|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | 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 | Исаев В.В. |
| 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

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.

Таблица 1. Вариант задания

|  |  |  |  |
| --- | --- | --- | --- |
| Вариант | Сжатие данных по методу Лемпеля-Зива LZ77 | Закодировать следующую фразу, используя код LZ78 | Закодировать фразу методами Шеннона-Фано |
| 8 | 010110110110100010001 | sarsalsarsanlasanl | Мой котёнок очень странный, Он не хочет есть сметану, К молоку не прикасался  И от рыбки отказался. |

# Решение

Сжатие данных— алгоритмическое преобразование данных, производимое с целью уменьшения занимаемого ими объёма. Применяется для более рационального использования устройств хранения и передачи данных. Обратная процедура называется восстановлением данных (распаковкой, декомпрессией).

Сжатие основано на устранении избыточности, содержащейся в исходных данных. Простейшим примером избыточности является повторение в тексте фрагментов. Подобная избыточность обычно устраняется заменой повторяющейся последовательности ссылкой на уже закодированный фрагмент с указанием его длины. Другой вид избыточности связан с тем, что некоторые значения в сжимаемых данных встречаются чаще других. Сокращение объёма данных достигается за счёт замены часто встречающихся данных короткими кодовыми словами, а редких — длинными (энтропийное кодирование).

Сжатие без потерь позволяет полностью восстановить исходное сообщение, так как не уменьшает в нем количество информации, несмотря на уменьшение длины. Такая возможность возникает только если распределение вероятностей на множестве сообщений не равномерное, например часть теоретически возможных в прежней кодировке сообщений на практике не встречается.

Кодирование длин серий (англ. run-length encoding, RLE) или кодирование повторов — алгоритм сжатия данных, заменяющий повторяющиеся символы (серии) на один символ и число его повторов. Серией называется последовательность, состоящая из нескольких одинаковых символов. При кодировании строка одинаковых символов, составляющих серию, заменяется строкой, содержащей сам повторяющийся символ и количество его повторов.

LZ77 и LZ78 — алгоритмы сжатия без потерь, опубликованные в статьях израильских математиков Авраама Лемпеля и Яакова Зива в 1977 и 1978 годах. Эти алгоритмы — наиболее известные варианты в семействе LZ\*, которое включает в себя также LZW, LZSS, LZMA и другие алгоритмы.

Оба алгоритма относятся к словарным методам, в отличие от других методов уменьшения избыточности, таких как RLE и арифметическое сжатие. LZ77 является алгоритмом со «скользящим окном», что эквивалентно неявному использованию словарного подхода, впервые предложенного в LZ78.

Принцип скользящего окна

Метод кодирования, согласно принципу скользящего окна, учитывает уже ранее встречавшуюся информацию, то есть информацию, которая уже известна для кодировщика и декодировщика (второе и последующие вхождения некоторой строки символов в сообщении заменяются ссылками на её первое вхождение).

Благодаря этому принципу алгоритмы LZ\* иногда называются методами сжатия с использованием скользящего окна. Скользящее окно можно представить в виде буфера (или более сложной динамической структуры данных), который организован так, чтобы запоминать «сказанную» ранее информацию и предоставлять к ней доступ. Таким образом, сам процесс сжимающего кодирования согласно LZ77 напоминает написание программы, команды которой позволяют обращаться к элементам «скользящего окна», и вместо значений сжимаемой последовательности вставлять ссылки на эти значения в «скользящем окне». Размер скользящего окна может динамически изменяться и составлять 2, 4 или 32 килобайта. Следует также отметить, что размер окна кодировщика может быть меньше или равен размеру окна декодировщика, но не наоборот.

Механизм кодирования совпадений

В стандартном алгоритме LZ77 совпадения кодируются парой:

длина совпадения (match length)

смещение (offset) или дистанция (distance)

В продолжение уже приведенной аналогии с программированием отметим, что в большинстве статей, посвященных алгоритму LZ77, кодируемая пара трактуется именно как команда копирования символов из скользящего окна с определенной позиции, или дословно как: «Вернуться в буфере символов на значение смещения и скопировать значение длины символов, начиная с текущей позиции».

Хотя для приверженцев императивного программирования такая интерпретация может показаться интуитивно понятной, она мало говорит о сущности алгоритма LZ77 как метода сжатия. Особенность данного алгоритма сжатия заключается в том, что использование кодируемой пары длина-смещение является не только приемлемым, но и эффективным в тех случаях, когда значение длины превышает значение смещения.

Пример с командой копирования не совсем очевиден: «Вернуться на 1 символ назад в буфере и скопировать 7 символов, начиная с текущей позиции». Каким образом можно скопировать 7 символов из буфера, когда в настоящий момент в буфере находится только 1 символ? Однако следующая интерпретация кодирующей пары может прояснить ситуацию: каждые 7 последующих символов совпадают с 1 символом перед ними.

Это означает, что каждый символ можно однозначно определить, переместившись назад в буфере — даже если данный символ ещё отсутствует в буфере на момент декодирования текущей пары длина-смещение. Такая кодируемая пара будет представлять собой многократное повторение последовательности символов, что представляет собой более общую форму RLE.

Недостатки

* невозможность кодирования подстрок, отстоящих друг от друга на расстоянии, большем длины словаря
* длина подстроки, которую можно закодировать, ограничена размером буфера
* малая эффективность при кодировании незначительного объёма данных

В отличие от LZ77, работающего с уже полученными данными, LZ78, предложенный в 1978 году, ориентируется на данные, которые только будут получены (LZ78 не использует «скользящее» окно, он хранит словарь из уже просмотренных фраз). 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.

Алгоритм Ше́ннона — Фанó — один из первых алгоритмов сжатия, который впервые сформулировали американские учёные Клод Шеннон и Роберт Фано. Данный метод сжатия имеет большое сходство с алгоритмом Хаффмана, который появился на несколько лет позже и является логическим продолжением алгоритма Шеннона. Алгоритм использует коды переменной длины: часто встречающийся символ кодируется кодом меньшей длины, редко встречающийся — кодом большей длины. Коды Шеннона — Фано — префиксные, то есть никакое кодовое слово не является префиксом любого другого. Это свойство позволяет однозначно декодировать любую последовательность кодовых слов.

Алгоритм Хаффмана — жадный алгоритм оптимального префиксного кодирования алфавита с минимальной избыточностью. Был разработан в 1952 году аспирантом Массачусетского технологического института Дэвидом Хаффманом при написании им курсовой работы. В настоящее время используется во многих программах сжатия данных.

В отличие от алгоритма Шеннона — Фано, алгоритм Хаффмана остаётся всегда оптимальным и для вторичных алфавитов m2 с более чем двумя символами.

