МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Кафедра «Вычислительная техника»

Дисциплина «Алгоритмы и структуры данных»

**Пояснительная записка**

**к курсовой работе по теме**

**«Burrows-Wheeler»**

Выполнили:

студенты группы ИВТАПбд-21

Кондратьев П.С. и Закамсков Б.И.

Проверил:

Фолунин В. А.

Ульяновск, 2017

**Содержание**

[1 Техническое задание 3](#_Toc502135130)

[2 Техническое проектирование 4](#_Toc502135131)

[2.1 Описание классов 5](#_Toc502135132)

[2.2 Взаимодействие классов 8](#_Toc502135133)

[2.3 Справка по основной части приложения 9](#_Toc502135134)

[3 Алгоритмическое обеспечение 16](#_Toc502135135)

[3.1 Описание используемых алгоритмов и их назначения 16](#_Toc502135136)

[3.2 Теоретическое обоснование сложности 18](#_Toc502135137)

[3.3 Эмпирическое обоснование сложности 22](#_Toc502135138)

[HashTab — определяем контрольные суммы файла 24](#_Toc502135139)

[Вывод 28](#_Toc502135140)

[Используемые ресурсы 29](#_Toc502135141)

[Приложение 1. Руководство пользователя 30](#_Toc502135142)

[Приложение 2. Исходный код 31](#_Toc502135144)

# 1 Техническое задание

Требуется создать приложение, позволяющее производить сжатие и распаковку данных при помощи преобразований Move-to-front и Барроуза-Уилера, а также кодирования Хаффмана. Приложение представляет собой exe файл проекта, который вызывает консоль Visual Studio в которой производиться последующие действия с вашим файлом на компьютере.

Программная часть приложения написана на языке программирования С++ с использованием среды разработки Visual Studio. При запуске exe файла появляется пользовательская консоль с интерфейсом. На первом этапе работы пользователю доступно только одно действие – выбор файла для загрузки из папки «Проект» на рабочем столе. Из выбранного файла считывается байты. Выбранный файл программа кодирует по этапно (в 3 этапа). Помимо исходного файла будут генерироваться еще файлы трех кодировок, а именно кодирование и декодирование классов MTF, BWT, HUFFMAN. Для реализации данного технического задания потребуется

**Реализовать:**

* Класс MoveToFront, осуществляющий MTF-преобразование;
* Класс BurrowsWheeler, осуществляющий преобразование Барроуза-Уилера;
* Класс, осуществляющий кодирование по Хаффману;
* Класс(ы) графического интерфейса приложения (можно использовать только консольный интерфейс, но это снижает оценку).

Основные алгоритмы в этой работе:

* Преобразование Move-to-front;
* Преобразование Барроуза-Уиллера;
* Кодирование Хаффмана.

Курсовая работа была поделена на две алгоритмические части:

* Написание двух классов: MTF, BWT Закамским Богданом
* Написание двух классов: BWT, HUFFMAN Кондратьевым Павлом

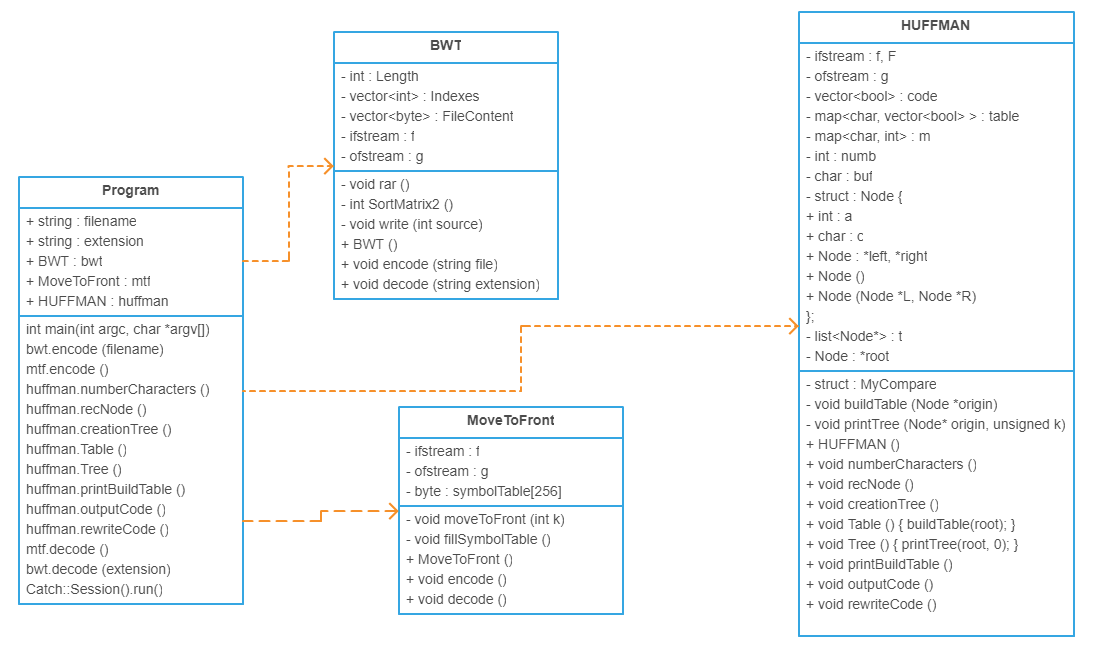
# 2 Техническое проектирование

Программная часть приложения состоит из трех классов. Эти классы представляют собой описание структур объектов одного вида с набором методов их обработки. Графическое представление некоторого количества классов и связей между ними называется [диаграммой классов](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0_%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D0%BE%D0%B2) (Рис. 1). При использовании классов все элементы кода программы, такие как переменные, константы, методы, процедуры и функции, обязаны принадлежать тому или иному классу. Сам класс в итоге определяется как список своих членов, а именно [полей](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%B5_%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D0%B0) ([свойств](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%BE%D0%B9%D1%81%D1%82%D0%B2%D0%BE_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29)) и [методов (функций, процедур](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29)).

Каждому члену класса можно установить модификатор доступа (за исключением статических конструкторов и некоторых других вещей). В большинстве [объектно-ориентированных языков программирования](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) поддерживаются следующие модификаторы доступа:

* private (закрытый, внутренний член класса) — обращения к члену допускаются только из [методов](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29) того класса, в котором этот член определён. Любые наследники класса уже не смогут получить доступ к этому члену. Наследование по типу private делает все члены родительского класса private-членами класса-наследника;
* public (открытый член класса) — обращения к члену допускаются из любого кода. Наследование по типу public не меняет модификаторов родительского класса.

## **2.1 Описание классов**

 Рис. 1. UML-диаграмма классов

Классы:

1. Класс HUFFMAN.

Данный класс содержит методы, осуществляющие чтение файла c компьютера и последующее кодирование этого файла по битно методом Хаффмана.

Класс содержит следующие поля:

* поле f, F имеют тип ifstream и режим доступа private, f содержит выбранный пользователем файл, а F содержит исходный закодированный файл;
* поле code имеет тип vector<bool> и режим доступа private, он содержит в себе пары 'символ-код';
* поле table имеет тип map <char, vector<bool>> и режим доступа private, содержащий в себе частоты вхождения букв из считанного фала;
* поля int numb и char buf нужны для подсчета восьми бит и записи их в закодированный файл и имеют режим доступа private;
* поле struct Node и list<Node\*> t являются дерево Хаффмана, в котором храниться исковое дерево кодов символов и имеют режим доступа private;
* Класс содержит следующие методы:
* метод buildTable с режимом доступом private, вызывается в методе Table с режимом доступом public, для создания пары 'символ-код' искомого файла;
* метод printTree с режимом доступом private, вызывается в методе Tree с режимом доступом public, для вывода дерева Хаффмана;
* метод numberCharacters с режимом доступом public, считаем частоты символов в исходном файле;
* метод recNode с режимом доступом public, записывает начальные узлы в список list для последующего создания дерева Хаффмана;
* метод creationTree с режимом доступом public, производит создание дерева Хаффмана;
* метод printBuildTable с режимом доступа public, вывод 'символ-код' искомого файла;
* метод outputCode с режимом доступа public, кодирование исходного файла и запись его в новый;
* метод rewriteCode с режимом доступа public, считывание закодированного файла и преобразование его обратно (декодирование);

1. Класс BWT.

