**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

Курсовая РАБОТА

**по дисциплине «Алгоритмы и структуры данных»**

Тема: Статическое кодирование и декодирование текстового файла методами Хаффмана и Фано-Шеннона – текущий контроль

|  |  |  |
| --- | --- | --- |
| Студентка гр. 6383 |  | Лавренкова Е.Л. |
| Преподаватель |  | Шолохова О. М. |

Санкт-Петербург

2017

**содержание**

|  |  |  |
| --- | --- | --- |
| 1. | Формулировка задания | 3 |
| 2. | Формальная постановка задачи | 3 |
| 3. | Теоретический материал | 3 |
| 4. | Описание алгоритма | 4 |
| 5. | Спецификация программы | 4 |
|  | 5.1. Описание данных | 5 |
|  | 5.2. Описание функций | 5 |
|  | 5.3. Интерфейс программы | 6 |
| 6. | Тестирование | 7 |
| 7. | Выводы | 9 |
|  | Приложение А. Исходный код | 10 |

1. **Формулировка задания.**

Статическое кодирование и декодирование текстового файла методами Хаффмана и Фано-Шеннона. Текущий контроль – генерация заданий и ответов в удобной форме.

1. **Формальная постановка задачи.**

На вход даётся файл с исходным сообщением. Требуется сгенерировать различные варианты заданий для проверки текущего контроля с ответами. Алгоритм пошагово выводится как в консоль, так и в результирующий файл.

1. **Теоретический материал.**

Сжатие информации - проблема, имеющая достаточно давнюю историю, гораздо более давнюю, нежели история развития вычислительной техники, которая (история) обычно шла параллельно с историей развития проблемы кодирования и шифровки информации.

Все алгоритмы сжатия оперируют входным потоком информации, минимальной единицей которой является бит, а максимальной - несколько бит, байт или несколько байт.

Целью процесса сжатия, как правило, является получение более компактного выходного потока информационных единиц из некоторого изначально некомпактного входного потока при помощи некоторого их преобразования.

Основными техническими характеристиками процессов сжатия и результатов их работы являются:

* степень сжатия (compress rating) или отношение (ratio) объемов исходного и результирующего потоков;
* скорость сжатия - время, затрачиваемое на сжатие некоторого объема информации входного потока, до получения из него эквивалентного выходного потока;
* качество сжатия - величина, показывающая на сколько сильно упакован выходной поток, при помощи применения к нему повторного сжатия по этому же или иному алгоритму.

Кодирования Хаффмана и Фано-Шеннона являются простыми алгоритмами для построения кодов переменной длины, имеющих минимальную среднюю длину. Эти весьма популярные алгоритмы служат основой многих компьютерных программ сжатия текстовой и графической информации. Некоторые из них используют непосредственно алгоритм Хаффмана, а другие берут его в качестве одной из ступеней многоуровневого процесса сжатия. Метод Хаффмана производит идеальное сжатие (то есть, сжимает данные до их энтропии).

1. **Описание алгоритма.**

1) Алгоритм чтения исходного текста

1. Входной файл (input.txt) считывается построчно, пока поток чтения «не упрётся» в конец файла.

2. Символы переноса заменяются символом «\_» для удобства вывода последующих алгоритмов кодирования и деревьев в консоль.

**Примечание:** после декодирования, символы переноса возвращается на свои места.

2) Метод Хаффмана:

1. На вход приходят упорядоченные по убыванию частот данные.

2. Выбираются две наименьших по частоте буквы алфавита, и создается родитель (сумма двух частот этих «листков»).

3. Потомки удаляются и вместо них записывается родитель, «ветви» родителя нумеруются: левой ветви ставится в соответствие «1», правой «0».

4. Шаг два повторяется до тех пор, пока не будет найден главный родитель — «корень».

3) Метод Фано-Шеннона:

1. На вход приходят упорядоченные по не возрастанию частот данные.

2. Находится середина, которая делит алфавит примерно на две части. Эти части (суммы частот алфавита) примерно равны. Для левой части присваивается «1», для правой «0», таким образом мы получим листья дерева.

3. Шаг 2 повторяется до тех пор, пока мы не получим единственный элемент последовательности, т.е. листок.

1. **Спецификация программы.**

Программа написана на языке С++ с использованием компилятора Miscosoft Visual Studio 2015.

* 1. **Структура данных.**

1. Бинарное дерево для метода Хаффмана, реализованное с помощью шаблонных функций.

namespace bin\_tree {

template <class T> struct b\_node {

b\_node()

{

l\_br = r\_br = 0;

};

T data;

b\_node<T> \* l\_br;

b\_node<T> \* r\_br;

};

...

