|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |   Институт Информационных технологий | |
|  | |
| Кафедра Математического обеспечения и стандартизации информационных технологий | |
|  | |
|  | |

|  |  |
| --- | --- |
| **ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 7** | |
| **по дисциплине** | |
| **«**Структуры и алгоритмы обработки данных**»**  **Тема: «Кодирование и сжатие данных методами без потерь.»** | |
|  | |
| Выполнил студент группы ИКБО-04-21 | Козин Г.А. |
| Принял преподаватель | Филатов А.С. |

|  |  |  |
| --- | --- | --- |
| Лабораторная работа выполнена | «01» ноября 2022г. | *(подпись студента)* |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись руководителя)* |

Москва 2022

**ВАРИАНТ № 10**

# **Цель работы**

Получение практических навыков и знаний по выполнению сжатия данных рассматриваемыми методами.

# **Задание**

1. Составить программу сжатия текста, используя метод RLE (run length encoding/кодирование длин серий/групповое кодирование). Рассчитать коэффициент сжатия в тестах.
   1. Провести тестирование на длинной серии повторяющихся символов.
   2. Провести тестирование на длинной серии неповторяющихся символов.
2. Составить программу сжатия текста алгоритмом Лемпеля-Зива (LZ77), LZ78. Рассчитать коэффициент сжатия в тестах.
   1. Провести тестирование реализации LZ77 в соответствии с индивидуальным вариантом задания, используя двухсимвольный алфавит (0, 1). Описать в отчете процесс восстановления полученного сжатого текста.
   2. Провести тестирование реализации LZ78 в соответствии с индивидуальным вариантом задания. Описать в отчете процесс восстановления полученного сжатого текста.
3. Составить программу сжатия и восстановления текста алгоритмами Шеннона-Фано и Хаффмана.
   1. Провести тестирование метода Шеннона-Фано в соответствии с индивидуальным вариантом задания. Представить в отчете таблицу формирования кода, изобразить префиксное дерево, рассчитать коэффициент сжатия.
   2. Провести тестирование метода Хаффмана на строке, содержащей ваше ФИО. Построить таблицу частот встречаемости символов, сформировать алфавит исходной строки и посчитать частоту вхождений и вероятность появления символов. Изобразить префиксное дерево Хаффмана. Показать кодирование каждого символа в строке. Рассчитать коэффициент сжатия, среднюю длину кода и дисперсию.

# **3. Решение**

Прежде чем рассказывать об алгоритмах сжатия, дадим определение:

Сжатие данных — алгоритмическое (обычно обратимое) преобразование данных, производимое с целью уменьшения занимаемого ими объёма. Применяется для более рационального использования устройств хранения и передачи данных.

В данной практической работе нам предлагается изучить алгоритмы сжатия:

1. RLE —это простейший алгоритм сжатия данных, основанный на поиске цепочек одинаковых символов. При кодировании строка одинаковых символов, составляющих серию, заменяется строкой, которая содержит сам повторяющийся символ и количество его повторов.
2. LZ — Алгоритм Лемпеля-Зива, это универсальный алгоритм сжатия данных без потерь.  Последовательно считываются символы входного потока и происходит проверка, существует ли в созданной таблице строк такая строка. Если такая строка существует, считывается следующий символ, а если строка не существует, в поток заносится код для предыдущей найденной строки, строка заносится в таблицу, а поиск начинается снова.
3. Шеннона-Фано — это метод энтропийного кодирования для сжатия мультимедийных данных без потерь, он присваивает код каждому символу на основе их вероятности появления.
4. Хаффман — жадный алгоритм оптимального префиксного кодирования алфавита с минимальной избыточностью.

Для реализации RLE алгоритма, были созданы функции EncodingRLE и DecodingRLE типа string. Функция EncodingRLE предназначена для кодирования по алгоритму RLE, DecodingRLE предназначена для декодирования:

|  |
| --- |
| string EncodingRLE(string line) {  if (line.length() > 0)  {  char first = line[0];  char second = line[1];  int count = 0;  string rleline = "";  string tmp = "";  for (int i = 1; i < line.length(); i++)  {  second = line[i];  if (first == second)  {  if (count == 0)  count = 2;  else if(count > 0)  count++;  else  {  rleline += to\_string(count) + tmp;  count = 2;  tmp = "";  }  }  else  {  if (count == 0)  {  count = -1;  tmp += first;  }  else if (count < 0)  {  count--;  tmp += first;  }  else  {  rleline += to\_string(count) + first;  count = 0;  }  }  first = second;  }  if(count > 0)  rleline += to\_string(count) + first;  else  rleline += to\_string(count-1) + tmp + second;  return rleline;  }  else  return ""; }  string DecodingRLE(string str) {  string res = "", temp = "";  int i = 0;  while (i < str.length()) {  if (str[i] == '-')  {  i++;  while (str[i] >= 48 && str[i] <= 57)  i++;  while (i < str.length() && !(str[i] >= 48 && str[i] <= 57) && str[i] != 45)  {  res += str[i];  i++;  }  }  else if (str[i] >= 48 && str[i] <= 57)  {  while (str[i] >= 48 && str[i] <= 57)  {  temp += str[i];  i++;  }  for (int j = 0; j < stoi(temp); j++)  res += str[i];  i++;  temp = "";  }  }  return res; } |

Для реализации LZ77 алгоритма, были созданы функции EncodingLZ77 и DecodingLZ77 типа string. Функция EncodingLZ77 предназначена для кодирования по алгоритму LZ77, DecodingLZ77 предназначена для декодирования:

|  |
| --- |
| string EncodingLZ77(string str) {  int maxWindow = 10;  int window1, window2;  string result = "";  for (int i = 0; i < str.length() - 1; i++) {  if (i + 1 <= maxWindow) window1 = i;  else window1 = maxWindow;   if (i + window1 < str.length()) window2 = window1;  else window2 = str.length() - i;    string str1 = str.substr(i - window1, window1);  string str2 = str.substr(i, window2);   int off = -1;  while (window2 > 0)  {  string str3 = str2.substr(0, window2);  off = str1.find(str3);  if (off != -1) break;  window2--;  }  if (off != -1)  {  result += "<" + to\_string(window1 - off) + "," + to\_string(window2) + ",";  result += str[i + window2];  result += ">";  i += window2;  }  else  {  result += "<0,0,";  result += str[i + window2];  result += ">";  }  }  result += "<0,0,";  result += str[str.length()-1];  result += ">";  return result + "-1"; }  string DecodingLZ77(string str) {  string result = "";  int tmp1 = 0;  int tmp2 = 0;  string buff = "";  bool flg = false;  for(int i = 0; i < str.length(); i++)  {  if (str[i] == '>')  flg = false;  if (flg)  buff += str[i];  if (str[i] == '<')  {  flg = true;  buff = "";  }  else if (str[i] == ',')  {  tmp1 = tmp2;  tmp2 = stoi(buff);  buff = "";  }  if(!flg)  {  if (tmp1 != 0 && tmp2 != 0)  buff = result.substr(result.length() - tmp1, tmp2) + buff;  result += buff;  flg = true;  }  }  return result; } |

