Федеральное агентство связи (Россвязь).

Федеральное государственное бюджетное образовательное учреждение «Сибирский государственный университет телекоммуникаций и информатики»

(СибГУТИ)

Кафедра прикладной математики и кибернетики

КУРСОВОЙ ПРОЕКТ

По дисциплине «Структуры и алгоритмы обработки данных»

Вариант 57

Выполнил: Студент 2 курса группы ИП-012

Харченко Е.В.

Проверил: Доцент кафедры Янченко Е. В.

# Содержание

[Содержание 2](#_Toc90501136)

[1. Постановка задачи 3](#_Toc90501137)

[2. Основные идеи и характеристики применяемых методов 4](#_Toc90501138)

[2.1. Метод сортировки 4](#_Toc90501139)

[2.2. Двоичный поиск 6](#_Toc90501140)

[2.3. Списки и очереди 7](#_Toc90501141)

[2.4. Вид дерева и поиск 8](#_Toc90501142)

[2.5. Метод кодирования 9](#_Toc90501143)

[3. Описание структур данных и использованных алгоритмов. 11](#_Toc90501144)

[4. Описание программы 12](#_Toc90501145)

[4.1. Основные переменные и структуры 12](#_Toc90501146)

[4.2. Описание подпрограмм 13](#_Toc90501147)

[5. Исходный текст программы 15](#_Toc90501148)

[6. Результаты 30](#_Toc90501149)

[7. Выводы 34](#_Toc90501150)

# 1. Постановка задачи

Хранящуюся в файле базу данных “Предприятие” (4000 записей) - загрузить динамически в оперативную память с формированием индексного массива как массива указателей, вывести на экран по 20 записей (строк) на странице с возможностью отказа от просмотра.

Упорядочить данные методом сортировки Уильямса-Флойда по дням(!) рождения и ФИО. Упорядоченные данные вывести на экран.

Предусмотреть возможность быстрого поиска по дню рождения в упорядоченной базе, в результате которого из записей с одинаковым ключом формируется очередь, содержимое очереди выводится на экран.

Из записей очереди построить дерево оптимального поиска (приближенный алгоритм А1) по номеру отдела, вывести на экран содержимое дерева и предусмотреть возможность поиска в дереве по запросу.

Закодировать файл базы данных статическим кодом Хаффмена, предварительно оценив вероятности всех встречающихся в ней символов. Построенный код вывести на экран, вычислить среднюю длину кодового слова и сравнить ее с энтропией исходного файла.

# 2. Основные идеи и характеристики применяемых методов

## 2.1. Метод сортировки

Метод сортировки Уильямса - Флойда основан на алгоритме построения пирамиды.

Последовательность **aL, aL+1, …, aR** называется пирамидой, если неравенство

ai≤ min (a2i, a2i+1) выполняется для всех i, для которых хотя бы один из элементов a2i и a2i+1 существует.

Свойства пирамиды:

1. Двустороннее усечение:

Если последовательность aL, aL+1, ..., аR-1, aR - пирамида, то aL+1, ..., aR-1 тоже пирамида.

**2.** Если a1, a2, ..., an– пирамида, то а1 – минимальный элемент пирамиды.

**3.** Если a1, ..., an– произвольная последовательность, то an/2,.., an– пирамида.

Построение пирамиды:

Пусть aL+1, …, aR - пирамида, необходимо добавить элемент Х, чтобы получить новую пирамиду aL, …, aR.

Новый элемент **добавляем в начало**, расширяя последовательность влево.

Если aLудовлетворяет **условию пирамиды**, то пирамида построена.

Иначе найдутся такие a2L или a2L+1 , что **не будут удовлетворять** условию пирамиды.

Возьмем минимальный элемент из a2L и a2L+1, обозначим его за aj и обменяем с aL.

В результате получим a’L ≤ a2L и a’L ≤ a2L+1, что удовлетворяет **условию пирамиды**.

Идея сортировки:

**Первый этап.** Построение пирамиды из элементов массива.

В соответствии со **свойством 3** правая часть массива уже пирамида. Будем добавлять по одному элементу слева, расширяя пирамиду, пока в нее не войдут все элементы массива.

Второй этап. Сортировка.

По **свойству 2** в пирамиде первый элемент минимальный. Производим двустороннее усечение пирамиды: уберем элементы а1 и аn. По **свойству 1** a2, .., an-1 – пирамида. Поставим элемент а1 на последнее место, а элемент аn добавим к пирамиде a2,..,an-1. Отсекаем последний элемент и повторяем действия, пока пирамида не исчезнет.