};

1. Бинарное дерево для метода Фано-Шеннона.

struct f\_node {

int left;

int right;

int priority;

string symbol;

f\_node(int a, int b)

{

left = a;

right = b;

}

f\_node()

{

left = 0;

right = 0;

}

};

1. Очередь с приоритетами, реализованная в виде шаблонного класса .

template <class T> class priority\_queue {

public:

struct queue\_cell

{

T data;

\_\_int64 priority;

};

queue\_cell \* head;

long size, top;

...

};

* 1. **Описание функций.**

void hman(string my\_str); //Метод Хаффмана

my\_str – строка для кодирования

void fano(string my\_str); //Метод Шеннона-Фано

my\_str – строка для кодирования

void print(b\_node<node> \* root, unsigned k); //Вывод дерева для метода Хаффмана

root – указатель на бин. дерево

k – глубина рекурсии

void print2(b\_node<f\_node> \* root, f\_node\* mas, int \*priority, string \*symbol, int k); //Вывод дерева для метода Шеннона-Фано

root – указатель на бин. дерево

mas – указатель на массив символов

priority – приоритет символа

symbol – текущий символ

k – глубина рекурсии

void BuildTable(b\_node<node> \*root, map<string, string> &table, ofstream &fout); //Построение дерева для метода Хаффмана

root – указатель на бин. Дерево

table – ссылка на ассоциативный массив

fout – ссылка на поток вывода в файл

void BuildTable2(b\_node<f\_node> \*root, map<string, string> &table, ofstream &fout); //Построение дерева для метода Шеннона-Фано

root – указатель на бин. Дерево

table – ссылка на ассоциативный массив

fout – ссылка на поток вывода в файл

int Share\_to\_Intervals(int L, int R, int p[], bool build); //Распределение по интервалам

L – левая граница

R – правая граница

void Fano\_Shennon(int left, int right, b\_node<f\_node> \*root, int prior[], string \*symbol, map<char, int> &dictionary, bool build); //Кодирование по дереву

left – левая граница

right – правая граница

symbol – текущий символ

dictionary – ссылка на ассоциативный массив

void Sort\_Symb\_Prior(string symbol[], int priority[], map<char, int> &dictionary); //Сортировка символов по частоте

symbol – текущий символ

priority – приоритет символа

dictionary – ссылка на ассоциативный массив

string Final\_Code(string str, map<string, string> &table); //Создание закодированного сообщения

str – строка для кодирования

table – ссылка на ассоциативный массив

string One\_Letter(string str, map<char, int> &dictionary, ofstream &fout); //Если в сообщении встречается всего один символ

string hman\_decode(b\_node<node> \* b, string encoded\_str); //Декодирование по методу Хаффмана

root – указатель на бин. Дерево

encoded\_str – строка для декодирования

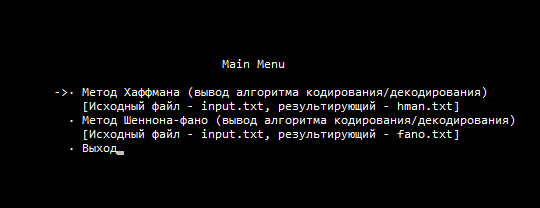
string fano\_decode(b\_node<f\_node> \* b, string encoded\_str); //Декодирование по методу Шеннона-Фано

root – указатель на бин. Дерево

encoded\_str – строка для декодирования

* 1. **Интерфейс программы.**

Реализовано простое консольное меню:



*Рис 1. Главное меню программы.*

Управление:

1. Стрелка «Вверх» - перейти к верхнему пункту меню.
2. Стрелка «Вниз» - перейти к нижнему пункту меню.
3. Кнопка «Enter» - выбрать текущий пункт.
4. **Тестирование**

Метод Хаффмана:

|  |  |  |  |
| --- | --- | --- | --- |
| Исходное сообщение | Закодированное сообщение | Коды символов | Декодированное сообщение |
| (пустой файл) | — | — | — |
| (отсутствие файла) | — | — | — |
| мама мыла раму | 100110011101000111100111011110110000 | 'р' = 1111  'л' = 1110  ' ' = 110  'м' = 10  'а' = 01  'ы' = 001  'у' = 000 | мама мыла раму |
| please give me 5 | 11001010010011110101111100010010000  11111011011110010 | ' ' = 111  's' = 1101  'p' = 1100  'm' = 1011  'l' = 1010  'i' = 1001  'g' = 1000  'e' = 01  'a' = 0011  '5' = 0010  'v' = 000 | please give me 5 |
| 1 once единожды! | 1011001111111101100110100101000000  11001111000010100010011010 | 'o' = 1111  'n' = 1110  'e' = 1101  'c' = 1100  '1' = 1011  '!' = 1010  'ы' = 1001  'о' = 1000  'н' = 0111  'и' = 0110  'ж' = 0101  'е' = 0100  ' ' = 001  'д' = 000 | 1 once единожды! |