Для реализации LZ78 алгоритма, были созданы функции EncodingLZ78 и DecodingLZ78 типа string. Функция EncodingLZ78 предназначена для кодирования по алгоритму LZ78, DecodingLZ78 предназначена для декодирования:

|  |
| --- |
| string EncodingLZ78(string str) {  vector<string> res;  bool flg = false;  string strResult = "";  int indicator = 0;  for (int i = 0; i < str.length(); i++)  {  string str1 = str.substr(i, 1);  int count = 1;  indicator = 0;  do  {  flg = false;  for (int j = 0; j < res.size(); j++)  {  if (res[j] == str1)  {  flg = true;  indicator = j+1;  break;  }  }  if (flg)  {  if (i + count >= str.length())  break;  count++;  str1 = str.substr(i, count);  }  } while (flg);  if (flg)  break;  else  res.push\_back(str1);  strResult += "<" + to\_string(indicator) + ",";  strResult += res[res.size() - 1][res[res.size() - 1].length() - 1];  strResult += ">";  i += count - 1;  }  if (!flg)  strResult += "<0,EOF>";  else  strResult += "<" + to\_string(indicator) + ",EOF>";  return strResult; }  string DecodingLZ78(string str) {  vector<string> array;  bool flg = false;  string buff = "";  int tmp = 0;  string result ="";  for(int i = 0; i < str.length(); i++)  {   if (str[i] == '>')  flg = false;  if (flg)  buff += str[i];  if (str[i] == '<')  {  flg = true;  buff = "";  }  else if(str[i] == ',')  {  tmp = stoi(buff);  buff = "";  }  if (!flg)  {  if (buff == "EOF")  buff = "";  if (tmp != 0)  buff = array[tmp - 1] + buff;  array.push\_back(buff);  result += buff;  }  }  return result; } |

Для реализации алгоритма Шеннона–Фано был создан класс ShannonAndFano, в котором содержатся public и private поля. В private поле содержатся методы для построения дерева, а в public конструктор и метод, предназначенный для кодирования:

|  |
| --- |
| class ShannonAndFano {  private:  vector<Symbol> symbols;  bool number = 0;  ShannonAndFano\* left = nullptr;  ShannonAndFano\* right = nullptr;  Symbol GetMaxSymbol();  void SetSymbols(vector<Symbol> symbols);  void CreateTree();  void AddingSymbol(Symbol symbol);  bool SearchCodeOfSymbol(char symbol, string& result);  public:  ShannonAndFano(bool number = 0, ShannonAndFano\* left = nullptr, ShannonAndFano\* right = nullptr);  string EncodingMethod(string data); }; |

Параметризированный конструктор ShannonAndFano класса ShannonAndFano:

|  |
| --- |
| ShannonAndFano::ShannonAndFano(bool number, ShannonAndFano\* left, ShannonAndFano\* right) {  this->number = number;  this->left = left;  this->right = right; } |

Метод SetSymbols класса ShannonAndFano, данный метод предназначен для установления текущего символа:

|  |
| --- |
| void ShannonAndFano::SetSymbols(vector<Symbol> symbols) {  this->symbols = symbols; } |

Метод GetMaxSymbol класса ShannonAndFano, данный метод предназначен для получения максимального символа:

|  |
| --- |
| Symbol ShannonAndFano::GetMaxSymbol() {  float max = 0.0f;  int maxPos = 0;  for (int i = 0; i < symbols.size(); i++)  {  if (symbols[i].frequency > max)  {  maxPos = i;  max = symbols[i].frequency;  }  }  Symbol result = symbols[maxPos];  symbols.erase(symbols.begin() + maxPos);  return result; } |

Метод CreateTree класса ShannonAndFano, данный метод предназначен для создания дерева, в последствии чего данный метод поможет с переводом в двоичную систему счисления:

|  |
| --- |
| void ShannonAndFano::CreateTree() {  vector<Symbol> leftSymbols;  vector<Symbol> rightSymbols;  vector<Symbol> saveSymbols = symbols;  while (getSumFrequency(leftSymbols) < getSumFrequency(saveSymbols) / 2.0f)  {  leftSymbols.push\_back(GetMaxSymbol());  }  while (symbols.size() > 0)  {  rightSymbols.push\_back(GetMaxSymbol());  }  symbols = saveSymbols;  ShannonAndFano\* leftNode = new ShannonAndFano(1);  leftNode->SetSymbols(leftSymbols);  if (leftSymbols.size() > 1)  {  leftNode->CreateTree();  }  ShannonAndFano\* rightNode = new ShannonAndFano(0);  rightNode->SetSymbols(rightSymbols);  if (rightSymbols.size() > 1)  {  rightNode->CreateTree();  }  left = leftNode;  right = rightNode; } |

Метод AddingSymbol класса ShannonAndFano, данный метод предназначен для добавления символов:

|  |
| --- |
| void ShannonAndFano::AddingSymbol(Symbol symbol) {  for (int i = 0; i < symbols.size(); i++)  {  if (symbols[i].symbol == symbol.symbol)  return;  }  symbols.push\_back(symbol); } |

Метод EncodingMethod класса ShannonAndFano, данный метод предназначен для кодирования данных:

|  |
| --- |
| string ShannonAndFano::EncodingMethod(string data) {  system("chcp 1251 > null");  string result;   for (int i = 0; i < data.length(); i++)  {  float frequency = (float)getNumOfOccurrences(data, data[i]) / (float)data.length();  AddingSymbol(Symbol{ data[i], frequency });  }   CreateTree();   string code;  int size = 0;  for (int i = 0; i < data.length(); i++)  {  code.clear();  SearchCodeOfSymbol(data[i], code);  code = code.substr(0, code.length() - 1);  size += code.size();  result += " " + code;  }  cout << "Коэффициент сжатия: " << float(data.size() \* 8) / (size\*2) << endl;  cout << "Закодированная информация:" << result << endl;   return result; } |

Для реализации алгоритма Хаффмана был создан класс HaffmanAlgorithm, в котором содержатся public и private поля. В private поле содержатся методы для построения дерева, а в public конструктор и метод, предназначенный для кодирования:

|  |
| --- |
| class HuffmanAlgorithm {  private:  vector<Symbol> symbols;  bool number = 0;  HuffmanAlgorithm\* left = nullptr;  HuffmanAlgorithm\* right = nullptr;  HuffmanAlgorithm\* GetMinNode(vector<HuffmanAlgorithm\*>& nodes);  void AddingNode(vector<HuffmanAlgorithm\*>& nodes, HuffmanAlgorithm\* node);  bool FindCodeOfSymbol(char symbol, string& result);  unsigned char GetBiteByBits(const string& bits); // By DR. DRE XD  public:  HuffmanAlgorithm(vector<Symbol> symbols, bool number = 0, HuffmanAlgorithm\* left = nullptr, HuffmanAlgorithm\* right = nullptr);  void EncodingFile(string fileInputName, string fileOutputName);  string Encoding(string data);  void printTreeWithTurn(int levelNum = 0); }; |

