|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | 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-10-21 | Черномуров С.А. |
| Принял преподаватель | Филатов А.С. |

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

Москва 2022

# **Цель работы**

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

# **Постановка задачи**

**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 процесс восстановления сжатого текста.
   2. Провести тестирование реализации LZ78 в соответствии с индивидуальным вариантом задания. Описать процесс восстановления 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. Применить алгоритм к большому текстовому файлу. Архивировать тот же файл любым архиватором. Выполнить сравнительный анализ сжатия этими способами.
4. Составить отчет, отобразив в нем описание выполнения всех этапов разработки, тестирования и код всей программы со скриншотами результатов тестирования.

Вариант №7. Условие задания:

|  |  |
| --- | --- |
| Упражнение 1 | **Сжатие данных по методу Лемпеля-Зива LZ77:**  110101011001100001001  **Закодировать следующую фразу, используя код LZ78:**  долделдолдилделдил  **Закодировать фразу методами Шеннона-Фано и Хаффмана:**  Тише, мыши, кот на крыше, А котята ещё выше. Кот пошёл за молоком, А котята кувырком. |

# **Решение**

**Объяснение алгоритмов применяемых функций**

**Сжатие методом RLE**

Метод compress класса RLE пробегает по входной строке data, считает максимальные длины подстрок, состоящих из одного символа, после чего заменяет такие подстроки по формату “символ подстроки” + “длина подстроки” и возвращает полученную закодированную строку.

|  |
| --- |
| // Метод сжатия методом RLE  string RLE::compress(string data) {  if (data == "") return "";  string compressedData = "", buffer = "";  buffer += data[0];  for (int i = 1; i < data.size(); i++) {  if (buffer[0] == data[i] || buffer == "") buffer += data[i];  if (buffer[0] != data[i] || i == data.size()-1) {  compressedData += buffer[0];  compressedData += to\_string(buffer.size());  compressedData += " ";  buffer = data[i];  }  }  if (buffer != "") {  compressedData += buffer[0];  compressedData += to\_string(buffer.size());  }  return compressedData;  } |

**Сжатие методом LZ77**

Метод compress класса LZ77 посимвольно проходит входную строку data и заменяет подстроки на ссылки их повторений в тексте, где такие подстроки уже встречались, каждая ссылка имеет формат <смещение, длина, символ (если еще не встречался)>. Смещение показывает насколько нужно сместиться от правого конца строки-буфера buffer, а длина показывает сколько символов из буфера нужно написать (с позиции “длина буфера”-“смещение”), чтобы отобразить нужную подстроку. Если смещение и длина равны нулю, значит следующий символ строки ранее не встречался, он требует отдельной обработки при декодировании.

|  |
| --- |
| // Метод сжатия методом LZ77  std::vector<LZ77::Node> LZ77::compress(const std::string data) {  vector<Node> answer = {};  string buffer = "";  for (int i = 0; i < data.size(); i++) {  if (buffer.find(data[i]) == -1) { //Если символ новый  buffer += data[i];  Node node;  node.length = 0;  node.offset = 0;  node.next = data[i];  answer.push\_back(node);  continue;  }  else {  for (int j = min(buffer.size(), data.size() - i); j > 0; j--) { // для всех подстрок правее длиной от 1 до (сколько осталось пройти)  string stringToFind = "";  for (int k = i; k < j + i; k++) { // создаем эти подстроки  stringToFind += data[k];  }  if (buffer.find(stringToFind) != -1) {  int offset = buffer.size() - buffer.rfind(stringToFind);  int length = stringToFind.size();  Node node;  node.length = length;  node.offset = offset;  node.next = '\0';  answer.push\_back(node);  buffer += stringToFind;  i += stringToFind.size() - 1;  break;  }  }  }  }  return answer;  } |

**Сжатие методом LZ78**

Метод compress класса LZ78 генерирует временный словарь. Изначально словарь dictionary пуст, метод пытается закодировать первый символ. Пока не пройдена вся строка, нужно пытаться увеличить кодируемый префикс, пока такой есть в словаре. Результатом кодирования будет структура “номер в словаре самого длинного найденного префикса” + ”символ, следующий за этим префиксом”. После кодирования префикс с приписанным символом добавляется в словарь и метод продолжает кодирование со следующего символа.

|  |
| --- |
| //Метод сжатия методом LZ78  vector<LZ78::Node> LZ78::compress(const string data) {  vector<string> dictionary = {};  vector<LZ78::Node> answer = {};  string stringToFind = "";  int position = 0;  while (position < data.size()) {  while (find(dictionary.begin(), dictionary.end(), stringToFind) != dictionary.end()) {  stringToFind += data[position];  position++;  }  if (stringToFind != "") {  int positionInDictionary = 0;  char lastSymbol = stringToFind[stringToFind.size() - 1];  string cuttedString = stringToFind;  cuttedString.pop\_back();  if (find(dictionary.begin(), dictionary.end(), cuttedString) != dictionary.end()) {  positionInDictionary = distance(dictionary.begin(), find(dictionary.begin(), dictionary.end(), cuttedString)); // номер слова по итератору  }  Node node;  node.next = lastSymbol;  node.position = positionInDictionary;  answer.push\_back(node);  }  dictionary.push\_back(stringToFind);  stringToFind = "";  }  return answer;  } |

**Сжатие методом Шеннона-Фано**

Метод getCodes разделяет массив вероятностей встречи символов в тексте probabilities оптимальным образом, после чего рекурсивно разделяет левый и правый подмассивы аналогичным образом, причем для левого подмассива к коду символов добавляется “0”, а для правого “1”. Если массив содержит лишь один символ, то ему присваивается рекурсивно полученный код.

|  |
| --- |
| // Метод получения кодов Шеннона-Фано  void ShannonFano::getCodes(vector<Node> probabilities, string code) {  int separateIndex = separate(probabilities);  //cout << separateIndex << "\n";  vector<Node> left = {};  vector<Node> right = {};  for (int i = 0; i < separateIndex; i++) {  left.push\_back(probabilities[i]);  }  for (int i = separateIndex; i < probabilities.size(); i++) {  right.push\_back(probabilities[i]);  }      if (left.size() > 1) getCodes(left, code + "0");  else {  //cout << left[0].symbol << " " << code+"0"<<"\n";  for (int i = 0; i < this->probabilities.size(); i++) {  if (probabilities[i].symbol == left[0].symbol) {  Code node;  node.code = code + "0";  node.symbol = left[0].symbol;  this->codes.push\_back(node);  break;  }  }  }  if (right.size() > 1) getCodes(right, code+"1");  else {  //cout << right[0].symbol << " " << code+"1"<<"\n";  for (int i = 0; i < this->probabilities.size(); i++) {  if (probabilities[i].symbol == right[0].symbol) {  Code node;  node.code = code + "1";  node.symbol = right[0].symbol;  this->codes.push\_back(node);  break;  }  }  }  } |