Метод Фано-Шеннона:

|  |  |  |  |
| --- | --- | --- | --- |
| Исходное сообщение | Закодированное сообщение | Коды символов | Декодированное сообщение |
| (пустой файл) | — | — | — |
| (отсутствие файла) | — | — | — |
| мама мыла раму | 10010011011011111110001101111001011110 | 'ы' = 11111  'у' = 11110  'р' = 1110  ' ' = 1101  'л' = 1100  'м' = 10  'а' = 0 | мама мыла раму |
| please give me 5 | 11011011001000111000011100110101111  00011110000011010 | 'v' = 1111  's' = 1110  'p' = 1101  'm' = 1100  'l' = 1011  'i' = 1010  'g' = 1001  'a' = 1000  ' ' = 011  '5' = 010  'e' = 00 | please give me 5 |
| 1 once единожды! | 1011001111111101100110100101000000  11001111000010100010011010 | 'o' = 1111  'n' = 1110  'e' = 1101  'c' = 1100  '1' = 1011  '!' = 1010  'ы' = 1001  'о' = 1000  'н' = 0111  'и' = 0110  'ж' = 0101  'е' = 0100  ' ' = 001  'д' = 000 | 1 once единожды! |

1. **Выводы.**

В ходе выполнения данной курсовой работы была изучена теория по основным понятиям и приёмам кодирования/декодирования информации алгоритмами Хаффмана и Фано-Шеннона, а также написана программа, выполняющая требуемые действия по кодированию информации. Были закреплены навыки работы с классами, со структурами данных: бинарное дерево, очередь с приоритетами, которые были реализованы на языке С++.

**приложение А**

**Исходный код**

1. ***Файл Queue.h***

#pragma once

#define ADD\_MEM 256

template <class T> class priority\_queue

{

public:

struct queue\_cell

{

T data;

\_\_int64 priority;

};

queue\_cell \* head;

long size, top;

void alloc\_mem(long count, bool is\_realloc)

{

if (!is\_realloc)

{

head = (queue\_cell \*)malloc(sizeof(queue\_cell)\* count);

size = count;

}

else

{

size = size + count;

head = (queue\_cell \*)realloc(head, sizeof(queue\_cell)\* size);

}

}

void swap(long i1, long i2)

{

queue\_cell tmp = head[i1];

head[i1] = head[i2];

head[i2] = tmp;

}

public:

priority\_queue()

{

alloc\_mem(ADD\_MEM, false);

top = -1;

}

template<class T> void push(T data0, \_\_int64 cur\_priority)

{

if (top + 1 >= size)

alloc\_mem(ADD\_MEM, true);

queue\_cell cur\_cell;

cur\_cell.data = data0;

cur\_cell.priority = cur\_priority;

head[++top] = cur\_cell;

for (long i = top - 1; i >= 0; i--)

{

if (head[i].priority <= cur\_priority)

swap(i, i + 1);

else

break;

}

}

template<class T> T pop(\_\_int64 & pr)

{

pr = head[top].priority;

return head[top--].data;

}

long get\_top()

{

return top;

}

~priority\_queue()

{

free(head);

}

};

1. ***Файл B\_tree.h***

#pragma once

namespace bin\_tree

{

template <class T> struct b\_node

{

b\_node()

{

l\_br = r\_br = 0;

};

T data;

b\_node<T> \* l\_br;

b\_node<T> \* r\_br;

};

template <class T> b\_node<T> \* create\_bt()

{

return 0;

}

template <class T> bool is\_null\_bt(b\_node<T> \* bt)

{

bool res = false;

if (bt == 0x0)

res = true;

return res;

}

template <class T> b\_node<T> \* cons\_bt(b\_node<T> \* bt1, b\_node<T> \* bt2, T root)

{

if (!(is\_null\_bt<T>(bt1) && is\_null\_bt<T>(bt2)))

{

b\_node<T> \* new\_bt = new b\_node<T>();

new\_bt->l\_br = bt1;

new\_bt->r\_br = bt2;

new\_bt->data = root;

return new\_bt;