Параметризированный конструктор HaffmanAlgorithm класса HaffmanAlgorithm:

|  |
| --- |
| HuffmanAlgorithm::HuffmanAlgorithm(vector<Symbol> symbols, bool number, HuffmanAlgorithm\* left, HuffmanAlgorithm\* right) {  this->symbols = symbols;  this->number = number;  this->left = left;  this->right = right; } |

Метод SecondPrintTree класса HaffmanAlgorithm предназначен для вывода дерева:

|  |
| --- |
| void HuffmanAlgorithm::SecondPrintTree(int levelNum) {  if (left == nullptr && right == nullptr)  cout << string(levelNum \* 4, ' ') << "(" << number << "|" << getString(symbols) << "|" << getSumFrequency(symbols) << ")" << endl;  else  {  right->printTreeWithTurn(levelNum + 1);  if(levelNum != 0)  cout << string(levelNum \* 4, ' ') << "(" << number << "|" << getString(symbols) << "|" << getSumFrequency(symbols) << ")" << endl;  else  cout << string(levelNum \* 4, ' ') << "(-|" << getString(symbols) << "|" << getSumFrequency(symbols) << ")" << endl;  left->printTreeWithTurn(levelNum + 1);  } } |

Метод EncodingFile класса HaffmanAlgorithm кодирования файла, на вход поступает файл с данными, которые нужно сжать, а на выходе получается выходной файл со сжатыми данными:

|  |
| --- |
| void HuffmanAlgorithm::EncodingFile(string fileInputName, string fileOutputName) {  //fileInputName += ".txt";  //fileOutputName += ".dot";  fileInputName;  fileOutputName;   ifstream fin(fileInputName);  string input;  string data;   while (getline(fin, input))  data += input;  fin.close();   string result;  vector<HuffmanAlgorithm\*> nodes;  for (unsigned long int i = 0; i < data.size(); i++)  {  float frequency = getNumOfOccurrences(data, data[i]);  AddingNode(nodes, new HuffmanAlgorithm( vector<Symbol> { Symbol { data[i], frequency } }, 0, nullptr, nullptr ));   }   while (nodes.size() > 1)  {  HuffmanAlgorithm\* minNodeLeft = GetMinNode(nodes);  minNodeLeft->number = 0;  HuffmanAlgorithm\* minNodeRight = GetMinNode(nodes);  minNodeRight->number = 1;   HuffmanAlgorithm\* node = new HuffmanAlgorithm(vector<Symbol> { Symbol { '-', getSumFrequency(minNodeLeft->symbols) + getSumFrequency(minNodeRight->symbols) }, Symbol{}}, 0, minNodeLeft, minNodeRight );  nodes.push\_back(node);  }   this->symbols = nodes[0]->symbols;  this->number = nodes[0]->number;  this->left = nodes[0]->left;  this->right = nodes[0]->right;   string code;  unsigned int currentBytePos = 0;  string remainder;  unsigned char currentByte = 0;   for (unsigned int i = 0; i < data.size(); i++)  {  code.clear();  nodes[0]->FindCodeOfSymbol(data[i], code);  code = code.substr(0, code.length() - 1);   if (currentBytePos >= 8)  {  while (remainder.size() > 8)  {  result += GetBiteByBits(remainder.substr(0, 8));  remainder = remainder.substr(8);  }  result += currentByte;  currentByte = 0;  currentByte += GetBiteByBits(remainder);  currentByte <<= remainder.size();  currentBytePos = 0;  currentBytePos += remainder.size();  remainder.clear();  }  if (currentBytePos + code.size() < 8)  {  currentByte += GetBiteByBits(code);  currentByte <<= code.size();  currentBytePos += code.size();  }  else  {  string part = code.substr(0, 8 - currentBytePos);  currentByte += GetBiteByBits(part);  remainder = code.substr(8 - currentBytePos);  currentBytePos += part.size();  }  }   ofstream fout(fileOutputName, ios::binary);  fout.write(result.c\_str(), sizeof(char) \* result.size());  fout.close(); } |

Метод Encoding класса HaffmanAlgorithm предназначен для сжатия и вывода данных, так же он вызывает другие методы:

|  |
| --- |
| string HuffmanAlgorithm::Encoding(string data) {  string result;   vector<HuffmanAlgorithm\*> nodes;  for (unsigned int i = 0; i < data.size(); i++)  {  float frequency = getNumOfOccurrences(data, data[i]);  AddingNode(nodes, new HuffmanAlgorithm(vector<Symbol> { Symbol { data[i], frequency } }, 0, nullptr, nullptr));  }   cout << "Таблица частотности:" << endl;  for (unsigned int i = 0; i < nodes.size(); i++)  {  cout << getString(nodes[i]->symbols) << " : " << getSumFrequency(nodes[i]->symbols) << " " <<  float(getSumFrequency(nodes[i]->symbols)) / data.size() << endl;  }   while (nodes.size() > 1)  {  HuffmanAlgorithm\* minNodeLeft = GetMinNode(nodes);  minNodeLeft->number = 0;  HuffmanAlgorithm\* minNodeRight = GetMinNode(nodes);  minNodeRight->number = 1;   HuffmanAlgorithm\* node = new HuffmanAlgorithm(vector<Symbol> { Symbol { '-', getSumFrequency(minNodeLeft->symbols) + getSumFrequency(minNodeRight->symbols) }, Symbol{}}, 0, minNodeLeft, minNodeRight);  nodes.push\_back(node);  }   this->symbols = nodes[0]->symbols;  this->number = nodes[0]->number;  this->left = nodes[0]->left;  this->right = nodes[0]->right;   cout << "Префиксное дерево Хаффмана:" << endl;  SecondPrintTree();   int size = 0;  string code;  for (int i = 0; i < data.size(); i++)  {  code.clear();  nodes[0]->FindCodeOfSymbol(data[i], code);  code = code.substr(0, code.length() - 1);  size += code.size();  result += code + " ";  }   cout << "Коэффициент сжатия: " << float(data.size() \* 8) / (size \* 2) << endl;  cout << "Средняя длина кода: " << (float)size / data.size() << endl;   return result; } |

# **4. Демонстрация работы программы**

После запуска программы, пользователю предлагается выбрать номер упражнения в меню. После выбора, пользователю выводится информация о том, какие переменные ему нужно ввести или результат заданий, в которых значение не нужно вводить. В случае правильного ввода всех данных, программа выдает результат упражнения.