Метод separate получает на вход вектор вероятностей встречи символов в тексте и возвращает индекс разделения этого массива согласно тому, чтобы суммы вероятностей элементов левее и правее индекса разделения были максимально близки друг к другу.

|  |
| --- |
| // Метод выбора оптимального места разделения массива вероятностей  int ShannonFano::separate(vector<Node> probabilities) {  int position = 1;  float minimalSum = 1.0;  int result=1;  while (position < probabilities.size() - 2) {  float sum1 = 0, sum2 = 0;  for (int i = 0; i < position; i++) {  sum1 += probabilities[i].probability;  }  for (int i = position; i < probabilities.size(); i++) {  sum2 += probabilities[i].probability;  }  if (abs(sum2 - sum1) < minimalSum) {  minimalSum = abs(sum2 - sum1);  result = position;  }  position++;  }  return result;  } |

Конструктор класса ShannonFano получает на вход строку data, кодирует ее методом Шеннона-Фано и записывает результат кодирования в свойство compressedData.

|  |
| --- |
| // Метод конструктора класса ShannonFano  ShannonFano::ShannonFano(const string data) {  this->data = data;  this->probabilities= getProbabilities(data);  getCodes(probabilities,"");  for (int i = 0; i < data.size(); i++) {  string code;  for (int j = 0; j < codes.size(); j++) {  if (codes[j].symbol == data[i]) {  code = codes[j].code;  break;  }  }  compressedData += code;  }  } |

Метод getProbabilities проходит входную строку data и считает вероятность встречи каждого уникального символа в тексте и записывает эти вероятности в вектор probabilities.

|  |
| --- |
| // Метод получения вероятностей встречи символов алфавите в тексте  vector<ShannonFano::Node> ShannonFano::getProbabilities(const string data) {  for (int i = 0; i < data.size(); i++) {  long counter = count(data.begin(), data.end(), data[i]);  int k = 0;  for (int j = 0; j < probabilities.size(); j++) {  if (probabilities[j].symbol == data[i]) break;  else k++;  }  if (k == probabilities.size()) {  Node node;  node.symbol = data[i];  node.probability = counter \* 1.0 / data.size();  probabilities.push\_back(node);  }  }  sort(probabilities.begin(), probabilities.end(), [](Node a, Node b) {return a.probability > b.probability; });  return this->probabilities;  } |

Метод decompress декодирует строку, закодированную методом Шеннона-Фано. В строку buffer добавляются символы из сжатой строки, и если полученная в буфере строка является кодом какого-либо символа, то буфер обнуляется, а этот код заменяется символом, соответствующим данному коду.

|  |
| --- |
| // Метод декодирования кодов Шеннона-Фано  void ShannonFano::decompress() {  string buffer = "";  for (int i = 0; i < compressedData.size(); i++) {  buffer += compressedData[i];  for (int j = 0; j < codes.size(); j++) {  if (buffer == codes[j].code) {  cout << codes[j].symbol;  buffer = "";  break;  }  }  }  } |

**Сжатие методом Хаффмана**

Конструктор класса Haffman принимает на вход строку data, записывает в структуру map количество вхождений каждого уникального символа, далее записывает в приоритетную очередь все найденные символы и их число вхождений в текст, далее составляет дерево оптимальных кодов для каждого символа и (по полученному дереву кодов) кодирует изначальный текст.

|  |
| --- |
| //Конструктор класса Haffman  Haffman::Haffman(string data)  {  unordered\_map<char, int> amount;  for (int i = 0; i < data.size(); i++) {  amount[data[i]]++;  }  auto comparator = [](Node\* a, Node\* b)->bool {return a->amount > b->amount; };  priority\_queue < Node\*, vector<Node\*>, decltype(comparator)> probabilities(comparator);      for (auto pair : amount) {  Node\* node = new Node;  node->symbol = pair.first;  node->amount = pair.second;  node->left = nullptr;  node->right = nullptr;  probabilities.push(node);  }    while (probabilities.size() != 1)  {  Node\* left = probabilities.top();  probabilities.pop();  Node\* right = probabilities.top();  probabilities.pop();  int sum = left->amount + right->amount;  Node\* node = new Node;  node->symbol = '\0';  node->amount = sum;  node->left = left;  node->right = right;  probabilities.push(node);  }  this->root = probabilities.top();  compress(root, "");  for (int i = 0; i < data.size(); i++) // получить закодированную строку  compressedData += haffmanCode[data[i]];  } |

Метод compress класса Haffman обходит дерево кодов и на листьях этого дерева записывает уникальные символы, встречающиеся в строке (присваивает каждому символу код Хаффмана).

|  |
| --- |
| // Метод создания кодов Хаффмана  void Haffman::compress(Node\* root, string code) {  if (root == nullptr) {  return;  }  if (root->left == nullptr && root->right == nullptr) {  haffmanCode[root->symbol] = code;  }  compress(root->left, code + "0");  compress(root->right, code + "1");  } |

Метод decode обходит дерево кодов Хаффмана и декодирует строку compressedData.

|  |
| --- |
| // Метод декодирования кодов Хаффмана  void Haffman::decode(Node\* root, int& index) {  if (root == nullptr) {  return;  }  if (root->left==nullptr && root->right==nullptr) {  cout << root->symbol;  return;  }  index++;  if (compressedData[index] == '0') {  decode(root->left, index);  }  else {  decode(root->right, index);  }  } |

Метод decompress циклично вызывает метод decode.

|  |
| --- |
| // Метод вызова метода декодирования кодов Хаффмана  void Haffman::decompress() {  int index = -1;  //cout << "\n" << compressedData.size() - 2;  while (index < (int)compressedData.size() - 2) { // декодировать строку  decode(root, index);  }  } |

**Описание работы пользовательского интерфейса**

Сначала выводится выбор задания, пользователь вводом с клавиатуры выбирает задание. После чего консоль очищается и отображается ввод в зависимости от выбранного задания. Введенные данные обрабатываются, результат обработки выводится на экран, программа автоматически перезапускается.