Данный класс содержит методы, осуществляющие чтение файла c компьютера и последующее преобразование этого файла методом Барроуза-Уилера.

Класс содержит следующие поля:

* поле f, g имеют тип ifstream и режим доступа private, f содержит исходный файл, а g содержит закодированный файл;
* поле Indexes имеет тип vector<int> и режим доступа private, он содержит в себе указатели на все циклически сдвинутые строки исходного файла;
* поле FileContent имеет тип vector<byte> и режим доступа private, содержащий в себе строки из исходного файла;
* поле Length имеет тип int и нужно для хранения длинны строки FileContent;

Класс содержит следующие методы:

* метод SortMatrix2 с режимом доступом private, вызывается в методе rar с режимом доступом private, для суффиксальной сортировки строки исходного файла;
* метод rar с режимом доступом private, вызывается в методе encode с режимом доступом public, для преобразования Барроуза-Уилера;
* метод encode с режимом доступом public, считываем строки из исходного файла и выполняем преобразование Барроуза-Уилера, с последующим записыванием в закодированный файл;
* метод decode с режимом доступом public, считываем строки из закодированного файла и выполняем обратное преобразование Барроуза-Уилера, с последующим записыванием в раскодированный файл;

1. Класс MoveToFront.

Данный класс содержит методы, осуществляющие чтение файла c компьютера и последующее преобразованием этого файла методом MoveToFront.

Класс содержит следующие поля:

* поле f, g имеют тип ifstream и режим доступа private, f содержит исходный файл, а g содержит закодированный файл;
* поле symbolTable имеет тип byte[ ] и режим доступа private, содержит в себе алфавит;

Класс содержит следующие методы:

* метод moveToFront с режимом доступом private, вызывается в методе encode с режимом доступом public, для перемещения символа в начало алфавита;
* метод fillSymbolTable с режимом доступом private, вызывается в методе encode и decode с режимом доступом public, для заполнения алфавита;
* метод encode с режимом доступом public, считываем строки из исходного файла и выполняем преобразование MoveToFront, с последующим записыванием в закодированный файл;
* метод decode с режимом доступом public, считываем строки из закодированного файла и выполняем обратное преобразование MoveToFront, с последующим записыванием в раскодированный файл;

## **2.2 Взаимодействие классов**

Зависимость — это слабая форма отношения использования, при котором изменение в спецификации одного влечёт за собой изменение другого, причём обратное не обязательно. Возникает, когда объект выступает, например, в форме параметра или локальной переменной.

Графически представляется штриховой стрелкой, идущей от зависимого элемента к тому, от которого он зависит.

На UML-диаграмме классов (Рис. 1) наглядно видна зависимость между классами. Представленная взаимосвязь между классами BWT, MoveToFront и HUFFAN называется зависимостью, так как в Porogram производиться вызов этих трех классов по очередности, при этом сначала производиться вызов BWT от исходного файла, после чего закодированный им файл передается в MoveToFront и затем HUFFAN.Также производится и декодирование только в обратном порядке.

## **2.3 Справка по основной части приложения**

**Методы класса BWT:**

* int SortMatrix2()

Производит сортировку суффиксов исходной строки и записывает результат в вектор Indexes.

Возвращает номер исходной строки в отсортированной матрице.

Код метода:

int BWT::SortMatrix2() {

int source = 0;

vector<int> count(Length);

for (int i = 0; i < Length; i++)

count[FileContent[i]]++;

vector<int> from(Length);

for (int i = 1; i < Length; i++)

from[i] = from[i - 1] + count[i - 1];

for (int i = 0; i < Length; i++)

Indexes[from[FileContent[i]]++] = i;

vector<int> groups(Length);

int groupsCount = 1;

groups[Indexes[0]] = 0;

for (int i = 1; i < Length; i++) {

if (FileContent[Indexes[i]] != FileContent[Indexes[i - 1]])

groupsCount++;

groups[Indexes[i]] = groupsCount - 1;

}

vector<int> newIndexes(Length), newGroups(Length);

for (int blockLength = 1; blockLength < Length; blockLength \*= 2) {

for (int i = 0; i < Length; i++)

newIndexes[i] = (Indexes[i] + Length - blockLength) % Length;

for (int i = 0; i < Length; i++)

count[i] = from[i] = 0;

for (int i = 0; i < Length; i++)

count[groups[newIndexes[i]]]++;

for (int i = 1; i < groupsCount; i++)

from[i] = from[i - 1] + count[i - 1];

for (int i = 0; i < Length; i++) {

if (newIndexes[i] == 0)

source = from[groups[newIndexes[i]]];

Indexes[from[groups[newIndexes[i]]]++] = newIndexes[i];

}

groupsCount = 1;

newGroups[Indexes[0]] = 0;

for (int i = 1; i < Length; i++) {

if (groups[Indexes[i]] != groups[Indexes[i - 1]] ||

groups[(Indexes[i] + blockLength) % Length] != groups[(Indexes[i - 1] + blockLength) % Length])

groupsCount++;

newGroups[Indexes[i]] = groupsCount - 1;

}

for (int i = 0; i < Length; i++)

groups[i] = newGroups[i];

}

return source;

}

* void rar()

Обновляет вектор Indexes(чистит и заполняет его индексами от 0 до Length), вызывает метод SortMatrix2. Возвращенный номер исходной строки записывает в массив и вызывает метод write для записи результата в закодированный файл.

Код метода:

void BWT::rar() {

Indexes.clear();

for (int i = 0; i < Length; i++) {

Indexes.push\_back(i);

}

int source = SortMatrix2();

int mass[4] = {};

for (int i = 3; i > -1; i--) {

if (source) {

mass[i] = source % 10;

source /= 10;

}

}

write(mass);

}

* void write(int source[])

Записывает номер исходной строки и результат сортировки в закодированный файл.

Аргументы:

* source[] – номер строки в отсортированной матрице.
* void encode(string file)

Считывает данные исходного файла и вызывает метод rar.

Аргументы:

* file – имя исходного файла.
* void decode(string extension)

Считывает номер закодированной строки в матрице и закодированную строку, методом обратного преобразования Барроуза-Уилера получает исходную строку и записывает раскодированную строку в файл.

Аргументы:

* extension – расширение исходного файла.

Код метода:

void BWT::decode(string extension) {

f.open("C:/Users/Bogdan/Desktop/rar/decodeMTF.bzip2", ios::out | ios::binary);

g.open("C:/Users/Bogdan/Desktop/rar/decodeBWT." + extension, ios::out | ios::binary);

char ch;

int size = 9999;

while (1) {

if (f.eof()) {

g.close();

f.close();

return;

}

vector<byte> bufIn;

int ko = 1000;

int primaryIndex = 0;

for (int i = 3; i > -1; i--) {

primaryIndex += ko \* (int)f.get();

ko /= 10;

}

if (f.eof()) {

g.close();

f.close();

return;

}

for (int i = 0; i < 9999; i++) {

ch = f.get();

if (f.eof()) {

size = bufIn.size();

break;

}

bufIn.push\_back(ch);

}

vector<byte> F(size);

vector<int> buckets(256, 0);

int i, j, k;

vector<int> indices(size);

for (i = 0; i < size; i++)

buckets[bufIn[i]]++;

for (i = 0, k = 0; i < 256; i++)

for (j = 0; j < buckets[i]; j++)

F[k++] = i;

for (i = 0, j = 0; i < 256; i++) {

while (j < size && i > F[j])

j++;

buckets[i] = j;

}

for (i = 0; i < size; i++)

indices[buckets[bufIn[i]]++] = i;

for (i = 0, j = primaryIndex; i < size; i++) {

j = indices[j];

g << bufIn[j];

}

}

g.close();

f.close();

}

**Методы класса MoveToFront:**

* moveToFront(int k)

Перемещает k-ый символ алфавите в начало.