Этот метод кодирования состоит из двух основных этапов:

1. Построение оптимального кодового дерева.
2. Построение отображения код-символ на основе построенного дерева.
3. 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.

|  |
| --- |
| string encodeRle(string value)  {  string ans = "";  int counter = 1;  char prev = value[0];  for (int i = 1; i < value.size(); ++i)  {  if (prev == value[i])  counter++;  else {  if (counter > 9)  {  char buffer[33];  \_itoa(counter, buffer, 36);  ans += buffer;  }  else  {  ans += to\_string(counter);  }  ans += prev;  prev = value[i];  counter = 1;  }  }  if (counter)  {  ans += to\_string(counter);  ans += prev;  prev = value[value.size() - 1];  }  return ans;  }  string decodeRle(string value)  {  string ans = "";  for (int i = 0; i < value.size() - 1; i += 2)  for (int j = 0; j < rebase(value[i]); ++j)  ans += value[i + 1];  return ans;  } |

1. 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.

|  |
| --- |
| vector<Sequence77> encodeLZ77(string s)  {  int maxWindow = 10;  vector<Sequence77> ans;  int sizeBuffer, sizeUnchecked;  for (int i = 0; i < s.length(); i++)  {  if (i + 1 <= maxWindow)  sizeBuffer = i;  else  sizeBuffer = maxWindow;  if (i + sizeBuffer < s.length())  sizeUnchecked = sizeBuffer;  else  sizeUnchecked = s.length() - i;  string buffer = s.substr(i - sizeBuffer, sizeBuffer);  string unchecked = s.substr(i, sizeUnchecked);  int off = -1;  while (true)  {  if (!sizeUnchecked)  break;  string str3 = unchecked.substr(0, sizeUnchecked);  off = buffer.find(str3);  if (off != -1)  break;  sizeUnchecked--;  if (sizeUnchecked <= 0)  break;  }  Sequence77 value;  value.next = s[i + sizeUnchecked];  if (off != -1)  {  value.length = sizeUnchecked;  value.offset = sizeBuffer - off;  ans.push\_back(value);  i += sizeUnchecked;  }  else  {  value.length = 0;  value.offset = 0;  ans.push\_back(value);  }  }  return ans;  }  vector<Sequence78> encodeLZ78(string s) {  string buffer = "";  map<string, int> dict = {};  vector<Sequence78> ans;  for (int i = 0; i < s.size(); ++i) {  if (dict.find(buffer + s[i]) != dict.end())  buffer += s[i];  else {  Sequence78 value;  value.next = s[i];  if (dict[buffer] == 0)  value.pos = dict[buffer];  else  value.pos = dict[buffer] - 1;  ans.push\_back(value);  dict[buffer + s[i]] = dict.size() + 1;  buffer = "";  }  }  if (buffer != "") {  char last\_ch = buffer[buffer.size() - 1];  buffer.pop\_back();  Sequence78 value;  if (dict.find(buffer + s[s.size() - 1]) != dict.end()) {  value.next = '$';  value.pos = dict[buffer + s[s.size() - 1]] - 1;  }  else {  value.next = last\_ch;  value.pos = dict[buffer];  }  ans.push\_back(value);  }  return ans;  }  string decodeLZ77(vector<Sequence77> parameters) {  string ans;  for (int i = 0; i < parameters.size(); i++) {  if (parameters[i].length == 0)  ans += parameters[i].next;  else {  int size = ans.length();  size -= parameters[i].offset;  string temp = ans.substr(size, parameters[i].length);  ans += temp + parameters[i].next;  }  }  return ans;  }  string decodeLZ78(vector<Sequence78> parameters) {  int currentWord = 0;  map<int, string> dict = { { currentWord, "" } };  string ans = "";  for (auto el : parameters) {  currentWord++;  string word = dict[el.pos];  if (el.next != '$')  word += el.next;  ans += word;  dict[currentWord] = word;  }  return ans;  } |

1. 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.

|  |
| --- |
| void ShannonFanoMethod(int begin, int end)  {  if ((begin + 1) == end || begin == end || begin> end)  {  if (begin == end || begin> end)  return;  p[end]->arr += '0';  p[begin]->arr += '1';  return;  }  else  {  double group1 = 0, group2 = 0;  double difference1 = 0, difference2 = 0;  for (int i = begin; i <= end - 1; i++)  group1 = group1 + p[i]->frequency;  group2 = group2 + p[end]->frequency;  difference1 = group1 - group2;  if (difference1 < 0)  difference1 = difference1 \* -1;  int k;  for (int j = 2; j != end - begin + 1; j++) {  k = end - j;  group1 = 0;  group2 = 0;  for (int i = begin; i <= k; i++)  group1 = group1 + p[i]->frequency;  for (int i = end; i > k; i--)  group2 = group2 + p[i]->frequency;  difference2 = group1 - group2;  if (difference2 < 0)  difference2 = difference2 \* -1;  if (difference2 >= difference1)  break;  difference1 = difference2;  }  k++;  for (int i = begin; i <= k; i++)  p[i]->arr += '1';  for (int i = k + 1; i <= end; i++)  p[i]->arr += '0';  ShannonFanoMethod(begin, k);  ShannonFanoMethod(k + 1, end);  }  }  string decodeShennonFano(string value, int size)  {  string ans = "";  string buffer = "";  vector<int> candidates;  for (int i = 0; i < value.size(); ++i)  {  buffer += value[i];  for (int j = 0; j < size; ++j)  {  if (buffer.size() != p[j]->arr.size())  continue;  bool isCandidate = true;  for (int k = 0; k < buffer.size(); ++k)  {  if (buffer[k] != p[j]->arr[k])  isCandidate = false;  }  if (isCandidate && find(candidates.begin(), candidates.end(), j) == candidates.end())  candidates.push\_back(j);  else candidates.erase(remove(candidates.begin(), candidates.end(), j), candidates.end());  }  if (candidates.size() == 1)  {  ans += p[candidates[0]]->symbol;  candidates.erase(candidates.begin());  buffer = "";  }  }  return ans;  }  int counter = 0;  string ans = "";  int i = counter;  while (i < str.size())  {  Node\* temp = root\_main;  while (true)  {  if (str[i] == '0')  {  if (temp->left != nullptr)  temp = temp->left;  else  {  ans += temp->symbol;  break;  }  }  else  {  if (temp->right != nullptr)  temp = temp->right;  else  {  ans += temp->symbol;  break;  }  }  i++;  counter = i;  }  if (flag\_ex)  i++;  else  i = counter;  }  return ans;  }  Node\* HuffmanMethod(int size)  {  int current = size;  MyQueue ans;  for (int i = 0; i <= size; ++i)  ans.push(p[i]);  while (ans.checkSize())  {  Node\* first = ans.pop();  Node\* second = ans.pop();  Node\* tmp = new Node('!', first->frequency + second->frequency, first, second);  ans.push(tmp);  }  return ans.pop();  } |

1. Составить отчет, отобразив в нем описание выполнения всех этапов разработки, тестирования и код всей программы со скриншотами результатов тестирования.

Отчет составлен, тестирование проведено успешно (подробнее в пункте 4 данного отчета). Полный код представлен в пункте 5 данного отчета.

# Тестирование

Тестирование приложения.