Характеристики:

Оценим трудоемкость сортировки, используя уже известную оценку трудоемкости построения пирамиды:

C = 2 ; M = + 2

На первом этапе построение пирамиды производится n/2 раз, на втором этапе – n-1 раз.

Очевидно, трудоемкость пирамидальной сортировки имеет порядок

O(nlog2n), *n→∞*.

Количество операций сравнения и пересылки оценивается следующими неравенствами:

C < 2 n log2n + n + 2 M < n log2n + 6.5n - 4

Пирамидальная сортировка не устойчива.

Метод практически не зависит от исходной упорядоченности массива.

## 2.2. Двоичный поиск

Идея поиска:

Алгоритм двоичного поиска в упорядоченном массиве сводится к следующему. Берём средний элемент отсортированного массива и сравниваем с ключом X. Возможны три варианта:

1. Выбранный элемент равен X. Поиск завершён.
2. Выбранный элемент меньше X. Продолжаем поиск в правой половине массива.
3. Выбранный элемент больше X. Продолжаем поиск в левой половине массива.

Если в массиве несколько элементов с одинаковым ключом, алгоритм двоичного поиска, используемый в курсовом проекте, находит самый левый из них. Для поиска остальных элементов с заданным ключом требуется просмотреть массив только в одном направлении – вправо от найденного элемента.

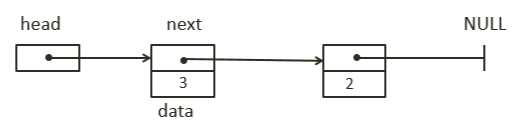
Характеристики:

На каждой итерации поиска необходимоодно сравнение. Количество итераций не больше, чем . Таким образом, трудоёмкость двоичного поиска:

С=O(log*n*), *n → ∞*.

## 2.3. Списки и очереди

Списком называется последовательность однотипных элементов, связанных между собой указателями.



Поле next является указателем на элемент списка и может занимать произвольное место в структуре элемента. Однако если оно является первым элементом структуры, то его адрес совпадает с адресом элемента списка, и это позволяет оптимизировать многие операции со списками. Поле data содержит информацию, которая будет учитываться при сортировке.

Рассмотрим два вида списков: стек и очередь. Стек характеризуется тем, что новый элемент добавляется в начало последовательности, а удаляться может только первый элемент списка. Стек реализует дисциплину обслуживания LIFO (Last Input, First Output). При добавлении в очередь новый элемент ставится в конец списка, удаляется первый элемент последовательности. Очередь реализует дисциплину обслуживания FIFO (FirstInput, FirstOutput)

## 2.4. Вид дерева и поиск

До сих пор предполагалось, что частота обращения ко всем вершинам дерева поиска одинакова. Однако встречаются ситуации, когда известна информация о вероятностях обращения к отдельным ключам. Обычно для таких ситуаций характерно постоянство ключей, т.е. в дерево не включаются новые вершины и не исключаются старые и структура дерева остается неизменной. Эту ситуацию иллюстрирует сканер транслятора, который определяет, является ли каждое слово программы (идентификатор) служебным. Статистические измерения на сотнях транслируемых программ могут в этом случае дать точную информацию об относительных частотах появления в тексте отдельных ключей.

Припишем каждой вершине дерева Vi вес wi, пропорциональный частоте поиска этой вершины (например, если из каждых 100 операций поиска 15 операций приходятся на вершину V1, то w1=15). Сумма весов всех вершин дает вес дерева W. Каждая вершина Vi расположена на высоте hi, корень расположен на высоте 1. Высота вершины равна количеству операций сравнения, необходимых для поиска этой вершины. Определим средневзвешенную высоту дерева с n вершинами следующим образом: hср=(w1h1+w2h2+…+wnhn)/W. Дерево поиска, имеющее минимальную средневзвешенную высоту, называется деревом оптимального поиска (ДОП).

Для построения дерева оптимального поиска по алгоритму А1 в качестве корня берем вершину с наибольшим весом, будем поступать так же для каждого поддерева

## 2.5. Метод кодирования

Метод Хаффмена

Алгоритм построения оптимального кода Хаффмена

1. Упорядочим символы исходного алфавита А={a1,…,an} по убыванию их вероятностей p1≥p2≥…≥pn.

2. Если А={a1,a2}, то a1→0, a2→1.

3. Если А={a1,…,aj,…,an} и известны коды <aj → bj >, j = 1,…,n ,то для {a1,…aj’ ,aj’’…,an}, p(aj)=p(aj’)+ p(aj’’), aj’ → bj0, aj’’ →bj1.