Аргументы:

* k – индекс символа в алфавите, который нужно переместить в начало.

Код метода:

void MoveToFront::moveToFront(int k) {

byte t = symbolTable[k];

for (int i = k - 1; i >= 0; i--)

symbolTable[i + 1] = symbolTable[i];

symbolTable[0] = t;

}

* void fillSymbolTable()

Обновляет алфавит, записанный в массиве symbolTable.

* void encode()

Вызывает метод fillSymbolTable, для заполнения алфавита, считывает символ из исходного файла, записывает его индекс в алфавите и вызывает метод moveToFront, который перемещает этот символ в начало алфавита.

Код метода:

void MoveToFront::encode() {

f.open("C:/Users/Bogdan/Desktop/rar/encodeBWT.bzip2", std::ios::out | std::ios::binary);

g.open("C:/Users/Bogdan/Desktop/rar/encodeMTF.bzip2", std::ios::out | std::ios::binary);

fillSymbolTable();

while (1) {

byte c = f.get();

if (f.eof()) {

f.close();

g.close();

return;

}

for (int i = 0; i < 256; i++) {

if (c == symbolTable[i]) {

g << (byte)i;

moveToFront(i);

break;

}

}

}

}

* void decode()

Вызывает метод fillSymbolTable, для заполнения алфавита, считывает символ ch из закодированного файла, записывает символ, стоящий на ch-том месте в алфавите и вызывает метод moveToFront, который перемещает этот символ в начало алфавита.

Код метода:

void MoveToFront::decode() {

f.open("C:/Users/Bogdan/Desktop/rar/decodeHUFFMAN.bzip2", std::ios::out | std::ios::binary);

g.open("C:/Users/Bogdan/Desktop/rar/decodeMTF.bzip2", std::ios::out | std::ios::binary);

fillSymbolTable();

while (1) {

byte c = f.get();

if (f.eof()) {

f.close();

g.close();

return;

}

g << symbolTable[c];

moveToFront(c);

}

}

**Методы класса HUFFMAN:**

* void numberCharacters()

Считает частоты символов в исходном файле.

Код метода:

void HUFFMAN::numberCharacters() {

f.open("C:/Users/Pavel/Desktop/rar/encodeMTF.bzip2", ios::out | ios::binary);

while (1)

{

char c = f.get();

if (f.eof())

break;

m[c]++;

}

}

* void creationTree()

Строит бинарное дерево. Выбираем 2 наименьших символа (чтобы количество вхождений было наименьшее) и строим от него новый узел, складывая значения двух предыдущих. После чего повторяем итерацию уже с новым узлом до тек пор, пока не построится дерево.

Код метода:

void HUFFMAN::creationTree() {

while (t.size() != 1) {

t.sort(MyCompare());

Node \*SonL = t.front();

t.pop\_front();

Node \*SonR = t.front();

t.pop\_front();

Node \*parent = new Node(SonL, SonR);

t.push\_back(parent);

}

root = t.front(); //root - указатель на вершину дерева

}

* void buildTable(Node \*origin)

Создает map и записывает в него соотношение “символ-код”.

Аргументы:

* \*origin – указатель на корень дерева;

Код метода:

void HUFFMAN::buildTable(Node \*origin) {

if (origin->left != NULL) {

code.push\_back(0);

buildTable(origin->left);

}

if (origin->right != NULL) {

code.push\_back(1);

buildTable(origin->right);

}

if (origin->left == NULL && origin->right == NULL)

table[origin->c] = code;

if (!code.empty())

code.pop\_back();

}

void HUFFMAN::printBuildTable() {

for (auto it = table.begin(); it != table.end(); it++) {

cout << it->first << ":";

for (auto iter = it->second.begin(); iter != it->second.end(); iter++) {

cout << \*iter;

}

cout << endl;

}

}

* void outputCode()

Считывает символы из исходного файла и записывает в закодированный файл бинарный код символов в соответствии с кодом в table.

Код метода:

void HUFFMAN::outputCode() {

f.clear(); f.seekg(0);// перемещаем указатель снова в начало файла

g.open("C:/Users/Pavel/Desktop/rar/encodeHUFFMAN.bzip2", ios::out | ios::binary);

while (1) {

char c = f.get();

if (f.eof()) {

g << buf << (char)numb;

break;

}

vector<bool> x = table[c];

for (int n = 0; n < x.size(); n++) {

buf = buf | x[n] << (7 - numb);

numb++;

if (numb == 8) {

numb = 0;

g << buf;

buf = 0;

}

}

}

numb = 0;

f.close();

g.close();

}

* void rewriteCode()

Производит декодирование файла. Считывает бит из файла пока не найдет нужную кодировку символа и записывает нужную кодировку в новый файл.

Код метода:

void HUFFMAN::rewriteCode() {

F.open("C:/Users/Pavel/Desktop/rar/encodeHUFFMAN.bzip2", ios::in | ios::binary);

g.open("C:/Users/Pavel/Desktop/rar/decodeHUFFMAN.bzip2", ios::out | ios::binary);

char ch1, ch2, ch3;

Node \*p = root;

bool byte = false;

ch3 = F.get(); ch2 = F.get();ch1 = F.get();

while (1) {

if (numb == 8) {

ch3 = ch2;

ch2 = ch1;

ch1 = F.get();

numb = 0;

}

if (F.eof()) {

for (int i = 0; i < (int)ch2; i++) {

byte = ch3 & 1 << (7 - i);

if (byte)

p = p->right;

else

p = p->left;

if (p->left == NULL && p->right == NULL) {

g << p->c;

p = root;

}

}

break;

}

byte = ch3 & 1 << (7 - numb);

numb++;

if (byte)

p = p->right;

else

p = p->left;

if (p->left == NULL && p->right == NULL) {

g << p->c;

p = root;

}

}

g.close();

F.close();

}

# 3 Алгоритмическое обеспечение

## **3.1 Описание используемых алгоритмов и их назначения**

Алгоритм Хаффмана:

Построение кода Хаффмана сводится к построению соответствующего бинарного дерева по следующему алгоритму:

Составим список кодируемых символов, при этом будем рассматривать один символ как дерево, состоящее из одного элемента c весом, равным частоте появления символа в строке.

Из списка выберем два узла с наименьшим весом.

Сформируем новый узел с весом, равным сумме весов выбранных узлов, и присоединим к нему два выбранных узла в качестве детей.

Добавим к списку только что сформированный узел вместо двух объединенных узлов.

Если в списке больше одного узла, то повторим пункты со второго по пятый.

Преобразование Барроуза — Уилера:

Алгоритм меняет порядок символов во входной строке таким образом, что повторяющиеся подстроки образуют на выходе идущие подряд последовательности одинаковых символов. Таким образом, сочетание BWT и RLE выполняет задачу сжатия исключением повторяющихся подстрок, то есть задачу, аналогичную алгоритмам LZ.

Кроме того, почти точно повторяющиеся (с незначительными отличиями) подстроки входного текста дают на выходе последовательности одинаковых символов, редко перемежающиеся другими символами. Если после этого выполнить шаг по замене каждого символа расстоянием до его предыдущей встречи (т. н. алгоритм move to front, MTF) — то полученный набор чисел будет иметь крайне удачное статистическое распределение для применения энтропийного сжатия типа Хаффмана или же арифметического.

На практике алгоритм сжатия вида BWT → MTF/RLE → Хаффман, применённый в архиваторе bzip2, немного превосходит лучшие реализации LZH по качеству сжатия при аналогичной скорости.