Метод RLE сжимает данные эффективно в том случае, когда в строке много одинаковых символов. В портивном случае, объем сжатого текста только увеличивается. Сжатие данных алгоритмом RLE представлено на рисунке 1.

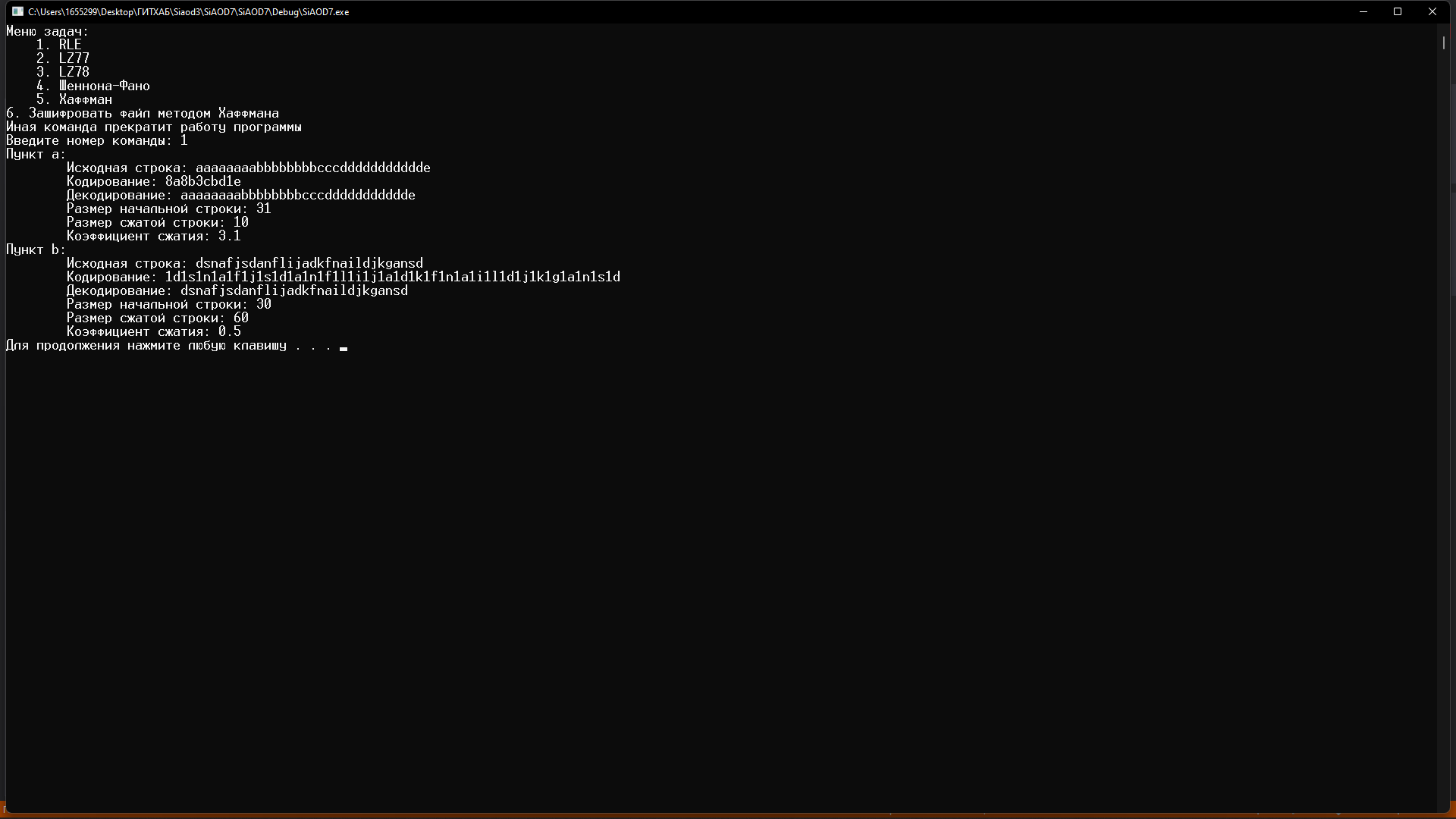


Рисунок 1. Сжатие данных алгоритмом RLE

Алгоритмы LZ77, LZ78 успешно сжимает вводимую строку. Сжатие данных алгоритмом LZ77 представлено на рисунке 2.

