Федеральное агентство связи

Сибирский государственный университет телекоммуникаций и информатики

Кафедра прикладной математики и кибернетики

Курсовой проект

«Структуры и алгоритмы обработки данных»

Вариант

Выполнил: студент группы

Проверил: ассистент кафедры ПМиК

Новосибирск 2021

Оглавление

[1. Постановка задачи 2](#_Toc533614154)

[2. Основные идеи и характеристики применяемых методов 3](#_Toc533614155)

[2.1. Метод сортировки 3](#_Toc533614156)

[2.2. Двоичный поиск 3](#_Toc533614157)

[2.3. Дерево и поиск по нему 4](#_Toc533614158)

[2.4. Метод кодирования 4](#_Toc533614159)

[3. Особенности реализации алгоритмов 6](#_Toc533614160)

[4. Описание программы 8](#_Toc533614161)

[4.1. Основные переменные и структуры 8](#_Toc533614162)

[4.2. Описание подпрограмм 9](#_Toc533614163)

[5. Текст программы 12](#_Toc533614164)

[6. Результаты 25](#_Toc533614165)

[7. Выводы 29](#_Toc533614166)

# Постановка задачи

Хранящуюся в файле библиогpафическую базу данных " Населенный пункт" загрузить динамически в оперативную память компьютера в виде списка, вывести на экран по 20 записей на странице с возможностью отказа от просмотра.

Упорядочить данные с помощью цифровой сортировки (Quick sort), построить по отсортированным данным индексный массив. 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 дерево оптимального поиска по ключу (АВЛ дерево) по дате, вывести на экран содержимое дерева и предусмотреть возможность поиска в дереве по запросу.

Закодировать файл базы данных кодом Шенона, 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.

База данных "Населенный пункт"

Стpуктуpа записи:

ФИО гражданина: текстовое поле 32 символа

фоpмат <Фамилия>\_<Имя>\_<Отчество>

Название улицы: текстовое поле 18 символов

Номер дома: целое число

Номер квартиры: целое число

Дата поселения: текстовое поле 10 символов

фоpмат дд-мм-гг

Пpимеp записи из БД:

Петpов\_Иван\_Федоpович\_\_\_\_\_\_\_\_\_\_\_

Ленина\_\_\_\_\_\_\_\_\_\_\_\_\_\_

10

67

29-02-65

Ваpианты условий упоpядочения и ключи поиска (К):

по улице и фио гражданина, К = улица. Ключ в дереве = дата поселения

# Основные идеи и характеристики применяемых методов

## Метод сортировки

Быстрая сортировка (Quick Sort)

Метод Хоара или метод быстрой сортировки заключается в следующем. Возьмём произвольный элемент массива х. Просматривая массив слева, найдём элемент ai ≥x. Просматривая массив справа, найдём aj ≤x. Поменяем местами ai и aJ . Будем продолжать процесс просмотра и обмена, до тех пор пока i не станет больше j. Тогда массив можно разбить на две части: в левой части все элементы не больше х, в правой части массива не меньше х. Затем к каждой части массива применяется тот же алгоритм.

Очевидно, трудоёмкость метода существенно зависит от выбора элемента х, который влияет на разделение массива. Максимальные значения М и С для метода быстрой сортировки достигаются при сортировке упорядоченных массивов (в прямом и обратном порядке). Тогда в этом случае в одной части остаётся только один элемент (минимальный или максимальный), а во второй – все остальные элементы. Выражения для М и С имеют следующий вид

M=*3(n-1),* C=*(n2+5n+4)/2*

Таким образом, в случае упорядоченных массивов трудоёмкость сортировки имеет квадратичный порядок.

Элемент *am* называется *медианой* для элементов *aL…aR,* если количество элементов меньших *am* равно количеству элементов больших *am* с точностью до одного элемента (если количество элементов нечётно). В примере буква К- медиана для КУРАПОВАЕ.