Пусть дан алфавит A={a1, a2, a3, a4, a5, a6} с вероятностями p1=0.36, p2=0.18, p3=0.18, p4=0.12, p5=0.09, p6=0.07. Будем складывать две наименьшие вероятности и включать суммарную вероятность на соответствующее место в упорядоченном списке вероятностей до тех пор, пока в списке не останется два символа. Тогда закодируем эти два символа 0 и 1. Далее кодовые слова достраиваются, как показано на рисунке:

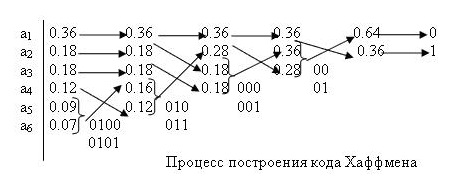
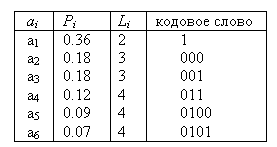


Таблица кода Хаффмена:



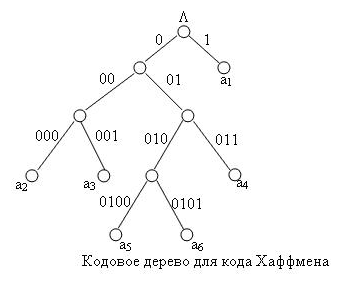
Посчитаем среднюю длину, построенного кода Хаффмена

Lср(P) = 1.0\*36 + 3\*0.18 + 3\*0.18 + 3\*0.12 + 4\*0.09 + 4\*0.07 = 2.44,

при этом энтропия данного источника равна

H = -(0.36\*log0.36 + 2\*0.18\*log0.18 +0.12\*log0.12+ 0.09\*log0.09 + 0.07\*log0.07) = 2.37

Код Хаффмена обычно строится и хранится в виде двоичного дерева, в листьях которого находятся символы алфавита, а на «ветвях» – 0 или 1. Тогда уникальным кодом символа является путь от корня дерева к этому символу, по которому все 0 и 1 собираются в одну уникальную последовательность.



# 3. Описание структур данных и использованных алгоритмов.

В ходе выполнения курсовой работы, помимо основных алгоритмов, потребовалось реализовать также несколько вспомогательных, необходимых для корректной работы программы.

1. Интерфейс программы

Вывод интерфейса главного меню происходит в главной функции main(). Пункт построения дерева отображается только после использования двоичного поиска хотя бы раз. С помощью команды system("color 03") текст приобретает приятный бирюзовый цвет.

2. Загрузка и вывод базы данных

Данные из файла через функцию fread считываются в массив структур типа record в самом начале программы. Создается массив индексов типа int и в цикле заполняется числами от 0 до 3999. Вывод имеет два варианта: все записи сразу или по 20 записей и возможностью выхода из режима просмотра. Сам вывод осуществляется через отсортированный массив индексов.

3. Вспомогательные функции и процедуры для сортировки данных

Для сортировки записей по дням рождения и фамилиям была создана функция сравнения Compare, которая сперва сравнивает первые два символа строки с датой рождения через функцию strncmp, если две даты равны, то функция сравнивает строки с ФИО.

4. Особенности реализации бинарного поиска

Переменная содержащая в себе введенный с клавиатуры искомый день рождения имеет тип int, который в последствии конвертируется в тип char. Функция BSearch ищет первый идущий по порядку день рождения и возвращает индекс этой записи. В цикле все индексы добавляются в очередь, которая через функцию PrintQueue выводится на экран.

5. Вспомогательные функции и процедуры для построения А1-дерева

Каждый раз при вызове пункта бинарного поиска в главном меню дерево очищается, массив весов заполняется новыми индексами и весами. Массив весов сортируется по убыванию с помощью модифицированной быстрой сортировки QuickSortV2.

6. Кодирование данных

Кодирование происходит в функции Encoding. В ней файл заново считывается в буфер, подсчитываются вероятности символов и записываются в массив P и сортируются по убыванию. Символы кодируются и в этой же функции выводятся на экран.