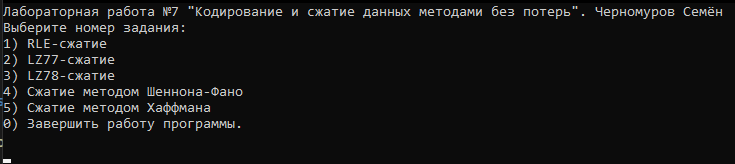


Рисунок 1. Интерфейс программы

# **Тестирование**

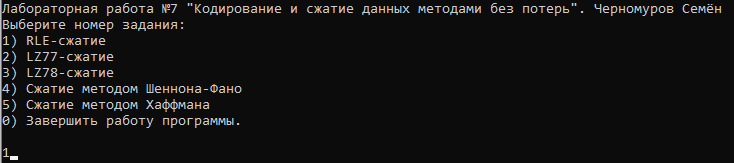


Рисунок 2. Выбор задания 1

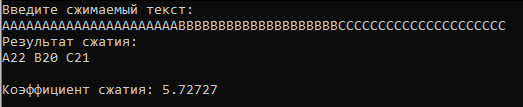


Рисунок 3. Результат работы программы для длинной серии повторяющихся символов (задание 1)

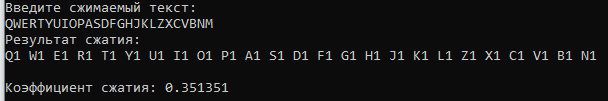


Рисунок 4. Результат работы программы для длинной серии неповторяющихся символов (задание 1)

Программа считывает строку, введенную с клавиатуры, после чего сжимает ее методом RLE. Для строки, содержащей длинные серии повторяющихся символов (AAAAAAAAAAAAAAAAAAAAAABBBBBBBBBBBBBBBBBBBBCCCCCCCCCCCCCCCCCCCCC) коэффициент сжатия составил 5.727 (сжатая строка “A22 B20 C21”), а для длинной серии неповторяющихся символов (QWERTYUIOPASDFGHJKLZXCVBNM) коэффициент сжатия составил 0.351 (сжатая строка “Q1 W1 E1 R1 T1 Y1 U1 I1 O1 P1 A1 S1 D1 F1 G1 H1 J1 K1 L1 Z1 X1 C1 V1 B1 N1”. Полученные коэффициенты сжатия говорят о том, что метод сжатия RLE эффективен для сжатия строк с большим количеством длинных подстрок, состоящих из одного символа. В обратном случае коэффициент сжатия уменьшается вплоть до того, что размер сжатой строки оказывается больше размера изначальной строки.

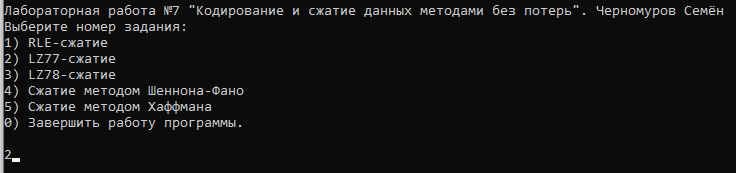


Рисунок 5. Выбор задания 2

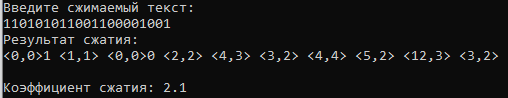


Рисунок 6. Результат выполнения программы (задание 2)

Программа считывает строку с клавиатуры, после чего выполняет сжатие по методу LZ77, и возвращает полученную последовательность кодов. Для строки “110101011001100001001” коэффициент сжатия составляет 2.1 (сжатая строка “<0,0>1 <1,1> <0,0>0 <2,2> <4,3> <3,2> <4,4> <5,2> <12,3> <3,2>”). Декодирование получившегося кода представляет собой следующий процесс:

1. Смещение и длина равны нулю, символ “1”, значит символ встречается впервые, записываем его в декодированную строку. Итоговая строка: “1”.
2. Смещение и длина равны единице, символа нет, значит возвращаемся на один символ влево, копируем 1 символ и записываем его в декодированную строку. Итоговая строка: “11”.
3. Смещение и длина равны нулю, символ “0”, значит символ встречается впервые, записываем его в декодированную строку. Итоговая строка: “110”.
4. Смещение и длина равны двойке, символа нет, значит возвращаемся на 2 символа влево, копируем 2 символа и записываем их в декодированную строку. Итоговая строка: “11010”.
5. Смещение равно четырем, длина трем, символа нет, значит возвращаемся на 4 символа влево, копируем 3 символа и записываем их в декодированную строку. Итоговая строка: “11010101”.
6. Смещение равно трем, длина двум, символа нет, значит возвращаемся на 3 символа влево, копируем 2 символа и записываем их в декодированную строку. Итоговая строка: “1101010110”.
7. Смещение и длина равны четырем, символа нет, значит возвращаемся на 4 символа влево, копируем 4 символа и записываем их в декодированную строку. Итоговая строка: “11010101100110”.
8. Смещение равно пяти, длина двум, символа нет, значит возвращаемся на 5 символов влево, копируем 2 символа и записываем их в декодированную строку. Итоговая строка: “1101010110011000”.
9. Смещение равно двенадцати, длина трем, символа нет, значит возвращаемся на 12 символов влево, копируем 3 символа и записываем их в декодированную строку. Итоговая строка: “1101010110011000010”.
10. Смещение равно трем, длина двум, символа нет, значит возвращаемся на 3 символа влево, копируем 2 символа и записываем их в декодированную строку. Итоговая строка: “110101011001100001001”.

Изначальная строка: “110101011001100001001”.

Декодированная строка: “110101011001100001001”.

Изначальная и декодированная строки совпадают, что говорит о корректном декодировании.

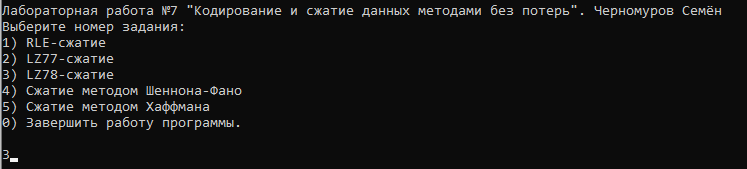


Рисунок 7. Выбор задания 3

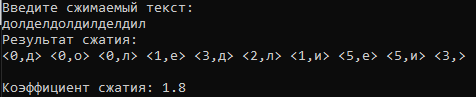


Рисунок 8. Результат работы программы (задание 3)

Программа считывает строку с клавиатуры, производит сжатие методом LZ78 и выводит результат сжатия на экран. Для строки “долделдолдилделдил” коэффициент сжатия составил 1.8 (сжатая строка “<0,д> <0,о> <0,л> <1,е> <3,д> <2,л> <1,и> <5,е> <5,и> <3,>”). Декодирование получившегося кода представляет собой следующий процесс:

1. Номер в словаре 0, символ “д”, значит просто добавляем “д” в словарь и в декодированную строку. Итоговый словарь: “д”. Итоговая строка: “д”.
2. Номер в словаре 0, символ “о”, значит просто добавляем “о” в словарь и в декодированную строку. Итоговый словарь: “д о”. Итоговая строка: “до”.
3. Номер в словаре 0, символ “л”, значит просто добавляем “л” в словарь и в декодированную строку. Итоговый словарь: “д о л”. Итоговая строка: “дол”.
4. Номер в словаре 1, символ “е”, значит берем первый элемент из словаря (“д”), добавляем к нему “е”, дописываем “де” в словарь и в декодированную строку. Итоговый словарь: “д о л де”. Итоговая строка: “долде”.
5. Номер в словаре 3, символ “д”, значит берем третий элемент из словаря (“л”), добавляем к нему “д”, дописываем “лд” в словарь и в декодированную строку. Итоговый словарь: “д о л де лд”. Итоговая строка: “долделд”.
6. Номер в словаре 2, символ “л”, значит берем второй элемент из словаря (“о”), добавляем к нему “л”, дописываем “ол” в словарь и в декодированную строку. Итоговый словарь: “д о л де лд ол”. Итоговая строка: “долделдол”.
7. Номер в словаре 1, символ “и”, значит берем первый элемент из словаря (“д”), добавляем к нему “и”, дописываем “ди” в словарь и в декодированную строку. Итоговый словарь: “д о л де лд ол ди”. Итоговая строка: “долделдолди”.
8. Номер в словаре 5, символ “е”, значит берем пятый элемент из словаря (“лд”), добавляем к нему “е”, дописываем “лде” в словарь и в декодированную строку. Итоговый словарь: “д о л де лд ол ди лде”. Итоговая строка: “долделдолдилде”.
9. Номер в словаре 5, символ “и”, значит берем пятый элемент из словаря (“лд”), добавляем к нему “и”, дописываем “лди” в словарь и в декодированную строку. Итоговый словарь: “д о л де лд ол ди лде лди”. Итоговая строка: “долделдолдилделди”.
10. Номер в словаре 3, символ отсутствует, значит берем третий элемент из словаря (“л”), дописываем “л” в декодированную строку. Итоговый словарь: “д о л де лд ол ди лде лди”. Итоговая строка: “долделдолдилделдил”.

Изначальная строка: “ долделдолдилделдил”.

Декодированная строка: “ долделдолдилделдил”.

Изначальная и декодированная строки совпадают, что говорит о корректном декодировании.

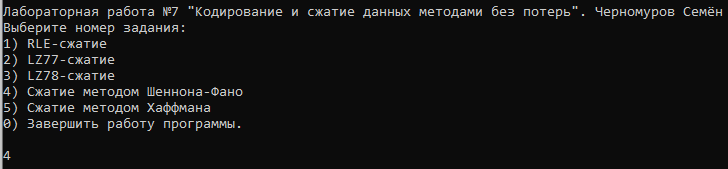


Рисунок 9. Выбор задания 4

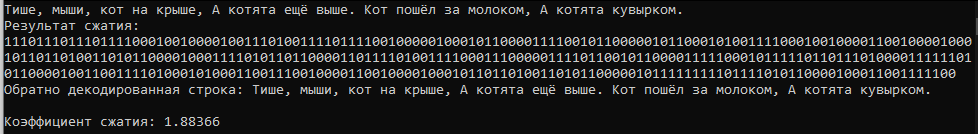


Рисунок 10. Результат работы программы (задание 4)

Программа получает на вход строку, введенную с клавиатуры, сжимает ее методом Шеннона-Фано, обратно декодирует полученную комбинацию нулей и единиц, после чего расчитывает коэффициент сжатия (для строки “Тише, мыши, кот на крыше, А котята ещё выше. Кот пошёл за молоком, А котята кувырком.” Коэффициент сжатия составляет 1.883, а сжатая строка “1110111011101111000100100001001110100111101111001000001000101100001111001011000001011000101001111000100100001100100001000101101101001101011000010001111010110110000110111101001111000111000001111011001011000011111000101111101101110100001111110101100001001100111101000101000110011100100001100100001000101101101001101011000001011111111101111010110000100011001111100”).

Составим таблицу формирования кодов:

Таблица 1. Таблица формирования кодов Шеннона-Фано

|  |  |  |  |
| --- | --- | --- | --- |
| Массив | Левый подмассив | Правый подмассив | Текущий код |
| “ ” о к т ш е , м ы а и р А я ё в . л Т н щ К п з у | “ ” о к т ш | е , м ы а и р А я ё в . л Т н щ К п з у | - |
| “ ” о к т ш | “ ” о | к т ш | 0 |
| “ ” о | “ ” | о | 00 |
| “ ” (Пробел) |  |  | 000 |
| о |  |  | 001 |
| к т ш | к | т ш | 01 |
| к |  |  | 010 |
| т ш | т | ш | 011 |
| т |  |  | 0110 |
| ш |  |  | 0111 |
| е , м ы а и р А я ё в . л Т н щ К п з у | е , м ы а и | р А я ё в . л Т н щ К п з у | 1 |
| е , м ы а и | е , м | ы а и | 10 |
| е , м | е | , м | 100 |
| е |  |  | 1000 |
| , м | , | м | 1001 |
| , |  |  | 10010 |
| м |  |  | 10011 |
| ы а и | ы | а и | 101 |
| ы |  |  | 1010 |
| а и | а | и | 1011 |
| а |  |  | 10110 |
| и |  |  | 10111 |
| р А я ё в . л Т н щ К п з у | р А я ё в | . л Т н щ К п з у | 11 |
| р А я ё в | р А | я ё в | 110 |
| р А | р | А | 1100 |
| р |  |  | 11000 |
| А |  |  | 11001 |
| я ё в | я | ё в | 1101 |
| я |  |  | 11010 |
| ё в | ё | в | 11011 |
| ё |  |  | 110110 |
| в |  |  | 110111 |
| . л Т н щ К п з у | . л Т | н щ К п з у | 111 |
| . л Т | . | л Т | 1110 |
| . |  |  | 11100 |
| л Т | л | т | 11101 |
| л |  |  | 111010 |
| Т |  |  | 111011 |
| н щ К п з у | н щ К | п з у | 1111 |
| н щ К | н | щ К | 11110 |
| н |  |  | 111100 |
| щ К | щ | К | 111101 |
| щ |  |  | 1111010 |
| К |  |  | 1111011 |
| п з у | п | з у | 11111 |
| п |  |  | 111110 |
| з у | з | у | 111111 |
| з |  |  | 1111110 |
| у |  |  | 1111111 |

В данной таблице желтым цветом обозначены финальные коды для каждого из символов.

