |  | 111001 | 12 |
| F | 2 | 1 | 1 | 1 | 0 | 1 | 0 |  | 111010 | 12 |
| m | 1 | 1 | 1 | 1 | 0 | 1 | 1 |  | 111011 | 6 |
| k | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1111000 | 7 |
| p | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1111001 | 7 |
| T | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1111010 | 7 |
| S | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1111011 | 7 |
| O | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1111100 | 7 |
| w | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1111101 | 7 |
| . | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1111110 | 7 |
| ' | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1111111 | 7 |

Незакодированная фраза – 129 \* 8 = 1032 бит.

Закодированная фраза – 573 бит.

Коэффициент сжатия: 0.555233.

Для метода Лемпеля-Зива LZ77:

Описание алгоритма:

1) Выбирается первый символ сообщения и заменяется на его код.

2) Выбираются следующие два символа и заменяются своими кодами. Одновременно этой комбинации двух символов присваивается свой код. Обычно это номер, равный числу уже использованных кодов.

3) Выбираются из исходного текста очередные 2, 3,...N символов до тех пор, пока не образуется еще не встречавшаяся комбинация. Тогда этой комбинации присваивается очередной код, и поскольку совокупность из первых N-1 символов уже встречалась, то она имеет свой код, который и записывается вместо этих N-1 символов.

4) Процесс продолжается до исчерпания исходного текста.

Используя алфавит (0, 1), закодируем следующую последовательность: *0100100010010000101*. Решение представлено в таблице 3.

Таблица 3 – Кодирование фразы алгоритмом LZ77

|  |  |
| --- | --- |
| Исходный текст | 0.10.01.00.010.0100.001.01 |
| LZ-код | 0.10.001.000.0110.1010.1001.011 |
| R | 2 3 |
| Вводимые коды | - 10 11 100 101 110 111 011 |

Для метода Лемпеля-Зива LZ78:

Описание алгоритма:

Алгоритм считывает символы сообщения до тех пор, пока накапливаемая подстрока входит целиком в одну из фраз словаря. Как только эта строка перестанет соответствовать хотя бы одной фразе словаря, алгоритм генерирует код, состоящий из индекса строки в словаре, которая до последнего введенного символа содержала входную строку, и символа, нарушившего совпадение. Затем в словарь добавляется введенная подстрока. Если словарь уже заполнен, то из него предварительно удаляют менее всех используемую в сравнениях фразу. Если в конце алгоритма мы не находим символ, нарушивший совпадения, то тогда мы выдаем код в виде: индекс строки в словаре без последнего символа, последний символ.

Используя алгоритм Лемпеля-Зива LZ78, закодируем последовательность: *упупапекапекаупуп*. Решение приведено в таблице 4.

Таблица 4 – Кодирование фразы алгоритмом LZ78

|  |  |  |
| --- | --- | --- |
| **Словарь** | **Считываемое содержимое** | **Код** |
|  | у | <0, у> |
| у = 1 | п | <0, п> |
| у = 1, п = 2 | уп | <1, п> |
| у =1, п = 2, уп = 3 | а | <0, а> |
| у =1, п = 2, уп = 3, а = 4 | пе | <2, е> |
| у =1, п = 2, уп = 3, а = 4, пе = 5 | к | <0, к> |
| у =1, п = 2, уп = 3, а = 4, пе = 5, к = 6 | ап | <4, п> |
| у =1, п = 2, уп = 3, а = 4, пе = 5, к = 6, ап = 7 | е | <0, е> |
| у =1, п = 2, уп = 3, а = 4, пе = 5, к = 6, ап = 7, е = 8 | ка | <6, а> |
| у =1, п = 2, уп = 3, а = 4, пе = 5, к = 6, ап = 7, е = 8, ка = 9 | упу | <3, у> |
| у =1, п = 2, уп = 3, а = 4, пе = 5, к = 6, ап = 7, е = 8, ка = 9, упу = 10 | п | <2, EOF> |

Задание 2:

Для метода Хаффмана:

Описание алгоритма:

1) На вход приходят упорядоченные по невозрастанию частот данные.

2) Выбираются две наименьших по частоте буквы алфавита, и создается родитель (сумма двух частот этих «листков»).

3) Потомки удаляются и вместо них записывается родитель, «ветви» родителя нумеруются: левой ветви ставится в соответствие «1», правой «0».

4) Шаг два повторяется до тех пор, пока не будет найден главный родитель — «корень».

