Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации

СибГУТИ

Кафедра ПМиК

Курсовой проект

«Структуры и алгоритмы обработки данных»

Вариант 129

Выполнил: cтудент группы ИП-015

Дорогин Т. А.

Проверил:

Солодов П.С.

Новосибирск 2021

СОДЕРЖАНИЕ

1. ПОСТАНОВКА ЗАДАЧ

2. ОСНОВНЫЕ ИДЕИ И ХАРАКТЕРИСТИКИ ПРИМЕНЯЕМЫХ МЕТОДОВ

2.1. Метод сортировки

2.2. Двоичный поиск

2.3. Дерево и поиск по дереву

2.4. Метод кодирования

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ АЛГОРИТМОВ

4. ОПИСАНИЕ ПРОГРАММЫ

4.1. Основные переменные и структуры

4.2. Описание подпрограмм

5. ТЕКСТ ПРОГРАММЫ

6. РЕЗУЛЬТАТЫ ПРОГРАММЫ

7. ВЫВОД

1. ПОСТАНОВКА ЗАДАЧИ

Хранящуюся в файле базу данных (4000 записей) загрузить динамически в оперативную память компьютера в виде списка, вывести на экран по 25 записей на странице с возможностью отказа от просмотра.

Упорядочить данные **по фамилиям замечательных людей**, используя **Метод Вильямса-Флойда**. Упорядоченные данные вывести на экран.

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

Из записей очереди построить **Двоичное Б-дерево** отличному от ключа сортировки, 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.

B = 1 (файл testBase1.dat)

Библиогpафическая база данных "Жизнь замечательных людей"

Стpуктуpа записи:

Автоp: текстовое поле 12 символов

фоpмат <Фамилия>\_<буква>\_<буква>

Заглавие: текстовое поле 32 символа

фоpмат <Имя>\_<Отчество>\_<Фамилия>

Издательство: текстовое поле 16 символов

Год издания: целое число

Кол-во стpаниц: целое число

Пpимеp записи из БД:

Кловский\_В\_Б

Лев\_Hиколаевич\_Толстой\_\_\_\_\_\_\_\_\_

Молодая\_гваpдия\_

1963

864

Ваpианты условий упоpядочения и ключи поиска (К):

C = 1 - по фамилиям(!) замечательных (!) людей, К = тpи пеpвые буквы фамилии;

# 2.ОСНОВНЫЕ ИДЕИ И ХАРАКТЕРИСТИКИ ПРИМЕНЯЕМЫХ МЕТОДОВ

## 2.1. Метод сортировки

Метод Вильямса-Флойда

Метод Вильямса-Флойда основан на алгоритме построения пирамиды. Последовательность ai , ai+1,…,ak называется (i,k)-пирамидой, если неравенство aj≤min(a2j , а2j+1) (\*) выполняется для каждого j, j=i,…,k для которого хотя бы один из элементов a2j , a2j+1 существует.

Процесс построения пирамиды выглядит следующим образом. Дана последовательность as+1,…,ak, которая является (s+1, k)-пирамидой. Добавим новый элемент х и поставим его на s-тую позицию в последовательности, т.е. пирамида всегда будет расширяться влево. Если выполняется (\*), то полученная последовательность – (s, k)-пирамида. Иначе найдутся элементы a2s+1,a2s такие, что либо a2s < as либо a2s+1 < as . Пусть имеет место первый случай, второй случай рассматривается аналогично. Поменяем местами элементы as и a2s . В результате получим новую последовательность as ’ ,as+1,…, a2s ’ ,…, ak. Повторяем все действия для элемента a2s ’ и т.д. пока не получим (s, k)-пирамиду.

Пирамидальная сортировка производится в два этапа. Сначала строится пирамида из элементов массива. По свойству (3) правая часть массива является (n/2+1, n)-пирамидой. Будем добавлять по одному элементу слева, расширяя пирамиду, пока в неё не войдут все элементы массива. Тогда по свойству (2) первый элемент последовательности – минимальный. Произведём двустороннее усечение: уберём элементы a1,an. По свойству (1) оставшаяся последовательность является (2, n-1)-пирамидой. Элемент a1 поставим на последнее место, а элемент an добавим к пирамиде a2,…,an-1 слева. Получим новую (1, n-1)-пирамиду. В ней первый элемент является минимальным. Поставим первый элемент пирамиды на позицию n-1, а элемент an-1 добавим к пирамиде a2,…,an-1, и т.д. В результате получим обратно отсортированный массив.

*2.2. Двоичный поиск*

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

\*Выбранный элемент равен X. Поиск завершён.

\*Выбранный элемент меньше X. Продолжаем поиск в правой половине массива.

\*Выбранный элемент больше X. Продолжаем поиск в левой половине массива.

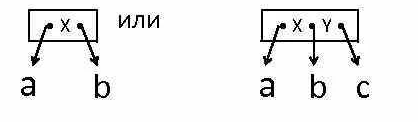
Из-за необходимости найти все элементы, соответствующие заданному ключу поиска, в курсовом проекте использовалась вторая версия двоичного поиска, которая из необходимых элементов находит самый левый, в результате чего для поиска остальных требуется просматривать лишь оставшуюся правую часть массива.

Верхняя оценка трудоёмкости алгоритма двоичного поиска такова. На каждой итерации поиска необходимо два сравнение для первой версии, одно сравнение для второй версии. Количество итераций не больше, чем . Таким образом, трудоёмкость двоичного поиска в обоих случаях: С=O(log n), n → ∞.

2.3. Дерево и поиск по дереву

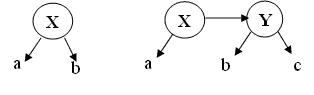
Двоичное Б-дерево

Двоичное Б-дерево состоит из узлов с одним или двумя элементами. Следовательно, каждая страница содержит две или три ссылки на поддеревья. На рисунке 1 показаны примеры страниц Б – дерева при m = 1.



**Рисунок 1 –** Виды вершин ДБД

Поэтому вновь рассмотрим задачу построения деревьев поиска в оперативной памяти компьютера. В этом случае неэффективным с точки зрения экономии памяти будет представление элементов внутри страницы в виде массива. Выход из положения – динамическое размещение на основе списочной структуры, когда внутри страницы существует список из одного или двух элементов.



**Рисунок 2 –** Списочная структура

Таким образом, страницы Б-дерева теряют свою целостность и элементы списков начинают играть роль вершин в двоичном дереве. Однако остается необходимость делать различия между ссылками на потомков (вертикальными) и ссылками на одном уровне (горизонтальными), а также следить, чтобы все листья были на одном уровне.

Очевидно, двоичные Б-деревья представляют собой альтернативу АВЛ-деревьям. При этом поиск в двоичном Б-дереве происходит как в обычном двоичном дереве.

Высота двоичного Б-дерева .