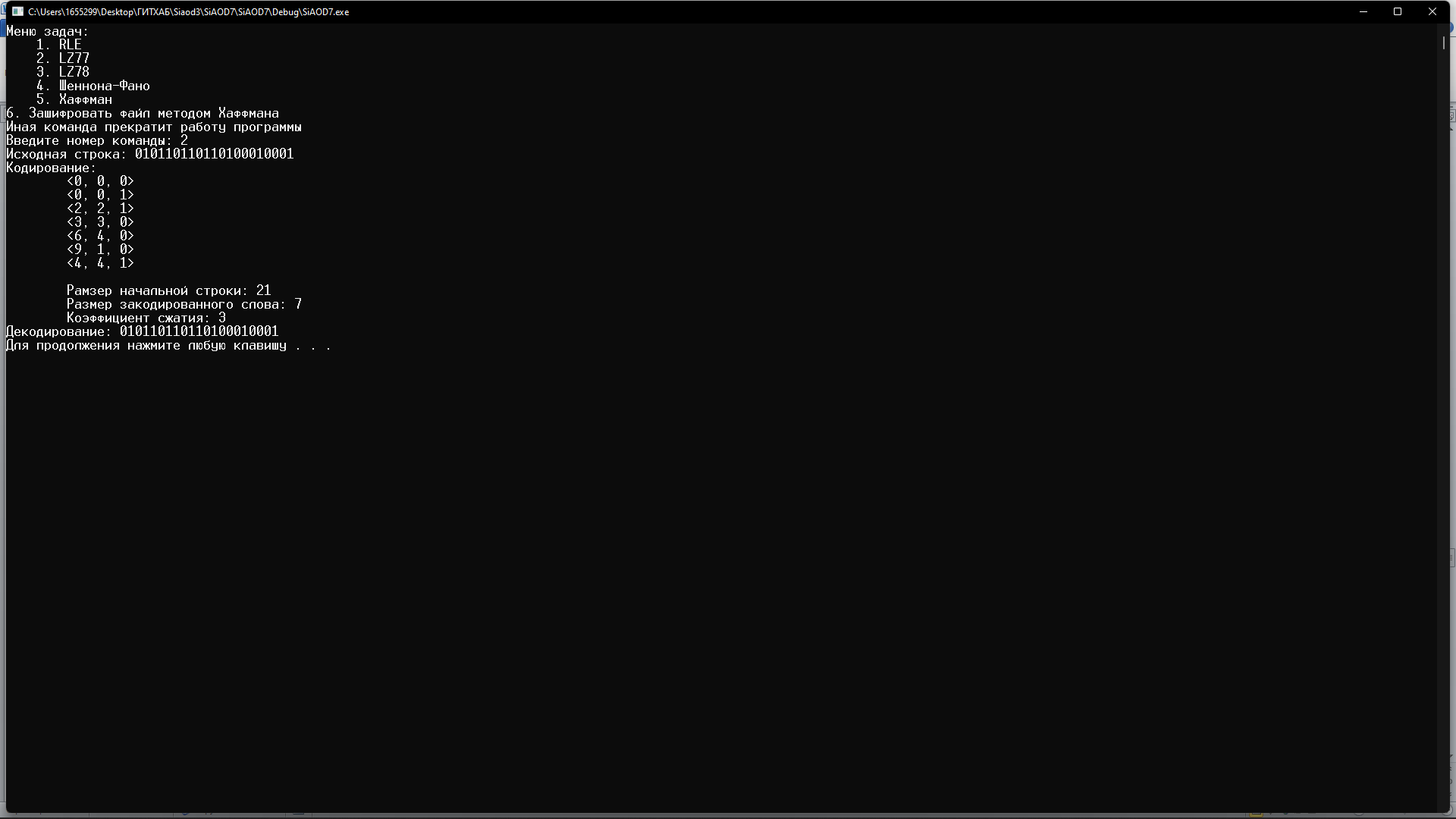
**

Рисунок 2. Сжатие данных алгоритмом LZ77

Сжатие данных алгоритмом LZ78 представлено на рисунке 3.

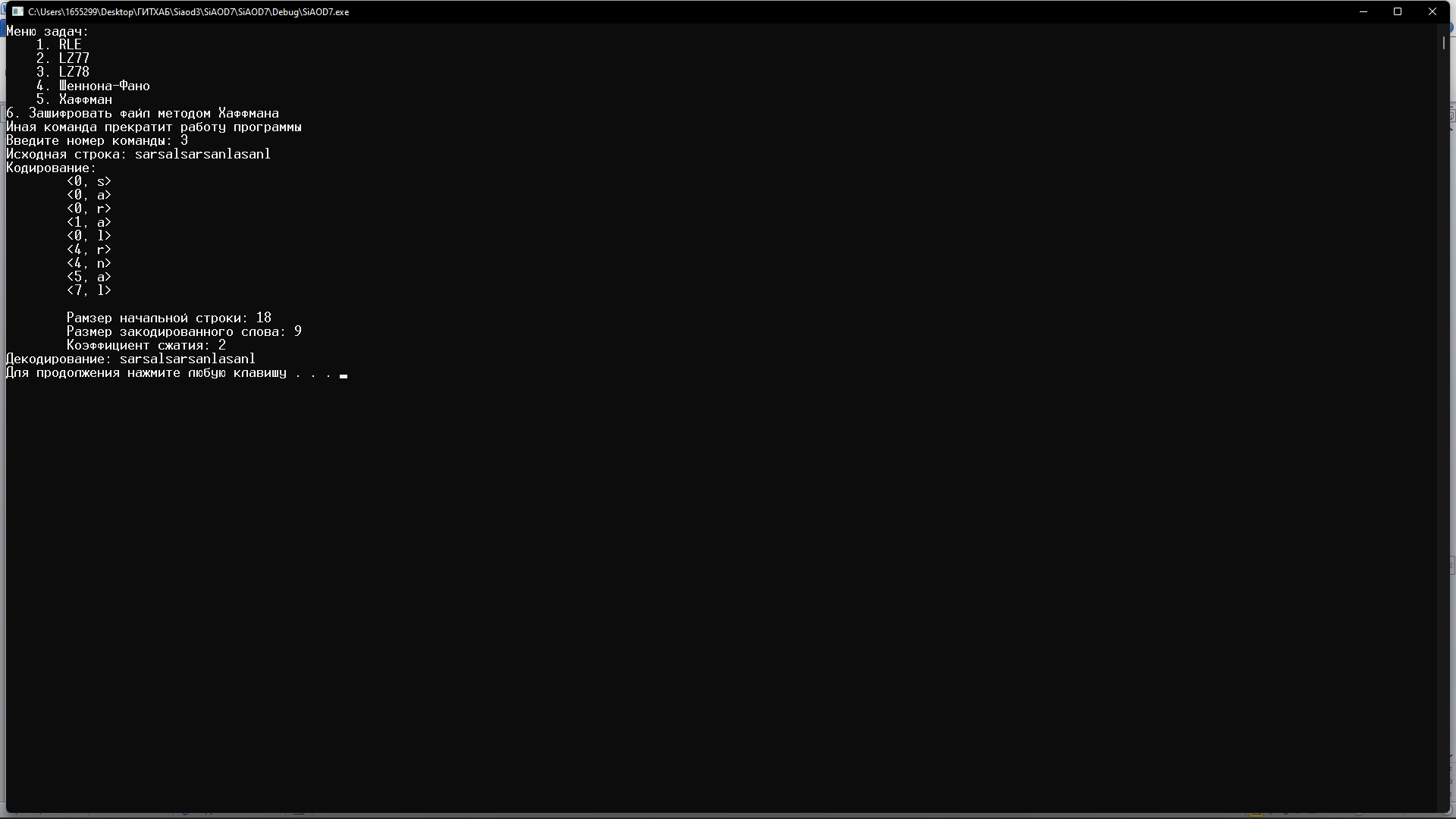


Рисунок 3. Сжатие данных алгоритмом LZ78

Алгоритм Шеннона-Фано плохо сжимает вводимые данные. Строка, зашифрованная с помощью метода Шеннона-Фано занимает больше места, чем начальная. Фрагмент сжатия данных методом Шеннона-Фано представлен на рисунке 6.