Описание алгоритма восстановления сжатого текста:

Следует начать с корня и прочитать первый бит сжатого файла. Если это нуль, следует двигаться по нижней ветке дерева; если это единица, то двигаться надо по верхней ветке дерева. Далее читается второй бит и происходит движение по следующей ветке по направлению к листьям. Когда декодер достигнет листа дерева, он узнает код первого несжатого символа. Процедура повторяется для следующего бита, начиная опять из корня дерева.

Произведем кодирование строки «*kuznetcov andrei alexsandrovich*». Построим таблицу частоты встречаемости символов, таблица 5, а также таблицу кодирования символов, таблица 6.

Таблица 5 – Частота встречаемости символов

|  |  |  |
| --- | --- | --- |
| **Буква** | **Кол-во вхождений** | **Частота встречаемости** |
| a | 3 | 0.096 |
| e | 3 | 0.096 |
| n | 3 | 0.096 |
| r | 2 | 0.06 |
| t | 2 | 0.06 |
| c | 2 | 0.06 |
| o | 2 | 0.06 |
| v | 2 | 0.06 |
| d | 2 | 0.06 |
| i | 2 | 0.06 |
| Пробел | 2 | 0.06 |
| h | 1 | 0.03 |
| s | 1 | 0.03 |
| x | 1 | 0.03 |
| l | 1 | 0.03 |
| k | 1 | 0.03 |
| u | 1 | 0.03 |
| z | 1 | 0.03 |
| t | 1 | 0.03 |

Проведем кодирование исходной строки: 01110 11100 01001 1111 000 11101 1000 1011 1001 0110 001 1111 1010 0101 000 1101 0110 001 01000 000 11001 01111 001 1111 1010 0101 1011 1001 1101 1000 11000

Коэффициент сжатия: 127 / 248 = 0.51

Таблица 6 – Кодировка символов Хаффманом

|  |  |
| --- | --- |
| **Символ** | **Кодирование** |
| n | 1111 |
| t | 11101 |
| u | 11100 |
| i | 1101 |
| x | 11001 |
| e | 000 |
| r | 0101 |
| a | 001 |
| Пробел | 0110 |
| l | 01000 |
| z | 01001 |
| k | 01110 |
| s | 01111 |
| c | 1000 |
| v | 1001 |
| d | 1010 |
| o | 1011 |
| h | 11000 |

Рассчитаем среднюю длину по формуле

где *s* – множество символов алфавита; – вероятность появления символа; – количество бит в коде символа.

Средняя длина символа: 0.096\*4 + 0.096\*3 + 0.096\*3 + 0.06\*4 + 0.06\*4 + 0.06\*5 + 0.06\*4 + 0.06\*4 + 0.06\*4 + 0.06\*4 + 0.06\*4 + 0.03\*5 + 0.03\*5 + 0.03\*5 + 0.03\*5 + 0.03\*5 + 0.03\*5 + 0.03\*5 + 0.03\*5 = 4.14

Вычислим дисперсию по формуле:

Дисперсия: 2 \* 0.096 \* (3 - 4.14)2 + 0.096 \* (4 - 4.14)2 + 7 \* 0.06 \* (4 - 4.14)2 + 0.06 \* (5 - 4.14)2 + 8 \* 0.03 \* (5 - 4.14)2 = 0.48

Файл сжатия: Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer rutrum elementum metus, vel commodo risus eleifend eget. Donec id tortor condimentum, mattis nibh a, lobortis leo. Vivamus vel elementum augue, eget accumsan purus. In eget augue eros. Suspendisse consectetur, mauris nec tempus suscipit, quam odio euismod urna, id consequat turpis erat volutpat nunc. Ut quis egestas nibh. Maecenas ultrices nibh leo, nec mollis nisi consequat non. Suspendisse potenti.

Размер файла до сжатия: 468 байт.

Размер файла после сжатия архиватором win.rar: 366 байт (коэффициент = 0.782).

Размер файла после сжатия разработанной программой (м. Хаффмана): 1971 бит = 247 байт (коэффициент = 0.521).

Код с реализацией алгоритма Хаффмана представлен в листинге 1.