Если рассматривать двоичное Б-дерево как обычное двоичное дерево, то его высота может увеличиться вдвое, т.е. . Для сравнения, в АВЛ-дереве даже в самом плохом случае h<1.44 log n. Поэтому сложность поиска в двоичном Б-дереве и в АВЛ-дереве одинакова по порядку величины.

При построении двоичного Б-дерева реже приходится переставлять вершины, поэтому АВЛ-деревья предпочтительней в тех случаях, когда поиск ключей происходит значительно чаще, чем добавление новых элементов. Кроме того, существует зависимость от особенностей реализации, поэтому вопрос о применение того или иного тапа деревьев следует решать индивидуально для каждого конкретного случая.

## 2.4. Метод кодирования

Адаптивный код Хаффмана

Алгоритм такого кодирования заключается в выполнении следующих действий:

1. Перед кодированием очередной буквы подсчитываются частоты появления в окне всех символов исходного алфавита А={a1, a2, ..., an}. Обозначим эти частоты как q(a1), q(a2), ..., q(an). Вероятности символов исходного алфавита оцениваются на основе значений частот символов в окне P(a1)= q(a1)/W, P(a2) =q(a2)/W, ..., P(an)= q(an)/W.
2. По полученному распределению вероятностей строится код Хаффмана для алфавита А.
3. Очередная буква кодируется при помощи построенного кода.
4. Окно передвигается на один символ вправо, вновь подсчитываются частоты встреч в окне букв алфавита, строится новый код для очередного символа, и так далее, пока не будет получен код всего сообщения.

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ АЛГОРИТМОВ

В ходе написания курсового проекта, помимо основных функций также использовались вспомогательные процедуры для более оптимальной и корректной работы программы.

1. Интерфейс программы

Для создания интерфейса использовалась функция void menu(), выводящая на экран список пунктов меню и она же позволяет пользователю выбирать эти пункты меню, а также закрыть саму программу.

2. Загрузка и вывод базы данных

Для загрузки базы данных отведена процедура Load, в которой производится считывание записей с файла testBase1.dat. Здесь же предусмотрена проверка на наличие файла. Данная процедура вызывается автоматически, в то время как остальные функции программы он может выбрать посредствам меню.

3. Вспомогательные функции для сортировки данных

Для сортировки данных используется функция HeapSort. Сортировка происходит по фамилии замечательного человека.

Вспомогательные функции и процедуры для построения двоичного Б-дерева.

Перед тем, чтобы вывести дерево нужно произвести поиск по первым трём буквам фамилии. После этого выбрать построение дерева, и оно будет отсортировано по заданному ключу. Вызывается процедура добавления элементов очереди в дерево, после выводится дерево tree.traverse(). После этого нужно выбрать в меню поиск по дереву, после чего программа ждёт ввода для начала поиска по дереву. Для поиска по году издательства, используется метод tree.search(y).

4. Кодирование данных

Кодирование базы данных начинается с процедуры int Reading\_Symbols(), которая вызывается из menu() отвечает за все процессы связанные с кодировкой. Функция void Fill\_Chance\_Meet\_of\_Symbols заполняет массив символов количеством их появления в файле. Далее вычисляется вероятность встречаемости каждого символа в файле. Процедура void CodingSymbols и void Huffman кодирует символы алфавитным кодом Хаффмана

4. ОПИСАНИЕ ПРОГРАММЫ

4.1. Основные структуры

struct record {

char Author[12];

char Title[32];

char Pub[16];

short int Year;

short int Pages;

};

struct Node {

record\* r;

};

class TreeNode {

record\*\* keys;

int t;

TreeNode\*\* C;

int n;

bool leaf;

public:

TreeNode(int temp, bool bool\_leaf);

void insertNonFull(record\* k);

void splitChild(int i, TreeNode\* y);

void traverse();

void Delete();

TreeNode\* search(int k);

friend class BTree;

};

class BTree {

TreeNode\* root;

int t;

public:

BTree(int temp) {

root = NULL;

t = temp;

}

void traverse() {

if (root != NULL)

root->traverse();

}

int real() {

if (root != NULL) return 1;

else return 0;

}

TreeNode\* search(int k) {

if (root == NULL) return NULL;

else return root->search(k);

}

void Delete() {

if (root != NULL)

root->Delete();

root = NULL;

}

void insert(record\* k);

};

struct List {

record r;

List\* next;

};

class Queue {

public:

List\* head, \* tail;

Queue() {

head = NULL;

tail = NULL;

}

~Queue() {

delete head;

delete tail;

}

void push(record& rec) {

List\* tmp = new List;

tmp->r = rec;

tmp->next = NULL;

if (head == NULL) head = tail = tmp;

else {

tail->next = tmp;

tail = tmp;

}

}

void printQueue() {

int i = 1;

List\* p = head;

while (p != NULL) {

cout << '[' << i << ']' << setw(12) << p->r.Author

<< setw(36) << p->r.Title

<< setw(20) << p->r.Pub

<< setw(8) << p->r.Year

<< setw(8) << p->r.Pages

<< endl;

p = p->next;

++i;

}

}

void Delete() {

List\* p = head, \*p1;

while (p != NULL) {

p1 = p;

p = p->next;

delete p1;

}

head = NULL;

tail = NULL;

}

};

4.2 Описание подпрограмм

1. Процедуры сортировки данных:

void HeapSort(Node\* arr); - основная функция сортировки.

void heapify(Node\* arr, int n, int i); - создание пирамид.

int ForHeapSort(Node\* arr, int a, int b); - для сравнения фамилий.

2. Функции и процедуры для поиска в отсортированной базе данных:

int FindCheck(Node node, char\* K); – функция сравнения ключа и фамилии в записи.

int FindByKey(Node\* arr, int\* indexl, int\* indexr); – функция быстрого поиска.

3. Методы построения двоичного Б-дерева:

TreeNode(int temp, bool bool\_leaf); \

void insertNonFull(record\* k); -- - создание записи в дереве

void splitChild(int i, TreeNode\* y); /

void traverse(); - выводит дерево

void Delete(); - очищает дерево

TreeNode\* search(int k); - получает ключ и ищет по дереву совпадения.

4.Процедуры и функции кодирования базы данных:

int Reading\_Symbols(); - основная функция кодирования для чтения и заполнения массивов.

void Fill\_Chance\_Meet\_of\_Symbols(char symbol, double\*\* array\_chance, int symbol\_alphabet); - заполнение массива символов их количеством встречи в файле.

void CodingSymbols(double\*\* array\_chance, int symbol\_alphabet); - основная функция кодирования.

void Huffman(int symbol\_alphabet, double\* probability, bool\*\* code\_word, int\* length); - кодирование методом Хаффмана.

void QuickSortForChance(double\*\* array\_chance, int left\_limit, int right\_limit); - сорировка массива шансов.

void Down(int symbol\_alphabet, int j, int\* length, bool\*\* code\_word); - кодирование сверху вниз.

int Up(int symbol\_alphabet, double q, double\* probability); - кодирование снизу вверх.