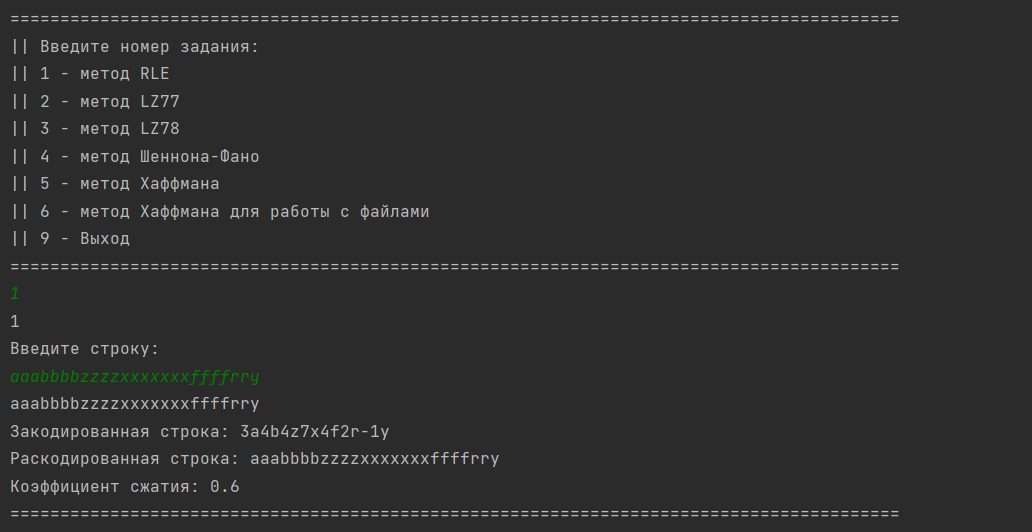


Рисунок 1. Метод RLE с повторяющимися символами

На рисунке 1 пользователь выбрал метод RLE, после чего он ввёл строку, которую надо сжать, заметим, что символы повторяются. После ввода, выводится закодированная строка, раскодированная строка и коэффициент сжатия.

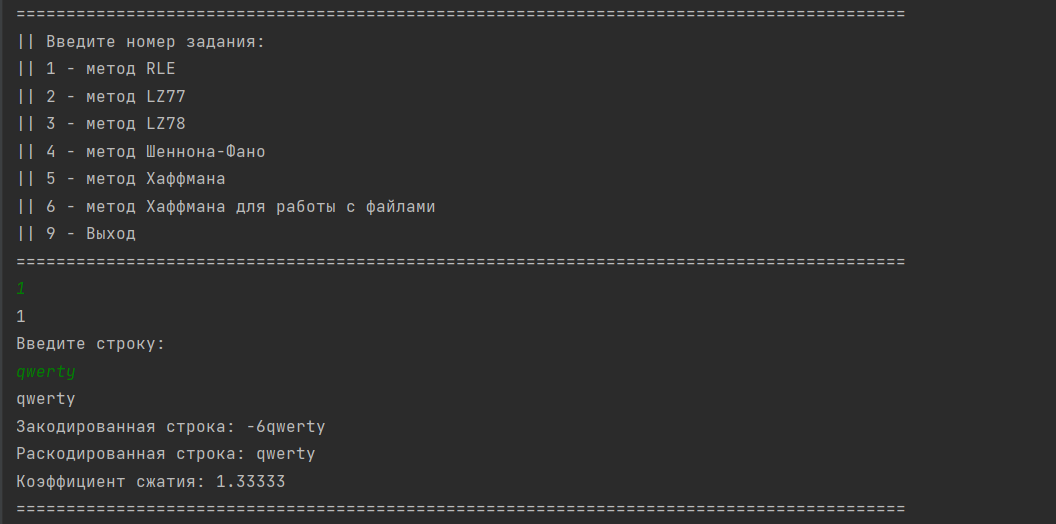


Рисунок 2. Метод RLE с не повторяющимися символами

На рисунке 2 пользователь выбрал метод RLE, после чего он ввёл строку, которую надо сжать, заметим, что символы не повторяются. После ввода, выводится закодированная строка, раскодированная строка и коэффициент сжатия.

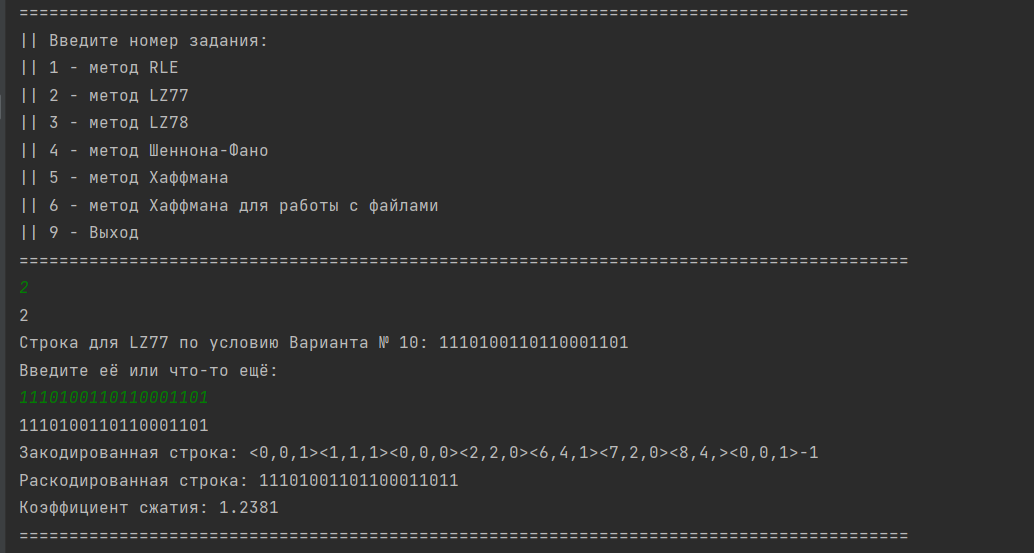


Рисунок 3. Метод LZ77

На рисунке 3 пользователь выбрал метод LZ77, после чего он ввёл строку, которую надо сжать, заметим, что ввести можно и индивидуальный вариант и что-то другое. После ввода, выводится закодированная строка, раскодированная строка и коэффициент сжатия.

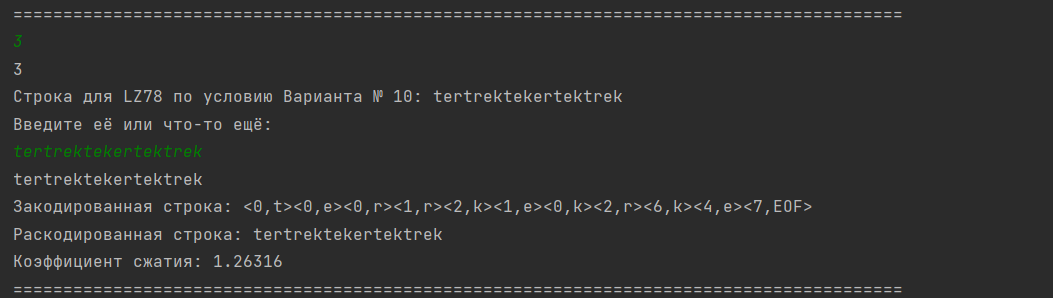


Рисунок 4. Вывод графа

На рисунке 4 пользователь выбрал метод LZ78, после чего он ввёл строку, которую надо сжать, заметим, что ввести можно и индивидуальный вариант и что-то другое. После ввода, выводится закодированная строка, раскодированная строка и коэффициент сжатия.