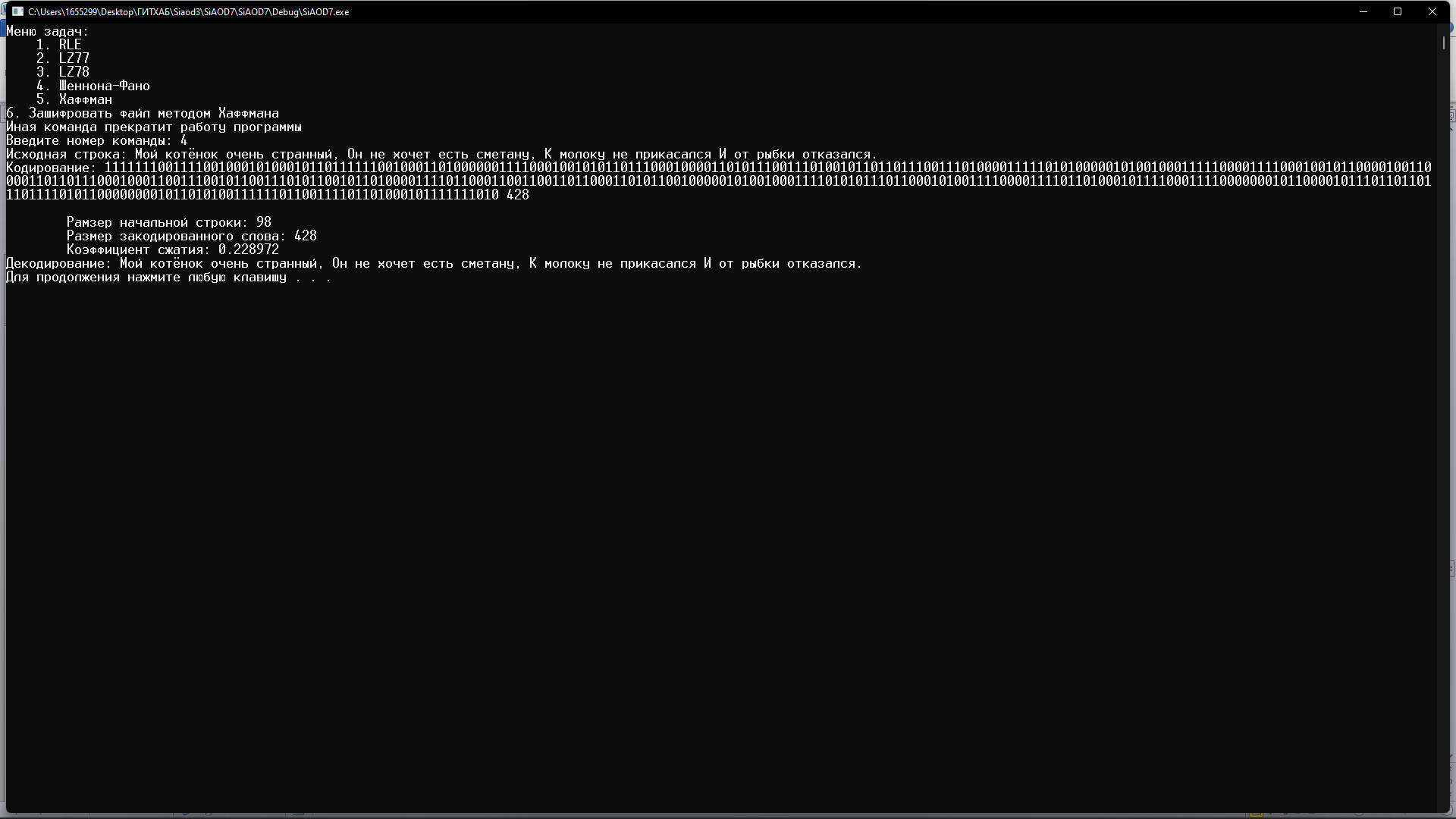


Рисунок 6. Фрагмент сжатия данных методом Шеннона-Фано

Алгоритм Хаффмана так же не эффективно сжимает строку. Сжатие данных методом Хаффмана представлено на рисунке 7.

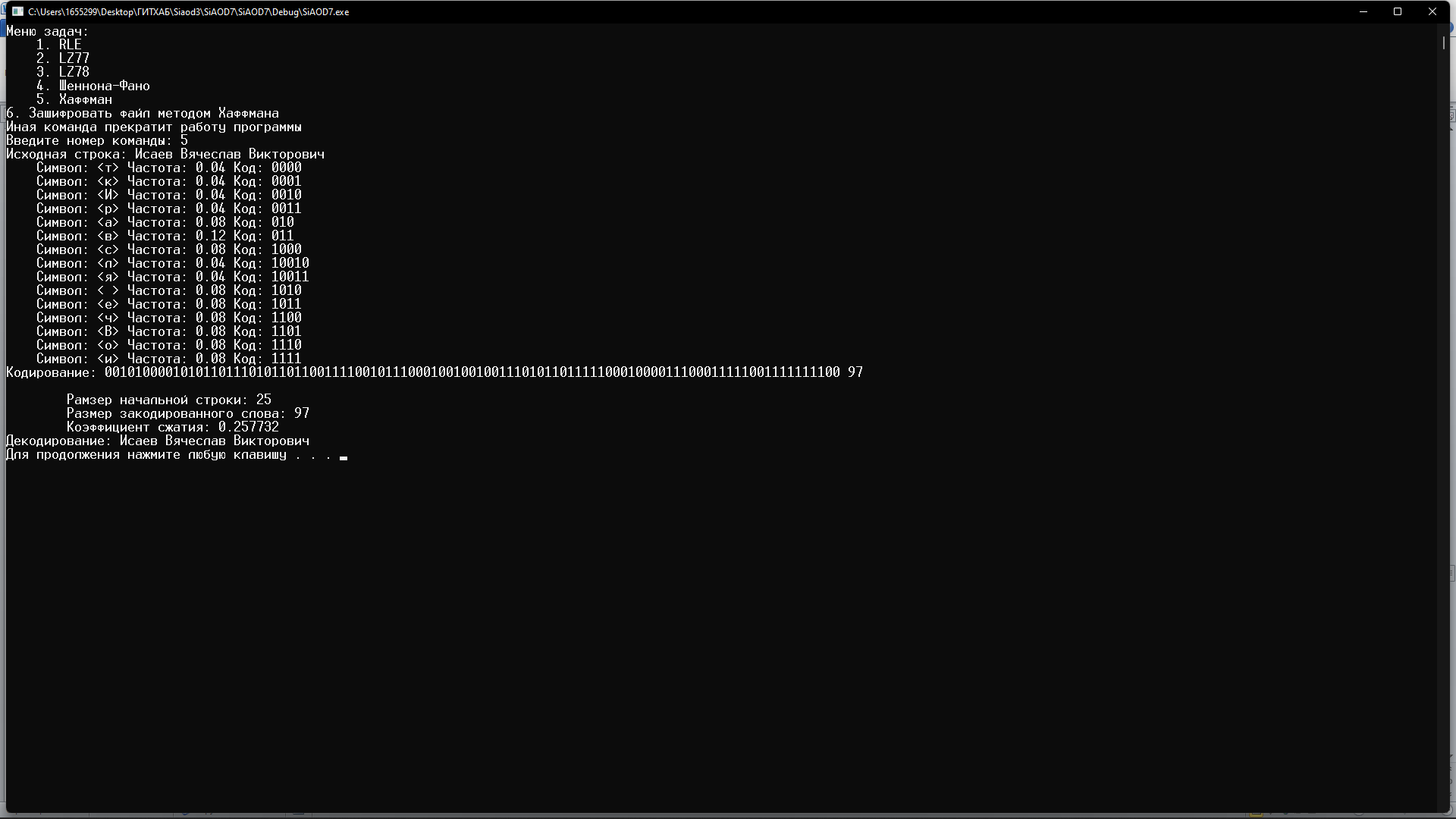


Рисунок 7. Сжатие данных методом Хаффмана

Сжатие большого текстового файла (90КБ) представлено на рисунке 8.