5.ТЕКСТ ПРОГРАММЫ

#include <iostream>

#include <cstdlib>

#include <cstdio>

#include <string.h>

#include <iomanip>

#include <math.h>

#define N 4000

using namespace std;

int num = 0;

struct record {

char Author[12];

char Title[32];

char Pub[16];

short int Year;

short int Pages;

};

struct Node {

record\* r;

};

class TreeNode {

record\*\* keys;

int t;

TreeNode\*\* C;

int n;

bool leaf;

public:

TreeNode(int temp, bool bool\_leaf);

void insertNonFull(record\* k);

void splitChild(int i, TreeNode\* y);

void traverse();

void Delete();

TreeNode\* search(int k);

friend class BTree;

};

class BTree {

TreeNode\* root;

int t;

public:

BTree(int temp) {

root = NULL;

t = temp;

}

void traverse() {

if (root != NULL)

root->traverse();

}

int real() {

if (root != NULL) return 1;

else return 0;

}

TreeNode\* search(int k) {

if (root == NULL) return NULL;

else return root->search(k);

}

void Delete() {

if (root != NULL)

root->Delete();

root = NULL;

}

void insert(record\* k);

};

TreeNode::TreeNode(int t1, bool leaf1) {

t = t1;

leaf = leaf1;

keys = new record\*[2 \* t - 1];

for (int i = 0; i < 2 \* t - 1; ++i) {

keys[i] = NULL;

}

C = new TreeNode \* [2 \* t];

for (int i = 0; i < 2 \* t; ++i) {

C[i] = NULL;

}

n = 0;

}

void TreeNode::traverse() {

int i;

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

if (leaf == false)

C[i]->traverse();

cout << "[" << ++num << "]" << setw(12) << keys[i]->Author

<< setw(36) << keys[i]->Title

<< setw(20) << keys[i]->Pub

<< setw(8) << keys[i]->Year

<< setw(8) << keys[i]->Pages

<< endl;

}

if (leaf == false)

C[i]->traverse();

}

void TreeNode::Delete() {

for (int i = 0; i < 2 \* t; i++) {

if (leaf == false)

C[i]->Delete();

}

for (int i = 0; i < 2 \* t - 1; i++) {

delete keys[i];

}

delete[] keys;

if (leaf == false)

C[2 \* t]->Delete();

delete[] C;

}

TreeNode\* TreeNode::search(int k) {

int i = 0;

while (i < n && k > keys[i]->Year)

i++;

if (keys[i] != NULL) {

while (keys[i]->Year == k) {

cout << "[" << ++num << "]" << setw(12) << keys[i]->Author

<< setw(36) << keys[i]->Title

<< setw(20) << keys[i]->Pub

<< setw(8) << keys[i]->Year

<< setw(8) << keys[i]->Pages

<< endl;

++i;

}

if (keys[i]->Year > k) i--;

}

if (leaf == true)

return NULL;

return C[i]->search(k);

}

void BTree::insert(record\* k) {

if (root == NULL) {

root = new TreeNode(t, true);

root->keys[0] = k;

root->n = 1;

}

else {

if (root->n == 2 \* t - 1) {

TreeNode\* s = new TreeNode(t, false);

s->C[0] = root;

s->splitChild(0, root);

int i = 0;

if (s->keys[0]->Year < k->Year)

i++;

s->C[i]->insertNonFull(k);

root = s;

}

else

root->insertNonFull(k);

}

}

void TreeNode::insertNonFull(record\* k) {

int i = n - 1;

if (leaf == true) {

while (i >= 0 && keys[i]->Year > k->Year) {

keys[i + 1] = keys[i];

i--;

}

keys[i + 1] = k;

n = n + 1;

}

else {

while (i >= 0 && keys[i]->Year > k->Year)

i--;

if (C[i + 1]->n == 2 \* t - 1) {

splitChild(i + 1, C[i + 1]);

if (keys[i + 1]->Year < k->Year)

i++;

}

C[i + 1]->insertNonFull(k);

}

}

void TreeNode::splitChild(int i, TreeNode\* y) {

TreeNode\* z = new TreeNode(y->t, y->leaf);

z->n = t - 1;

for (int j = 0; j < t - 1; j++)

z->keys[j] = y->keys[j + t];

if (y->leaf == false) {

for (int j = 0; j < t; j++)

z->C[j] = y->C[j + t];

}

y->n = t - 1;

for (int j = n; j >= i + 1; j--)

C[j + 1] = C[j];

C[i + 1] = z;

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

keys[j + 1] = keys[j];

keys[i] = y->keys[t - 1];

n = n + 1;

}

record\* Load(Node\* pointers, int\* check\_load);

void HeapSort(Node\* arr);

void heapify(Node\* arr, int n, int i);

int ForHeapSort(Node\* arr, int a, int b);

int FindCheck(Node node, char\* K);

int FindByKey(Node\* arr, int\* indexl, int\* indexr);

int Reading\_Symbols();

void Fill\_Chance\_Meet\_of\_Symbols(char symbol, double\*\* array\_chance, int symbol\_alphabet);

void CodingSymbols(double\*\* array\_chance, int symbol\_alphabet);

void Huffman(int symbol\_alphabet, double\* probability, bool\*\* code\_word, int\* length);

void QuickSortForChance(double\*\* array\_chance, int left\_limit, int right\_limit);

void Down(int symbol\_alphabet, int j, int\* length, bool\*\* code\_word);

int Up(int symbol\_alphabet, double q, double\* probability);

struct List {

record r;

List\* next;

};

class Queue {

public:

List\* head, \* tail;

Queue() {

head = NULL;

tail = NULL;

}

~Queue() {

delete head;

delete tail;

}

void push(record& rec) {

List\* tmp = new List;

tmp->r = rec;

tmp->next = NULL;

if (head == NULL) head = tail = tmp;

else {

tail->next = tmp;

tail = tmp;

}

}

void printQueue() {

int i = 1;

List\* p = head;

while (p != NULL) {

cout << '[' << i << ']' << setw(12) << p->r.Author

<< setw(36) << p->r.Title

<< setw(20) << p->r.Pub

<< setw(8) << p->r.Year

<< setw(8) << p->r.Pages

<< endl;

p = p->next;

++i;

}

}

void Delete() {

List\* p = head, \*p1;

while (p != NULL) {

p1 = p;

p = p->next;

delete p1;

}

head = NULL;

tail = NULL;

}

};

int main() {

record\* records = NULL;

Node\* pointers = new Node[N];

Queue q;

BTree tree(2);

int choice, check\_load = 0;

records = Load(pointers, &check\_load);

if (check\_load == -1) return -1;

HeapSort(pointers);

int exit\_flag = 1, clear\_flag = 0;

while (exit\_flag) {

cout << "[1]Show unsorted data"

<< endl << "[2]Show sorted data"

<< endl << "[3]Find by last name of a wonderful person"

<< endl << "[4]Build the tree"