# 4. Описание программы

## 4.1. Основные переменные и структуры

Основные переменные:

record\* tt - указатель на массив записей

queue\* head – указатель на очередь найденных по запросу элементов

Vertex\* Root – указатель на корень A1-дерева

Основные структуры:

Структура записи:

struct record

{

char a[30];

short int b;

char c[22];

char d[10];

};

Структура очереди:

struct queue

{

int number;

queue\* next;

};

Структура вершины дерева:

struct Vertex {

int number;

Vertex \*left;

Vertex \*right;

};

## 4.2. Описание подпрограмм

Кодирование Хаффмана:

int Up(int n, float q, float \*P) – находит в массиве P место, куда вставить число q, и вставляет его, сдвигая вниз остальные элементы.

void Down(int n, int j, int \*L, int \*\*C) – формирует кодовые слова.

void Huffman(float \*P, int \*L, int \*\*C, int n) – основная часть кода Хаффмана.

void Encoding() – вычисляет вероятности, энтропию, среднюю длину кодового слова, выводит коды слов на экран.

Формирование и операции с деревом:

void QuickSortV2(int\*\* A, int L, int R) – сортирует массив по убыванию.

void addVertex(Vertex\*& p, int data, record\* tt) – добавляет вершину в дерево.

void A1(Vertex\*& p, int size, int\*\* VW, record\* tt) – создает дерево.

void DeleteTree(Vertex\* p) – очищает дерево.

void Obhod(Vertex\* p, record\* tt) – обход дерева слева направо.

int SizeTree(Vertex\* p) – возвращает размер дерева.

int TreeSearch(Vertex\* p, record\* tt, short int &X, bool& flag) – поиск и вывод искомых элементов.

Бинарный поиск и операции с очередью:

int BSearch(int A[], char X[], record\* tt) – поиск элемента в массиве по ключу.

void PrintQueue(queue\* head, record\* tt) – вывод очереди.

void BSearchQueue(int A[], record\* tt, queue\*& head) – добавление элементов в очередь.

Сортировка и вывод записей:

int Compare(record\* tt, int A, int B, int n) – сравнение записей по дням рождения и ФИО.

void Heap(int A[], int L, int R, record\* tt) – формирование пирамиды.

void HeapSort(int A[], int N, record\* tt) – сортировка массива.

void PrintDataBase(int A[], record\* tt) – вывод всех записей базы данных.

void PrintDataBaseBy20(int A[], record\* tt) – вывод записей по 20 штук.

# 5. Исходный текст программы

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <string.h>

#include <conio.h>

#include <iomanip>

#include <stdlib.h>

#include <cmath>

#include <Windows.h>

#include <fstream>

using namespace std;

const int M = 256;

float entropy = (float)(0);

float midlength = (float)(0);

struct record

{

char a[30];

short int b;

char c[22];

char d[10];

};

struct queue

{

int number;

queue\* next;

};

struct Vertex {

int number;

Vertex \*left;

Vertex \*right;

};

int Up(int n, float q, float \*P);

void Down(int n, int j, int \*L, int \*\*C);

void Huffman(float \*P, int \*L, int \*\*C, int n);

void Encoding();

void swap(int\* a, int\* b);

void QuickSortV2(int\*\* A, int L, int R);

void addVertex(Vertex\*& p, int data, record\* tt);

void A1(Vertex\*& p, int size, int\*\* VW, record\* tt);

void DeleteTree(Vertex\* p);

void Obhod(Vertex\* p, record\* tt);

int SizeTree(Vertex\* p);

int TreeSearch(Vertex\* p, record\* tt, short int &X, bool& flag);

int BSearch(int A[], char X[], record\* tt);

void PrintBSearch(int A[], record\* tt);

void PrintQueue(queue\* head, record\* tt);

void BSearchQueue(int A[], record\* tt, queue\*& head);

int Compare(record\* tt, int A, int B, int n);

void Heap(int A[], int L, int R, record\* tt);

void HeapSort(int A[], int N, record\* tt);

void PrintDataBase(int A[], record\* tt);

void PrintDataBaseBy20(int A[], record\* tt);

int main()

{

SetConsoleCP(1251);

FILE\* fp;

fp = fopen("testBase2.dat", "rb");

record\* tt = new record[4000];

if(fread((record\*)tt, sizeof(record), 4000, fp));

system("color 03");

fclose(fp);

Vertex\* Root = NULL;

bool flag = 0, tree\_flag = 0;

int A[4000], i1=0;

short int key;

int\*\* VW;

VW = new int\*[2];

for (int i = 0; i < 2; i++)

VW[i] = new int[4000];

char switcher;

queue\* head = NULL;

queue\* temp;

for(int i=0; i<4000; i++){

A[i]=i;