Составим префиксное дерево:

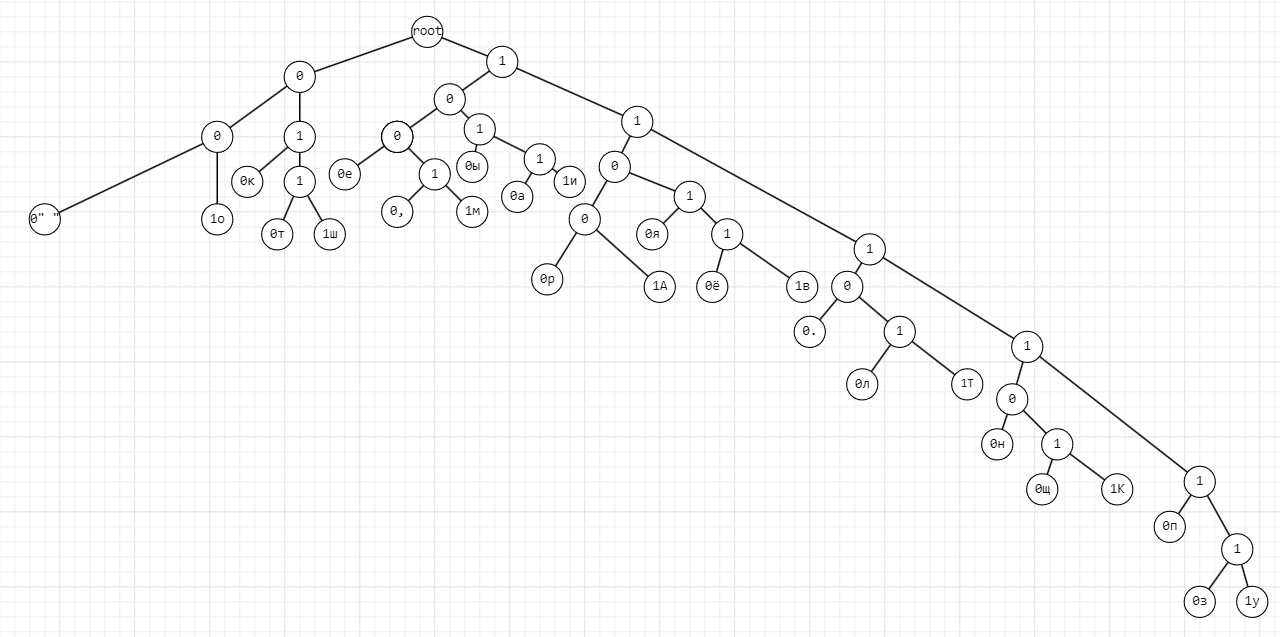


Рисунок 11. Префиксное дерево для алгоритма Шеннона-Фано

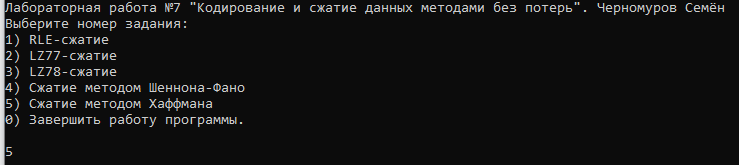


Рисунок 12. Выбор задания 5

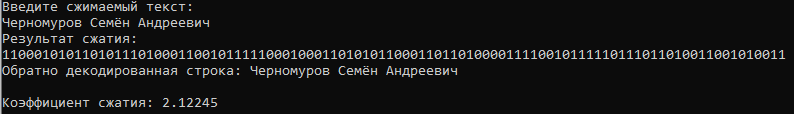


Рисунок 13. Результат работы программы (задание 5)

Программа получает на вход строку, введенную с клавиатуры, сжимает ее методом Хаффмана, обратно декодирует полученную комбинацию нулей и единиц, после чего расчитывает коэффициент сжатия (для строки “Черномуров Семён Андреевич” коэффициент сжатия составляет 2.122, а результатом сжатия полученной строки является: “11000101011010111010001100101111100010001101010110001101101000011110010111110111011010011001010011”).

Построим таблицу частот встречаемости символов, вероятности их появления в тексте, а также их кодов:

Таблица 2. Таблица встречаемости символов в тексте

|  |  |  |  |
| --- | --- | --- | --- |
| Символ | Число вхождений в текст | Вероятность появления в тексте | Код Хаффмана для символа |
| Ч | 1 | 0.038 | 11000 |
| н | 3 | 0.115 | 010 |
| е | 4 | 0.1153 | 101 |
| р | 3 | 0.115 | 011 |
| о | 2 | 0.076 | 1110 |
| м | 2 | 0.076 | 1000 |
| у | 1 | 0.038 | 11001 |
| в | 2 | 0.076 | 001 |
| Пробел | 2 | 0.076 | 000 |
| С | 1 | 0.038 | 11010 |
| ё | 1 | 0.038 | 11011 |
| А | 1 | 0.038 | 11110 |
| д | 1 | 0.038 | 11111 |
| и | 1 | 0.038 | 10010 |
| ч | 1 | 0.038 | 10011 |

Построим префиксное дерево Хаффмана:

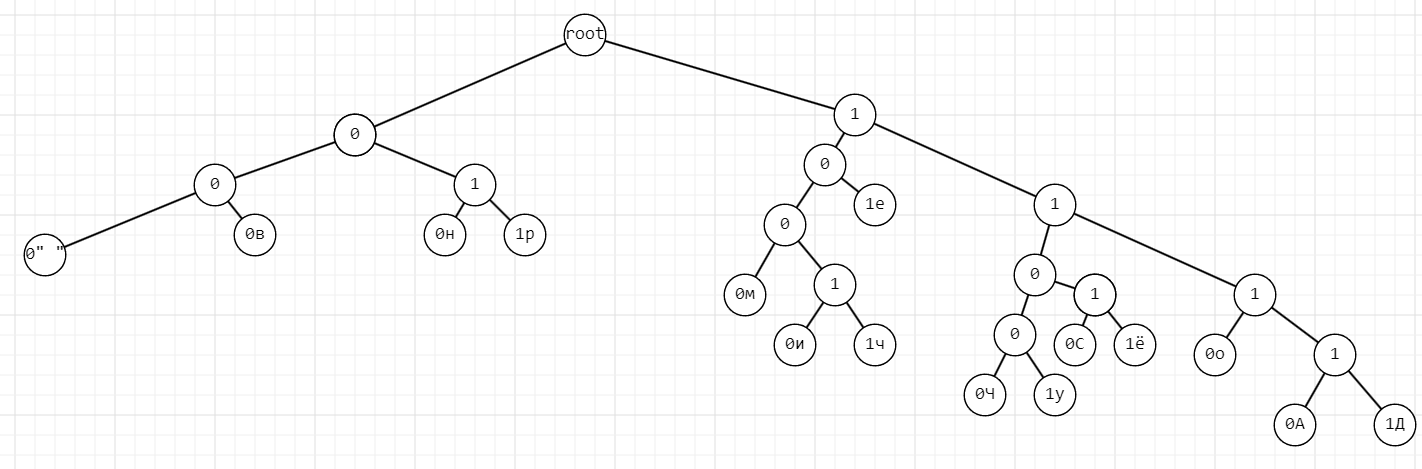


Рисунок 14. Префиксное дерево кодов Хаффмана

Расчитаем среднюю длину кода. Для этого сложим длины всех кодов и разделим полученное значение на количество кодовых слов. Средняя длина кода: 4.2 бит/символ.

Расчитаем дисперсию. Для этого сосчитаем сумму квадратов отклонений от средней длины каждого кода и разделим на количество кодов. Дисперсия равна: 0.826.

Применим сжатие методом Хаффмана для отрывка из книги «Разумный инвестор» и сравним результат с сжатием с помощью архиватора WinRAR:

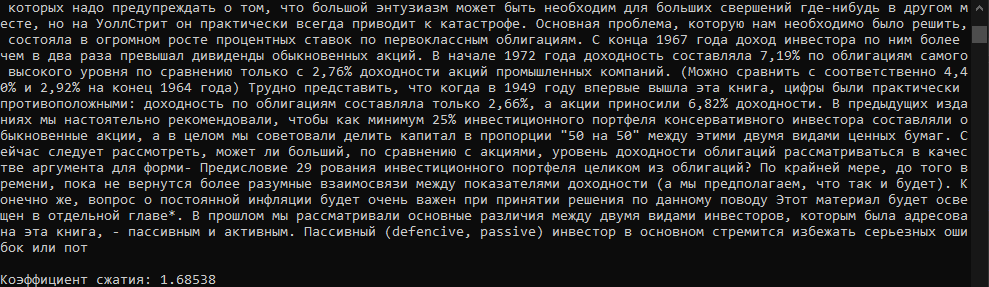


Рисунок 15. Результат работы программы для большого текста (задание 5)

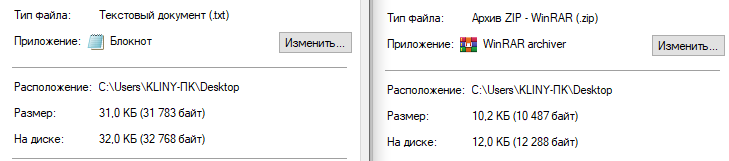


Рисунок 16. Сжатие фрагмента с использованием WinRAR

Коэффициент сжатия с помощью метода Хаффмана составил 1.685, а с помощью архиватора WinRAR 3.031.

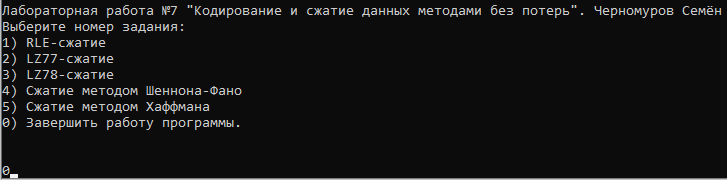


Рисунок 16. Выбор завершения работы программы

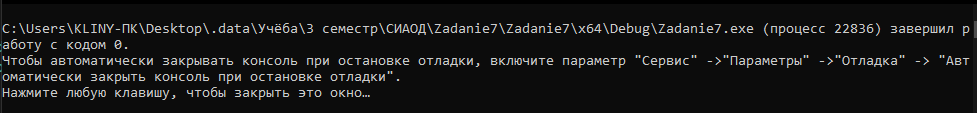


Рисунок 17. Завершение работы программы

При выборе функции завершения работы программы программа завершает свою работу.

Из результатов выполнения программы видно, что программа работает корректно.

# **Вывод**

В результате выполнения работы я:

1. Получил практические навыки и знания по выполнению сжатия данных методами RLE, LZ77, LZ78, Шеннона-Фано, Хаффмана.
2. Закрепил знания по тестированию корректности работы программы.
3. Закрепил навыки создания пользовательских интерфейсов.

# **Исходный код программы**

**Файл main.cpp (основной алгоритм программы)**

|  |
| --- |
| #include "rle.h"  #include "lz77.h"  #include "lz78.h"  #include "shannon-fano.h"  #include "haffman.h"  #include <iostream>  #include <windows.h>  using namespace std;  int main() {  SetConsoleCP(1251);  SetConsoleOutputCP(1251);  setlocale(LC\_ALL, "");  cout << "Лабораторная работа №7 \"Кодирование и сжатие данных методами без потерь\". Черномуров Семён\n";  cout << "Выберите номер задания:\n" <<  "1) RLE-сжатие\n" <<  "2) LZ77-сжатие\n" <<  "3) LZ78-сжатие\n" <<  "4) Сжатие методом Шеннона-Фано\n" <<  "5) Сжатие методом Хаффмана\n" <<  "0) Завершить работу программы.\n\n";  int choice1;  do {  cin >> choice1;  if (choice1 != 1 && choice1 != 2 && choice1 != 3 && choice1 != 4 && choice1 != 5 && choice1 != 0) cout << "Введено неверное значение, попробуйте снова.\n";  } while (choice1 != 1 && choice1 != 2 && choice1 != 3 && choice1 != 4 && choice1 != 5 && choice1 != 0);  system("cls");  switch (choice1)  {  case 1: {  cout << "Введите сжимаемый текст:\n";  string data;  cin.ignore(32767, '\n');  getline(cin, data);  string compressedData = RLE::compress(data);  cout << "Результат сжатия:\n" << compressedData;  cout << "\n\n" << "Коэффициент сжатия: " << data.size() \* 1.0 / compressedData.size();  break;  }  case 2: {  cout << "Введите сжимаемый текст:\n";  string data;  cin.ignore(32767, '\n');  getline(cin, data);  vector<LZ77::Node> compressedData = LZ77::compress(data);  cout << "Результат сжатия:\n";  for (int i = 0; i < compressedData.size(); i++) {  cout << "<" << compressedData[i].offset << "," << compressedData[i].length << ">" << compressedData[i].next<<" ";  }  cout << "\n\n" << "Коэффициент сжатия: " << data.size() \* 1.0 / compressedData.size();  break;  }  case 3: {  cout << "Введите сжимаемый текст:\n";  string data;  cin.ignore(32767, '\n');  getline(cin, data);  vector<LZ78::Node> compressedData = LZ78::compress(data);  cout << "Результат сжатия:\n";  for (int i = 0; i < compressedData.size(); i++) {  cout << "<" << compressedData[i].position << "," << compressedData[i].next << "> ";  }  cout << "\n\n" << "Коэффициент сжатия: " << data.size() \* 1.0 / compressedData.size();  break;  }  case 4: {  cout << "Введите сжимаемый текст:\n";  string data;  cin.ignore(32767, '\n');  getline(cin, data);  ShannonFano obj(data);  cout << "Результат сжатия:\n";  cout << obj.compressedData<<"\n";  cout << "Обратно декодированная строка: ";  obj.decompress();  cout << "\n\n" << "Коэффициент сжатия: " << data.size() \* 8.0 / obj.compressedData.size();  break;  }  case 5: {  cout << "Введите сжимаемый текст:\n";  string data;  cin.ignore(32767, '\n');  getline(cin, data);  Haffman obj(data);  cout << "Результат сжатия:\n";  cout << obj.compressedData << "\n";  cout << "Обратно декодированная строка: ";  obj.decompress();  cout << "\n\n" << "Коэффициент сжатия: " << data.size() \* 8.0 / obj.compressedData.size();  break;  }  case 0:  return 0;  }  cout << "\n\n";  main();  } |