}

else

return 0;

}

template <class T> void destroy\_bt(b\_node<T> \* bt)

{

if (!is\_null\_bt(bt))

{

destroy\_bt(get\_l\_br(bt));

destroy\_bt(get\_r\_br(bt));

delete bt;

}

}

template <class T> b\_node<T> \* make\_root(T t\_data)

{

b\_node<T> \* new\_bt = new b\_node<T>();

new\_bt->data = t\_data;

new\_bt->l\_br = 0;

new\_bt->r\_br = 0;

return new\_bt;

}

};

1. ***Файл Process.h***

#pragma once

#include <string>

#include "B\_tree.h"

#include "Queue.h"

using namespace bin\_tree;

using namespace std;

namespace proc {

class My {

private:

struct node; //Дерево для Хаффмана

struct f\_node; //Дерево для Шеннона-Фано

public:

void hman(string my\_str); //Метод Хаффмана

void fano(string my\_str); //Метод Шеннона-Фано

void print(b\_node<node> \* root, unsigned k, ofstream &fout); //Вывод дерева для метода Хаффмана

void print2(b\_node<f\_node> \* root, f\_node\* mas, int \*priority, string \*symbol, int k, ofstream &fout); //Вывод дерева для метода Шеннона-Фано

void BuildTable(b\_node<node> \*root, map<string, string> &table, ofstream &fout); //Построение дерева для метода Хаффмана

void BuildTable2(b\_node<f\_node> \*root, map<string, string> &table, ofstream &fout); //Построение дерева для метода Шеннона-Фано

int Share\_to\_Intervals(int L, int R, int p[], bool build); //Распределение по интервалам

void Fano\_Shennon(int left, int right, b\_node<f\_node> \*root, int prior[], string \*symbol, map<char, int> &dictionary, bool build); //Кодирование по дереву

void Sort\_Symb\_Prior(string symbol[], int priority[], map<char, int> &dictionary); //Сортировка символов по частоте

string Final\_Code(string str, map<string, string> &table); //Создание закодированного собщения

string One\_Letter(string str, map<char, int> &dictionary, ofstream &fout); //Если в сообщении встречается всего один символ

string hman\_decode(b\_node<node> \* b, string encoded\_str); //Декодирование по методу Хаффмана

string fano\_decode(b\_node<f\_node> \* b, string encoded\_str); //Декодирование по методу Шеннона-Фано

};

}

1. ***Файл Process.cpp***

#include <iostream>

#include <fstream>

#include <map>

#include <conio.h>

#include "b\_tree.h"

#include "queue.h"

#include "process.h"

#include <string>

#include <fstream>

using namespace std;

namespace proc {

struct My::node {

\_\_int64 priority;

string symbol;

};

struct My::f\_node {

int left;

int right;

int priority;

string symbol;

f\_node(int a, int b)

{

left = a;

right = b;

}

f\_node()

{

left = 0;

right = 0;

}

};

void My::hman(string my\_str) {

bool open = false;

ofstream fout("hman.txt", ios\_base::app);

if (fout.is\_open())

open = true;

string result = "";

cout << " [Метод Хаффмана] " << endl << endl;

cout << "Введено сообщение: " << char(34) << my\_str << char(34) << "\nРазмером: " << my\_str.length() \* 8 << " бит";

fout << "Введено сообщение: " << char(34) << my\_str << char(34) << "\nРазмером: " << my\_str.length() \* 8 << " бит";

int\* priority = new int[64];

string\* symbol = new string[64];

for (int i = 0; i < 64; i++) {

priority[i] = 0;

symbol[i].clear();

}

map<string, string> table; // ассоциативный массив связывающий символ и его код

map<char, int> dictionary; //ассоциативный массив связывающий символ и его количество вхождений

//b\_node<node> \* tree;

priority\_queue<b\_node<node>\*>\* queue;

queue = new priority\_queue<b\_node<node>\*>;

\_\_int64 p1, p2;

b\_node<node> \*bt1, \*bt2, \*ht;

ht = new b\_node<node>;

for (unsigned int i = 0; i < my\_str.length(); i++) {

char tmp = my\_str[i];

dictionary[tmp]++;

}

if (dictionary.size() == 1) {//обрабатываем, если в файле используется одна буква

fout << "\n\nТаблица кодов:" << endl;

cout << "\n\nТаблица кодов:" << endl;

result = One\_Letter(my\_str, dictionary, fout);

fout << "\n\nЗакодированное сообщение: " << result << "\nРазмером: " << result.length() << " бит" << endl;

cout << "\n\nЗакодированное сообщение: " << result << "\nРазмером: " << result.length() << " бит" << endl; //вывод закодированного сообщения

for (unsigned i = 0; i < 80; i++)

cout << "-";

fout << "Декодированное сообщение: " << my\_str << endl;

cout << "Декодированное сообщение: " << my\_str << endl;