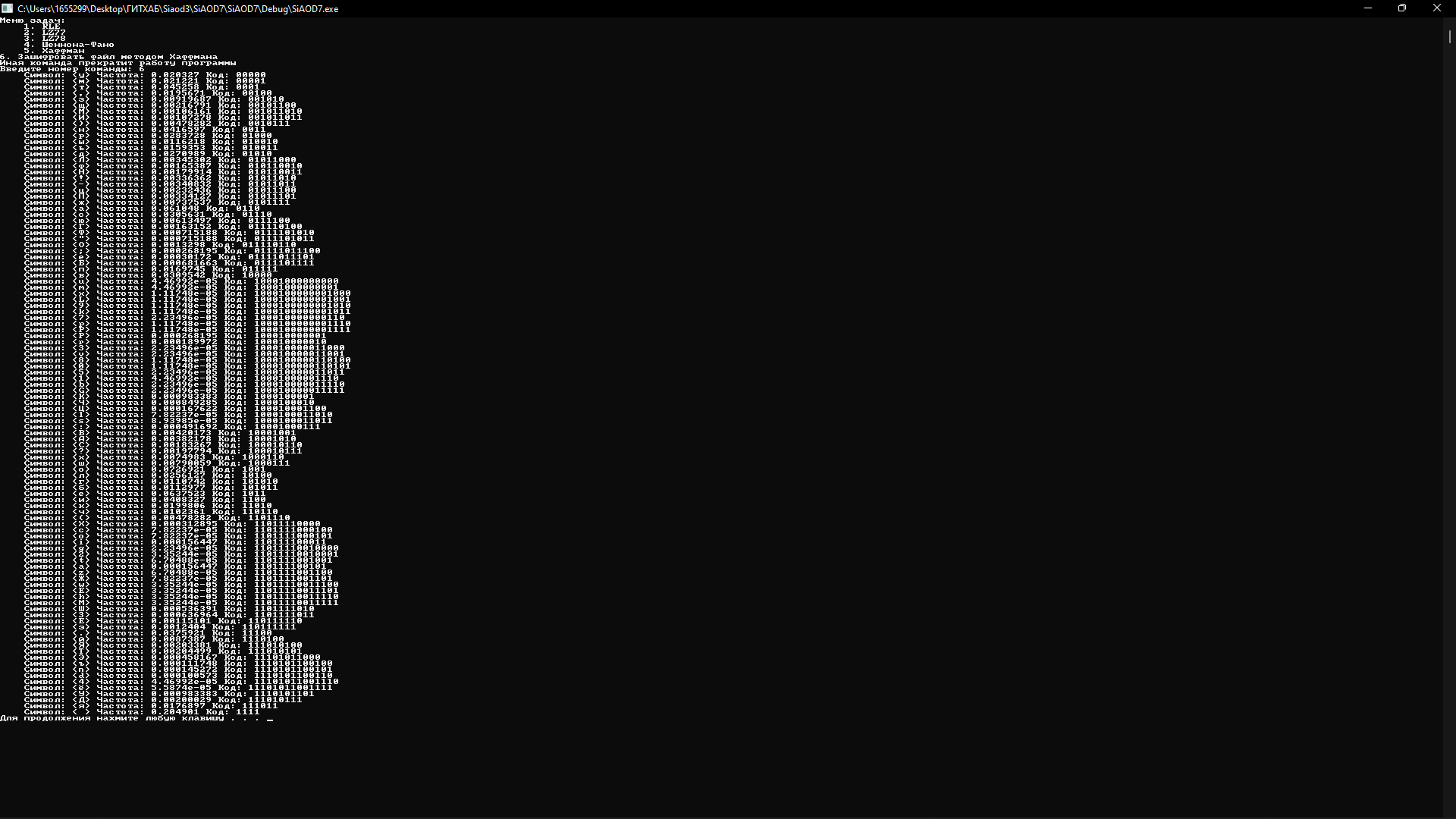


Рисунок 8. Сжатие большого текстового файла

Заданный файл ー 90КБ. Файл, сжатый с помощью алгоритма Хаффмана ー 428КБ. Файл, сжатый с помощью 7ZIP-архиватора ー 37КБ.

Таким образом, после проведения тестирования можно подтвердить, что программа работает правильно.

# Вывод

В ходе выполнения практической работы были выполнены все задания, а также:

1. Изучен принцип работы алгоритмов сжатия данных;
2. Улучшены знания о рекурсии;
3. Улучшены знания о деревьях;
4. Изучен принцип работы векторов.