<< endl << "[5]Search in tree by year"

<< endl << "[6]Encode Data Base by Huffman"

<< endl << "[7]Clear screen"

<< endl << "[8]Exit"

<< endl;

cout << "The choice is yours: ";

cin >> choice;

switch (choice) {

case 1:

{

int i = 0, j = 0;

while (i < N) {

if (j == 25) {

cout << "========================================================================"

<< endl << "Print more? (0 - NO, 1 - YES): ";

cin >> choice;

if (choice == 0) break;

else j = 0;

}

cout << '[' << i << ']' << setw(12) << records[i].Author

<< setw(36) << records[i].Title

<< setw(20) << records[i].Pub

<< setw(8) << records[i].Year

<< setw(8) << records[i].Pages

<< endl;

++i;

++j;

}

} break;

case 2:

{

int i = 0, j = 0;

while (i < N) {

if (j == 25) {

cout << "========================================================================"

<< endl << "Print more? (0 - NO, 1 - YES): ";

cin >> choice;

if (choice == 0) break;

else j = 0;

}

cout << '[' << i << ']' << setw(12) << pointers[i].r->Author

<< setw(36) << pointers[i].r->Title

<< setw(20) << pointers[i].r->Pub

<< setw(8) << pointers[i].r->Year

<< setw(8) << pointers[i].r->Pages

<< endl;

++i;

++j;

}

} break;

case 3:

{

int indexl = 0, indexr = 0;

if (FindByKey(pointers, &indexl, &indexr)) {

if (q.head != NULL) q.Delete();

for (int i = indexl; i <= indexr; ++i) q.push(\*pointers[i].r);

q.printQueue();

}

else {

cout << "Node not found!" << endl;

}

} break;

case 4:

{

if (q.head == NULL) {

cout << "Queue not found!" << endl;

}

else {

List\* p = q.head;

if (tree.real()) tree.Delete();

while (p != NULL) {

tree.insert(&p->r);

p = p->next;

}

tree.traverse();

num = 0;

}

} break;

case 5:

{

if (tree.real()) {

int y;

cin >> y;

tree.search(y);

num = 0;

}

else {

cout << "Tree not found!" << endl;

}

} break;

case 6:

{

Reading\_Symbols();

} break;

case 7:

{

clear\_flag = 1;;

} break;

case 8:

{

exit\_flag = 0;

q.Delete();

} break;

}

system("PAUSE");

cout << endl;

if (clear\_flag) {

system("cls");

clear\_flag = 0;

}

}

delete[] pointers;

delete[] records;

return EXIT\_SUCCESS;

}

record\* Load(Node\* pointers, int \*check\_load) {

FILE\* fp;

fopen\_s(&fp, "testBase1.dat", "rb");

if (fp == NULL) {

cout << "PROBLEM" << endl;

system("pause");

\*check\_load = -1;

return NULL;

}

record\* records = new record[N];

fread((record\*)records, sizeof(record), N, fp);

fclose(fp);

for (int i = 0; i < N; ++i) pointers[i].r = &records[i];

return records;

}

int FindCheck(Node node, char\* K) {

int check = 0, m = 0;

for (int i = 0; i < 32; ++i) {

if (check == 2) {

m = i;

break;

}

if (node.r->Title[i] == ' ') check++;

}

check = 0;

for (int i = 0; i < 3; ++i) {

if (check == 0) {

if (node.r->Title[m + i] > K[i]) check = 1;

if (node.r->Title[m + i] < K[i]) check = 2;

}

else break;

}

return check;

}

int FindByKey(Node\* arr, int\* indexl, int\* indexr) {

char\* K = new char[4];

for (int i = 0; i < 3; ++i) {

cin >> K[i];

}

int l = 0, r = N - 1, mid, flag = 1;

while ((l <= r) && flag) {

mid = (l + r) / 2;

if (FindCheck(arr[mid], K) == 0) {

flag = 0;

\*indexl = mid;

}

else if (FindCheck(arr[mid], K) == 1) r = mid - 1;

else l = mid + 1;

}

\*indexr = \*indexl;

if (flag == 0) {

while (flag == 0) {

int i = \*indexl - 1;

if (FindCheck(arr[i], K) == 0) \*indexl -= 1;

else flag = 1;

}

flag = 0;

while (flag == 0) {

int i = \*indexr + 1;

if (FindCheck(arr[i], K) == 0) \*indexr += 1;

else flag = 1;

}

} else flag = 0;

delete[] K;

return flag;

}

int ForHeapSort(Node\* arr, int a, int b) {

int check1 = 0, check2 = 0, m1 = 0, m2 = 0, k1 = 0, k2 = 0;

for (int i = 0; i < 32; ++i) {

if (check1 == 2) {

m1 = i;

++check1;

}

else {

if (arr[a].r->Title[i] == ' ') check1++;

}

if (check2 == 2) {

m2 = i;

++check2;

}

else {

if (arr[b].r->Title[i] == ' ') check2++;

}

}

for (int i = 0; i < 32; ++i) {

if (arr[a].r->Title[m1 + i] > arr[b].r->Title[m2 + i]) {

return 1;

}

else if (arr[a].r->Title[m1 + i] < arr[b].r->Title[m2 + i]) {

return 0;

}

}

return 0;

}

void heapify(Node\* arr, int n, int i)

{

int largest = i;

int left = 2 \* i + 1;

int right = 2 \* i + 2;

if (left < n && ForHeapSort(arr, left, largest))

largest = left;

if (right < n && ForHeapSort(arr, right, largest))

largest = right;

if (largest != i) {

swap(arr[i], arr[largest]);

heapify(arr, n, largest);

}

}

void HeapSort(Node\* arr) {

for (int i = N / 2 - 1; i >= 0; i--)

heapify(arr, N, i);

for (int i = N - 1; i >= 0; i--) {

swap(arr[0], arr[i]);

heapify(arr, i, 0);

}

}

int Reading\_Symbols()

{

double\*\* array\_chance;

FILE\* file;

double entropy = 0.0;

int symbol\_alphabet = 256;

int sum = 0;

char symbol = '\0';

fopen\_s(&file, "testBase1.dat", "rb");

if (file == NULL)

{

return 0;

}

array\_chance = new double\*[symbol\_alphabet];

for (int i = 0; i < symbol\_alphabet; i++)

{

array\_chance[i] = new double[2];

array\_chance[i][0] = 0;

array\_chance[i][1] = 0;

}

while (fread(&symbol, sizeof(char), 1, file) && (!feof(file)))

{

Fill\_Chance\_Meet\_of\_Symbols(symbol, array\_chance, symbol\_alphabet);

}

fclose(file);

for (int i = 0; i < symbol\_alphabet; i++)

sum += array\_chance[i][1];

cout << "!!!SUM == " << sum << endl;

for (int i = 0; i < symbol\_alphabet; i++)

array\_chance[i][1] = (array\_chance[i][1] / sum);

for (int i = 0; i < symbol\_alphabet; i++)

entropy = (entropy + log(array\_chance[i][1]) / log(2) \* array\_chance[i][1]);

entropy = entropy \* -1;

