Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

Национальный исследовательский ядерный университет «МИФИ»

Обнинский институт атомной энергетики–

Филиал федерального государственного автономного образовательного учреждения высшего образования

«Национальный исследовательский ядерный университет «МИФИ»

(ИАТЭ НИЯУ МИФИ)

Отделение Интеллектуальных кибернетических систем

Направление подготовки: 01.03.02 Прикладная математика и информатика, профиль – Прикладная информатика

УДК 51.37

Утверждаю

Директор ИАТЭ НИЯУ МИФИ

\_\_\_\_\_\_\_\_\_\_\_\_\_Т.А. Осипова

«\_\_\_» \_\_\_\_\_\_\_\_\_2021 г.

ЛАБОРАТОРНАЯ РАБОТА № 2

по теме:

практическая реализация префиксных

методов кодирования

Студент гр. М-Б20 \_\_\_\_\_\_\_\_\_\_\_\_\_\_ С.М. Заграевская

Преподаватель,

к.ф.-м.н., доцент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.В. Мышев

Обнинск 2021

СОДЕРЖАНИЕ

I. Введение 3

1.1. Постановка цели 3

1.2. Постановка задачи и этапы выполнения работы 3

1.3. Основные понятия и определения 4

II. Основная часть 7

2.1. Листинг программы 7

2.2. Тестирование 16

III. Заключение 19

**Введение**

**1.1. Постановка цели**

Цель работы: разработка и реализация компьютерных технологий сжатия информации на основе методов кодирования и способов виртуализации в каналах передачи и хранения в виде готового программного продукта, приобретение базовых знаний построения моделей алгоритмов и процедур программных компонент таких технологий, освоение навыков интеллектуального анализа информационных объектов на уровне их «физического и виртуального» представления, оценки эффективности и конкурентоспособности технологии продукта.

**1.2. Постановка задач и этапы выполнения работы**

1. Разработка моделей алгоритмов и процедур программного компонента сжатия информационного объекта (файлы и другие логические структуры) согласно заданию.
2. Реализация программного интерфейса ввода-вывода файловых структур с внешних устройств (ВУ) в ОЗУ и обратно с учетом особенностей периферии.
3. Разработка логических схем моделей построения заголовков словарей и алгоритмы их реализации.
4. Реализация п.3 в виде элемента технологической цепочки общего программного компонента технологии сжатия.
5. Реализация алгоритмов программных компонент преобразования файлов на бинарных полях памяти.
6. Отладка программных компонент.
7. Тестирование и контрольные примеры.
8. Расчеты и анализ. Выводы.
9. Прием-сдача программного компонента в диалоговом режиме.

**1.3.** **Основные понятия и определения**

Основной информационной и статистической характеристикой кодируемого (сжимаемого) объекта является таблица информационной насыщенности.

Основным элементом в схеме алгоритма кодирования (сжатия) и декодирования (разжатия) информационных объектов (файлов) как бинарного множества, на котором определено (задано) информационное пространство типа *<Х, Nm>* (см. лаб. работу №1), является **словарь,** представляющий собой матрицу типа ***А*(2*,L*)**. Входной алфавит словаря задается элементами первой строки, которые представляют собой бинарные цепочки фиксированной длинны, а выходной алфавит – это элементы второй строки, определенные как бинарные цепочки переменной длины. Словарь в функциональной структуре программного компонента задается в виде сегмента памяти, который представляет собой логическую и структурированную область памяти, которую назовем **заголовком**. Поэтому разработка логической схемы заголовка является важным элементом в технологической цепочке построения кодера и декодера.

Основным моментом является построение модели алгоритма и процедуры формирования элементов выходного алфавита словаря. Здесь их необходимо рассматривать как виртуальные объекты и работать как с данными типа битовой строки, т.к. такой тип данных поддерживают все современные процессоры и любая среда (высокого и низкого уровней) программирования.