Минимальная трудоемкость метода Хоара достигается в случае, когда на каждом шаге алгоритма в качестве ведущего элемента выбирается медиана массива. Количество сравнений в этом случае C=*(n+1)*log*(n+1)-(n+1)*. Количество пересылок зависит от положения элементов, но не может быть больше одного обмена на два сравнения. Поэтому количество пересылок – величина того же порядка, что и число сравнений. Асимптотические оценки для средних значений М и С имеют следующий вид

С=О(*n* log *n*), М=О(*n* log *n*) при *n → ∞*.

Метод Хоара неустойчив.

## 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 курсовой работе 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 f_02. Таким образом, трудоёмкость двоичного поиска в обоих случаях

f_03

## Дерево и поиск по нему

АВЛ дерево

Как было показано выше, ИСДП обеспечивает минимальное среднее время поиска. Однако перестройка дерева после случайного включения вершины – довольно сложная операция. СДП дает среднее время поиска на 40 % больше, но процедура построения достаточно проста. Возможное промежуточное решение – введение менее строгого определения сбалансированности. Одно из таких определений было предложено Г. М. Адельсон – Вельским и Е. М. Ландисом (1962).

Дерево поиска называется сбалансированным по высоте, или АВЛ – деревом, если для каждой его вершины высоты левого и правого поддеревьев отличаются не более чем на 1.

На рисунке 39 приведены примеры деревьев, одно из которых является АВЛ-деревом, а другое – нет. В выделенной вершине нарушается баланс высот левого и правого поддеревьев.

Рисунок 39 Пример АВЛ-дерева и не АВЛ-дерева

Заметим, что ИСДП является также и АВЛ – деревом. Обратное утверждение не верно.

Адельсон – Вельский и Ландис доказали теорему, гарантирующую, что АВЛ-дерево никогда не будет в среднем по высоте превышать ИСДП более, чем на 45% независимо от количества вершин:

log(n+1) ≤ hАВЛ(n) < 1,44 log(n+2) – 0,328 при n→∞.

Таким образом, лучший случай сбалансированного по высоте дерева – ИСДП, худший случай – плохое АВЛ – дерево. Плохое АВЛ – дерево это АВЛ-дерево, которое имеет наименьшее число вершин при фиксированной высоте. Рассмотрим процесс построения плохого АВЛ-дерева. Возьмём фиксированную высоту h и построим АВЛ – дерево с минимальным количеством вершин. Обозначим такое дерево через Th. Ясно, что Т0 – пустое дерево, Т1 – дерево с одной вершиной. Для построения Тh при h > 1 будем брать корень и два поддерева с минимальным количеством вершин.

h=1

**T1**

h=2

**T2**

h=3

**T3**

h=4

**T4**

T3

T2

**T5**

T3

T4

h=5

Рисунок 40 Деревья Фибоначчи

Одно поддерево должно быть высотой h–1, а другое высотой h–2. Поскольку принцип их построения очень напоминает построение чисел Фибоначчи, то такие деревья называют деревьями Фибоначчи: Th = < Th-1, x, Th-2 >. Число вершин в Th определяется следующим образом:

n0 = 0, n1 = 1, nh = nh-1 + 1 + nh-2

Повороты при балансировке

Рассмотрим, что может произойти при включении новой вершины в сбалансированное по высоте дерево. Пусть r – корень АВЛ-дерева, у которого имеется левое поддерево (ТL) и правое поддерево (TR). Если добавление новой вершины в левое поддерево приведет к увеличению его высоты на 1, то возможны три случая:

если hL = hR, то ТL и TR станут разной высоты, но баланс не будет нарушен;

если hL < hR, то ТL и TR станут равной высоты, т. е. баланс даже улучшится;

если hL > hR, то баланс нарушиться и дерево необходимо перестраивать.

Введём в каждую вершину дополнительный параметр Balance (показатель баланса), принимающий следующие значения:

-1, если левое поддерево на единицу выше правого;

0, если высоты обоих поддеревьев одинаковы;

1, если правое поддерево на единицу выше левого.

Если в какой-либо вершине баланс высот нарушается, то необходимо так перестроить имеющееся дерево, чтобы восстановить баланс в каждой вершине. Для восстановления баланса будем использовать процедуры поворотов АВЛ-дерева.