}

else {

fout << "\n\nСортируем символы по убыванию приоритета:" << endl;

cout << "\n\nСортируем символы по убыванию приоритета:" << endl;

for (map<char, int>::iterator it = dictionary.begin(); it != dictionary.end(); ++it) { //пробегаемся по ассоц.массиву и добавляем элементы в приорит.очередь

b\_node<node>\* tmp = new b\_node<node>();

tmp->data.symbol = it->first;

tmp->data.priority = it->second;

queue->push(tmp, it->second);

}

//выполняем пока не останется один элемент в очереди

while ((queue->get\_top()) != 0) { //создание дерева Хаффмана

fout << "\n";

cout << "\n";

bt1 = queue->pop<b\_node<node>\*>(p1); //делаем pop последнего

bt2 = queue->pop<b\_node<node>\*>(p2); //и предпоследнего элементов

ht = cons\_bt<node>(bt1, bt2, node()); //присоединяем два предыдущих элемента к пустому узлу

ht->data.priority = ht->l\_br->data.priority + ht->r\_br->data.priority; //задаем общий приоритет отца

ht->data.symbol = ht->l\_br->data.symbol + ht->r\_br->data.symbol; //общую строку для отца

queue->push<b\_node<node>\*>(ht, p1 + p2); //добавляем полученное дерево в приор. очередь

fout << "Текущие элементы: ";

cout << "Текущие элементы: ";

for (int i = 0; i <= queue->top; i++) {

fout << "'" << queue->head[i].data->data.symbol << "'(" << queue->head[i].data->data.priority << ") ";

cout << "'" << queue->head[i].data->data.symbol << "'(" << queue->head[i].data->data.priority << ") ";

}

cout << endl;

fout << endl;

print(ht, 0, fout); //вывод дерева Хаффмана

}

fout << "\n\nТаблица кодов:" << endl;

cout << "\n\nТаблица кодов:" << endl;

BuildTable(ht, table, fout); //построение и вывод кодов символов

result = Final\_Code(my\_str, table);

fout << "\nЗакодированное сообщение: " << result << "\nРазмером: " << result.length() << " бит" << endl;

cout << "\nЗакодированное сообщение: " << result << "\nРазмером: " << result.length() << " бит" << endl; //вывод закодированного сообщения

for (unsigned i = 0; i < 80; i++)

cout << "-";

string decoded = hman\_decode(ht, result);

cout << "Декодированное сообщение: " << decoded << endl; //Вывод декодированного сообщения

for (unsigned i = 0; i < decoded.length(); i++) {

if (decoded[i] == char(95))

decoded[i] = char(13);

}

fout << endl << "Декодированное сообщение: " << decoded << endl;

}

fout << "==============================================================" << endl;

delete[] priority;

dictionary.clear();

table.clear();

delete[] symbol;

fout.close();

if (open)

cout << "\nЛог действий успешно записан в файл hman.txt" << endl;

}

void My::fano(string my\_str) {

bool open = false;

ofstream fout("fano.txt", ios\_base::app);

if (fout.is\_open())

open = true;

string result = "";

bool build = true;

int \* priority = new int[64];

string \* symbol = new string[64];

for (int i = 0; i < 64; i++) {

priority[i] = 0;

symbol[i].clear();

}

map<string, string> table; // ассоциативный массив связывающий символ и его код

map<char, int> dictionary; //ассоциативный массив связывающий символ и его количество вхождений

fout << "[Метод Шеннона-Фано]" << endl;

cout << " [Метод Шеннона-Фано] " << endl;

fout << "Введено сообщение: " << char(34) << my\_str << char(34) << "\nРазмером: " << my\_str.length() \* 8 << " бит";

cout << "Введено сообщение: " << char(34) << my\_str << char(34) << "\nРазмером: " << my\_str.length() \* 8 << " бит";

//Open\_Map\_Shen(file, priority, symbol, dictionary); //чтение из файла и запись в массивы

for (unsigned int i = 0; i < my\_str.length(); i++) {

char tmp = my\_str[i];

dictionary[tmp]++;

}

int i1 = -1;

for (map<char, int>::iterator it = dictionary.begin(); it != dictionary.end(); ++it) {

i1++;

priority[i1] = it->second;

symbol[i1] = it->first;