Преобразование MoveToFront:

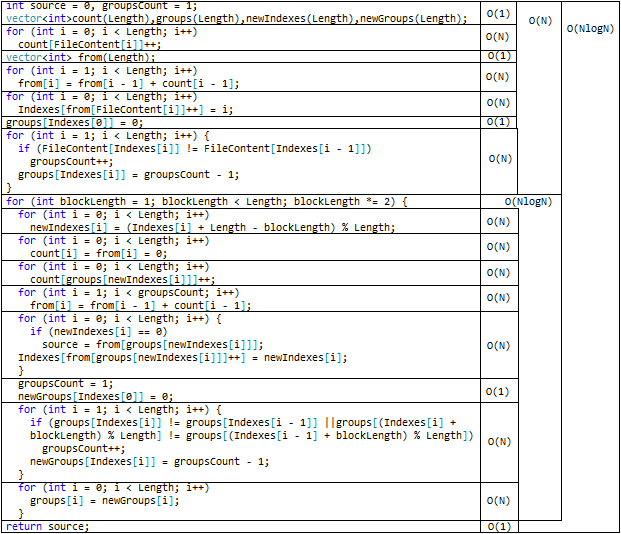
Основной идеей преобразования является замена каждого входного символа его номером в специальном стеке недавно использованных символов. Последовательности идентичных символов, к примеру, будут заменены (начиная со второго символа) на последовательность нулей. Если же символ долго не появлялся во входной последовательности, он будет заменен большим числом. Преобразование заменяет последовательность входных символов на последовательность целых чисел, если во входных данных было много локальных корреляций, то среди этих чисел будут преобладать небольшие, лучше сжимаемые энтропийным кодированием, чем исходные данные.

Часто используется при преобразовании байтов. Изначально каждое возможное значение байта записывается в список, в ячейку с номером, равным значению байта, т.е. (0, 1, 2, 3, …, 255). В процессе обработки данных этот список изменяется. Первый обработанный символ заменяется самим собой, после чего элемент, соответствующий этому символу, перемещается в голову списка (сдвигая элементы с 0 по своё положение на 1 вправо). Последующие символы кодируются номером элемента, содержащего их значение. После кодирования каждого символа эти элементы также продвигаются к голове списка.

## **3.2 Теоретическое обоснование сложности**

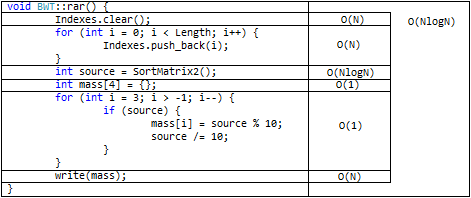
Вычислительная сложность - понятие в [информатике](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0) и [теории алгоритмов](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D0%BE%D0%B2), обозначающее функцию зависимости объёма работы, которая выполняется некоторым алгоритмом, от размера входных данных.

int BWT::SortMatrix2(){

****

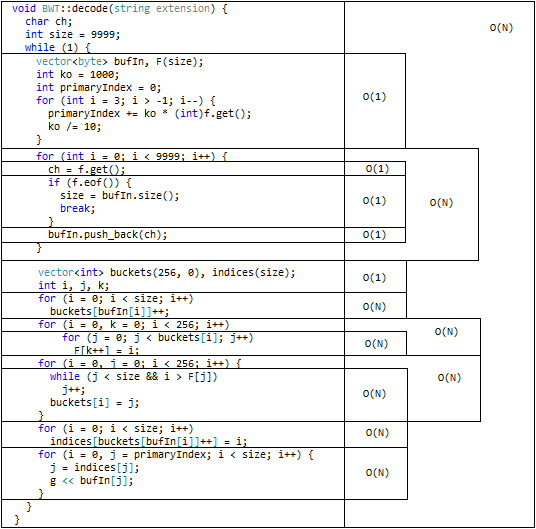
}

Теоретическая оценка сложности метода BWT::SortMatrix2 - O(NlogN).

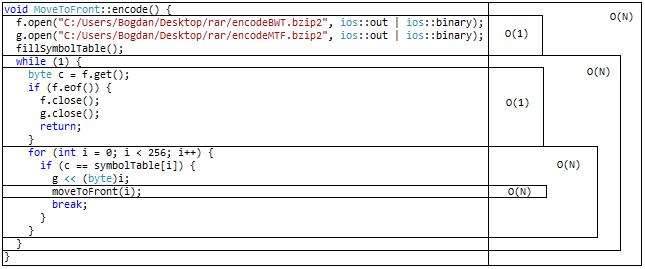


Теоретическая оценка сложности метода BWT::rar - O(NlogN).

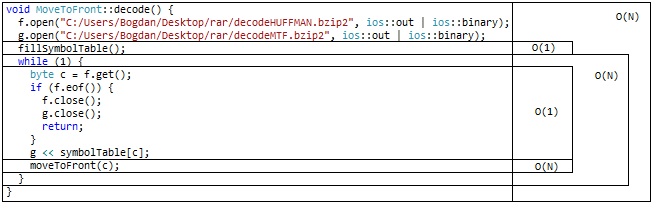
Так как в методе encode выполняется поблочное считывание символов и вызов rar, то теоретическая оценка сложности метода encode – O(NlogN).

****

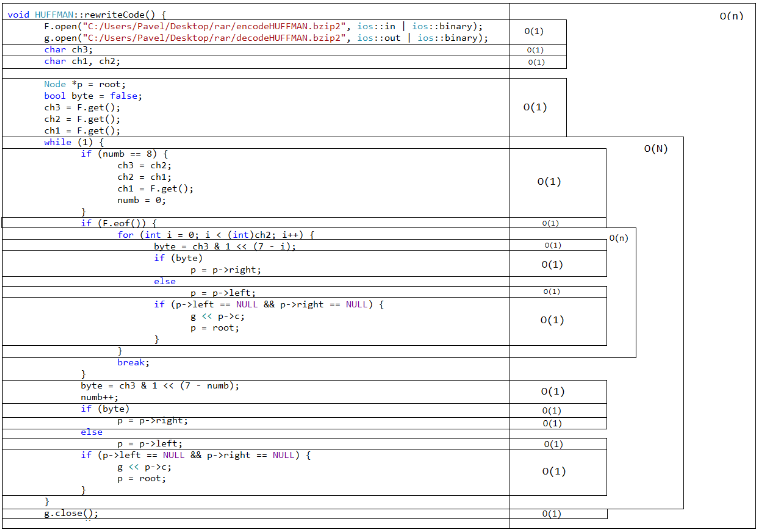
Теоретическая оценка сложности метода BWT::decode - O(N).

****

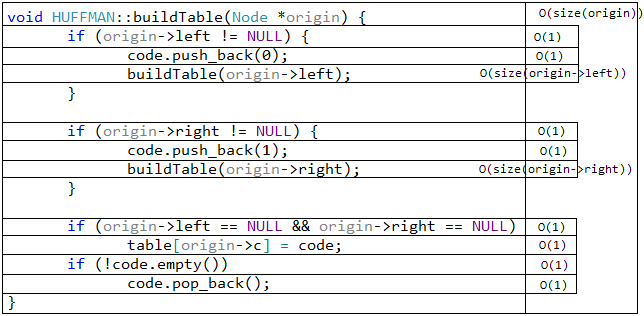
Теоретическая оценка сложности метода MoveToFront::encode - O(N).

****

Теоретическая оценка сложности метода MoveToFront::decode - O(N).

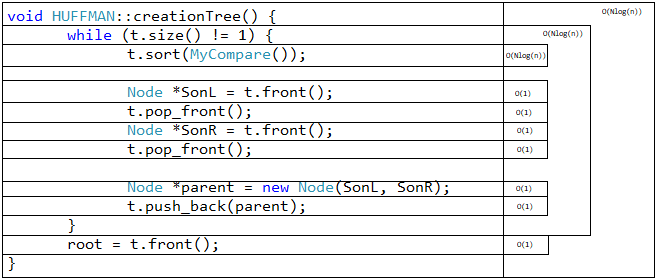
****

Теоретическая оценка сложности метода HUFFMAN::rewriteCode - O(N).