**Метод Хаффмена**

Методика разработки алгоритмов и процедур технологий сжатия по этому методу достаточно проста и включает в себя формирование двух элементов: треугольную матрицу вероятностей и построение кодового древа. Треугольная матрица вероятностей имеет размерность **(*L,L*)**, где ***L*** – количество элементов входного алфавита. Элементы столбцов этой матрицы – это вероятности. Первый столбец задается как вероятности элементов входного алфавита, а элементы последующих столбцов формируются по методике Хаффмена. Процесс формирования матрицы поясняется таблицей.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Знаки | Вероятности | Вспомогательные столбцы | | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| *Z1* | 0,22 | 0,22 | 0,22 | 0,26 | 0,32 | 0,42 | 0,58 | 1 |
| *Z2* | 0,2 | 0,20 | 0,20 | 0,22 | 0,26 | 0,32 | 0,42 |  |
| Z3 | 0,16 | 0,16 | 0,16 | 0,20 | 0,22 | 0,26 |  |  |
| *Z4* | 0,16 | 0,16 | 0,16 | 0,16 | 0,20 |  |  |  |
| Z5 | 0,1 | 0,10 | 0,16 | 0,16 |  |  |  |  |
| *Z6* | 0,1 | 0,10 | 0,10 |  |  |  |  |  |
| *Z7* | 0,04 | 0,06 |  |  |  |  |  |  |
| *Z8* | 0,02 |  |  |  |  |  |  |  |

Кодовое древо для формирования элементов выходного алфавита строится на основе матрицы вероятностей. Из точки (первый элемент последнего столбца), соответствующий вероятности 1, направляем две ветви, причем ветви с большей вероятностью присваиваем – 1, а с меньшей – 0. Такое последовательное ветвление продолжаем до тех пор, пока не дойдем до вероятности каждого элемента входного алфавита. На рис. 1 показана иллюстрация этого процесса.

После построения кодового древа, двигаясь сверху вниз записываем, (определяем) элементы выходного алфавита. Словарь будет иметь следующий вид, в котором символы *Z1,Z2,Z3,Z4,Z5,Z6,Z7,Z8* первой строки обозначают буквы входного алфавита и соответствующие им бинарные цепочки фиксированной длины, а бинарные цепочки переменной длины во второй строке – это буквы выходного алфавита.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Z1* | *Z2* | *Z3* | *Z4* | Z5 | *Z6* | *Z7* | *Z8* |
| 01 | 00 | 111 | 110 | 100 | 1011 | 10101 | 10100 |

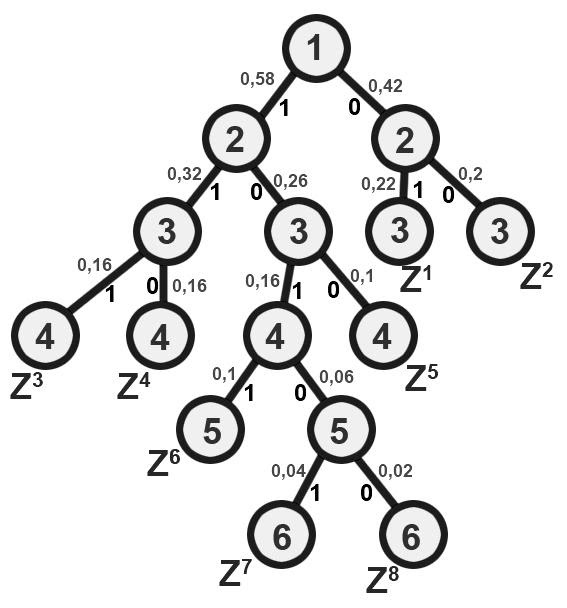
https://lh3.googleusercontent.com/3HnqUUwAK6nBcJprqEVTyjx9oH2NKO51-P7OYoV-4BbpaGjTqZtkxuHloOcgIUG8qbe4kufPoun3zvUC5IrnT6-8_oS_UI_nZgmCS6cvfL3iuJIcxCR4znju53Qy7tCakQbv2rY

Рис. 1. Построение кодового древа

# Основная часть

# 2.1. Листинг программы

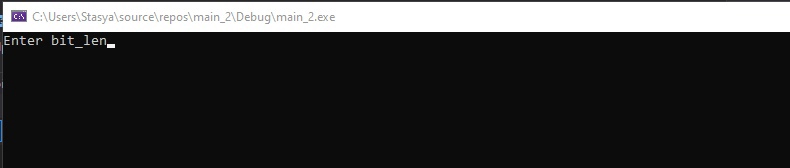
[#include](https://vk.com/im?sel=354876940&st=%23include) <iostream>  
[#include](https://vk.com/im?sel=354876940&st=%23include) <vector> //динамический массив  
[#include](https://vk.com/im?sel=354876940&st=%23include) <map> //ассоциативный массив  
[#include](https://vk.com/im?sel=354876940&st=%23include) <list> //список  
[#include](https://vk.com/im?sel=354876940&st=%23include) <fstream>  
using namespace std;  
  
class Node  
{  
public:  
int freq;  
int symbol;  
Node\* left, \* right;  
  