}

if (dictionary.size() == 1) {

fout << "\n\nТаблица кодов:" << endl;

cout << "\n\nТаблица кодов:" << endl;

result = One\_Letter(my\_str, dictionary, fout);

fout << "\n\nЗакодированное сообщение: " << result << "\nРазмером: " << result.length() << " бит" << endl;

cout << "\n\nЗакодированное сообщение: " << result << "\nРазмером: " << result.length() << " бит" << endl;

for (unsigned i = 0; i < 80; i++)

cout << "-";

fout << endl << "Декодированное сообщение: " << my\_str << endl;

cout << "Декодированное сообщение: " << my\_str << endl;

}

else {

Sort\_Symb\_Prior(symbol, priority, dictionary); //сортировка массивов: символы и приоритеты

cout << "\n\nСортируем символы по убыванию приоритета:" << endl;

fout << "\n\nСортируем символы по убыванию приоритета:" << endl;

for (unsigned int i = 0; i < dictionary.size(); i++) {

fout << "'" << symbol[i] << "'(" << priority[i] << ")";

cout << "'" << symbol[i] << "'(" << priority[i] << ")";

if ((dictionary.size() - i) != 1) {

cout << "->";

fout << "->";

}

}

b\_node < f\_node > \* root = make\_root(f\_node(0, dictionary.size()));

Fano\_Shennon(0, dictionary.size() - 1, root, priority, symbol, dictionary, true); //построение дерева

f\_node \* mas;

mas = new f\_node[100];

fout << "\n\nПостроение дерева Фано-Шеннона:" << endl;

cout << "\n\nПостроение дерева Фано-Шеннона:" << endl;

print2(root, mas, priority, symbol, 0, fout);

fout << "\n\nТаблица кодов для каждого символа:" << endl;

cout << "\n\nТаблица кодов для каждого символа:" << endl;

BuildTable2(root, table, fout);

result = Final\_Code(my\_str, table);

fout << "Закодированное сообщение: " << result << "\nРазмером: " << result.length() << " бит" << endl;

cout << "Закодированное сообщение: " << result << "\nРазмером: " << result.length() << " бит" << endl;

for (unsigned i = 0; i < 80; i++)

cout << "-";

string decoded = fano\_decode(root, result);

cout << "Декодированное сообщение: " << decoded << endl; //Вывод декодированного сообщения

for (unsigned i = 0; i < decoded.length(); i++) {

if (decoded[i] == char(95))

decoded[i] = char(13);

}

fout << endl << "Декодированное сообщение: " << decoded << endl;

}

fout << "==============================================================" << endl;

delete[] priority;

dictionary.clear();

table.clear();

delete[] symbol;

fout.close();

if (open)

cout << "\nЛог действий успешно записан в файл fano.txt" << endl;

}

void My::print(b\_node<node> \* root, unsigned k, ofstream &fout) {

if (root != NULL)

{

print(root->r\_br, k + 5, fout);

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

cout << " ";

fout << " ";

}

if (!root->data.symbol.empty()) {

fout << "'" << root->data.symbol << "'(" << root->data.priority << ")" << endl;

cout << "'" << root->data.symbol << "'(" << root->data.priority << ")" << endl;

}

else {

fout << root->data.priority << endl;

cout << root->data.priority << endl;

}

print(root->l\_br, k + 5, fout);

}

}

void My::print2(b\_node<f\_node> \* root, f\_node\* mas, int \*priority, string \*symbol, int k, ofstream &fout) {

if (root != NULL)

{

print2(root->r\_br, mas, priority, symbol, k + 5, fout);

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

cout << " ";

fout << " ";

}

string s;

int n = 0;

for (int i = root->data.left; i <= root->data.right; i++)

{

s += symbol[i];

n += priority[i];

}

if (!root->data.symbol.empty()) {

fout << "'" << s << "'(" << n << ")" << endl;

cout << "'" << s << "'(" << n << ")" << endl;

}

else {

fout << "'" << s << "'(" << n << ")" << endl;

cout << "'" << s << "'(" << n << ")" << endl;

}

print2(root->l\_br, mas, priority, symbol, k + 5, fout);

}

}

void My::BuildTable(b\_node<node> \*root, map<string, string> &table, ofstream &fout) {

static std::string s = "";

if (root->l\_br != NULL&&root->r\_br != NULL)

{

s += "1";

BuildTable(root->r\_br, table, fout);

s.erase(s.size() - 1, 1);

s += "0";