****

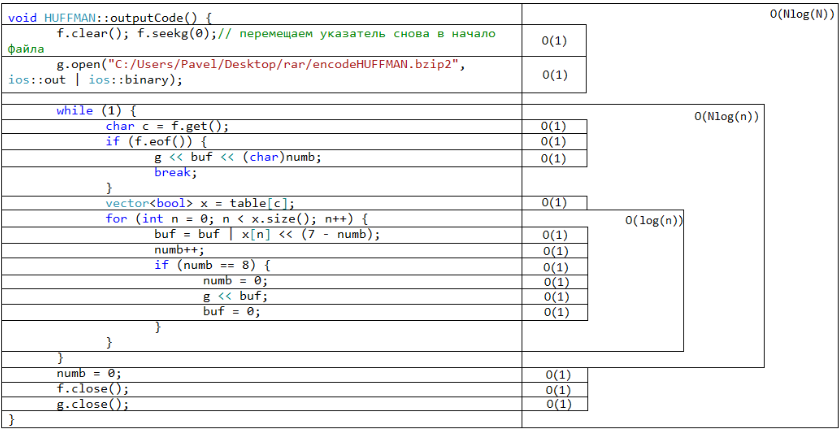
Теоретическая оценка сложности метода

HUFFMAN::buildTable - O(N).

****

Теоретическая оценка сложности метода

HUFFMAN::creationTree - O(NlogN).

****

Теоретическая оценка сложности метода

HUFFMAN::outputCode - O(NlogN).

**Теоретическая оценка сложности программы**

O(NlogN) + O(NlogN) + O(NlogN) + O(N) + O(N) +

O(N) + O(N) + O(N) + O(NlogN) + O(NlogN) = O(NlogN)

## **3.3 Эмпирическое обоснование сложности**

Теоретический расчёт даёт сложность N\*logN выполнения алгоритма. Она соотносится с реальными результатами работы следующим образом:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Размер  данных N, КБ | Время работы  программы, с. | Значение NlogN | коэффициент k | k\*NlogN |
| 8 | 0,203397 | 24 | k = <T>/<NlogN> | 0,010149166 |
| 16 | 0,307826 | 64 | 0,000422882 | 0,027064442 |
| 32 | 0,345727 | 160 |  | 0,067661106 |
| 64 | 0,548365 | 384 |  | 0,162386654 |
| 128 | 0,779472 | 896 |  | 0,378902194 |
| 256 | 1,559577 | 2048 |  | 0,866062157 |
| 512 | 2,89733 | 4608 |  | 1,948639853 |
| 1024 | 4,45319 | 10240 |  | 4,330310784 |
| 2048 | 10,3678 | 22528 |  | 9,526683725 |
| 4096 | 22,8125 | 49152 |  | 20,78549176 |
| 8192 | 38,8634 | 106496 |  | 45,03523215 |
| среднее | 8,3138584 | 19660 |  |  |

Из графика видно, что при выбранном значении поправочного коэффициента реальные результаты работы программы близки с рассчитанными теоретически.

# HashTab — определяем контрольные суммы файла

Хеш — это определенный код, соответствующий определенной данной единицы информации, уникальный просчитанный математически образ конкретного файла. При малейшем изменении файла сразу изменяется и хэш-сумма этого файла. С помощью такой проверки обеспечивается защита конкретного файла от изменения.

Контрольная сумма (хеш) — определенное значение, рассчитанное для данных с помощью известных алгоритмов. Предназначается для проверки целостности данных при передаче. Наиболее распространенными алгоритмами являются: CRC32, MD5 и SHA-1.

CRC32 — (Cyclic redundancy code) Циклический избыточный код. используется в работе программ архиваторов.

MD5 — используется не только для проверки целостности данных, но и позволяет получить довольно надежный идентификатор файла. Последний часто используется при поиске одинаковый файлов на компьютере, чтобы не сравнивать все содержимое, а сравнить только хеш.

SHA-1 — используется для проверки целостности загружаемых данных программой BitTorrent.

Пример. Вы хотите установить Windows или игру. Скачиваете образ и хотите убедиться в целостности файла. Все ли правильно скачалось и ничего ли не потерялось при передаче. Самым простым способом это сделать — сравнить контрольные суммы скачанного файла с теми, которые дает автор или раздающий. Если суммы совпадают, то ошибок в образе нет.

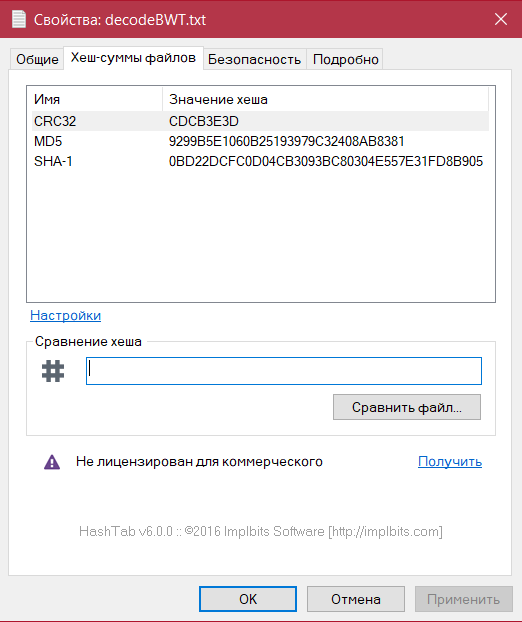
Программа HashTab представляет собой плагин или утилиту, которая интегрируется в операционную систему Windows и при необходимости используется.

Пример работы HashTab:

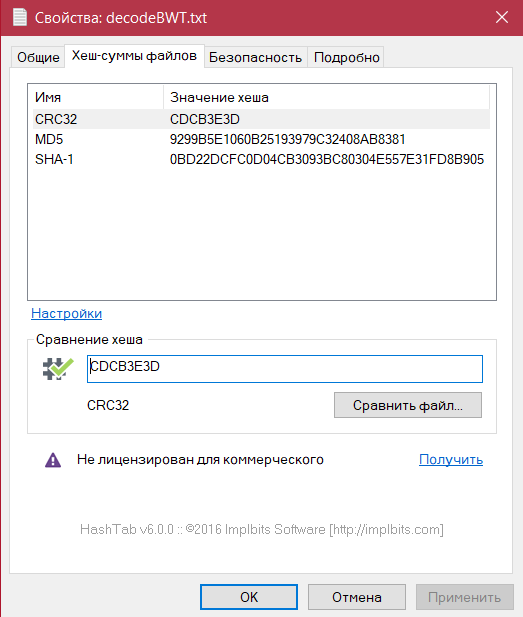
При установке программа HashTab интегрируется в окно свойств Проводника. После установки программы HashTab на ваш компьютер, вы можете проверять хэш-суммы файлов. Для этого кликните по какому-нибудь файлу правой кнопкой мыши.

В контекстном меню выберите пункт «Свойства». После открытия окна, в окне «Свойства» вы увидите новую вкладку «Хеш-суммы файлов».