CodingSymbols(array\_chance, symbol\_alphabet);

cout << "Entropy: " << entropy << endl;

for (int i = 0; i < symbol\_alphabet; i++)

{

delete[] array\_chance[i];

}

delete[] array\_chance;

return 0;

}

void Fill\_Chance\_Meet\_of\_Symbols(char symbol, double\*\* array\_chance, int symbol\_alphabet)

{

for (int i = 0; i < symbol\_alphabet; i++)

{

if (array\_chance[i][0] == symbol)

{

array\_chance[i][1]++;

return;

}

if (array\_chance[i][1] == 0)

{

array\_chance[i][0] = symbol;

array\_chance[i][1]++;

return;

}

}

}

void CodingSymbols(double\*\* array\_chance, int symbol\_alphabet)

{

double average\_code\_length = 0.0;

double \*probability = new double[symbol\_alphabet + 1];

int \*length = new int[symbol\_alphabet + 1];

bool \*\*code\_word = new bool\*[symbol\_alphabet + 1];

for (int i = 0; i < symbol\_alphabet + 1; i++)

{

probability[i] = 0;

length[i] = 0;

code\_word[i] = new bool[20];

}

for (int i = 0; i < symbol\_alphabet + 1; i++)

{

for (int j = 0; j < 20; j++)

{

code\_word[i][j] = 0;

}

}

QuickSortForChance(array\_chance, 0, symbol\_alphabet - 1);

for (int i = 1; i < symbol\_alphabet + 1; i++)

{

probability[i] = array\_chance[i - 1][1];

}

Huffman(symbol\_alphabet, probability, code\_word, length);

cout << endl << "Symbol Probability Code word Word length" << endl;

for (int i = 1; i < symbol\_alphabet + 1; i++)

{

average\_code\_length += array\_chance[i - 1][1] \* length[i];

printf("%hu)%c\t%f\t\t", i, (char)array\_chance[i - 1][0], array\_chance[i - 1][1]);

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

{

cout << code\_word[i][j];

}

cout << " " << length[i] << endl;

}

cout << "Average length of code words : " << average\_code\_length << endl;

delete[] probability;

delete[] length;

for (int i = 0; i < symbol\_alphabet + 1; i++)

{

delete[] code\_word[i];

}

delete[] code\_word;

}

void Huffman(int symbol\_alphabet, double \*probability, bool \*\*code\_word, int \*length)

{

if (symbol\_alphabet == 2)

{

code\_word[1][0] = 0;

length[1] = 1;

code\_word[2][0] = 1;

length[2] = 1;

}

else

{

double q = probability[symbol\_alphabet - 1] + probability[symbol\_alphabet];

int j = Up(symbol\_alphabet, q, probability);

Huffman(symbol\_alphabet - 1, probability, code\_word, length);

Down(symbol\_alphabet, j, length, code\_word);

}

}

void QuickSortForChance(double\*\* array\_chance, int left, int right)

{

int j = right, i = left;

double X = array\_chance[(left + right) / 2][1];

double temp, m;

while (i <= j)

{

while (array\_chance[i][1] > X)

{

i++;

}

while (array\_chance[j][1] < X)

{

j--;

}

if (i <= j)

{

m = array\_chance[j][0];

array\_chance[j][0] = array\_chance[i][0];

array\_chance[i][0] = m;

temp = array\_chance[j][1];

array\_chance[j][1] = array\_chance[i][1];

array\_chance[i][1] = temp;

i++;

j--;

}

}

if (left < j)

{

QuickSortForChance(array\_chance, left, j);

}

if (i < right)

{

QuickSortForChance(array\_chance, i, right);

}

return;

}

void Down(int symbol\_alphabet, int j, int \*length, bool \*\*code\_word)

{

bool S[20] = { 0 };

for (int i = 0; i < 20; i++)

{

S[i] = code\_word[j][i];

}

int l = length[j];

for (int i = j; i <= symbol\_alphabet - 2; i++)

{

for (int k = 0; k < 20; k++)

{

code\_word[i][k] = code\_word[i + 1][k];

}

length[i] = length[i + 1];

}

for (int i = 0; i < 20; i++)

{

code\_word[symbol\_alphabet - 1][i] = S[i];

code\_word[symbol\_alphabet][i] = S[i];

}

code\_word[symbol\_alphabet - 1][l] = 0;

code\_word[symbol\_alphabet][l] = 1;

length[symbol\_alphabet - 1] = l + 1;

length[symbol\_alphabet] = l + 1;

}

int Up(int symbol\_alphabet, double q, double \*probability)

{

int j = 1;

for (int i = symbol\_alphabet - 1; i > 1; i--)

{

if (probability[i - 1] <= q)

{

probability[i] = probability[i - 1];

}

else

{

j = i;

break;