# Исходный код программы

Source.cpp

|  |
| --- |
| #define \_CRT\_SECURE\_NO\_WARNINGS  #include <iostream>  #include <algorithm>  #include "MyQueue.h"  #include "LZ.h"  #include <fstream>  using namespace std;  Node\*\* p = nullptr;  int rebase(char a)  {  switch (a)  {  case '0':  return 0;  break;  case '1':  return 1;  break;  case '2':  return 2;  break;  case '3':  return 3;  break;  case '4':  return 4;  break;  case '5':  return 5;  break;  case '6':  return 6;  break;  case '7':  return 7;  break;  case '8':  return 8;  break;  case '9':  return 9;  case 'a':  return 10;  break;  case 'b':  return 11;  break;  case 'c':  return 12;  break;  case 'd':  return 13;  break;  case 'e':  return 14;  break;  case 'f':  return 15;  break;  case 'g':  return 16;  break;  case 'h':  return 17;  break;  case 'i':  return 18;  break;  case 'j':  return 19;  break;  case 'k':  return 20;  break;  case 'l':  return 21;  break;  case 'm':  return 22;  break;  case 'n':  return 23;  break;  case 'o':  return 24;  break;  case 'p':  return 25;  break;  case 'q':  return 26;  break;  case 'r':  return 27;  break;  case 's':  return 28;  break;  case 't':  return 29;  break;  case 'u':  return 30;  break;  case 'v':  return 31;  break;  case 'w':  return 32;  break;  case 'x':  return 33;  break;  case 'y':  return 34;  break;  case 'z':  return 35;  break;  default:  break;  }  return NULL;  }  string encodeRle(string value)  {  string ans = "";  int counter = 1;  char prev = value[0];  for (int i = 1; i < value.size(); ++i)  {  if (prev == value[i])  counter++;  else {  if (counter > 9)  {  char buffer[33];  \_itoa(counter, buffer, 36);  ans += buffer;  }  else  {  ans += to\_string(counter);  }  ans += prev;  prev = value[i];  counter = 1;  }  }  if (counter)  {  ans += to\_string(counter);  ans += prev;  prev = value[value.size() - 1];  }  return ans;  }  string decodeRle(string value)  {  string ans = "";  for (int i = 0; i < value.size() - 1; i += 2)  for (int j = 0; j < rebase(value[i]); ++j)  ans += value[i + 1];  return ans;  }  void ShannonFanoMethod(int begin, int end)  {  if ((begin + 1) == end || begin == end || begin> end)  {  if (begin == end || begin> end)  return;  p[end]->arr += '0';  p[begin]->arr += '1';  return;  }  else  {  double group1 = 0, group2 = 0;  double difference1 = 0, difference2 = 0;  for (int i = begin; i <= end - 1; i++)  group1 = group1 + p[i]->frequency;  group2 = group2 + p[end]->frequency;  difference1 = group1 - group2;  if (difference1 < 0)  difference1 = difference1 \* -1;  int k;  for (int j = 2; j != end - begin + 1; j++) {  k = end - j;  group1 = 0;  group2 = 0;  for (int i = begin; i <= k; i++)  group1 = group1 + p[i]->frequency;  for (int i = end; i > k; i--)  group2 = group2 + p[i]->frequency;  difference2 = group1 - group2;  if (difference2 < 0)  difference2 = difference2 \* -1;  if (difference2 >= difference1)  break;  difference1 = difference2;  }  k++;  for (int i = begin; i <= k; i++)  p[i]->arr += '1';  for (int i = k + 1; i <= end; i++)  p[i]->arr += '0';  ShannonFanoMethod(begin, k);  ShannonFanoMethod(k + 1, end);  }  }  string decodeShennonFano(string value, int size)  {  string ans = "";  string buffer = "";  vector<int> candidates;  for (int i = 0; i < value.size(); ++i)  {  buffer += value[i];  for (int j = 0; j < size; ++j)  {  if (buffer.size() != p[j]->arr.size())  continue;  bool isCandidate = true;  for (int k = 0; k < buffer.size(); ++k)  {  if (buffer[k] != p[j]->arr[k])  isCandidate = false;  }  if (isCandidate && find(candidates.begin(), candidates.end(), j) == candidates.end())  candidates.push\_back(j);  else candidates.erase(remove(candidates.begin(), candidates.end(), j), candidates.end());  }  if (candidates.size() == 1)  {  ans += p[candidates[0]]->symbol;  candidates.erase(candidates.begin());  buffer = "";  }  }  return ans;  }  void getCodes(string value, int size)  {  cout << " Символ\t Частота Двоичный код";  for (int i = size - 1; i >= 0; i--)  {  cout << "\n " << " <" << p[i]->symbol << ">" << "\t" << p[i]->frequency << "\t";  for (int j = 0; j <= p[i]->arr.size(); j++)  cout << p[i]->arr[j];  }  }  string encodeText(string value, int size)  {  string ans = "";  for (int i = 0; i < value.size(); ++i)  for (int j = 0; j < size; ++j)  if (value[i] == p[j]->symbol)  ans += p[j]->arr;  return ans;  }  string decodeHuffman(string str, Node\* root\_main, bool flag\_ex)  {  int counter = 0;  string ans = "";  int i = counter;  while (i < str.size())  {  Node\* temp = root\_main;  while (true)  {  if (str[i] == '0')  {  if (temp->left != nullptr)  temp = temp->left;  else  {  ans += temp->symbol;  break;  }  }  else  {  if (temp->right != nullptr)  temp = temp->right;  else  {  ans += temp->symbol;  break;  }  }  i++;  counter = i;  }  if (flag\_ex)  i++;  else  i = counter;  }  return ans;  }  void printLeaves(Node\* root, string path)  {  if (!root->left && !root->right)  {  cout << " Символ: <" << root->symbol << "> Частота: " << root->frequency << " Код: " << path << "\n";  root->arr = path;  return;  }  if (root->left != nullptr)  printLeaves(root->left, path + "0");  if (root->right != nullptr)  printLeaves(root->right, path + "1");  }  Node\* HuffmanMethod(int size)  {  int current = size;  MyQueue ans;  for (int i = 0; i <= size; ++i)  ans.push(p[i]);  while (ans.checkSize())  {  Node\* first = ans.pop();  Node\* second = ans.pop();  Node\* tmp = new Node('!', first->frequency + second->frequency, first, second);  ans.push(tmp);  }  return ans.pop();  }  int CountUniqueSymbols(string input)  {  string tmp = "";  tmp += input[0];  for (int i = 0; i < input.size(); ++i)  if (count(tmp.begin(), tmp.end(), input[i]) == 0)  tmp += input[i];  p = new Node \*[tmp.size()];  for (int i = 0; i < tmp.size(); ++i)  p[i] = new Node(tmp[i], double(count(input.begin(), input.end(), tmp[i])) / double(input.size()));  return tmp.size();  }  void sortByfrequency(int size)  {  for (int j = 1; j < size; j++)  for (int i = 0; i < size - 1; i++)  if (p[i]->frequency > p[i + 1]->frequency)  swap(p[i], p[i + 1]);  }  int main() {  setlocale(0, "RUS");  while (true)  {  system("pause");  system("cls");  cout << "Меню задач:"  << "\n 1. RLE\n " << "2. LZ77\n " << "3. LZ78\n " << "4. Шеннона-Фано\n " << "5. Хаффман\n" << "6. Зашифровать файл методом Хаффмана\n" << "Иная команда прекратит работу программы\n";  int choice;  cout << "Введите номер команды: ";  cin >> choice;  switch (choice)  {  case 1:  {  string a = "aaaaaaaabbbbbbbbcccddddddddddde";  string b = "dsnafjsdanflijadkfnaildjkgansd";  cout << "Пункт а: " << "\n\tИсходная строка: " << a << "\n\tКодирование: " << encodeRle(a) << "\n\tДекодирование: " << decodeRle(encodeRle(a)) <<"\n\tРазмер начальной строки: "<<a.size()<<"\n\tРазмер сжатой строки: "<< encodeRle(a).size() <<"\n\tКоэффициент сжатия: "<< (double)a.size() / (double)encodeRle(a).size() <<"\n";  cout << "Пункт b: " << "\n\tИсходная строка: " << b << "\n\tКодирование: " << encodeRle(b) << "\n\tДекодирование: " << decodeRle(encodeRle(b))<< "\n\tРазмер начальной строки: " << b.size() << "\n\tРазмер сжатой строки: " << encodeRle(b).size() << "\n\tКоэффициент сжатия: " << (double)b.size() / (double)encodeRle(b).size() << "\n";  break;  }  case 2:  {  string lz77 = "010110110110100010001";  cout << "Исходная строка: " << lz77 << "\n";  vector<Sequence77> ans = encodeLZ77(lz77);  cout << "Кодирование:\n";  for (auto el : ans)  cout << "\t<" << el.offset << ", " << el.length << ", " << el.next << ">\n";  cout << "\n\tРамзер начальной строки: " << lz77.size() << "\n\tРазмер закодированного слова: " << ans.size() << "\n\tКоэффициент сжатия: " << (double)lz77.size() / (double)ans.size() << "\n";  cout << "Декодирование: " << decodeLZ77(ans) << "\n";  break;  }  case 3:  {  string lz78 = "sarsalsarsanlasanl";  cout << "Исходная строка: " << lz78 << "\n";  vector<Sequence78> ans2 = encodeLZ78(lz78);  cout << "Кодирование:\n";  for (auto el : ans2)  cout << "\t<" << el.pos << ", " << el.next << ">\n";  cout << "\n\tРамзер начальной строки: " << lz78.size() << "\n\tРазмер закодированного слова: " << ans2.size() << "\n\tКоэффициент сжатия: " << (double)lz78.size() / (double)ans2.size() << "\n";  cout << "Декодирование: " << decodeLZ78(ans2) << "\n";  break;  }  case 4:  {  string shenonFano = "Мой котёнок очень странный, Он не хочет есть сметану, К молоку не прикасался И от рыбки отказался.";  cout << "Исходная строка: " << shenonFano << "\n";  int size = CountUniqueSymbols(shenonFano);  sortByfrequency(size);  ShannonFanoMethod(0, size - 1);  cout << "Кодирование: ";  string ans = encodeText(shenonFano, size);  cout << ans << " " << ans.size() << "\n";  cout << "\n\tРамзер начальной строки: " << shenonFano.size() << "\n\tРазмер закодированного слова: " << ans.size() << "\n\tКоэффициент сжатия: " << (double)shenonFano.size() / (double)ans.size() << "\n";  cout << "Декодирование: " << decodeShennonFano(ans, size) << "\n";  break;  }  case 5:  {  string haffman = "Исаев Вячеслав Викторович";  cout << "Исходная строка: " << haffman << "\n";  int size = CountUniqueSymbols(haffman);  sortByfrequency(size);  Node\* ans = HuffmanMethod(size - 1);  printLeaves(ans, "");  cout << "Кодирование: ";  string text = encodeText(haffman, size);  cout << text << " " << text.size() << "\n";  cout << "\n\tРамзер начальной строки: " << haffman.size() << "\n\tРазмер закодированного слова: " << text.size() << "\n\tКоэффициент сжатия: " << (double)haffman.size() / (double)text.size() << "\n";  cout << "Декодирование: " << decodeHuffman(text, ans, 0) << "\n";  break;  }  case 6:  {  string s;  string book;  ifstream f;  f.open("book.txt");  while (getline(f, s))  {  book += s;  }  f.close();  int bam = CountUniqueSymbols(book);  sortByfrequency(bam);  Node\* kek = HuffmanMethod(bam - 1);  printLeaves(kek, "");  string coded = encodeText(book,bam);  ofstream out;  out.open("bookhaffman.txt");  out << coded;  out.close();  break;  }  default:  cout << "Завершение работы";  return 0;  break;  }  }  } |