*Листинг 1*

|  |
| --- |
| #include <iostream> #include <string> #include <queue> #include <unordered\_map> using namespace std;  struct Node {  char ch;  int freq;  Node\* left, \* right; };  Node\* getNode(char ch, int freq, Node\* left, Node\* right) {  Node\* node = new Node();   node->ch = ch;  node->freq = freq;  node->left = left;  node->right = right;   return node; }  struct comp {  bool operator()(Node\* l, Node\* r)  {  // highest priority item has lowest frequency  return l->freq > r->freq;  } };  void encode(Node\* root, string str,  unordered\_map<char, string>& huffmanCode) {  if (root == nullptr)  return;   // found a leaf node  if (!root->left && !root->right) {  huffmanCode[root->ch] = str;  }   encode(root->left, str + "0", huffmanCode);  encode(root->right, str + "1", huffmanCode); }  void decode(Node\* root, int& index, string str) {  if (root == nullptr) {  return;  }   // found a leaf node  if (!root->left && !root->right)  {  cout << root->ch;  return;  }   index++;   if (str[index] == '0')  decode(root->left, index, str);  else  decode(root->right, index, str); }  void buildHuffmanTree(string text) {  unordered\_map<char, int> freq;  for (char ch : text) {  freq[ch]++;   }   priority\_queue<Node\*, vector<Node\*>, comp> pq;   for (auto pair : freq) {  pq.push(getNode(pair.first, pair.second, nullptr, nullptr));  }   while (pq.size() != 1)  {  Node\* left = pq.top(); pq.pop();  Node\* right = pq.top(); pq.pop();   int sum = left->freq + right->freq;  pq.push(getNode('\0', sum, left, right));  }   Node\* root = pq.top();   unordered\_map<char, string> huffmanCode;  encode(root, "", huffmanCode);   cout << "Кодирование Хаффманом:\n";  for (auto pair : huffmanCode) {  cout << pair.first << " " << pair.second << '\n';  }   cout << "\nНачальная строка:\n" << text << '\n';  string str = "";  for (char ch : text) {  str += huffmanCode[ch];  }   cout << "\nЗакодированная строка:\n" << str << '\n';   cout << "\nРазмер закодированной строки:\n" << str.size() << '\n';    cout << "\nДекодированная строка:\n";  int index = -1;  while (index < (int)str.size() - 2) {  decode(root, index, str);  }  }  int main() {  setlocale(LC\_ALL, "Russian");  //string text = "kuznetcov andrei alexsandrovich";  string text = "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer rutrum elementum metus, vel commodo risus eleifend eget. Donec id tortor condimentum, mattis nibh a, lobortis leo. Vivamus vel elementum augue, eget accumsan purus. In eget augue eros. Suspendisse consectetur, mauris nec tempus suscipit, quam odio euismod urna, id consequat turpis erat volutpat nunc. Ut quis egestas nibh. Maecenas ultrices nibh leo, nec mollis nisi consequat non. Suspendisse potenti.";  buildHuffmanTree(text);  return 0; } |
|  |

Для метода Шеннона-Фано:

Описание алгоритма:

1) Символы первичного алфавита сортируются по невозрастанию.

2) Находится середина алфавита, которая делит на две примерно равные части. Для первой части присваивается двоичная цифра «0», а для второй цифра «1», таким образом мы получим листья дерева.

3) Полученные части рекурсивно делятся, и их частям назначаются соответствующие двоичные цифры в префиксном коде.

Описание алгоритма восстановления сжатого текста:

Для декодирования сжатого текста строится то же дерево, которое было построено на этапе сжатия. Начиная с корневого узла, из сжатого потока битов выделяется по одному биту. Если бит является нулевым, происходит перемещение влево, если единичным - вправо. Процесс продолжается до тех пор, пока не будет найден лист дерева, т.е. символ. Полученный символ выводится в поток восстановленных данных. Затем начинается тот же процесс с корневого узла дерева с целью извлечения следующего бита.

Файл сжатия: Pellentesque congue lacinia magna quis vehicula. Nam at rutrum est. Nulla elementum fringilla dolor. In sed accumsan metus. Sed sit amet pretium est. Phasellus interdum sodales ex ut rhoncus. Pellentesque consequat ut libero a suscipit. Proin vitae volutpat orci, id tempor mi. Mauris tincidunt arcu ut tellus ornare finibus. Aenean est lacus, commodo a sem non, maximus rutrum dolor. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris viverra ipsum nulla, eu pulvinar sapien fringilla congue.

Размер файла до сжатия: 4344 бит.

Размер файла после сжатия архиватором win.rar: 3 344 бит (коэффициент = 0.77).

Размер файла после сжатия разработанной программой (м. Шеннона-Фано): 2299 бит (коэффициент = 0.53).