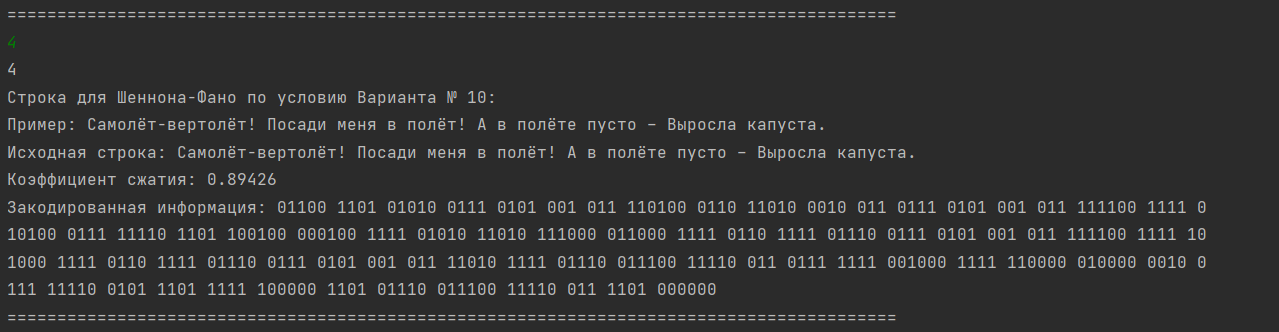


Рисунок 5. метод Шеннона-Фано

На рисунке 5 пользователь выбрал метод Шеннона-Фано, ему вывелся пример, который используется в сжатии. Потом вывелся коэффициент сжатия и закодированная информация.

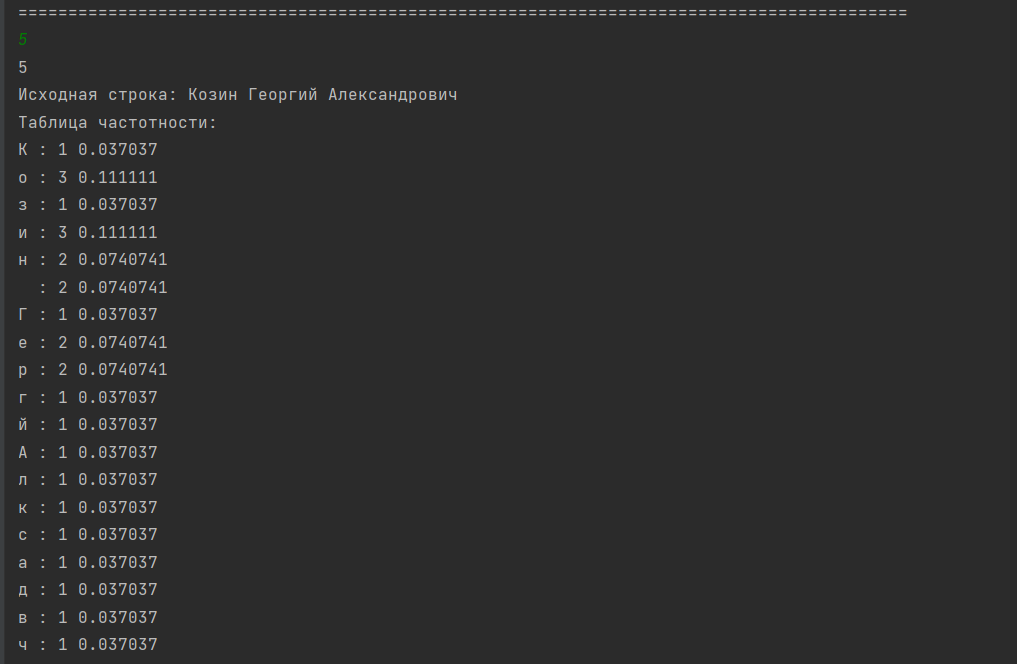


Рисунок 6. метод Хаффмана, часть 1

На рисунке 6 пользователь выбрал метод Хаффмана, после чего вывелась строка для кодировки, которую надо сжать, заметим, что после этого выводится таблица частности.

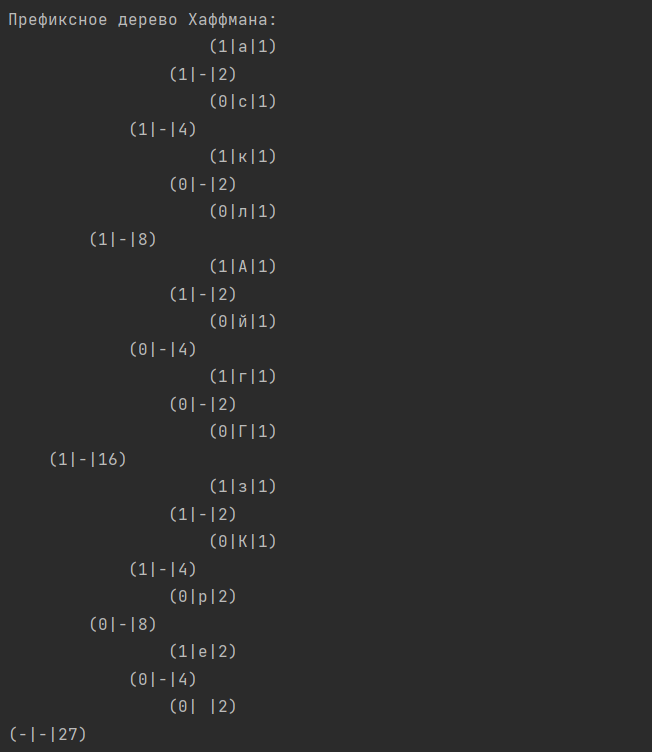


Рисунок 7. метод Хаффмана, часть 2

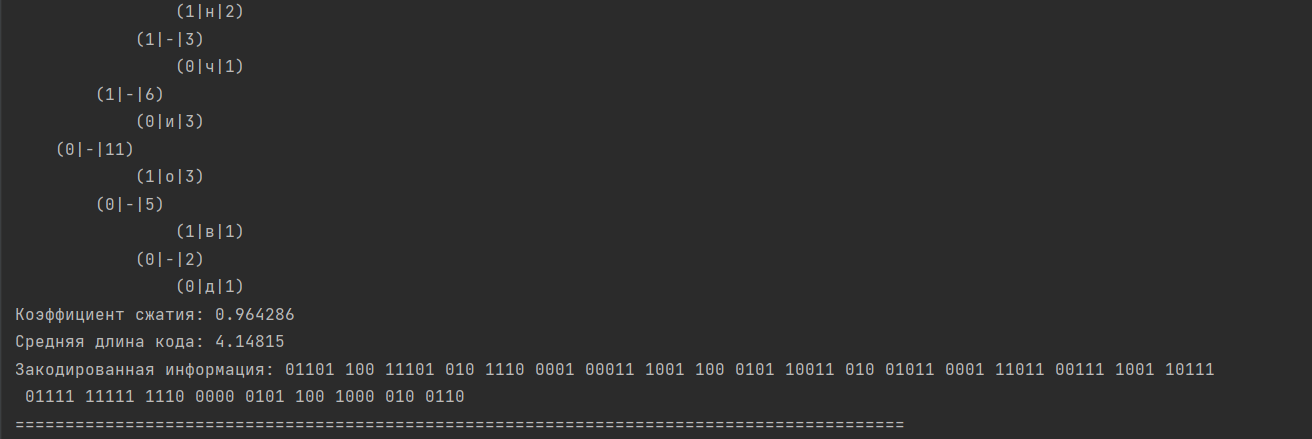


Рисунок 8. метод Хаффмана, часть 3

На рисунках 7 и 8 демонстрируются префиксное дерево метода Хаффмана, коэффициент сжатия, средняя длина кода и закодированная информация.

