МИНЕСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧЕРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«Брестский государственный технический университет»

Кафедра «Интеллектуальные информационные технологии»

Лабораторная работа №2

По дисциплине «Основы информационных технологий»

За 5 семестр

Тема: «Построение кода сжатия информации»

Выполнила:

студентка 3 курса

группы АС-56

Карпенко М.В.

Проверил:

Пролиско Е.Е.

Брест 2021

***Задание***

На основе программы написанной для частотных таблиц, написать

программу построения кодов сжатия информации используя метод ШеннонаФано и метод Хаффмена. Имя результирующего файла аналогичен имени

исходного файла с дополнительным расширением .haf если кодировка

проводилась методом Хаффменна, и .s-f если кодировка проводилась методом

Шеннона-Фэно. Например, обрабатывая файл text01.doc методом Хаффмена

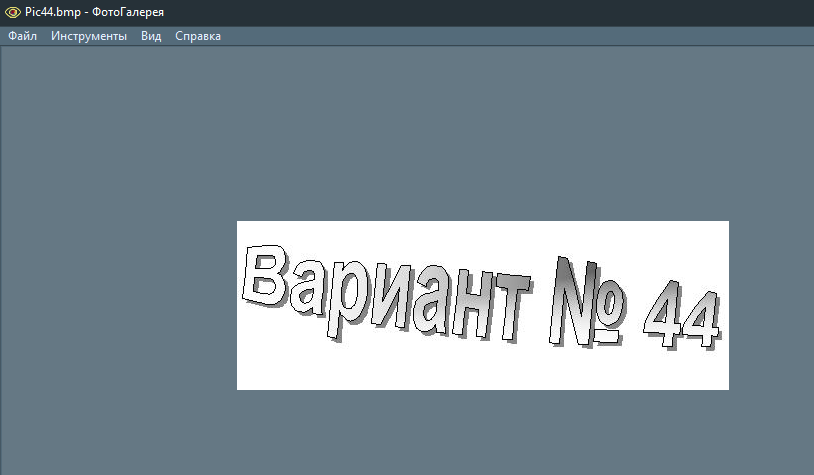
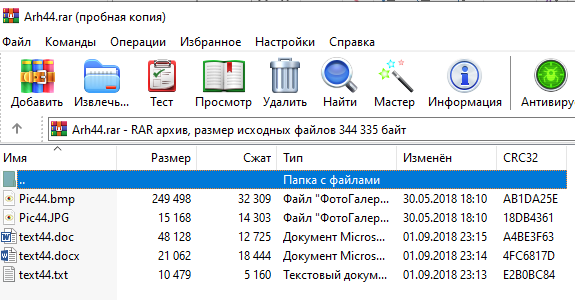
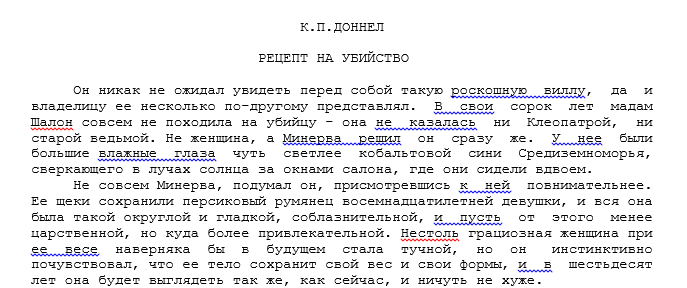
получим файл c кодовой таблицей с именем text01.doc.haf, если же этот файл

обработан методом Шеннона-Фaно, то в результате получим файл с именем

text01.doc.s-f.

***Вариант 44***

*Исходные файлы:*

*Код программы:*

#include <iostream>

#include <fstream>

#include <string>

#include <algorithm>

#include <vector>

#include <math.h>

#include<queue>

using namespace std;

//узел дерева

struct ByteStruct

{

int byte; // значение узла

int repeats; // Повторения

ByteStruct\* lNode, \* rNode; // указатель на левый и правый лист

ByteStruct() : byte(0), repeats(0), lNode(nullptr), rNode(nullptr) { } // Конструктор

ByteStruct(int Byte, int Repeats) : byte(Byte), repeats(Repeats), lNode(nullptr), rNode(nullptr) { } //констуктор

};

//сравнение байтов

struct ByteCompare

{

bool operator()(ByteStruct l, ByteStruct r)

{

return (l.repeats > r.repeats);

}

bool operator()(ByteStruct\* l, ByteStruct\* r)