Код с реализацией алгоритма Хаффмана представлен в листинге 2.

*Листинг 2*

|  |
| --- |
| #include <iostream> #include <vector> #include <string> #include <algorithm> #include <map>  using namespace std;  map<char, string> alphabet\_code; map<string, char> alphabet\_decode; vector<pair<char, int>> alphabet; string decode\_str = "";  // Информационный узел дерева struct Node {  string key; // значение узла  Node\* parent; // родитель  Node\* left; // левый "ребенок"  Node\* right; // правый "ребенок"   // конструктор  Node(string key)  {  this->key = key;  this->left = nullptr;  this->right = nullptr;  this->parent = nullptr;  }; };   Node\* code\_tree = new Node(""); // кодовое дерево   vector<pair<char, int>> make\_alphabet\_tabel(string str) {  map<char, int> unsorted\_alphabet;  vector<pair<char, int>> result\_alphabet;    for (int i = 0; i < str.length(); i++)  {  if (unsorted\_alphabet.count(str[i]) == 0) unsorted\_alphabet[str[i]] = 1; // если буква встретилась впервые  else unsorted\_alphabet[str[i]]++;  }    copy(unsorted\_alphabet.begin(), unsorted\_alphabet.end(), back\_inserter(result\_alphabet)); // копируем map в вектор пар  // сортируем вектор по неубыванию  sort(result\_alphabet.begin(), result\_alphabet.end(), [](const auto& x, const auto& y) { return x.second > y.second; });  return result\_alphabet; }  void make\_tree(int start, int end, Node\* parent, char code) {  if (code == '0') {  parent->left = new Node(parent->key + code); // дополняем код символа  parent = parent->left; // спускаемся влево  }  else if (code == '1') {  parent->right = new Node(parent->key + code); // дополняем код символа  parent = parent->right; // спускаемся вправо  }  if (start == end) { // дошли до листа дерева  alphabet\_code[alphabet[start].first] = parent->key; // поплняем словарь буква - кода  alphabet\_decode.insert(make\_pair(parent->key, alphabet[start].first));  return;  }   int size = 0;  for (int i = start; i < end; i++) {  size += alphabet[i].second;  }  // ищем середину  int left\_size = 0;  int left\_it = start;  int middle = left\_it;  while (left\_size + alphabet[left\_it].second < (double)size \* 0.5 && left\_it < end) {  left\_size += alphabet[left\_it].second;  left\_it++;  middle++;  }  make\_tree(start, middle, parent, '0'); // рекурсивно идем по левой части  make\_tree(middle + 1, end, parent, '1'); // рекурсивно идем по правой части }   void ShannonFanoDecode(string code, string s, Node\* current) {  // exit criteria  if (s.length() == 0)  {  cout << alphabet\_decode[code] << endl;  return;  }  // if we have 0 and left node  else if (s[0] == '0' && current->left)  ShannonFanoDecode(code + "0", s.substr(1), current->left);  // if we have 1 and right node  else if (s[0] == '1' && current->right)  ShannonFanoDecode(code + "1", s.substr(1), current->right);  // if we dont have nodes  else if (s[0] == '0' || s[0] == '1')  {  cout << alphabet\_decode[code];  ShannonFanoDecode("", s, code\_tree);  }  else ShannonFanoDecode(code, s.substr(1), current); }  int main() {  setlocale(LC\_ALL, "Russian");  //string s = "One, two, Freddy's coming for you Three, four, better lock your door Five, six, grab a crucifix Seven, eight, gonna stay up late.";  string s = "Pellentesque congue lacinia magna quis vehicula. Nam at rutrum est. Nulla elementum fringilla dolor. In sed accumsan metus. Sed sit amet pretium est. Phasellus interdum sodales ex ut rhoncus. Pellentesque consequat ut libero a suscipit. Proin vitae volutpat orci, id tempor mi. Mauris tincidunt arcu ut tellus ornare finibus. Aenean est lacus, commodo a sem non, maximus rutrum dolor. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris viverra ipsum nulla, eu pulvinar sapien fringilla congue.";   alphabet = make\_alphabet\_tabel(s);  cout << "Частотный алфавит:\n"; // выводим алфавит  for (int i = 0; i < alphabet.size(); i++)  cout << "\"" << alphabet[i].first << "\" - " << alphabet[i].second << endl;   make\_tree(0, alphabet.size() - 1, code\_tree, NULL); // формируем дерево   string res\_code = ""; // результат кодирования  cout << "Код, полученный алгоритмом Шеннона-Фано:\n";  for (int i = 0; i < s.length(); i++) {  res\_code += alphabet\_code[s[i]];  }   cout << res\_code;  cout << endl;  cout << "Незакодированная фраза: " << s.length() \* 8 << " бит" << endl;  cout << "Закодированная фраза: " << res\_code.length() << " бит" << endl;  cout << "Коэффициент сжатия: " << (double)res\_code.length() / (s.length() \* 8) << endl;  cout << "Рез-т декодирования: ";  ShannonFanoDecode("", res\_code, code\_tree);  return 0; } |
|  |