}

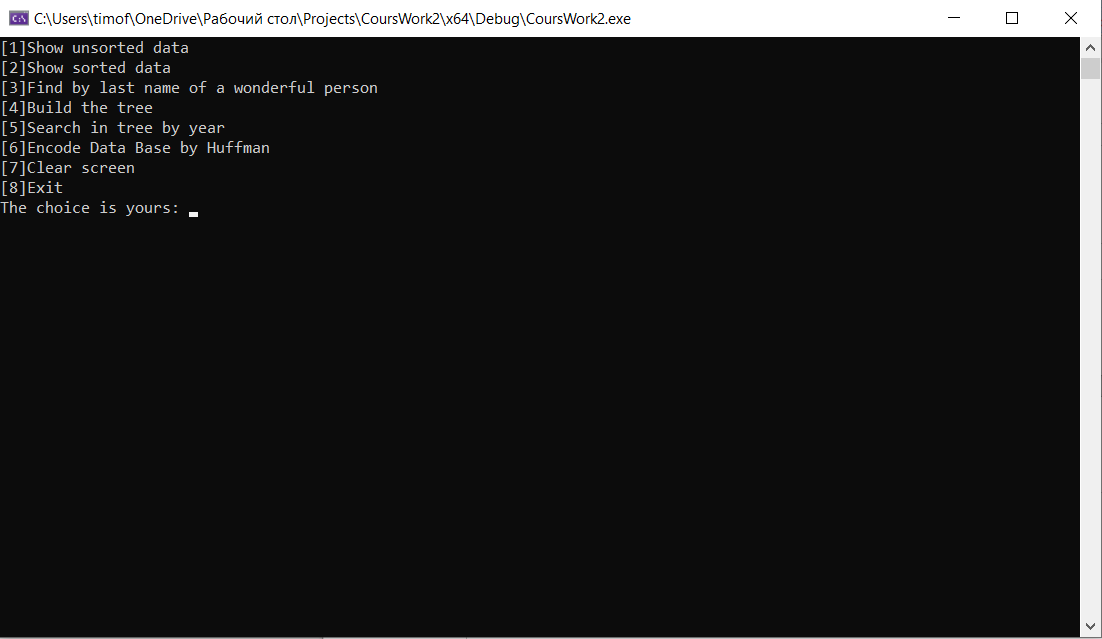
}

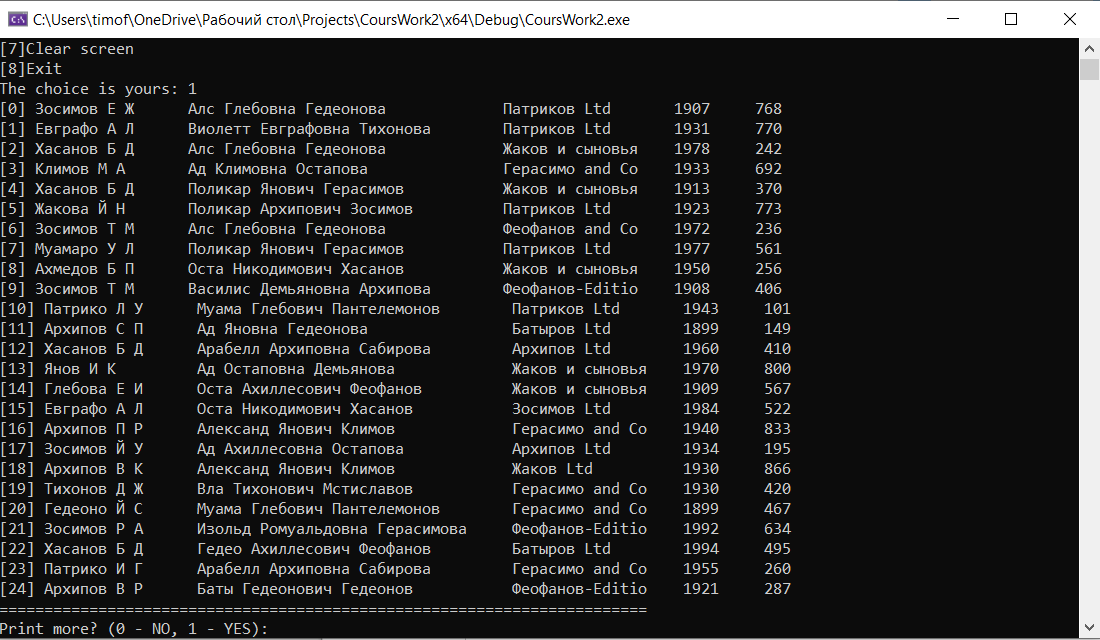
probability[j] = q;

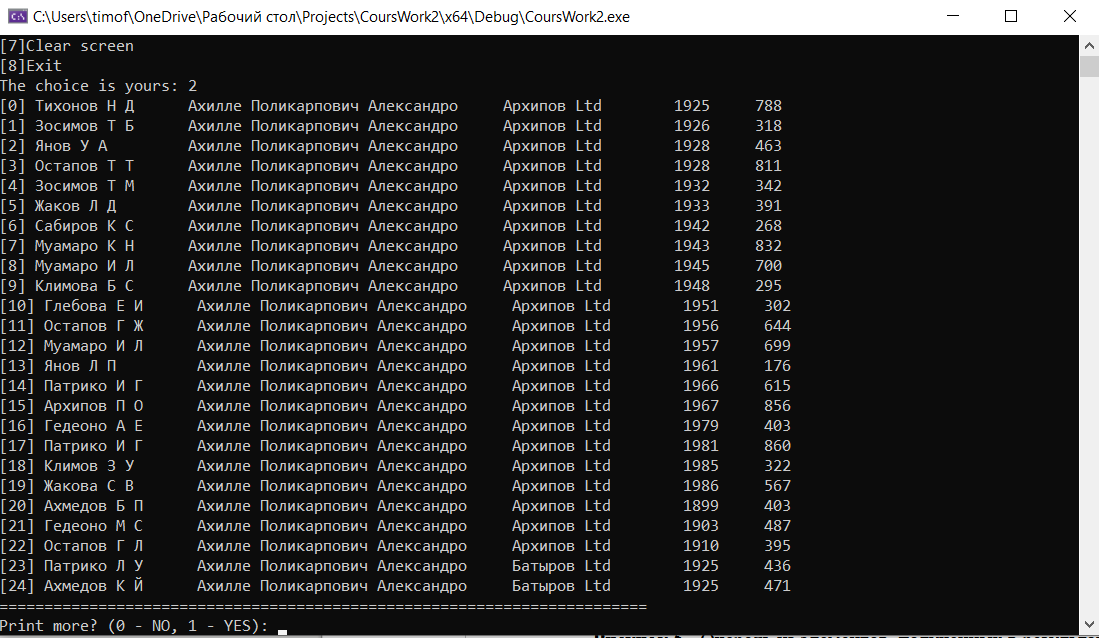
return j;