## Метод кодирования

Код Щеннона

Код Шеннонапозволяет построить почти оптимальный код с длинами кодовых слов . Тогда по теореме Шеннона из п. 5.1

.

Код Шеннона, удовлетворяющий этому соотношению, строится следующим образом:

1. Упорядочим символы исходного алфавита А={*a*1*,a*2*,…,an*} по убыванию их вероятностей: *p*1*≥p*2*≥p*3*≥…≥pn*.
2. Вычислим величины *Qi*:, которые называются *кумулятивные вероятности*

*Q*0*=*0*, Q*1*=p*1*, Q*2*=p*1*+p*2*, Q*3*=p*1*+p*2*+p*3*, … , Qn=*1.

1. Представим *Qi* в двоичной системе счисления и возьмем в качестве кодового слова первые  знаков после запятой .

Для вероятностей, представленных в виде десятичных дробей, удобно определить длину кодового слова *Li* из соотношения

, .

**Пример.** Пусть дан алфавит A={*a*1*, a*2*, a*3*, a*4*, a*5*, a*6} с вероятностями *p*1=0.36, *p*2=0.18, *p*3=0.18, *p*4=0.12, *p*5=0.09, *p*6=0.07. Построенный код приведен в таблице 6.

Таблица 6 Код Шеннона

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *ai* | *Pi* | *Qi* | *Li* | кодовое слово |
| *a*1  *a*2  *a*3  *a*4  *a*5  *a*6 | 1/22≤0.36<1/2  1/23≤0.18<1/22  1/23≤0.18<1/22  1/24≤0.12<1/23  1/24≤0.09<1/23  1/24≤0.07<1/23 | 0  0.36  0.54  0.72  0.84  0.93 | 2  3  3  4  4  4 | 00  010  100  1011  1101  1110 |

Построенный код является префиксным. Вычислим среднюю длину кодового слова и сравним ее с энтропией. Значение энтропии вычислено при построении кода Хаффмана в п. 5.2 (*H* = 2.37), сравним его со значением средней длины кодового слова кода Шеннона

*Lср*= 0.36**.**2+(0.18+0.18)**.**3+(0.12+0.09+0.07)**.**4=2.92< 2.37+1,

что полностью соответствует утверждению теоремы Шеннона.

# 3. Особенности реализации алгоритмов

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

1. *Интерфейс программы*

Интерфейс программы реализован в main части используя цикл while с возможностью выхода из него, при выборе пятого пункта. Выбор пунктов меню реализован с помощью функции switch( ).

1. *Загрузка и вывод базы данных*

База данных открывается в int main( ) и считывается в список, в котором хранится база данных. Считывание производится независимо от желания пользователя, в то время как большинство остальных функций он может выбрать посредствам меню. После считывания в список структур, файл закрывается.

За вывод элементов базы данных отвечает процедура void print(basa \*\*mas) которая представляет возможность просмотра базы данных постранично.

1. *Вспомогательные функции и процедуры для сортировки данных*

База данных сортируется после запуска программы. Для сортировки базы данных используется процедура void qs(basa \*\*mas, int left, int right). Она сортирует список сначала по полям улицы и ФИО гражданина. Для быстрого доступа к отсортированной и неотсортированной базе данных, перед вызовом процедуры сортировки, делается индексный массив оригинальной базы данных.

1. *Особенности реализации бинарного поиска*

Бинарный поиск по отсортированной базе данных осуществляется в процедуре void search(basa \*\*mas, int left, int right). Результатом работы является очередь из всех элементов, удовлетворяющих условию ключа.

1. *Вспомогательные функции и процедуры для построения АВЛ-дерева.*

Построение дерева осуществляется в процедуре void addAVL(spis \*p,vertex \*&point). Записи заносятся в дерево в процедуре void add\_vertex(vertex \*&p, list \*\*mas, int w). Обход дерева слева направо осуществляется в процедуре void Obhod(vertex \*v). Поиск по дереву выполняется в процедуре void searchAVL (vertex \*root, char \*data).