**Файл haffman.h (заголовочный файл класса Haffman)**

|  |
| --- |
| #pragma once  #ifndef HAFFMAN\_H  #define HAFFMAN\_H  #include <string>  #include <queue>  #include <unordered\_map>  #include <functional>  class Haffman {  private:    struct Node  {  char symbol;  int amount;  Node\* left;  Node\* right;  };  Node\* root;  std::unordered\_map<char, std::string> haffmanCode;  void compress(Node\* root, std::string code);  void decode(Node\* root, int& index);    public:  Haffman(std::string data);  void decompress();  std::string compressedData = "";  };  #endif |

**Файл haffman.cpp (реализация методов класса Haffman)**

|  |
| --- |
| #include "haffman.h"  #include <windows.h>  #include <iostream>  using namespace std;  //Конструктор класса Haffman  Haffman::Haffman(string data)  {  unordered\_map<char, int> amount;  for (int i = 0; i < data.size(); i++) {  amount[data[i]]++;  }  auto comparator = [](Node\* a, Node\* b)->bool {return a->amount > b->amount; };  priority\_queue < Node\*, vector<Node\*>, decltype(comparator)> probabilities(comparator);      for (auto pair : amount) {  Node\* node = new Node;  node->symbol = pair.first;  node->amount = pair.second;  node->left = nullptr;  node->right = nullptr;  probabilities.push(node);  }    while (probabilities.size() != 1)  {  Node\* left = probabilities.top();  probabilities.pop();  Node\* right = probabilities.top();  probabilities.pop();  int sum = left->amount + right->amount;  Node\* node = new Node;  node->symbol = '\0';  node->amount = sum;  node->left = left;  node->right = right;  probabilities.push(node);  }  this->root = probabilities.top();  compress(root, "");  for (int i = 0; i < data.size(); i++) // получить закодированную строку  compressedData += haffmanCode[data[i]];  }  // Метод создания кодов Хаффмана  void Haffman::compress(Node\* root, string code) {  if (root == nullptr) {  return;  }  if (root->left == nullptr && root->right == nullptr) {  haffmanCode[root->symbol] = code;  }  compress(root->left, code + "0");  compress(root->right, code + "1");  }  // Метод декодирования кодов Хаффмана  void Haffman::decode(Node\* root, int& index) {  if (root == nullptr) {  return;  }  if (root->left==nullptr && root->right==nullptr) {  cout << root->symbol;  return;  }  index++;  if (compressedData[index] == '0') {  decode(root->left, index);  }  else {  decode(root->right, index);  }  }  // Метод вызова метода декодирования кодов Хаффмана  void Haffman::decompress() {  int index = -1;  //cout << "\n" << compressedData.size() - 2;  while (index < (int)compressedData.size() - 2) { // декодировать строку  decode(root, index);  }  }  //int main()  //{  // SetConsoleCP(1251);  // SetConsoleOutputCP(1251);  // setlocale(0, "");  //  // Haffman obj(data);  // cout << obj.compressedData;  // obj.decompress();  // return 0;  //} |

**Файл lz77.h (заголовочный файл класса LZ77)**

|  |
| --- |
| #pragma once  #ifndef LZ77\_H  #define LZ77\_H  #include <vector>  #include <string>  class LZ77  {  public:  struct Node {  int offset;  int length;  char next;  };  static std::vector<Node> compress(const std::string data);  };  #endif // !LZ77\_H |

**Файл lz77.cpp (реализация методов класса LZ77)**

|  |
| --- |
| #include "lz77.h"  #include <iostream>  using namespace std;  // Метод сжатия методом LZ77  std::vector<LZ77::Node> LZ77::compress(const std::string data) {  vector<Node> answer = {};  string buffer = "";  for (int i = 0; i < data.size(); i++) {  if (buffer.find(data[i]) == -1) { //Если символ новый  buffer += data[i];  Node node;  node.length = 0;  node.offset = 0;  node.next = data[i];  answer.push\_back(node);  continue;  }  else {  for (int j = min(buffer.size(), data.size() - i); j > 0; j--) { // для всех подстрок правее длиной от 1 до (сколько осталось пройти)  string stringToFind = "";  for (int k = i; k < j + i; k++) { // создаем эти подстроки  stringToFind += data[k];  }  if (buffer.find(stringToFind) != -1) {  int offset = buffer.size() - buffer.rfind(stringToFind);  int length = stringToFind.size();  Node node;  node.length = length;  node.offset = offset;  node.next = '\0';  answer.push\_back(node);  buffer += stringToFind;  i += stringToFind.size() - 1;  break;  }  }  }  }  return answer;  } |

**Файл lz78.h (заголовочный файл класса LZ78)**

|  |
| --- |
| #pragma once  #ifndef LZ78\_H  #define LZ78\_H  #include <vector>  #include <string>  class LZ78  {  public:  struct Node {  int position;  char next;  };  static std::vector<Node> compress(const std::string data);  };  #endif // !LZ78\_H |