}

HeapSort(A,4000,tt);

while(true){

system("CLS");

if(flag)

cout << "Press 1 for full list" <<

endl << "Press 2 for print by 20" <<

endl << "Press 3 for search" <<

endl << "Press 4 for code" <<

endl <<"Press 5 for tree search" << endl << ">:";

else

cout << "Press 1 for full list" <<

endl << "Press 2 for print by 20" <<

endl << "Press 3 for search" <<

endl << "Press 4 for code" << endl << ">:";

switcher = getch();

cout << endl;

switch(switcher){

case '1':

PrintDataBase(A,tt);

break;

case '2':

PrintDataBaseBy20(A,tt);

break;

case '3':

BSearchQueue(A,tt,head);

DeleteTree(Root);

Root = NULL;

temp = head;

i1 = 0;

while (temp != NULL) {

VW[0][i1] = temp->number;

temp = temp->next;

i1++;

}

for (int i = 0; i < i1; i++)

VW[1][i] = rand() % 100+1;

A1(Root, i1, VW, tt);

flag = 1;

break;

case '5':

i1 = 1;

system("CLS");

cout << "Tree:" << endl;

Obhod(Root,tt);

cout << "Size = " << SizeTree(Root) << endl;

cout << "Enter search number of department:";

cin >> key;

TreeSearch(Root, tt, key, tree\_flag);

if(!tree\_flag)

cout << "Nothing found!" << endl;

tree\_flag = 0;

cout << "Press any button";

getch();

break;

case '4':

system("CLS");

Encoding();

cout << "Press any button";

getch();

break;

default:

delete[] tt;

return 0;

}

}

}

//Functions

int Up(int n, float q, float \*P)

{

int j = n-1;

for(int i = n-1; i >= 2;i--){

if(P[i-1] < q){

P[i] = P[i-1];

}

else {

j = i;

P[j] = q;

return j;

}

}

j = 1;

P[j] = q;

return j;

}

void Down(int n, int j, int \*L, int \*\*C)

{

int S[n+1];

for(int i = 1; i <= n; i++) S[i] = C[j][i];

int l = L[j];

for(int i = j; i <= n-2; i++){

for(int t = 1; t <= n; t++){

C[i][t] = C[i+1][t];

}

L[i] = L[i+1];

}

for(int i = 1; i <= n; i++) C[n-1][i] = S[i];

for(int i = 1; i <= n; i++) C[n][i] = S[i];

C[n-1][l+1] = 0;

C[n][l+1] = 1;

L[n-1] = l+1;

L[n] = l+1;

}

void Huffman(float \*P, int \*L, int \*\*C, int n)

{

if(n == 2){

C[1][1] = 0;

L[1] = 1;

C[2][1] = 1;

L[2] = 1;

}else{

float q = P[n-1]+P[n];

int j = Up(n, q, P);

Huffman(P, L, C, n-1);

Down(n, j, L, C);

}

}

void Encoding()

{

SetConsoleCP(1251);

int i, j = 1, sum = 0;

float Lm = 0, H = 0;

long int n = 0;

char temp\_char;

FILE \*f;

f = fopen("testBase2.dat", "rb");

float\*\* char\_freq = new float\*[2];

for (i = 0; i < 2; i++) {

char\_freq[i] = new float[M];

}

for (i = 0; i < M; i++){

char\_freq[0][i] = i;

char\_freq[1][i] = 0;

}

fseek(f , 0 , SEEK\_END);

long lSize = ftell(f);

rewind (f);

unsigned char \* buffer = (unsigned char\*) malloc(sizeof(char) \* lSize);

if(fread(buffer, 1, lSize, f));

for(i = 0; i < sizeof(char) \* lSize; i++){

char\_freq[1][buffer[i]] += 1;

}

for (i = 0; i < M; i++){

if(char\_freq[1][i] > 0){

n++;

}

sum += char\_freq[1][i];

}

float P[n+1], P1[n+1];

for (i = 0; i < M; i++){

if(char\_freq[1][i] > 0){

P[j] = char\_freq[1][i] / sum;

j++;

}

}

float tmp;

for (i = 1; i<n-1; i++){

for (j=n-1; j>i; j--){

if (P[j] > P[j-1]) {

tmp = P[j-1];

P[j-1] = P[j];

P[j] = tmp;

}

}

}

for (i = 1; i<=n; i++){

P1[i] = P[i];

}

for (i = 0; i<n-1; i++){

for (j=n-1; j>i; j--){

if(char\_freq[1][j] > char\_freq[1][j-1]){

tmp = char\_freq[0][j-1];

char\_freq[0][j-1] = char\_freq[0][j];

char\_freq[0][j] = tmp;

tmp = char\_freq[1][j-1];

char\_freq[1][j-1] = char\_freq[1][j];

char\_freq[1][j] = tmp;