{

return (l->repeats > r->repeats);

}

};

class ftable

{

protected:

string fileName; // Имя файла

double entropy()

{

double freqOfByte[256], entropy = 0;

fill(freqOfByte, freqOfByte + 256, 0); //заполнение массива нулями

for (int i = 0; i < table.size(); ++i)

{

freqOfByte[i] = (double)table[i].repeats / Symbols; // Считаем частоту

entropy += (log10(1.0 / freqOfByte[i]) / log10(2.0)) \* freqOfByte[i]; // Считаем энтропию по формуле Шеннона

}

Entropy = entropy;

return entropy;

}

void length()

{

int repeats = 0;

for ( vector<ByteStruct>::const\_iterator byteIterator = table.begin(); byteIterator != table.end(); byteIterator++)

{

repeats += byteIterator->repeats; // Все повторения

}

Repeats = repeats;

}

void sortAndDeleteZeros()

{

sort(table.begin(), table.end(), ByteCompare()); // Сортируем

// Удаляем нулевые повторения

for ( vector<ByteStruct>::const\_iterator byteIterator = table.begin(); byteIterator != table.end(); )

{

if ((\*byteIterator).repeats == 0)

{

byteIterator = table.erase(byteIterator);

}

else

{

++byteIterator;

}

}

}

public:

double Entropy = 0.0; // Энтропия

int Repeats = 0; // Повторения

int Symbols = 0; // Символов в файле

ftable() : fileName("") { } //констуктор

ftable( string \_fileName) : fileName(\_fileName) { }

void newFile( string \_fileName)

{

fileName = \_fileName;

}

vector<ByteStruct> table; // Таблица повторений

void buildTable()

{

int symbolCount = 0;

if (fileName == "")

{

return;

}

table.erase(table.begin(), table.end()); // Очищаем таблицу

ifstream file;

file.open(fileName, ifstream::binary); // Открываем файл в бинарном режиме

if (file.is\_open())

{

file.seekg(0, file.end);

symbolCount = file.tellg(); // Считаем количество символов в файле

Symbols = symbolCount;

file.seekg(0, file.beg);

}

unsigned char\* data; // Содержимое файла

data = new unsigned char[symbolCount]; // Выделяем память

file.read((char\*)data, symbolCount); // Заполняем массив

file.close();

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

{

table.push\_back(ByteStruct(i, 0)); // Даем значения байтам

}

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

{

table[data[i]].repeats += 1; // Считаем повторения байтов, номер байта сразу является индеком массива

}

sortAndDeleteZeros();

entropy();

length();

delete[] data; // Освобождаем память

}

void writeToFile( string FileName)

{

cout << "Write Table of Freqs of " + fileName << endl;

newFile(FileName);

buildTable();

ofstream outputFile(fileName + ".tab"); // Создаем выходной файл

outputFile << Entropy << endl;

for (int i = 0; i < table.size(); i++)

{

outputFile << "\n" << table[i].byte << "\t" << table[i].repeats; // Вывод частот

}

outputFile.close(); // Закрываем файл

}

~ftable() { }

};

class sfano

{

protected:

string fileName;

ftable FTable;

struct SfanoCode

{

ByteStruct byte; // Байт

string code; // Код

SfanoCode(ByteStruct Byte, string Code) : byte(Byte), code(Code) { }

SfanoCode() : byte(ByteStruct(0, 0)), code("") { }

};

//построение дерева

void saveCode( vector<SfanoCode>& sfCodes, int left, int right)

{

if (left == right)

return;

int leftPosition = left; // Начало списка

int rightPosition = right; // Конец списка

int leftSum = 0;

int rightSum = 0;

// Ищем середину

while (leftPosition < rightPosition)

{

if (leftSum <= rightSum)

{

leftSum += sfCode[leftPosition].byte.repeats;

leftPosition++;

}

else

{

rightSum += sfCode[rightPosition].byte.repeats;

rightPosition--;

}

}

// Подсчитываем коды

for (int leftPointer = left; leftPointer < leftPosition; leftPointer++)

{

sfCode[leftPointer].code += "1";

}

for (int rightPointer = leftPosition; rightPointer <= right; rightPointer++)

{

sfCode[rightPointer].code += "0";

}

// Идем влево

saveCode(sfCode, left, leftPosition - 1);

// Идем вправо

saveCode(sfCode, leftPosition, right);

}

double length()

{

double length = 0;

int allRepeats = FTable.Repeats;

for (int i = 0; i < sfCode.size(); i++)