**Файл lz78.cpp (реализация методов класса LZ78)**

|  |
| --- |
| #include "lz78.h"  #include <algorithm>  #include <iostream>  using namespace std;  //Метод сжатия методом LZ78  vector<LZ78::Node> LZ78::compress(const string data) {  vector<string> dictionary = {};  vector<LZ78::Node> answer = {};  string stringToFind = "";  int position = 0;  while (position < data.size()) {  while (find(dictionary.begin(), dictionary.end(), stringToFind) != dictionary.end()) {  stringToFind += data[position];  position++;  }  if (stringToFind != "") {  int positionInDictionary = 0;  char lastSymbol = stringToFind[stringToFind.size() - 1];  string cuttedString = stringToFind;  cuttedString.pop\_back();  if (find(dictionary.begin(), dictionary.end(), cuttedString) != dictionary.end()) {  positionInDictionary = distance(dictionary.begin(), find(dictionary.begin(), dictionary.end(), cuttedString)); // номер слова по итератору  }  Node node;  node.next = lastSymbol;  node.position = positionInDictionary;  answer.push\_back(node);  }  dictionary.push\_back(stringToFind);  stringToFind = "";  }  return answer;  } |

**Файл rle.h (заголовочный файл класса RLE)**

|  |
| --- |
| #pragma once  #ifndef RLE\_H  #define RLE\_H  #include<string>  class RLE {  public:  static std::string compress(std::string data);  };  #endif |

**Файл rle.cpp (реализация методов класса RLE)**

|  |
| --- |
| #include "rle.h"  #include <iostream>  using namespace std;  // Метод сжатия методом RLE  string RLE::compress(string data) {  if (data == "") return "";  string compressedData = "", buffer = "";  buffer += data[0];  for (int i = 1; i < data.size(); i++) {  if (buffer[0] == data[i] || buffer == "") buffer += data[i];  if (buffer[0] != data[i] || i == data.size()-1) {  compressedData += buffer[0];  compressedData += to\_string(buffer.size());  compressedData += " ";  buffer = data[i];  }  }  if (buffer != "") {  compressedData += buffer[0];  compressedData += to\_string(buffer.size());  }  return compressedData.erase(compressedData.size()-3,3);  } |

**Файл shannon-fano.h (заголовочный файл класса ShannonFano)**

|  |
| --- |
| #pragma once  #ifndef SHANNON\_H  #define SHANNON\_H  #include <string>  #include <vector>  class ShannonFano {  private:  struct Node {  char symbol;  float probability;  };  struct Code {  char symbol;  std::string code="";  };  std::vector<Node> probabilities;  std::vector<Code> codes;  std::string data;  std::vector<Node> getProbabilities(const std::string data);  int separate(std::vector<Node>);  void getCodes(std::vector<Node> probabilities, std::string code);  public:    std::string compressedData="";  ShannonFano(const std::string data);  void decompress();  };  #endif // !SHANNON\_H |

**Файл shannon-fano.cpp (реализация методов класса ShannonFano)**

|  |
| --- |
| #include "shannon-fano.h"  #include <algorithm>  #include <iostream>  #include <windows.h>  using namespace std;  // Метод получения кодов Шеннона-Фано  void ShannonFano::getCodes(vector<Node> probabilities, string code) {  int separateIndex = separate(probabilities);  //cout << separateIndex << "\n";  vector<Node> left = {};  vector<Node> right = {};  for (int i = 0; i < separateIndex; i++) {  left.push\_back(probabilities[i]);  }  for (int i = separateIndex; i < probabilities.size(); i++) {  right.push\_back(probabilities[i]);  }      if (left.size() > 1) getCodes(left, code + "0");  else {  //cout << left[0].symbol << " " << code+"0"<<"\n";  for (int i = 0; i < this->probabilities.size(); i++) {  if (probabilities[i].symbol == left[0].symbol) {  Code node;  node.code = code + "0";  node.symbol = left[0].symbol;  this->codes.push\_back(node);  break;  }  }  }  if (right.size() > 1) getCodes(right, code+"1");  else {  //cout << right[0].symbol << " " << code+"1"<<"\n";  for (int i = 0; i < this->probabilities.size(); i++) {  if (probabilities[i].symbol == right[0].symbol) {  Code node;  node.code = code + "1";  node.symbol = right[0].symbol;  this->codes.push\_back(node);  break;  }  }  }  }  // Метод выбора оптимального места разделения массива вероятностей  int ShannonFano::separate(vector<Node> probabilities) {  int position = 1;  float minimalSum = 1.0;  int result=1;  while (position < probabilities.size() - 2) {  float sum1 = 0, sum2 = 0;  for (int i = 0; i < position; i++) {  sum1 += probabilities[i].probability;  }  for (int i = position; i < probabilities.size(); i++) {  sum2 += probabilities[i].probability;  }  if (abs(sum2 - sum1) < minimalSum) {  minimalSum = abs(sum2 - sum1);  result = position;  }  position++;  }  return result;  }  // Метод конструктора класса ShannonFano  ShannonFano::ShannonFano(const string data) {  this->data = data;  this->probabilities= getProbabilities(data);  getCodes(probabilities,"");  for (int i = 0; i < data.size(); i++) {  string code;  for (int j = 0; j < codes.size(); j++) {  if (codes[j].symbol == data[i]) {  code = codes[j].code;  break;  }  }  compressedData += code;  }  }  // Метод получения вероятностей встречи символов алфавите в тексте  vector<ShannonFano::Node> ShannonFano::getProbabilities(const string data) {  for (int i = 0; i < data.size(); i++) {  long counter = count(data.begin(), data.end(), data[i]);  int k = 0;  for (int j = 0; j < probabilities.size(); j++) {  if (probabilities[j].symbol == data[i]) break;  else k++;  }  if (k == probabilities.size()) {  Node node;  node.symbol = data[i];  node.probability = counter \* 1.0 / data.size();  probabilities.push\_back(node);  }  }  sort(probabilities.begin(), probabilities.end(), [](Node a, Node b) {return a.probability > b.probability; });  return this->probabilities;  }  // Метод декодирования кодов Шеннона-Фано  void ShannonFano::decompress() {  string buffer = "";  for (int i = 0; i < compressedData.size(); i++) {  buffer += compressedData[i];  for (int j = 0; j < codes.size(); j++) {  if (buffer == codes[j].code) {  cout << codes[j].symbol;  buffer = "";  break;  }  }  }  }  //int main() {  // SetConsoleCP(1251);  // SetConsoleOutputCP(1251);  // setlocale(0, "");  // //const string data = "ABCABCDDEHFGY";  // ShannonFano obj(data);  // cout << obj.compressedData<<"\n";  // obj.decompress();  //} |