}

}

}

int \*\*C = new int\* [n+1];

for(int i=0;i<n+1;i++){

C[i] = new int[n+1];

}

int\* L = new int[n+1];

Huffman(P, L, C, n);

j = 1;

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

cout << j << ": " << (char)char\_freq[0][i-1] << "\t" << P1[i-1] << "\t";

j++;

for(int j = 1; j <= L[i]; j++){

cout << C[i][j] << " ";

}

cout << endl;

}

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

Lm += P1[i] \* L[i];

}

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

H -= P1[i] \* log2(P1[i]);

}

cout << "Average length = " << Lm << endl << "Entropy = " << H << endl << endl;

}

void swap(int\* a, int\* b)

{

int temp;

temp = \*a;

\*a = \*b;

\*b = temp;

}

void QuickSortV2(int\*\* A, int L, int R)

{

while (L < R)

{

int x = A[1][L];

int i = L;

int j = R;

while (i <= j)

{

while (A[1][i] > x)

i++;

while (A[1][j] < x)

j--;

if (i <= j)

{

swap(&A[0][i], &A[0][j]);

swap(&A[1][i], &A[1][j]);

i++;

j--;

}

}

if (j - L > R - i) {

QuickSortV2(A, i, R);

R = j;

}

else {

QuickSortV2(A, L, j);

L = i;

}

}

}

void addVertex(Vertex\*& p, int data, record\* tt)

{

if (p == NULL)

{

p = new Vertex;

p->number = data;

p->left = NULL;

p->right = NULL;

}

else if (tt[data].b <= tt[p->number].b) {

addVertex(p->left, data, tt);

}

else if (tt[data].b > tt[p->number].b){

addVertex(p->right, data, tt);

}

}

void A1(Vertex\*& p, int size, int\*\* VW, record\* tt)

{

QuickSortV2(VW, 0, size-1);

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

addVertex(p, VW[0][i], tt);

}

}

void DeleteTree(Vertex\* p)

{

if(p){

DeleteTree(p->right);

DeleteTree(p->left);

delete p;

}

}

void Obhod(Vertex\* p, record\* tt)

{

if (p != NULL)

{

Obhod(p->left,tt);

cout<<tt[p->number].a<<setw(4)<<"|"<<tt[p->number].b<<"|\t"<<tt[p->number].c<<"|"<<tt[p->number].d<<"|"<<p->number<<endl;

Obhod(p->right,tt);

}

}

int SizeTree(Vertex\* p)

{

int n;

if (p == NULL) n = 0;

else

n = 1 + SizeTree(p->left) + SizeTree(p->right);

return n;

}

int TreeSearch(Vertex\* p, record\* tt, short int &X, bool& flag)

{

if (p)

{

if(tt[p->number].b > X){

TreeSearch(p->left, tt, X, flag);

}

else if(tt[p->number].b < X){

TreeSearch(p->right, tt, X, flag);

}

else if(tt[p->number].b == X){

TreeSearch(p->left, tt, X, flag);

cout<<tt[p->number].a<<setw(4)<<"|"<<tt[p->number].b<<"|\t"<<tt[p->number].c<<"|"<<tt[p->number].d<<"|"<<p->number<<endl;

flag = 1;

}

}

}

int BSearch(int A[], char X[], record\* tt)

{

int L = 0, R = 3999, m;

while (L < R) {

m = (L+R)/2;

if (strncmp(tt[A[m]].d,X,2)>0)

L = m+1;

else

R = m;

}

if (strncmp(tt[A[R]].d,X,2) == 0)

return R;

return -1;

}

void PrintQueue(queue\* head, record\* tt){

queue\* temp = head;

int j = 1;

while (temp != NULL) {

cout<<j<<".\t"<<tt[temp->number].a<<setw(4)<<"|"<<tt[temp->number].b<<"|\t"<<tt[temp->number].c<<"|"<<tt[temp->number].d<<"|"<<endl;

temp = temp->next;

j++;

}

cout << "Press any button";

getch();

}

void BSearchQueue(int A[], record\* tt, queue\*& head){

head = NULL;

queue\* temp = head;

system("CLS");

int key, element, j=1;

char char\_key[2];

cout << "Enter birthday:";

cin >> key;

if(key < 100){

if (key < 10){

itoa(key, char\_key, 10);

char\_key[1] = char\_key[0];

char\_key[0] = '0';

element = BSearch(A, char\_key, tt);

}

else{

itoa(key, char\_key, 10);

element = BSearch(A, char\_key, tt);

}

}

else{

element = -1;