Node() { left = right = NULL; }  
  
Node(Node\* L, Node\* R) // перегрузка конструктора на левого и правого сына  
{  
left = L;  
right = R;  
freq = L->freq + R->freq;  
}  
};  
  
struct MyCompare  
{  
bool operator()(const Node\* l, const Node\* r) const { return l->freq < r->freq; }  
};  
  
vector<bool> code;  
map<int, vector<bool> > table;  
  
void BuildTable(Node\* root)  
{  
if (root->left != NULL)  
{  
code.push\_back(0); //если слева не 0 -в вектор 0  
BuildTable(root->left); //запускаем рекурсивную функцию для левого сына  
}  
  
if (root->right != NULL)  
{  
code.push\_back(1);  
BuildTable(root->right);  
}  
  
if (root->left == NULL && root->right == NULL) table[root->symbol] = code;  
  
if (code.size())  
code.pop\_back();  
}  
  
void DrawTREE(Node\* Head, int Level)  
{  
if (Head != NULL)  
{  
DrawTREE(Head->left, Level + 1);  
printf("%\*d %d\n", Level \* 3, Head->symbol, Head->freq);  
DrawTREE(Head->right, Level + 1);  
}  
}  
  
int main()  
{  
////// считаем частоты символов  
ifstream f("1.txt", ios::in | ios::binary);  
if (!f.is\_open())  
cout « "Eror";  
map<int, int> m;  
char c;  
int bit\_len = 7;  
cout « "Enter bit\_len";  
cin » bit\_len;  
list<bool> bits;  
bool done = false;  
int index;  
  
while (!done)//пока не конец файла  
{  
index = 0;  
while (bits.size() < bit\_len)  
{  
c = f.get();  
if (c == EOF)  
{  
done = true;  
c = 0;  
break;  
}  
else  
for (int i = 0; i < 8; i++)  
bits.push\_back((c & (1 « (7 - i))) ? 1 : 0);  
}  
if (bits.size())  
{  
for (int i = 0; i < bit\_len; i++)  
{  
index «= 1;  
if (bits.size())  
{  
index |= bits.front();  
bits.pop\_front();  
}  
}  
m[index]++;  
}  
}  
//m.erase(0);  
  
/\*  
t=3  
s=2  
=1  
r=1  
1=1  
64 bits->0111 0100 0111 0011 0111 0100 0010 0000  
0111 0011 0111 0100 0111 0010 0011 0001  
in map  
0111 0100 0111 0011 = 29811 = 00  
0111 0100 0010 0000 = 29728 = 01  
0111 0011 0111 0100 = 29556 = 10  
0111 0010 0011 0001 = 29233 = 11  
0001 1011 XOR 0xFF  
E4  
\*/  
////// записываем начальные узлы в список list  
  
list<Node\*> t;// массив указателей на узел  
for (map<int, int>::iterator itr = m.begin(); itr != m.end(); ++itr) // iterator проходит по элементам массива  
{  
Node\* p = new Node; // кладем в список: создаем в дин памяти новый узел  
p->symbol = itr->first; // c-символ  
p->freq = itr->second; //a -число  
t.push\_back(p);  
} // добавляем в list указатель на объекты  
  
////// создаем дерево  
  
while (t.size() != 1) //пока в списке не останется 1 элемент  
{  
t.sort(MyCompare()); // сортируем  
  
Node\* SonL = t.front(); //левый сын присваиваю первого элементту в списке  
t.pop\_front(); // 1 элемент удаляю  
Node\* SonR = t.front(); //правый сын ,который был вторым,а стал первым  
t.pop\_front(); // удаляем  
  
Node\* parent = new Node(SonL, SonR); //создаем отца  
t.push\_back(parent); // кладем в список  
  