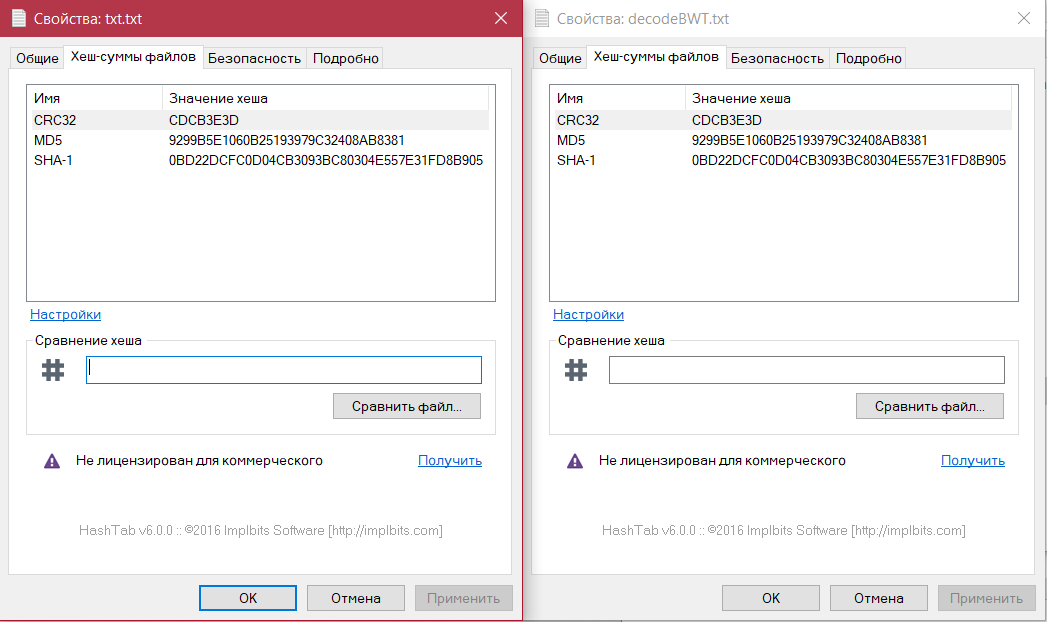
При нажатии на вкладку «Хеш-суммы файлов» появляется окно со значениями контрольных сумм этого файла.



Для сравнения хеш-сумм файлов нужно будет перетянуть файл в поле «Сравнение хеша». Если значения хэша файлов совпадают, то появится зеленый флажок.



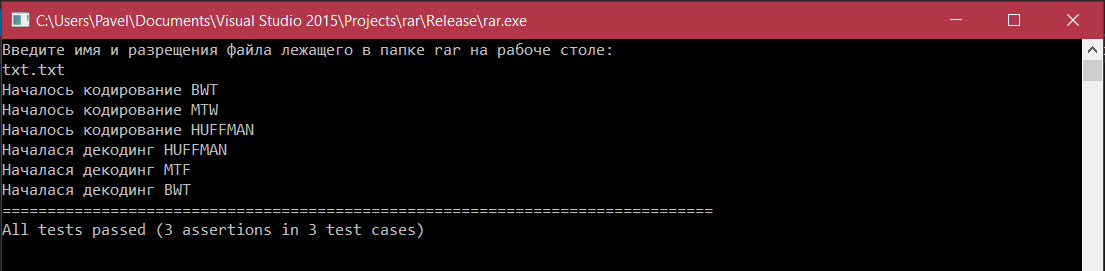
Можно также одновременно проверить два файла поодиночке и сравнить результат в двух окнах. На этом изображении видно, что контрольные суммы двух файлов совпадают (txt.txt – исходный файл, decodeBWT.txt – файл после декодирования зарезервированного файла txt.txt).



Пример сжатия Burrows-Wheeler и 7-Zip:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Расширение файла | txt | jpg | png | mp3 | exe |
| Размер файла | 187KB | 1,03MB | 549KB | 8,84MB | 15MB |
| Burrows-Wheeler | 81,9KB | 1,03MB | 549KB | 8,53MB | 14,9MB |
| 7-Zip | 63,8KB | 1,03MB | 552KB | 8,04MB | 14,9MB |

Также были проведены 3 проверки (трех классов: BWT, MTF, HUFMAN) с помощью библиотеки Catch.



# Вывод

В ходе выполнения работы были рассмотрены такие алгоритмы как MoveToFront, BurrowsWheeler, Huffman. Каждый алгоритм делает крупицу работы во всем проект, что после слияния этих алгоритмов создается приложение, позволяющее производить сжатие и распаковку данных.

В процессе выполнения курсовой работы также было изучена библиотека Catch для тестирования классов MoveToFront, BurrowsWheeler, Huffman, в которой сравнивался файл до кодирования и после декодирования. Также осуществили работу с программой HashTab, которая предназначается для проверки целостности данных. Этой программой были проверенный: исходный файл и файл после декодирования, после чего убедились, что их хеш-суммы равны.

На практике алгоритм сжатия вида BWT → MTF/RLE → Хаффман, применённый в архиваторе bzip2, немного превосходит лучшие реализации LZH по качеству сжатия при аналогичной скорости.

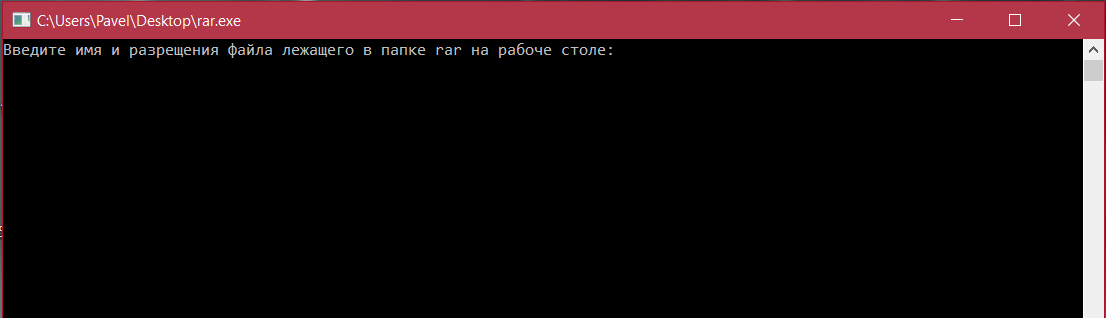
# Используемые ресурсы

1. Алгоритм Хаффмана: [Электронный ресурс] URL: https://neerc.ifmo.ru/wiki/index.php?title=Алгоритм\_Хаффмана (дата обращения: 11.12.2017).
2. Преобразование MTF: [Электронный ресурс] URL: https://neerc.ifmo.ru/wiki/index.php?title=Преобразование\_MTF (дата обращения: 11.12.2017).
3. Преобразование Барроуза-Уилера: [Электронный ресурс] URL: https://neerc.ifmo.ru/wiki/index.php?title=Преобразование\_Барроуза-Уиллера (дата обращения: 11.12.2017).