MyQueue.cpp

|  |
| --- |
| #include "MyQueue.h"  void MyQueue::push(Node\* newValue)  {  NodeQueue\* top = new NodeQueue(newValue);  if (!head)  {  head = top;  return;  }  NodeQueue\* current = head;  while (current->next && current->value->frequency < newValue->frequency)  current = current->next;  if (!current->next)  current->next = top;  else  {  NodeQueue\* tmp = current->next;  current->next = top;  top->next = tmp;  }  }  Node\* MyQueue::pop()  {  NodeQueue\* tmp = head;  head = head->next;  return tmp->value;  }  bool MyQueue::checkSize()  {  if (head && head->next)  return true;  return false;  } |

LZ.cpp

|  |
| --- |
| #include "LZ.h"  #include <iostream>  vector<Sequence77> encodeLZ77(string s)  {  int maxWindow = 10;  vector<Sequence77> ans;  int sizeBuffer, sizeUnchecked;  for (int i = 0; i < s.length(); i++)  {  if (i + 1 <= maxWindow)  sizeBuffer = i;  else  sizeBuffer = maxWindow;  if (i + sizeBuffer < s.length())  sizeUnchecked = sizeBuffer;  else  sizeUnchecked = s.length() - i;  string buffer = s.substr(i - sizeBuffer, sizeBuffer);  string unchecked = s.substr(i, sizeUnchecked);  int off = -1;  while (true)  {  if (!sizeUnchecked)  break;  string str3 = unchecked.substr(0, sizeUnchecked);  off = buffer.find(str3);  if (off != -1)  break;  sizeUnchecked--;  if (sizeUnchecked <= 0)  break;  }  Sequence77 value;  value.next = s[i + sizeUnchecked];  if (off != -1)  {  value.length = sizeUnchecked;  value.offset = sizeBuffer - off;  ans.push\_back(value);  i += sizeUnchecked;  }  else  {  value.length = 0;  value.offset = 0;  ans.push\_back(value);  }  }  return ans;  }  vector<Sequence78> encodeLZ78(string s) {  string buffer = "";  map<string, int> dict = {};  vector<Sequence78> ans;  for (int i = 0; i < s.size(); ++i) {  if (dict.find(buffer + s[i]) != dict.end())  buffer += s[i];  else {  Sequence78 value;  value.next = s[i];  if (dict[buffer] == 0)  value.pos = dict[buffer];  else  value.pos = dict[buffer] - 1;  ans.push\_back(value);  dict[buffer + s[i]] = dict.size() + 1;  buffer = "";  }  }  if (buffer != "") {  char last\_ch = buffer[buffer.size() - 1];  buffer.pop\_back();  Sequence78 value;  if (dict.find(buffer + s[s.size() - 1]) != dict.end()) {  value.next = '$';  value.pos = dict[buffer + s[s.size() - 1]] - 1;  }  else {  value.next = last\_ch;  value.pos = dict[buffer];  }  ans.push\_back(value);  }  return ans;  }  string decodeLZ77(vector<Sequence77> parameters) {  string ans;  for (int i = 0; i < parameters.size(); i++) {  if (parameters[i].length == 0)  ans += parameters[i].next;  else {  int size = ans.length();  size -= parameters[i].offset;  string temp = ans.substr(size, parameters[i].length);  ans += temp + parameters[i].next;  }  }  return ans;  }  string decodeLZ78(vector<Sequence78> parameters) {  int currentWord = 0;  map<int, string> dict = { { currentWord, "" } };  string ans = "";  for (auto el : parameters) {  currentWord++;  string word = dict[el.pos];  if (el.next != '$')  word += el.next;  ans += word;  dict[currentWord] = word;  }  return ans;  } |

Myqueue.h

|  |
| --- |
| #pragma once  #include <iostream>  using namespace std;  struct Node  {  char symbol;  double frequency;  string arr = "";  Node\* left;  Node\* right;  Node(char s, double f, Node\* l = nullptr, Node\* r = nullptr) : symbol(s), frequency(f), left(l), right(r) {};  };  struct NodeQueue  {  Node\* value;  NodeQueue\* next;  NodeQueue(Node\* newValue) : value(newValue), next(nullptr) {};  };  class MyQueue  {  private:  NodeQueue\* head = nullptr;  public:  void push(Node\*);  Node\* pop();  bool checkSize();  }; |

LZ.h

|  |
| --- |
| #pragma once  #include <vector>  #include <map>  #include <string>  using namespace std;  struct Sequence77  {  int offset;  int length;  char next;  };  struct Sequence78  {  int pos;  char next;  };  vector<Sequence77> encodeLZ77(string s);  vector<Sequence78> encodeLZ78(string s);  string decodeLZ77(vector<Sequence77>);  string decodeLZ78(vector<Sequence78>); |