}

if (element == -1){

cout << "Key was not found!" << endl << "Press any button";

getch();

return;

}

while(strncmp(tt[A[element]].d,char\_key,2)==0){

temp = new queue;

temp->number = A[element];

temp->next = head;

head = temp;

j++;

element++;

if ((element == 4000) or (element == -1))

break;

}

PrintQueue(head,tt);

}

int Compare(record\* tt, int A, int B, int n){

if (strncmp(tt[A].d,tt[B].d,2)<0) return 1;

if (strncmp(tt[A].d,tt[B].d,2)>0) return 0;

if (strcmp(tt[A].a,tt[B].a)<0) return 1;

if (strcmp(tt[A].a,tt[B].a)>0) return 0;

return n;

}

void Heap(int A[], int L, int R, record\* tt){

int X = A[L], j, i=L;

while(1){

j = 2\*i;

if(j > R){

break;

}

if(j < R){

if(Compare(tt, A[j+1], A[j], 1)) j++;

}

if(Compare(tt, X, A[j], 1)){

break;

}

A[i]=A[j];

i=j;

}

A[i]=X;

}

void HeapSort(int A[], int N, record\* tt){

N--;

int L=N/2, R=N, t;

while(L>=0){

Heap(A,L,N,tt);

L--;

}

while(R>0){

t=A[0];

A[0]=A[R];

A[R]=t;

R--;

Heap(A,0,R,tt);

}

}

void PrintDataBase(int A[], record\* tt){

system("CLS");

int i, j=1;

for (i=3999; i >= 0; i--){

cout<<j<<".\t"<<tt[A[i]].a<<setw(4)<<"|"<<tt[A[i]].b<<"|\t"<<tt[A[i]].c<<"|"<<tt[A[i]].d<<"|"<<endl;

j++;

}

cout << "Press any button";

getch();

}

void PrintDataBaseBy20(int A[], record\* tt){

int i = 3999, lim = 3979, j=1;

char switcher;

while(true){

system("CLS");

for (; i > lim ; i--){

cout<<j<<".\t"<<tt[A[i]].a<<setw(4)<<"|"<<tt[A[i]].b<<"|\t"<<tt[A[i]].c<<"|"<<tt[A[i]].d<<"|"<<endl;

j++;

}

lim -= 20;

cout << "1 - next 20/0 - exit:";

while(true){

switcher = getch();

if((switcher == '1') or (switcher == '0'))

break;

}

if ((switcher == '0') or (i<19))

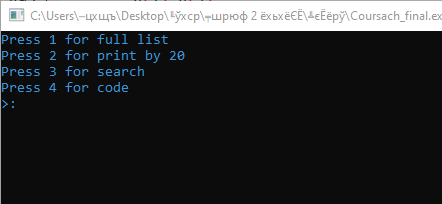
break;

}

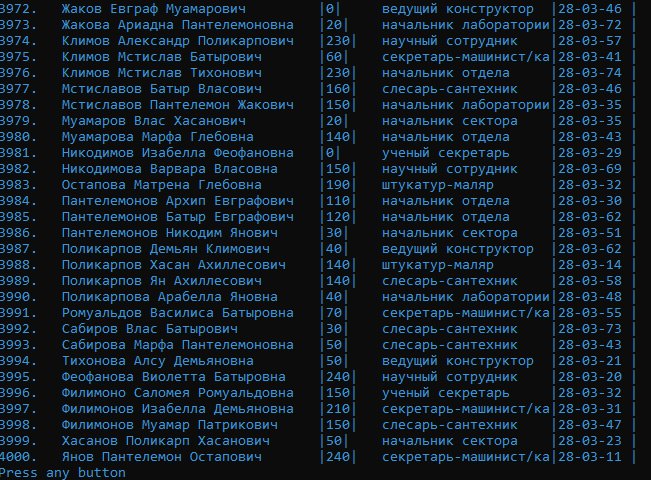
}

# 6. Результаты

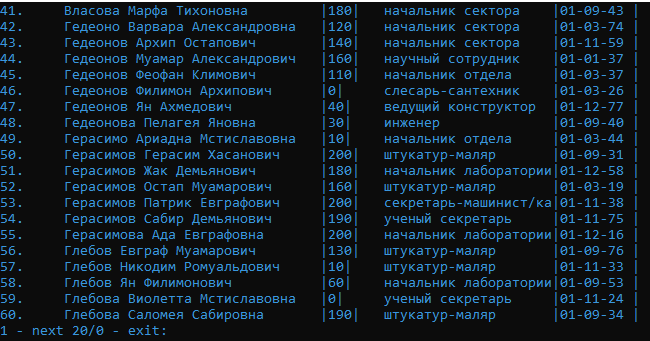
Главное меню (до поиска):