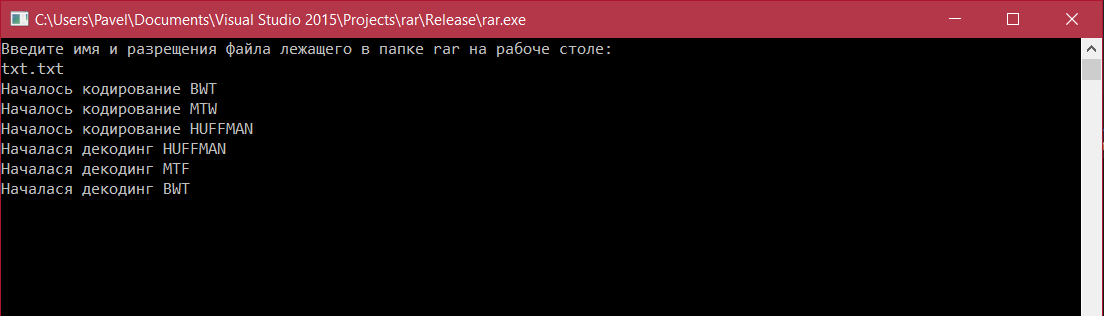
# Приложение 1

# Руководство пользователя

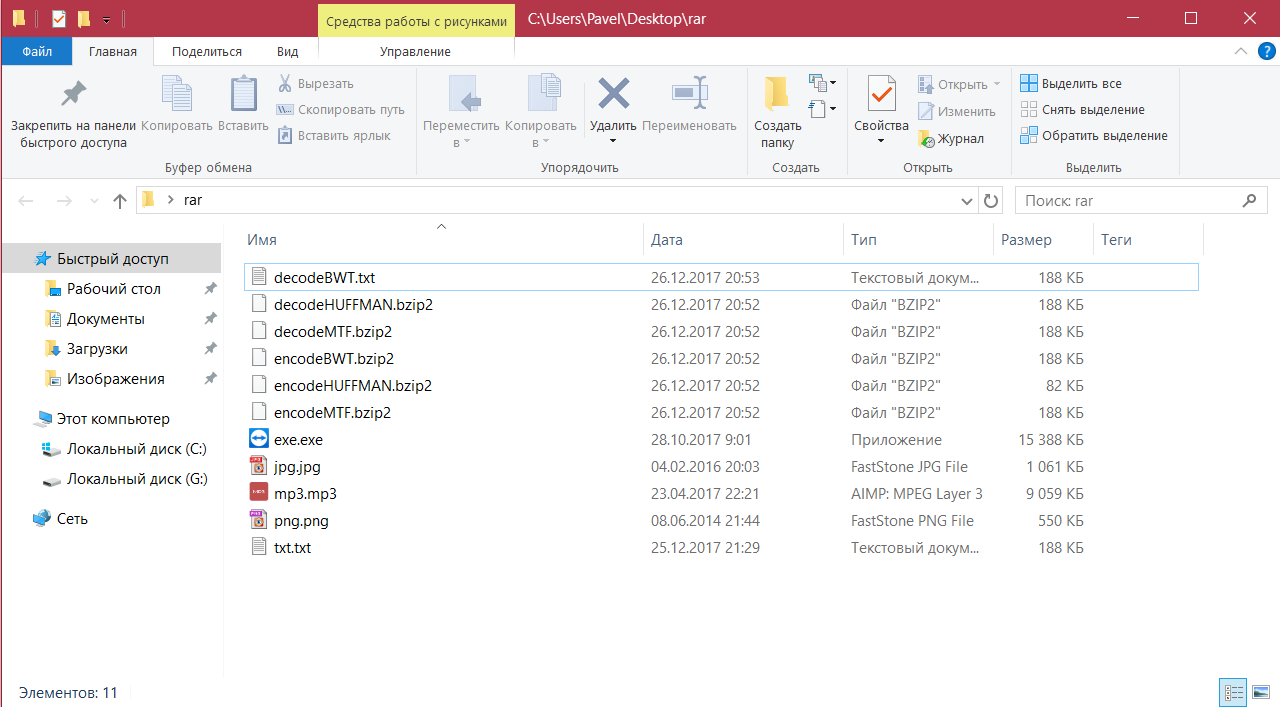
При запуске программы (rar.exe), который находиться на рабочем столе, появляется консольное окно.



В котором пользователь указывает название и расширение файла который они хотя зашифровать. После чего как файл был введен, пользователь увидит сначала поэтапное шифрование после чего поэтапное дешифрование его файла.



Результат проделанной программы можно посмотреть в папке rar, которая находиться на рабочем столе. Где было создано кодирование и декодирование каждого из алгоритмов архиватора. Файл с именем encodeHUFFMAN.bzip2 является закодированным файлом, а decodeBWT.txt - раскодированным.



# Приложение 2

**Исходный код:**

**Класс BWT:**

1. BWT.h

#pragma once

#include <iostream>

#include <string>

#include <cstring>

#include <vector>

#include <fstream>

#include <algorithm>

using namespace std;

typedef unsigned char byte;

class BWT {

int Length;

vector<int> Indexes;

vector<byte> FileContent;

ifstream f;

ofstream g;

void rar();

int SortMatrix2();

void write(int source[]);

public:

BWT();

void encode(string file);

void decode(string extension);

};

1. BWT.cpp

#include <iostream>

#include <string>

#include <cstrin>

#include <vector>

#include <fstream>

#include <algorithm>

#include "BWT.h"

using namespace std;

typedef unsigned char byte;

BWT::BWT() {}

void BWT::rar() {

Indexes.clear();

for (int i = 0; i < Length; i++) {

Indexes.push\_back(i);

}

int source = SortMatrix2();

int mass[4] = {};

for (int i = 3; i > -1; i--) {

if (source) {

mass[i] = source % 10;

source /= 10;

}

}

write(mass);

}

int BWT::SortMatrix2() {

int source = 0;

vector<int> count(Length);

for (int i = 0; i < Length; i++)

count[FileContent[i]]++;

vector<int> from(Length);

for (int i = 1; i < Length; i++)

from[i] = from[i - 1] + count[i - 1];

for (int i = 0; i < Length; i++)

Indexes[from[FileContent[i]]++] = i;

vector<int> groups(Length);

int groupsCount = 1;

groups[Indexes[0]] = 0;

for (int i = 1; i < Length; i++) {

if (FileContent[Indexes[i]] != FileContent[Indexes[i - 1]])

groupsCount++;

groups[Indexes[i]] = groupsCount - 1;

}

vector<int> newIndexes(Length), newGroups(Length);

for (int blockLength = 1; blockLength < Length; blockLength \*= 2) {

for (int i = 0; i < Length; i++)

newIndexes[i] = (Indexes[i] + Length - blockLength) % Length;

for (int i = 0; i < Length; i++)

count[i] = from[i] = 0;

for (int i = 0; i < Length; i++)

count[groups[newIndexes[i]]]++;

for (int i = 1; i < groupsCount; i++)

from[i] = from[i - 1] + count[i - 1];

for (int i = 0; i < Length; i++) {

if (newIndexes[i] == 0)

source = from[groups[newIndexes[i]]];

Indexes[from[groups[newIndexes[i]]]++] = newIndexes[i];

}

groupsCount = 1;

newGroups[Indexes[0]] = 0;

for (int i = 1; i < Length; i++) {

if (groups[Indexes[i]] != groups[Indexes[i - 1]] ||

groups[(Indexes[i] + blockLength) % Length] != groups[(Indexes[i - 1] + blockLength) % Length])

groupsCount++;

newGroups[Indexes[i]] = groupsCount - 1;

}

for (int i = 0; i < Length; i++)

groups[i] = newGroups[i];

}

return source;

}

void BWT::write(int source[]) {

for (int i = 0; i < 4; i++)

g << (char)source[i];

for (int i = 0; i < Length; i++) {

g << FileContent[(Indexes[i] - 1 + Length) % Length];

}

}

void BWT::encode(string file) {

Length = 9999;

f.open("C:/Users/Pavel/Desktop/rar/" + file, ios::out | ios::binary);

g.open("C:/Users/Pavel/Desktop/rar/encodeBWT.bzip2", ios::out | ios::binary);

char ch;

while (1) {

FileContent.clear();

for (int i = 0; i < 9999; i++) {

ch = f.get();

if (f.eof()) {

Length = FileContent.size();

rar();

f.close();

g.close();

return;

}

FileContent.push\_back(ch);

}

rar();

}

}

void BWT::decode(string extension) {

f.open("C:/Users/Pavel/Desktop/rar/decodeMTF.bzip2", ios::out | ios::binary);

g.open("C:/Users/Pavel/Desktop/rar/decodeBWT." + extension, ios::out | ios::binary);

char ch;

int size = 9999;

while (1) {

if (f.eof()) {

g.close();

f.close();

return;

}

vector<byte> bufIn;

int ko = 1000;

int primaryIndex = 0;

for (int i = 3; i > -1; i--) {

primaryIndex += ko \* (int)f.get();

ko /= 10;

}

if (f.eof()) {

g.close();

f.close();

return;

}

for (int i = 0; i < 9999; i++) {

ch = f.get();

if (f.eof()) {

size = bufIn.size();

break;

}

bufIn.push\_back(ch);