BuildTable(root->l\_br, table, fout);

s.erase(s.size() - 1, 1);

}

else

{

fout << "'" << root->data.symbol << "'" << " = " << s << endl;

cout << "'" << root->data.symbol << "'" << " = " << s << endl;

table[root->data.symbol] = s;

}

}

void My::BuildTable2(b\_node<f\_node> \*root, map<string, string> &table, ofstream &fout) {

static std::string s = "";

if (root->l\_br != NULL&&root->r\_br != NULL)

{

s += "1";

BuildTable2(root->r\_br, table, fout);

s.erase(s.size() - 1, 1);

s += "0";

BuildTable2(root->l\_br, table, fout);

s.erase(s.size() - 1, 1);

}

else

{

fout << "'" << root->data.symbol << "'" << " = " << s << endl;

cout << "'" << root->data.symbol << "'" << " = " << s << endl;

table[root->data.symbol] = s;

}

}

int My::Share\_to\_Intervals(int L, int R, int p[], bool build) {

if (R - L == 1)

return L;

else

{

int schet1 = 0;

int schet2 = 0;

for (int i = L; i <= R; i++)

{

schet1 = schet1 + p[i];

}

if (build)

cout << "Cумма=" << schet1 << endl;

int d = schet1 / 2;

int i = L - 1;

int pos = L;

bool flag = false;

while (true)

{

if ((schet2 + p[pos]) <= d)

{

flag = true;

schet2 += p[pos];

i++;

pos++;

}

else

break;

}

if (!flag)i++;

return i;

}

}

void My::Fano\_Shennon(int left, int right, b\_node<f\_node> \*root, int prior[], string \*symbol, map<char, int> &dictionary, bool build) {

if (right - left != 0)

{

if (build)

cout << "\n\nLEFT = " << left << " RIGHT = " << right << endl;

f\_node ln;

f\_node rn;

int n = Share\_to\_Intervals(left, right, prior, build);

if (build)

cout << "Точка разбиения: " << n << endl;

int p = 0;

int p1 = 0;

for (int i = left; i <= n; i++)

p += prior[i];

for (int i = n + 1; i <= right; i++)

p1 += prior[i];

ln.left = left;

ln.right = n;

ln.symbol = symbol[n];

ln.priority = p;

rn.left = n + 1;

rn.right = right;

rn.symbol = symbol[n + 1];

rn.priority = p1;

if (build) {

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

{

cout << "'" << symbol[i] << "'(" << prior[i] << ") ";

if (i == n)

cout << " | ";

}

}

//\_getch();

root->l\_br = make\_root(ln);

root->r\_br = make\_root(rn);

if (ln.priority>rn.priority)

{

root->l\_br = make\_root(rn);

root->r\_br = make\_root(ln);

}

Fano\_Shennon(left, n, root->l\_br, prior, symbol, dictionary, build);

Fano\_Shennon(n + 1, right, root->r\_br, prior, symbol, dictionary, build);

}

}

void My::Sort\_Symb\_Prior(string symbol[], int priority[], map<char, int> &dictionary) {

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

{

for (unsigned int j = 0; j < dictionary.size() - i - 1; j++)

{

string temp2;

int temp1 = 0;

if (priority[j] < priority[j + 1])

{

temp1 = priority[j];

temp2 = symbol[j];

priority[j] = priority[j + 1];

symbol[j] = symbol[j + 1];

priority[j + 1] = temp1;

symbol[j + 1] = temp2;

}

}

}

}

string My::Final\_Code(string str, map<string, string> &table) {

string result;

//cout << "Закодированное сообщение:" << endl;

for (unsigned int i = 0; i < str.length(); i++) {

string tmp;

tmp.push\_back(str[i]);

//cout << table[tmp];

result += table[tmp];

}

//cout << endl;

return result;

}

string My::One\_Letter(string str, map<char, int> &dictionary, ofstream &fout) {

string result;

//cout << dictionary.begin()->first << "(" << dictionary.begin()->second << ")\n" << endl;

//cout << dictionary.begin()->first << "(" << dictionary.begin()->second << ")\n" << endl;

fout << "'" << dictionary.begin()->first << "' = 1";

cout << "'" << dictionary.begin()->first << "' = 1";

char tmp = str[0];

//cout << "\nЗакодированное сообщение:" << endl;

for (int i = 0; i < dictionary[tmp]; i++)

result += "1";

//cout << "1";

return result;