{

length += sfCode[i].byte.repeats \* sfCode[i].code.length(); // Ищем общую длину

}

return length / allRepeats;

}

public:

sfano() { }

sfano( string FileName)

{

fileName = FileName;

FTable = ftable(fileName);

FTable.buildTable();

}

vector<SfanoCode> sfCode;

void buildSfanoTable()

{

sfCode.erase(sfCode.begin(), sfCode.end());

for (int i = 0; i < FTable.table.size(); ++i)

{

sfCode.push\_back(SfanoCode(ByteStruct(FTable.table[i].byte, FTable.table[i].repeats), "")); // Добавляем байты в таблицу

}

saveCode(sfCode, 0, FTable.table.size() - 1);

}

void newFile( string FileName)

{

fileName = FileName;

FTable = ftable(fileName);

FTable.buildTable();

}

void writeToFile( string FileName)

{

cout << "Write SF of " + fileName << endl;

newFile(FileName);

buildSfanoTable();

ofstream outputFile(fileName + ".s-f"); // Создаем выходной файл

outputFile << FTable.Entropy << '\n' << length() << endl;

for (int i = sfCode.size() - 1; i >= 0; i--)

{

outputFile << "\n" << sfCode[i].byte.byte << "\t" << sfCode[i].code;

}

outputFile.close(); // Закрываем файл

}

~sfano() { }

};

class huff

{

protected:

string fileName;

ftable FTable;

void saveCode(ByteStruct\* koren, string huffcod)

{

if (!koren)

{

return;

}

if (koren->byte != 256)

{

codes.push\_back(HuffmanCode(\*koren, huffcod));

}

saveCode(koren->lNode, huffcod + "0");

saveCode(koren->rNode, huffcod + "1");

}

double length()

{

double length = 0;

int allRepeats = FTable.Repeats;

for (int i = 0; i < codes.size(); i++)

{

length += codes[i].byte.repeats \* codes[i].code.length();

}

return length / allRepeats;

}

public:

huff() { }

huff( string FileName)

{

fileName = FileName;

FTable = ftable(fileName);

FTable.buildTable();

}

struct HuffmanCode

{

ByteStruct byte;

string code;

HuffmanCode(ByteStruct Byte, string Code) : byte(Byte), code(Code) { }

HuffmanCode() : byte(ByteStruct(0, 0)), code("") { }

};

vector<HuffmanCode> codes;

void buildHuffTable()

{

ByteStruct\* left, \* right, \* top;

priority\_queue<ByteStruct\*, vector<ByteStruct\*>, ByteCompare> huffDerevo; // Создаем дерево

for (int i = 0; i < FTable.table.size(); ++i)

huffDerevo.push(new ByteStruct(FTable.table[i].byte, FTable.table[i].repeats)); // Заполняем дерево

while (huffDerevo.size() != 1)

{

left = huffDerevo.top(); // Достаем левый лист

huffDerevo.pop();

right = huffDerevo.top(); // Достаем правый лист

huffDerevo.pop();

top = new ByteStruct(256, left->repeats + right->repeats);

top->lNode = left;

top->rNode = right;

huffDerevo.push(top); // Добавляем общий

}

codes.erase(codes.begin(), codes.end());

saveCode(huffDerevo.top(), "");

}

void newFile( string FileName)

{

fileName = FileName;

FTable = ftable(fileName);

FTable.buildTable();

}

void writeToFile( string FileName)

{

cout << "Write HAF of " + fileName << endl;

newFile(FileName);

buildHuffTable();

ofstream outputFile(fileName + ".haf"); // Создаем выходной файл

outputFile << FTable.Entropy << '\n' << length() << endl;

for (int i = 0; i < codes.size(); i++)

{

outputFile << "\n" << codes[i].byte.byte << "\t" << codes[i].code;

}

outputFile.close(); // Закрываем файл

}

~huff() { }

};

int main()

{

string F[6] = { "Arh44.rar", "Pic44.bmp", "Pic44.JPG", "text44.doc", "text44.docx", "text44.txt" };

ftable fr; //создаем объект класса

for (int i = 0; i < 6; i++) //передаем объекту файлы

{

fr.writeToFile(F[i]);

}

huff haf;

for (int i = 0; i < 6; i++)

{

haf.writeToFile(F[i]);

}

sfano sfn;

for (int i = 0; i < 6; i++)

{

sfn.writeToFile(F[i]);

}

return 0;

}

*Результат работы программы:*