}

vector<byte> F(size);

vector<int> buckets(256, 0);

int i, j, k;

vector<int> indices(size);

for (i = 0; i < size; i++)

buckets[bufIn[i]]++;

for (i = 0, k = 0; i < 256; i++)

for (j = 0; j < buckets[i]; j++)

F[k++] = i;

for (i = 0, j = 0; i < 256; i++) {

while (j < size && i > F[j])

j++;

buckets[i] = j;

}

for (i = 0; i < size; i++)

indices[buckets[bufIn[i]]++] = i;

for (i = 0, j = primaryIndex; i < size; i++) {

j = indices[j];

g << bufIn[j];

}

}

g.close();

f.close();

}

**Класс MTF:**

1. MTF.h

#pragma once

#include <list>

#include <iostream>

#include <iterator>

#include <fstream>

#include <sstream>

#include <vector>

typedef unsigned char byte;

using namespace std;

class MoveToFront {

ifstream f;

ofstream g;

byte symbolTable[256];

//выполняем сдвиг в алфавите

void moveToFront(int k);

//заполняем алфавит

void fillSymbolTable();

public:

MoveToFront();

void encode();

void decode();

};

1. MTF.cpp

#include <list>

#include <iostream>

#include <iterator>

#include <fstream>

#include <sstream>

#include <vector>

#include "MTF.h"

typedef unsigned char byte;

using namespace std;

byte symbolTable[256];

MoveToFront::MoveToFront() {}

//выполняем сдвиг в алфавите

void MoveToFront::moveToFront(int k) {

byte t = symbolTable[k];

for (int i = k - 1; i >= 0; i--)

symbolTable[i + 1] = symbolTable[i];

symbolTable[0] = t;

}

//заполняем алфавит

void MoveToFront::fillSymbolTable() {

for (int i = 0; i < 256; i++)

symbolTable[i] = i;

}

void MoveToFront::encode() {

f.open("C:/Users/Pavel/Desktop/rar/encodeBWT.bzip2", std::ios::out | std::ios::binary);

g.open("C:/Users/Pavel/Desktop/rar/encodeMTF.bzip2", std::ios::out | std::ios::binary);

fillSymbolTable();

while (1) {

byte c = f.get();

if (f.eof()) {

f.close();

g.close();

return;

}

for (int i = 0; i < 256; i++) {

if (c == symbolTable[i]) {

g << (byte)i;

moveToFront(i);

break;

}

}

}

}

void MoveToFront::decode() {

f.open("C:/Users/Pavel/Desktop/rar/decodeHUFFMAN.bzip2", std::ios::out | std::ios::binary);

g.open("C:/Users/Pavel/Desktop/rar/decodeMTF.bzip2", std::ios::out | std::ios::binary);

fillSymbolTable();

while (1) {

byte c = f.get();

if (f.eof()) {

f.close();

g.close();

return;

}

g << symbolTable[c];

moveToFront(c);

}

}

**Класс HUFFMAN:**

1. HUFFMAN.h

#pragma once

#include <iostream>

#include <vector>

#include <map>

#include <list>

#include <fstream>

using namespace std;

class HUFFMAN {

ifstream f, F;

ofstream g;

vector<bool> code;

map<char, vector<bool> > table;

map<char, int> m;

int numb; char buf;

struct Node {

public:

int a;

char c;

Node \*left, \*right;

Node();

Node(Node \*L, Node \*R);

};

list<Node\*> t;

Node \*root;

struct MyCompare;

void buildTable(Node \*origin);

void printTree(Node\* origin, unsigned k);

public:

HUFFMAN();

void numberCharacters();

void recNode();

void creationTree();

void Table() { buildTable(root); }

void Tree() { printTree(root, 0); }

void printBuildTable();

void outputCode();

void rewriteCode();

};

1. HUFFMAN.cpp

#include <iostream>

#include <vector>

#include <map>

#include <list>

#include <fstream>

#include "HUFFMAN.h"

using namespace std;

HUFFMAN::Node::Node() { left = right = NULL; }

HUFFMAN::Node::Node(Node \*L, Node \*R) {

left = L;

right = R;

a = L->a + R->a;

c = NULL;

}

struct HUFFMAN::MyCompare {

bool operator()(const Node\* l, const Node\* r) const { return l->a < r->a; }

};

void HUFFMAN::buildTable(Node \*origin) {

if (origin->left != NULL) {

code.push\_back(0);

buildTable(origin->left);

}

if (origin->right != NULL) {

code.push\_back(1);

buildTable(origin->right);

}

if (origin->left == NULL && origin->right == NULL)

table[origin->c] = code;

if (!code.empty())

code.pop\_back();

}

void HUFFMAN::printTree(Node\* origin, unsigned k = 0) {

if (origin != NULL) {

printTree(origin->left, k + 3);

for (unsigned i = 0; i < k; i++) {

cout << " ";

}

if (origin->c) {

cout << origin->a << " (" << origin->c << ")" << endl;

}

else {

cout << origin->a << endl;

}

printTree(origin->right, k + 3);

}

}

HUFFMAN::HUFFMAN() {

m = {};

numb = 0, buf = 0;

}

void HUFFMAN::numberCharacters() {

f.open("C:/Users/Pavel/Desktop/rar/encodeMTF.bzip2", ios::out | ios::binary);

while (1)

{

char c = f.get();

if (f.eof())

break;

m[c]++;

}

}

void HUFFMAN::recNode() {

for (map<char, int>::iterator itr = m.begin(); itr != m.end(); ++itr)

{

Node \*p = new Node;

p->c = itr->first;

p->a = itr->second;

t.push\_back(p);

}

}

void HUFFMAN::creationTree() {

while (t.size() != 1) {

t.sort(MyCompare());

Node \*SonL = t.front();

t.pop\_front();

Node \*SonR = t.front();

t.pop\_front();

Node \*parent = new Node(SonL, SonR);

t.push\_back(parent);

}

root = t.front(); //root - указатель на вершину дерева

}

void HUFFMAN::printBuildTable() {

for (auto it = table.begin(); it != table.end(); it++) {

cout << it->first << ":";

for (auto iter = it->second.begin(); iter != it->second.end(); iter++) {

cout << \*iter;

}

cout << endl;

}

}

void HUFFMAN::outputCode() {

f.clear(); f.seekg(0);// перемещаем указатель снова в начало файла

g.open("C:/Users/Pavel/Desktop/rar/encodeHUFFMAN.bzip2", ios::out | ios::binary);

while (1) {

char c = f.get();

if (f.eof()) {

g << buf << (char)numb;

break;

}

vector<bool> x = table[c];

for (int n = 0; n < x.size(); n++) {

buf = buf | x[n] << (7 - numb);

numb++;

if (numb == 8) {

numb = 0;

g << buf;

buf = 0;

}

}

}

numb = 0;

f.close();

g.close();

}

void HUFFMAN::rewriteCode() {

F.open("C:/Users/Pavel/Desktop/rar/encodeHUFFMAN.bzip2", ios::in | ios::binary);

g.open("C:/Users/Pavel/Desktop/rar/decodeHUFFMAN.bzip2", ios::out | ios::binary);

char ch3;

char ch1, ch2;

Node \*p = root;

bool byte = false;

ch3 = F.get();

ch2 = F.get();

ch1 = F.get();

while (1) {

if (numb == 8) {

ch3 = ch2;

ch2 = ch1;

ch1 = F.get();

numb = 0;

}

if (F.eof()) {

for (int i = 0; i < (int)ch2; i++) {

byte = ch3 & 1 << (7 - i);

if (byte)

p = p->right;

else

p = p->left;

if (p->left == NULL && p->right == NULL) {

g << p->c;

p = root;

}

}

break;

}

byte = ch3 & 1 << (7 - numb);

numb++;

if (byte)

p = p->right;

else

p = p->left;

if (p->left == NULL && p->right == NULL) {

g << p->c;

p = root;

}

}

g.close();

F.close();

}