}

string My::hman\_decode(b\_node<node> \* b, string encoded\_str) {

string curr, decoded;

unsigned i = 0;

while (i < encoded\_str.length()) {

b\_node<node> \* p = b;

curr = "";

while (p->l\_br != NULL && p->r\_br != NULL) {

curr += encoded\_str[i];

i++;

if (curr[curr.length() - 1] == '0') {

p = p->l\_br;

if (encoded\_str.length() < 160)

cout << "Идём влево, текущий код: " << curr << endl;

}

else {

p = p->r\_br;

if (encoded\_str.length() < 160)

cout << "Идём вправо, текущий код: " << curr << endl;

}

}

if (encoded\_str.length() < 160)

cout << "-> Декодирован символ: '" << p->data.symbol << "'" << endl;

decoded += p->data.symbol;

//cout << p->data.symbol << endl;

}

return decoded;

}

string My::fano\_decode(b\_node<f\_node> \* b, string encoded\_str) {

string curr, decoded;

unsigned i = 0;

while (i < encoded\_str.length()) {

b\_node<f\_node> \* p = b;

curr = "";

while (p->l\_br != NULL && p->r\_br != NULL) {

curr += encoded\_str[i];

i++;

if (curr[curr.length() - 1] == '0') {

p = p->l\_br;

if (encoded\_str.length() < 160)

cout << "Идём влево, текущий код: " << curr << endl;

}

else {

p = p->r\_br;

if (encoded\_str.length() < 160)

cout << "Идём вправо, текущий код: " << curr << endl;

}

}

if (encoded\_str.length() < 160)

cout << "-> Декодирован символ: '" << p->data.symbol << "'" << endl;

decoded += p->data.symbol;

//cout << p->data.symbol << endl;

}

return decoded;

}

}

1. ***Файл Main.cpp***

#include <iostream>

#include <fstream>

#include <map>

#include <conio.h>

#include <string>

#include <windows.h>

#include "process.h"

using namespace std;

using namespace proc;

HANDLE console = GetStdHandle(STD\_OUTPUT\_HANDLE); //Консольный хук

COORD CursorPosition;

void gotoXY(int, int); //Перемещения по консоли

void main() {

setlocale(LC\_ALL, "Russian");//Русские символы в консоле

string str;

My lets;

ifstream in("input.txt");

if (!in.is\_open()) {

cout << "Файл input.txt не найден!" << endl;

system("pause");

return;

}

char buff[1024] = { 0 };

//Считывание файла в одну строку (переносы строк заменяются символом "\_")

while (!in.eof()) {

in.getline(buff, 1024);

buff[(sizeof(buff) / sizeof(buff[0])) - 1] = NULL;

str += buff;

str += '\_';

}

str.pop\_back();//Удаление последнего символа

in.close();

if (str.length() == 0) {

cout << "Исходный файл пуст!" << endl;

getchar();

return;

}

start:

system("cls");

int menu\_item = 0, x = 7;

bool running = true;

gotoXY(30, 5); cout << " Main Menu";

gotoXY(10, 7); cout << "->";

while (running)

{

gotoXY(12, 7); cout << "• Метод Хаффмана (вывод алгоритма кодирования/декодирования)";

gotoXY(12, 8); cout << " [Исходный файл - input.txt, результирующий - hman.txt]";

gotoXY(12, 9); cout << "• Метод Шеннона-фано (вывод алгоритма кодирования/декодирования)";

gotoXY(12, 10); cout << " [Исходный файл - input.txt, результирующий - fano.txt]";

gotoXY(12, 11); cout << "• Выход";

system("pause>nul"); // the >nul bit causes it the print no message

if (GetAsyncKeyState(VK\_DOWN) && x < 11) //нажата кнопка "Верх"

{

gotoXY(10, x); cout << " ";

x += 2;

gotoXY(10, x); cout << "->";

menu\_item++;

continue;

}

if (GetAsyncKeyState(VK\_UP) && x > 7) //нажата кнопка "Вниз"

{

gotoXY(10, x); cout << " ";

x -= 2;

gotoXY(10, x); cout << "->";

menu\_item--;

continue;

}

if (GetAsyncKeyState(VK\_RETURN)) { //нажата кнопка "Enter"

system("cls");

switch (menu\_item) {

case 0: {

lets.hman(str);

running = false;

break;

}

case 1: {

lets.fano(str);

running = false;

break;

}

case 2: {

cout << "Вы завершили работу программы." << endl;

running = false;

getchar();

return;

}

}

}

}

getchar();

goto start;

}

void gotoXY(int x, int y)

{

CursorPosition.X = x;

CursorPosition.Y = y;

SetConsoleCursorPosition(console, CursorPosition);

}