}  
  
Node\* root = t.front(); //root - указатель на вершину дерева  
  
////// создаем пары 'символ-код':  
DrawTREE(root, 0);  
BuildTable(root);  
  
f.clear(); f.seekg(0); // перемещаем указатель снова в начало файла  
  
ofstream g("output.txt", ios::out | ios::binary);  
char buf = 0;  
int count = 0;  
done = false;  
bits.clear();  
while (!done)//пока не конец файла  
{  
index = 0;  
while (bits.size() < bit\_len)  
{  
c = f.get();  
if (c == EOF)  
{  
done = true;  
c = 0;  
}  
for (int i = 0; i < 8; i++)  
bits.push\_back((c & (1 « (7 - i))) ? 1 : 0);  
}  
for (int i = 0; i < bit\_len; i++)  
{  
index «= 1;  
index |= bits.front();  
bits.pop\_front();  
}  
vector<bool> x = table[index];  
for (int n = 0; n < x.size(); n++)  
{ // buf=00000000 побитово складываем  
buf |= x[n] « (7 - count);  
count++;  
if (count == 8) { count = 0; g « buf; buf = 0; }  
}  
}  
if (count)  
g « buf;  
  
f.close();  
g.close();  
  
///// считывание из файла output.txt и преобразование обратно  
  
ifstream F("output.txt", ios::in |

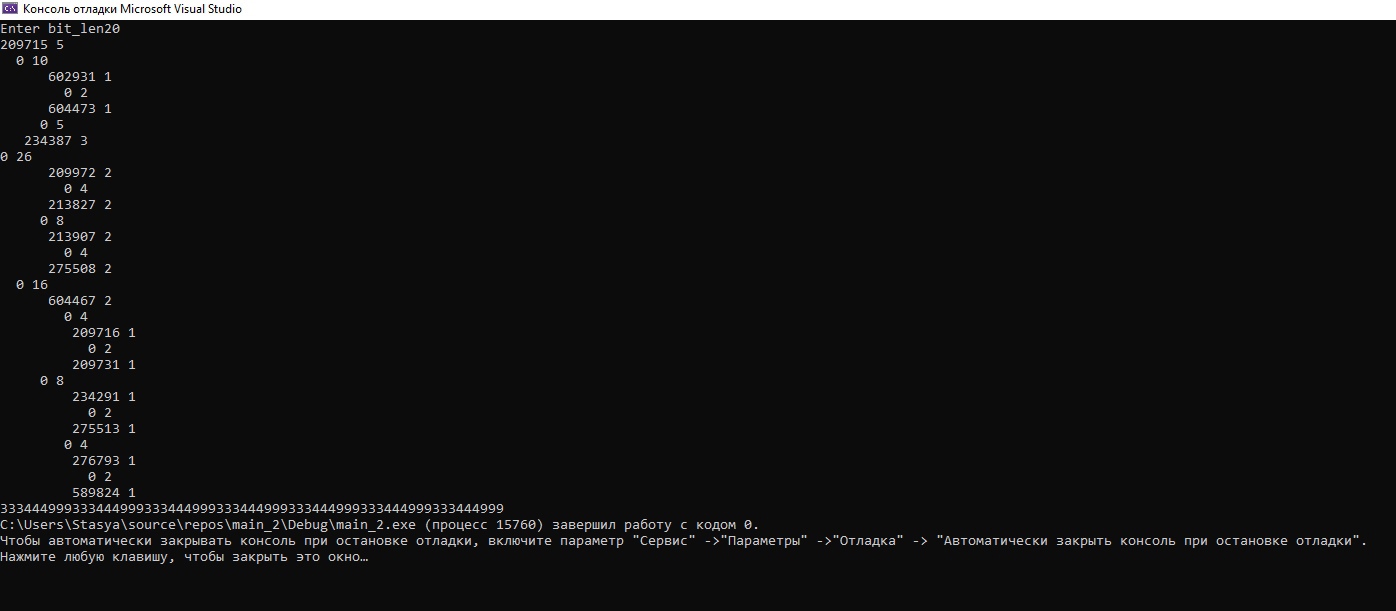
ios::binary);  
  
setlocale(LC\_ALL, "Russian"); // чтоб русские символы отображались в командной строке  
  