# Тестирование

Проведем тестирование программы Шеннона-Фано, рисунки 2-3, и программы Хаффмана, рисунки 4-5, на примере фразы индивидуального варианта, фразы: One, two, Freddy's coming for you Three, four, better lock your door Five, six, grab a crucifix Seven, eight, gonna stay up late.

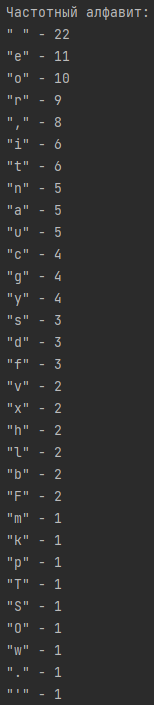


Рисунок 2 – Тестирование программы Шеннона-Фано

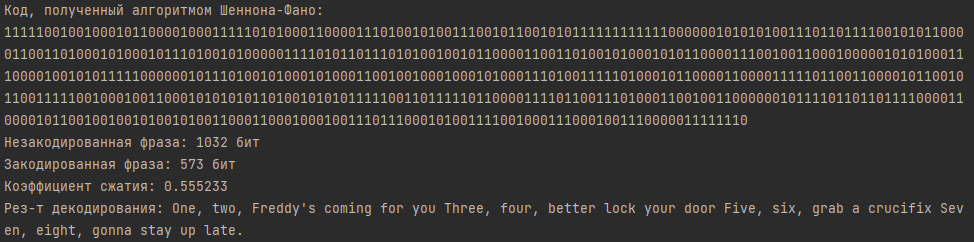


Рисунок 3 – Продолжение тестирования программы Шеннона-Фано

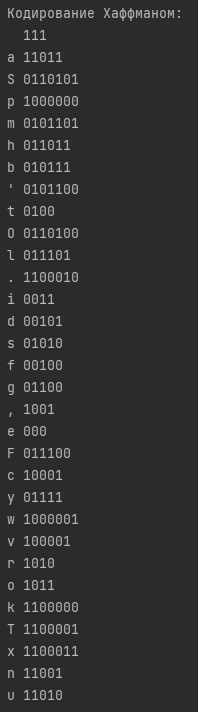


Рисунок 4 – Тестирование программы Хаффмана

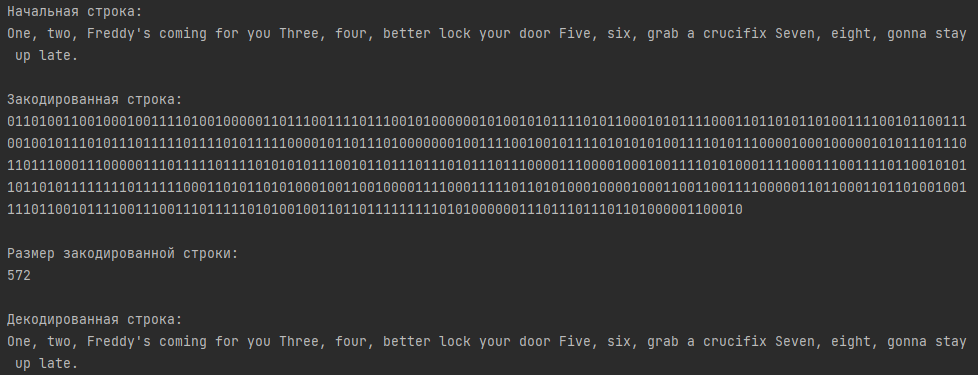


Рисунок 5 – Продолжение тестирования программы Хаффмана

# Вывод

В результате выполнения работы были получены знания и практические навыки по выполнению операций и применению алгоритмов для работы с различными алгоритмами сжатия и кодирования данных.