}

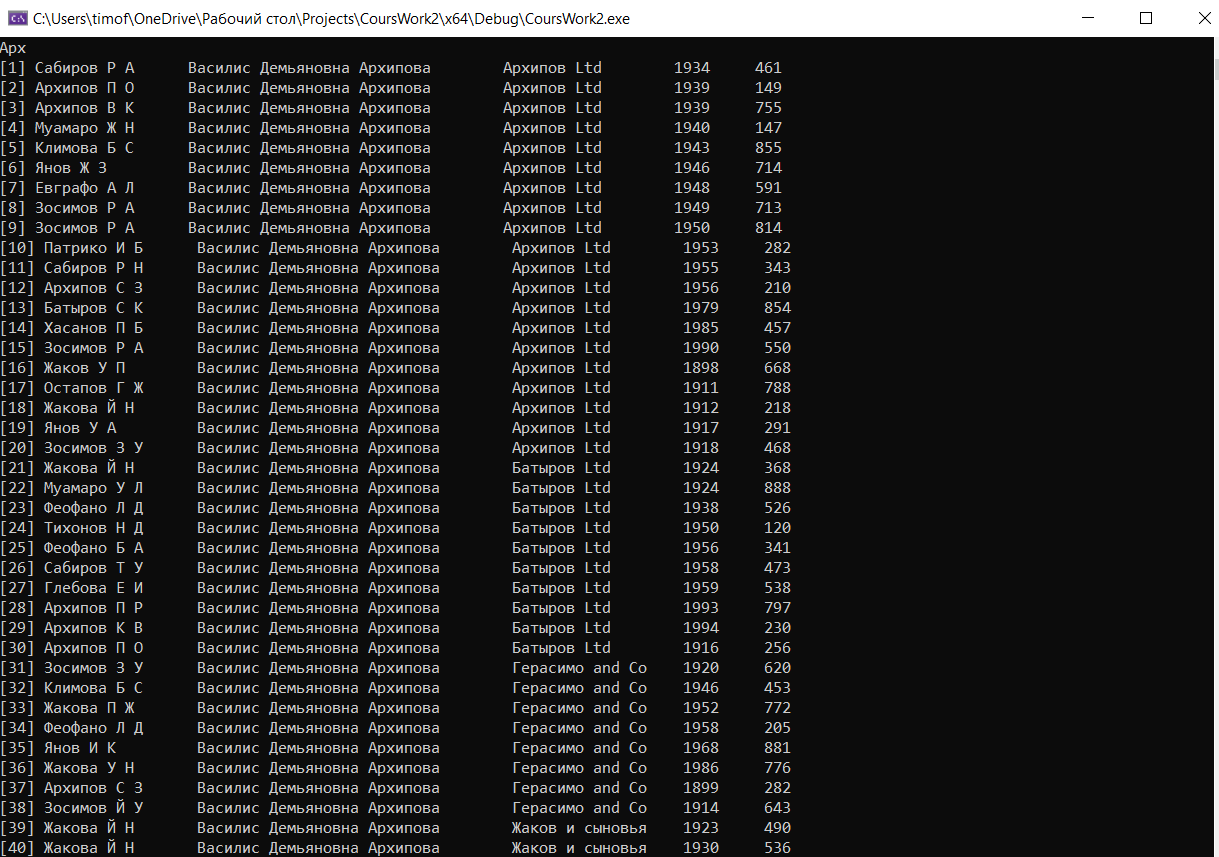
6. РЕЗУЛЬТАТЫ ПРОГРАММЫ

**Рисунок 1 -** Меню

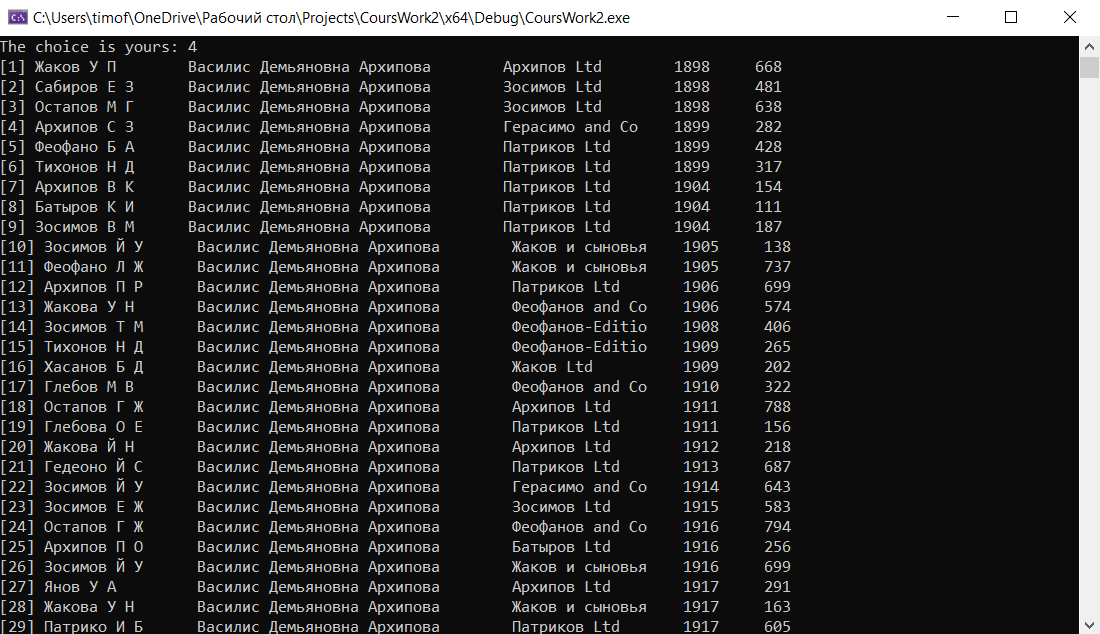
**Рисунок 2 -** Несортированная база данных



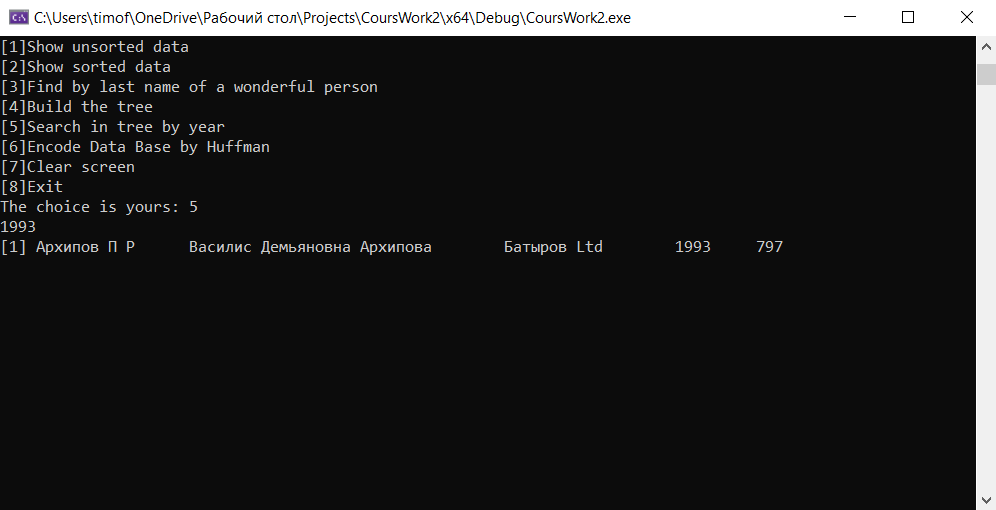
**Рисунок 3 -** Сортированная по фмилиям замечательных людей база данных