# **Вывод**

В результате выполнения данной практической работы, были получены практические навыки выполнения по выполнению сжатия данных рассматриваемыми методами.

# **Исходный код программы**

Main.cpp

|  |
| --- |
| #include "ShannonFano.h" #include "Huffman.h" #include "RLEandLZ.h" using namespace std;  int main() {  system("chcp 1251 > null");  while(true)  {  cout << "========================================================================================="<< endl;  cout << "|| Введите номер задания:"<< endl;  cout << "|| 1 - метод RLE" << endl;  cout << "|| 2 - метод LZ77" << endl;  cout << "|| 3 - метод LZ78" << endl;  cout << "|| 4 - метод Шеннона-Фано" << endl;  cout << "|| 5 - метод Хаффмана" << endl;  cout << "|| 6 - метод Хаффмана для работы с файлами" << endl;  cout << "|| 9 - Выход" << endl;  cout << "========================================================================================="<< endl;  int menu;  cin >> menu;  if(menu == 1)  {  string str;  cout << "Введите строку:" << endl;  getline(cin >> ws, str);  str = EncodingRLE(str);  int size = str.length();  cout << "Закодированная строка: " << str << endl;  str = DecodingRLE(str);  cout << "Раскодированная строка: " << str << endl;  cout << "Коэффициент сжатия: " << ((float)size / (float)str.length()) << endl;  }  else if (menu == 2)  {  cout << "Строка для LZ77 по условию Варианта № 10: 1110100110110001101" << endl;  cout << "Введите её или что-то ещё:" << endl;  string str;  cin >> str;  str = EncodingLZ77(str);  int size = 0;  for (int i = 0; i < str.length(); i++)  {  if (str[i] != '<' && str[i] != '>' && str[i] != ',')  {  size++;  }  }  cout << "Закодированная строка: " << str << endl;  str = DecodingLZ77(str);  cout << "Раскодированная строка: " << str << endl;  cout << "Коэффициент сжатия: " << ((float)size / (float)str.length()) << endl;  }  else if(menu == 3)  {  cout << "Строка для LZ78 по условию Варианта № 10: tertrektekertektrek" << endl;  cout << "Введите её или что-то ещё:" << endl;  string str;  cin >> str;  str = EncodingLZ78(str);  int size = 0;  for (int i = 0; i < str.length(); i++)  {  if (str[i] != '<' && str[i] != '>' && str[i] != ',')  size++;  }  cout << "Закодированная строка: " << str << endl;  str = DecodingLZ78(str);  cout << "Раскодированная строка: " << str << endl;  cout << "Коэффициент сжатия: " << ((float)size / (float)str.length()) << endl;   }  else if (menu == 4)  {  cout << "Строка для Шеннона-Фано по условию Варианта № 10:" << endl;  cout << "Пример: Самолёт-вертолёт! Посади меня в полёт! А в полёте пусто – Выросла капуста." << endl;  //cout << "Введите её или что-то ещё:" << endl;  string str = "Самолёт-вертолёт! Посади меня в полёт! А в полёте пусто – Выросла капуста.";  //cin >> str;  cout << "Исходная строка: " << str << endl;  ShannonAndFano SaF;  SaF.EncodingMethod(str);  }  else if (menu == 5)  {  string str = "Козин Георгий Александрович";  cout << "Исходная строка: " << str << endl;  vector<Symbol> symbols;  HuffmanAlgorithm huffman(symbols);  str = huffman.Encoding(str);  cout << "Закодированная информация: " << str << endl;  }  else if (menu == 6)  {  string inputFileName, outputFileName;  cout << "Введите название файла с входными файлами:" << endl;  cin >> inputFileName;  cout << "Введите название файла, в который будет записан результат:" << endl;  cin >> outputFileName;  vector<Symbol> symbols;  HuffmanAlgorithm huffman(symbols);  huffman.EncodingFile(inputFileName, outputFileName);  }  else if (menu == 9)  {  cout << "Конец программы !" << endl;  exit(0);  }  } } |

RLEandLZ.h

|  |
| --- |
| #ifndef MYFINALSIAOD\_7\_RLEANDLZ\_H #define MYFINALSIAOD\_7\_RLEANDLZ\_H  string EncodingRLE(string line) {  if (line.length() > 0)  {  char first = line[0];  char second = line[1];  int count = 0;  string rleline = "";  string tmp = "";  for (int i = 1; i < line.length(); i++)  {  second = line[i];  if (first == second)  {  if (count == 0)  count = 2;  else if(count > 0)  count++;  else  {  rleline += to\_string(count) + tmp;  count = 2;  tmp = "";  }  }  else  {  if (count == 0)  {  count = -1;  tmp += first;  }  else if (count < 0)  {  count--;  tmp += first;  }  else  {  rleline += to\_string(count) + first;  count = 0;  }  }  first = second;  }  if(count > 0)  rleline += to\_string(count) + first;  else  rleline += to\_string(count-1) + tmp + second;  return rleline;  }  else  return ""; }  string DecodingRLE(string str) {  string res = "", temp = "";  int i = 0;  while (i < str.length()) {  if (str[i] == '-')  {  i++;  while (str[i] >= 48 && str[i] <= 57)  i++;  while (i < str.length() && !(str[i] >= 48 && str[i] <= 57) && str[i] != 45)  {  res += str[i];  i++;  }  }  else if (str[i] >= 48 && str[i] <= 57)  {  while (str[i] >= 48 && str[i] <= 57)  {  temp += str[i];  i++;  }  for (int j = 0; j < stoi(temp); j++)  res += str[i];  i++;  temp = "";  }  }  return res; }  string EncodingLZ77(string str) {  int maxWindow = 10;  int window1, window2;  string result = "";  for (int i = 0; i < str.length() - 1; i++) {  if (i + 1 <= maxWindow) window1 = i;  else window1 = maxWindow;   if (i + window1 < str.length()) window2 = window1;  else window2 = str.length() - i;    string str1 = str.substr(i - window1, window1);  string str2 = str.substr(i, window2);   int off = -1;  while (window2 > 0)  {  string str3 = str2.substr(0, window2);  off = str1.find(str3);  if (off != -1) break;  window2--;  }  if (off != -1)  {  result += "<" + to\_string(window1 - off) + "," + to\_string(window2) + ",";  result += str[i + window2];  result += ">";  i += window2;  }  else  {  result += "<0,0,";  result += str[i + window2];  result += ">";  }  }  result += "<0,0,";  result += str[str.length()-1];  result += ">";  return result + "-1"; }  string DecodingLZ77(string str) {  string result = "";  int tmp1 = 0;  int tmp2 = 0;  string buff = "";  bool flg = false;  for(int i = 0; i < str.length(); i++)  {  if (str[i] == '>')  flg = false;  if (flg)  buff += str[i];  if (str[i] == '<')  {  flg = true;  buff = "";  }  else if (str[i] == ',')  {  tmp1 = tmp2;  tmp2 = stoi(buff);  buff = "";  }  if(!flg)  {  if (tmp1 != 0 && tmp2 != 0)  buff = result.substr(result.length() - tmp1, tmp2) + buff;  result += buff;  flg = true;  }  }  return result; }  string EncodingLZ78(string str) {  vector<string> res;  bool flg = false;  string strResult = "";  int indicator = 0;  for (int i = 0; i < str.length(); i++)  {  string str1 = str.substr(i, 1);  int count = 1;  indicator = 0;  do  {  flg = false;  for (int j = 0; j < res.size(); j++)  {  if (res[j] == str1)  {  flg = true;  indicator = j+1;  break;  }  }  if (flg)  {  if (i + count >= str.length())  break;  count++;  str1 = str.substr(i, count);  }  } while (flg);  if (flg)  break;  else  res.push\_back(str1);  strResult += "<" + to\_string(indicator) + ",";  strResult += res[res.size() - 1][res[res.size() - 1].length() - 1];  strResult += ">";  i += count - 1;  }  if (!flg)  strResult += "<0,EOF>";  else  strResult += "<" + to\_string(indicator) + ",EOF>";  return strResult; }  string DecodingLZ78(string str) {  vector<string> array;  bool flg = false;  string buff = "";  int tmp = 0;  string result ="";  for(int i = 0; i < str.length(); i++)  {   if (str[i] == '>')  flg = false;  if (flg)  buff += str[i];  if (str[i] == '<')  {  flg = true;  buff = "";  }  else if(str[i] == ',')  {  tmp = stoi(buff);  buff = "";  }  if (!flg)  {  if (buff == "EOF")  buff = "";  if (tmp != 0)  buff = array[tmp - 1] + buff;  array.push\_back(buff);  result += buff;  }  }  return result; } #endif |