Node\* p = root;  
count = 0; char byte;  
byte = F.get();  
bits.clear();  
while (!F.eof())  
{  
bool b = byte & (1 « (7 - count));  
if (b) p = p->right; else p = p->left;  
if (p->left == NULL && p->right == NULL)  
{  
if (p->freq)  
{  
//cout « (char)p->symbol;  
for (int i = bit\_len - 1; i >= 0; i--)  
bits.push\_back((p->symbol & (1 « i)) ? 1 : 0);  
while (bits.size() >= 8)  
{  
c = 0;  
for (int i = 0; i < 8; i++)  
{  
c «= 1;  
c |= bits.front();  
bits.pop\_front();  
}  
cout « c;  
}  
p->freq--;  
}  
p = root;  
}  
count++;  
if (count == 8) { count = 0; byte = F.get(); }  
}  
if (bits.size())  
{  
while (bits.size() < 8)  
bits.push\_back(0);  
c = 0;  
for (int i = 0; i < 8; i++)  
{  
c «= 1;  
c |= bits.front();  
bits.pop\_front();  
}  
cout « c;  
}  
F.close();  
  
return 0;  
}

# 2.2. Тестирование

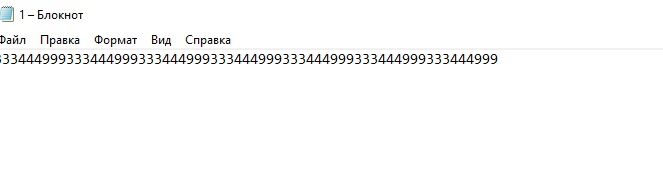
В целях тестирования программы было проведено 3 теста, результаты которых представлены ниже:

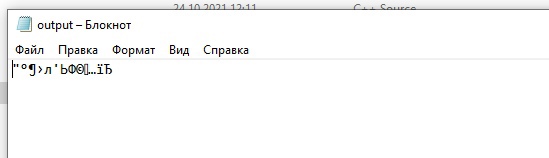
*Тест 1*



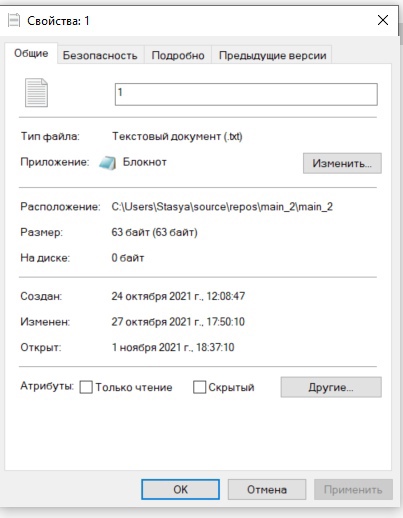


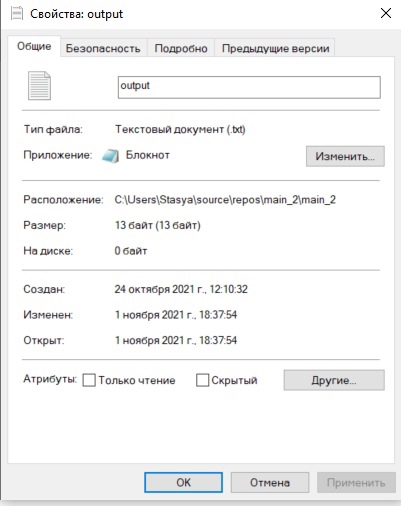
*Тест 2*





*Тест 3*





Программа отработала верно.

**ЗАКЛЮЧЕНИЕ**

Результатом данной лабораторно работы явилось создание программы на языке программирования С++, реализующей функции кодера и декодера на основе префиксных методов в виде программного компонента. Объем реализуемых функций созданного ПО включает:

* ввод-вывод файловых структур с ВУ в ОЗУ и обратно определенных и описанных, как бинарные объекты;
* реализация преобразования информационного объекта в виртуальном адресном пространстве бинарного поля ОЗУ;
* реализация сжатия (архивации) файла и его запись на ВУ;
* реализация распаковки файла и его запись на ВУ;
* реализация процедуры анализа файла на предмет оптимизации выбора параметров его сжатия и определения интегральных характеристик.

Через решение поставленной задачи были изучены и апробированы методы сжатия информации на основе методов кодирования и способов виртуализации в каналах передачи и хранения в виде готового программного продукта, приобретены базовые знания построения моделей алгоритмов и процедур программных компонент таких технологий, сформированы навыки интеллектуального анализа информационных объектов на уровне их «физического и виртуального» представления, оценки эффективности и конкурентоспособности технологии продукта.