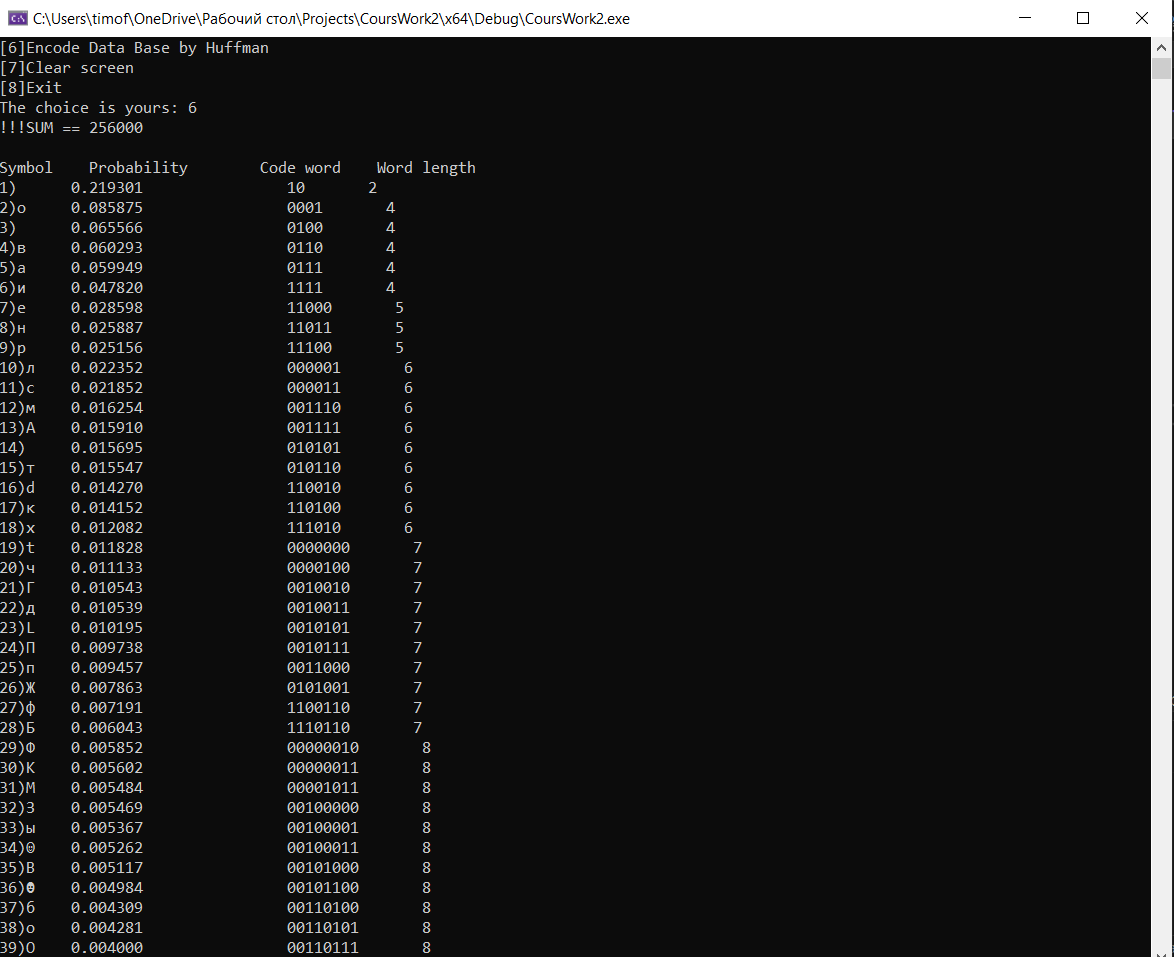
**Рисунок 4 -** Поиск по сортированному массиву по ключу «Арх»

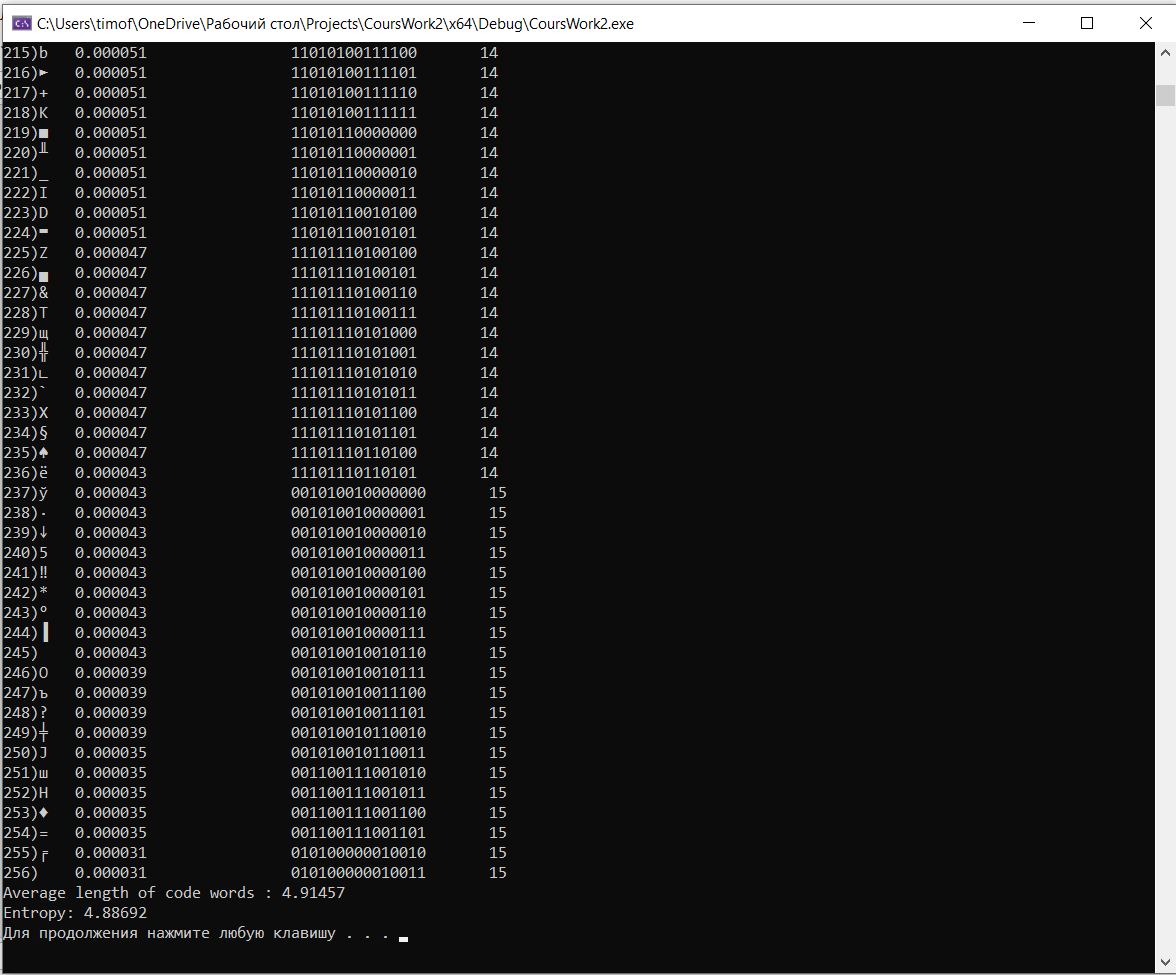


**Рисунок 5 —** Вывод двоичного Б-дерева



**Рисунок 6 -** Поиск по дереву по ключу «1993»





**Рисунок 7 и 8 -** Вы вод результата кодирования

7.ВЫВОД

В ходе выполнения данного курсового проекта были выполнены все поставленные задачи, а именно были реализованы алгоритмы: чтение базы, сортировки, поиск, построение двоичного Б-дерева, поиск по дереву и кодирование базы данных.

В результате кодирования были получены данные подтверждающие теоретические сведения. К которым относится: величины средней длины слова и энтропии.

Реализованные алгоритмы в данной работе представляют минимальный набор процедур для представления и полной обработки базы данных, а также отличаются достаточно высоким быстродействием.