ShannonAndFano.h

|  |
| --- |
| #ifndef SHANNONFANO\_H #define SHANNONFANO\_H #include "Functions.h"  class ShannonAndFano {  private:  vector<Symbol> symbols;  bool number = 0;  ShannonAndFano\* left = nullptr;  ShannonAndFano\* right = nullptr;  Symbol GetMaxSymbol();  void SetSymbols(vector<Symbol> symbols);  void CreateTree();  void AddingSymbol(Symbol symbol);  bool SearchCodeOfSymbol(char symbol, string& result);  public:  ShannonAndFano(bool number = 0, ShannonAndFano\* left = nullptr, ShannonAndFano\* right = nullptr);  string EncodingMethod(string data); };  #endif |

ShannonAndFano.cpp

|  |
| --- |
| #include "ShannonFano.h"  ShannonAndFano::ShannonAndFano(bool number, ShannonAndFano\* left, ShannonAndFano\* right) {  this->number = number;  this->left = left;  this->right = right; }  void ShannonAndFano::SetSymbols(vector<Symbol> symbols) {  this->symbols = symbols; }  Symbol ShannonAndFano::GetMaxSymbol() {  float max = 0.0f;  int maxPos = 0;  for (int i = 0; i < symbols.size(); i++)  {  if (symbols[i].frequency > max)  {  maxPos = i;  max = symbols[i].frequency;  }  }  Symbol result = symbols[maxPos];  symbols.erase(symbols.begin() + maxPos);  return result; }   void ShannonAndFano::CreateTree() {  vector<Symbol> leftSymbols;  vector<Symbol> rightSymbols;  vector<Symbol> saveSymbols = symbols;  while (getSumFrequency(leftSymbols) < getSumFrequency(saveSymbols) / 2.0f)  {  leftSymbols.push\_back(GetMaxSymbol());  }  while (symbols.size() > 0)  {  rightSymbols.push\_back(GetMaxSymbol());  }  symbols = saveSymbols;  ShannonAndFano\* leftNode = new ShannonAndFano(1);  leftNode->SetSymbols(leftSymbols);  if (leftSymbols.size() > 1)  {  leftNode->CreateTree();  }  ShannonAndFano\* rightNode = new ShannonAndFano(0);  rightNode->SetSymbols(rightSymbols);  if (rightSymbols.size() > 1)  {  rightNode->CreateTree();  }  left = leftNode;  right = rightNode; }   void ShannonAndFano::AddingSymbol(Symbol symbol)  {  for (int i = 0; i < symbols.size(); i++)  {  if (symbols[i].symbol == symbol.symbol)  return;  }  symbols.push\_back(symbol);  }  bool ShannonAndFano::SearchCodeOfSymbol(char symbol, string& result) {  if (symbols.size() == 1)  {  if (symbols[0].symbol == symbol)  {  result += to\_string(number);  return true;  }  else  return false;  }  else  {  if (left->SearchCodeOfSymbol(symbol, result))  {  result += to\_string(number);  return true;  }  else if (right->SearchCodeOfSymbol(symbol, result))  {  result += to\_string(number);  return true;  }  return false;  } }  string ShannonAndFano::EncodingMethod(string data) {  system("chcp 1251 > null");  string result;   for (int i = 0; i < data.length(); i++)  {  float frequency = (float)getNumOfOccurrences(data, data[i]) / (float)data.length();  AddingSymbol(Symbol{ data[i], frequency });  }   CreateTree();   string code;  int size = 0;  for (int i = 0; i < data.length(); i++)  {  code.clear();  SearchCodeOfSymbol(data[i], code);  code = code.substr(0, code.length() - 1);  size += code.size();  result += " " + code;  }  cout << "Коэффициент сжатия: " << float(data.size() \* 8) / (size\*2) << endl;  cout << "Закодированная информация:" << result << endl;   return result; } |

Huffman.h

|  |
| --- |
| #ifndef HUFFMAN\_H #define HUFFMAN\_H #include "Functions.h" #include <fstream>  class HuffmanAlgorithm {  private:  vector<Symbol> symbols;  bool number = 0;  HuffmanAlgorithm\* left = nullptr;  HuffmanAlgorithm\* right = nullptr;  HuffmanAlgorithm\* GetMinNode(vector<HuffmanAlgorithm\*>& nodes);  void AddingNode(vector<HuffmanAlgorithm\*>& nodes, HuffmanAlgorithm\* node);  bool FindCodeOfSymbol(char symbol, string& result);  unsigned char GetBiteByBits(const string& bits); // By DR. DRE XD  public:  HuffmanAlgorithm(vector<Symbol> symbols, bool number = 0, HuffmanAlgorithm\* left = nullptr, HuffmanAlgorithm\* right = nullptr);  void EncodingFile(string fileInputName, string fileOutputName);  string Encoding(string data);  void SecondPrintTree(int levelNum = 0); };  #endif |