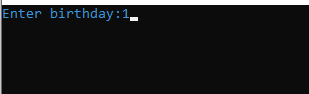
Вывод всех записей файла:



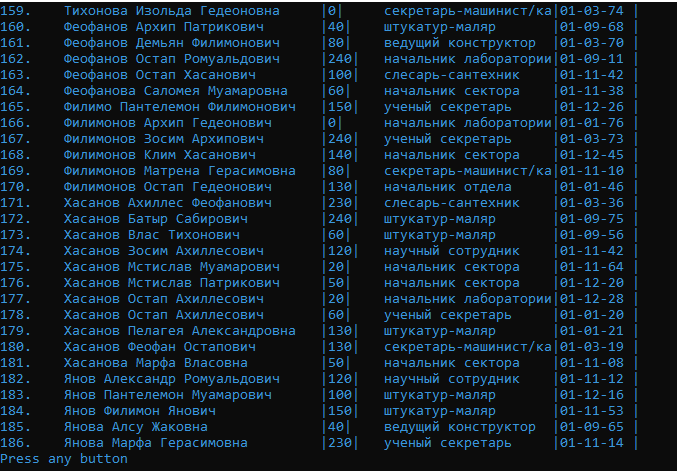
Вывод записей по 20:



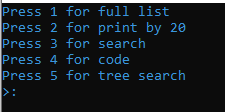
Бинарный поиск 1:



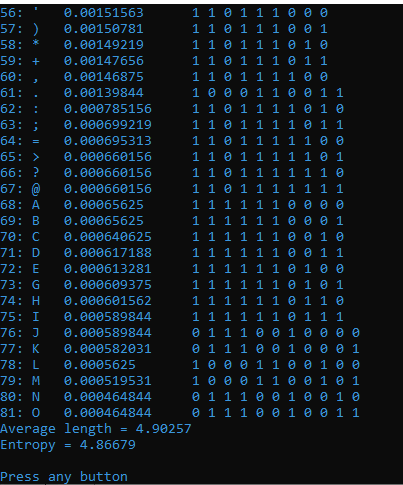
Бинарный поиск 2:



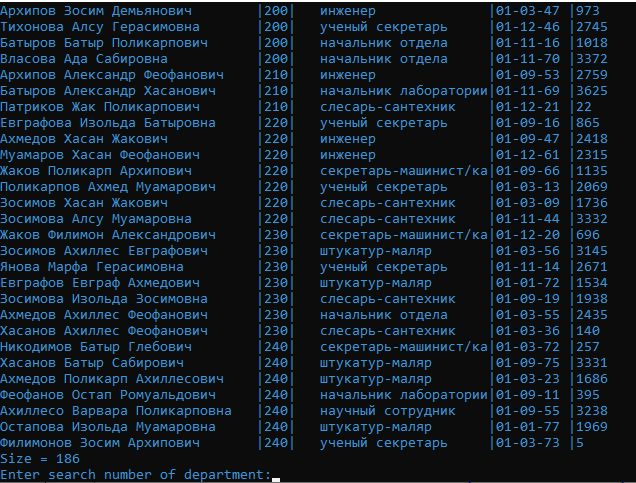
Главное меню (после поиска):



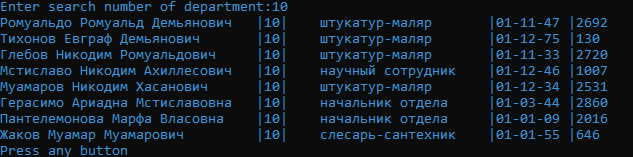
Кодирование:



Поиск в дереве 1:



Поиск в дереве 2:



# 7. Выводы

В ходе выполнения данной работы была создана программа, предоставляющая функционал по обработке базы данных “Предприятие”.

А именно использование данной программы позволяет:

1. Вывести содержимое БД
2. Отсортировать записи методом Уильямса-Флойда по дню рождения и ФИО
3. Выполнить двоичный поиск по дню рождения
4. Построить дерево поиска по ключу (А1-дерево)
5. Выполнить поиск в дереве по номеру отдела
6. Закодировать базу кодом Хаффмана

По моему мнению, задача, поставленная в начале работы над курсовым проектом, была успешно выполнена.

Благодаря данному проекту я научился производить обработку объемных данных, а также, пройдя весь путь написания программы для курсового проекта, я познал все тонкости предоставленных алгоритмов.