Huffman.cpp

|  |
| --- |
| #include "Huffman.h"  HuffmanAlgorithm::HuffmanAlgorithm(vector<Symbol> symbols, bool number, HuffmanAlgorithm\* left, HuffmanAlgorithm\* right) {  this->symbols = symbols;  this->number = number;  this->left = left;  this->right = right; }  HuffmanAlgorithm\* HuffmanAlgorithm::GetMinNode(vector<HuffmanAlgorithm\*>& nodes) {  float min = 1000.f;  int minPos = 0;  for (unsigned int i = 0; i < nodes.size(); i++)  {  float sumFrequency = getSumFrequency(nodes[i]->symbols);  if (sumFrequency < min)  {  minPos = i;  min = sumFrequency;  }  }  HuffmanAlgorithm\* result = nodes[minPos];  nodes.erase(nodes.begin() + minPos);   return result; }  void HuffmanAlgorithm::AddingNode(vector<HuffmanAlgorithm\*>& nodes, HuffmanAlgorithm\* node) {  for (unsigned int i = 0; i < nodes.size(); i++)  {  if (nodes[i]->symbols[0].symbol == node->symbols[0].symbol)  return;  }  nodes.push\_back(node); }  void HuffmanAlgorithm::SecondPrintTree(int levelNum) {  if (left == nullptr && right == nullptr)  cout << string(levelNum \* 4, ' ') << "(" << number << "|" << getString(symbols) << "|" << getSumFrequency(symbols) << ")" << endl;  else  {  right->SecondPrintTree(levelNum + 1);  if(levelNum != 0)  cout << string(levelNum \* 4, ' ') << "(" << number << "|" << getString(symbols) << "|" << getSumFrequency(symbols) << ")" << endl;  else  cout << string(levelNum \* 4, ' ') << "(-|" << getString(symbols) << "|" << getSumFrequency(symbols) << ")" << endl;  left->SecondPrintTree(levelNum + 1);  } }  string HuffmanAlgorithm::Encoding(string data) {  string result;   vector<HuffmanAlgorithm\*> nodes;  for (unsigned int i = 0; i < data.size(); i++)  {  float frequency = getNumOfOccurrences(data, data[i]);  AddingNode(nodes, new HuffmanAlgorithm(vector<Symbol> { Symbol { data[i], frequency } }, 0, nullptr, nullptr));  }   cout << "Таблица частотности:" << endl;  for (unsigned int i = 0; i < nodes.size(); i++)  {  cout << getString(nodes[i]->symbols) << " : " << getSumFrequency(nodes[i]->symbols) << " " <<  float(getSumFrequency(nodes[i]->symbols)) / data.size() << endl;  }   while (nodes.size() > 1)  {  HuffmanAlgorithm\* minNodeLeft = GetMinNode(nodes);  minNodeLeft->number = 0;  HuffmanAlgorithm\* minNodeRight = GetMinNode(nodes);  minNodeRight->number = 1;   HuffmanAlgorithm\* node = new HuffmanAlgorithm(vector<Symbol> { Symbol { '-', getSumFrequency(minNodeLeft->symbols) + getSumFrequency(minNodeRight->symbols) }, Symbol{}}, 0, minNodeLeft, minNodeRight);  nodes.push\_back(node);  }   this->symbols = nodes[0]->symbols;  this->number = nodes[0]->number;  this->left = nodes[0]->left;  this->right = nodes[0]->right;   cout << "Префиксное дерево Хаффмана:" << endl;  SecondPrintTree();   int size = 0;  string code;  for (int i = 0; i < data.size(); i++)  {  code.clear();  nodes[0]->FindCodeOfSymbol(data[i], code);  code = code.substr(0, code.length() - 1);  size += code.size();  result += code + " ";  }   cout << "Коэффициент сжатия: " << float(data.size() \* 8) / (size \* 2) << endl;  cout << "Средняя длина кода: " << (float)size / data.size() << endl;   return result; }  unsigned char HuffmanAlgorithm::GetBiteByBits(const string& bits) {  unsigned char result = 0;  for (int i = bits.size() - 1; i >= 0; i--)  result += (bits[i] - '0') \* powf(2, (bits.size() - 1) - i);  return result; }  bool HuffmanAlgorithm::FindCodeOfSymbol(char symbol, string& result) {  if (symbols.size() == 1)  {  if (symbols[0].symbol == symbol)  {  result += to\_string(number);  return true;  }  else  return false;  }  else  {  if (left->FindCodeOfSymbol(symbol, result))  {  result += to\_string(number);  return true;  }  else if (right->FindCodeOfSymbol(symbol, result))  {  result += to\_string(number);  return true;  }  return false;  } }  void HuffmanAlgorithm::EncodingFile(string fileInputName, string fileOutputName) {  //fileInputName += ".txt";  //fileOutputName += ".dot";  fileInputName;  fileOutputName;   ifstream fin(fileInputName);  string input;  string data;   while (getline(fin, input))  data += input;  fin.close();   string result;  vector<HuffmanAlgorithm\*> nodes;  for (unsigned long int i = 0; i < data.size(); i++)  {  float frequency = getNumOfOccurrences(data, data[i]);  AddingNode(nodes, new HuffmanAlgorithm( vector<Symbol> { Symbol { data[i], frequency } }, 0, nullptr, nullptr ));   }   while (nodes.size() > 1)  {  HuffmanAlgorithm\* minNodeLeft = GetMinNode(nodes);  minNodeLeft->number = 0;  HuffmanAlgorithm\* minNodeRight = GetMinNode(nodes);  minNodeRight->number = 1;   HuffmanAlgorithm\* node = new HuffmanAlgorithm(vector<Symbol> { Symbol { '-', getSumFrequency(minNodeLeft->symbols) + getSumFrequency(minNodeRight->symbols) }, Symbol{}}, 0, minNodeLeft, minNodeRight );  nodes.push\_back(node);  }   this->symbols = nodes[0]->symbols;  this->number = nodes[0]->number;  this->left = nodes[0]->left;  this->right = nodes[0]->right;   string code;  unsigned int currentBytePos = 0;  string remainder;  unsigned char currentByte = 0;   for (unsigned int i = 0; i < data.size(); i++)  {  code.clear();  nodes[0]->FindCodeOfSymbol(data[i], code);  code = code.substr(0, code.length() - 1);   if (currentBytePos >= 8)  {  while (remainder.size() > 8)  {  result += GetBiteByBits(remainder.substr(0, 8));  remainder = remainder.substr(8);  }  result += currentByte;  currentByte = 0;  currentByte += GetBiteByBits(remainder);  currentByte <<= remainder.size();  currentBytePos = 0;  currentBytePos += remainder.size();  remainder.clear();  }  if (currentBytePos + code.size() < 8)  {  currentByte += GetBiteByBits(code);  currentByte <<= code.size();  currentBytePos += code.size();  }  else  {  string part = code.substr(0, 8 - currentBytePos);  currentByte += GetBiteByBits(part);  remainder = code.substr(8 - currentBytePos);  currentBytePos += part.size();  }  }   ofstream fout(fileOutputName, ios::binary);  fout.write(result.c\_str(), sizeof(char) \* result.size());  fout.close(); } |