1. *Кодирование данных*

Кодирование данных начинается с процедуры void FStart(), которая открывает файл базы данных для чтения, заполняет массив структур для алфавита кодовых слов всеми возможными символами, считывает из файла символы и считает их вероятности, закрывает файл и сортирует полученный алфавит по вероятностям. В процедуре void shanon() считается длина кодового слова и само кодовое слово. В процедуре void FPrint() осуществляется подсчет и вывод средней длины кодового слова, энтропии, а также выводит символы, их вероятности, длины кодовых слов и сами кодовые слова на монитор.

# Описание программы

## Основные переменные и структуры

struct basa {

char a[32];

char b[18];

short int c;

short int d;

char e[10];

};

Структура для хранения элемента базы данных. Всего таких элементов:

const int N = 4000;

struct spis {

spis \*next;

spis \*prev;

basa \*data;

};

Структура для создания списка, в котором хранятся элементы базы данных.

struct vertex

{

basa\* pole;

int bal;

vertex \*left;

vertex \*right;

vertex \*next;

};

Структура, представляющая АВЛ-дерево.

vertex \*tree = new vertex; - указатель на корень дерева

struct SHANON {

float p;

float q;

int l;

char a;

};

Структура, представляющая собой информацию о символе для формирования кодового слова.

SHANON A[M]; - массив для символов

const int M = 256 – число символов в алфавите

float entropy = (0) - энтропия

float length = (0) – средняя длина кодового слова

int v=0; - счётчик уникальных символов в файле

int sum =0 – счётчик всех символов в файле

## Описание подпрограмм

Процедуры работы с базой данных:

1. Void Print(list \*base) - визуальный вывод списка на консоль.

Процедуры сортировки:

1. void qs(basa\* mas[], int left, int right) – сортирует базу данных по улице и ФИО гражданина.

Процедуры и функции для поиска в отсортированной базе данных:

1. void search(basa \*mas[], int left, int right) - бинарный поиск по ключу года.
2. int compare(basa \*a, basa \*b) - сравнение полей
3. void push\_queue(spis \*c, spis \*\*b) - формирование очереди из найденных элементов

Процедуры построения АВЛ-дерева:

1. void addAVL(spis \*p, vertex \*&point) – построение АВЛ-дерева.
2. void ll(vertex\* &point) - вспомогательная функция балансировки АВЛ-дерева
3. void rr(vertex\* &point) - вспомогательная функция балансировки АВЛ-дерева
4. void lr(vertex\* &point) - вспомогательная функция балансировки АВЛ-дерева
5. void rl(vertex\* &point) - вспомогательная функция балансировки АВЛ-дерева
6. int compare\_date(basa \*a,basa \*b) - сравнение поля даты
7. void Obhod(vertex \*v) – вывод дерева.
8. void searchAVL (vertex \*root, char \*data) – поиск в дереве.

Процедуры и функции кодирования базы данных:

1. void FStart() – считывание символов базы данных.
2. void shanon() – подсчет вероятностей символов, преобразование алфавита и создание кодовых слов.
3. void FPrint() – вывод статистики и алфавита с вероятностями.

Основная программа:

1. main( ) - основная программа, в которой выводится меню, а также в зависимости от выбранного пункта меню вызываются соответствующие процедуры и функции.

# Текст программы

#include <fstream>

#include <iostream>

#include <conio.h>

#include <Windows.h>

#include <iomanip>

#include <cstdio>

#include <cstring>

#include <cstdlib>

#include <cmath>

#include <iomanip>

#include <conio.h>

#include <ctime>

using namespace std;

const int M=256;

const int N=4000;

int sum=0;

int code[M][M];

double entropy = 0, length = 0;

int v=0;

bool up = false;

struct basa {

char a[32];

char b[18];

short int c;

short int d;

char e[10];

};

struct SHANON {

float p;

float q;

int l;

char a;

};

SHANON A[M];

struct vertex

{

basa\* pole;

int bal;

vertex \*left;

vertex \*right;

vertex \*next;

};

vertex \*tree = new vertex;

struct spis {

spis \*next;

spis \*prev;

basa \*data;

};

spis \*head = NULL, \*tail = NULL;

void print(basa\* mas[]);

void qs(basa\* mas[], int left, int right);

void search(basa \*mas[], int left, int right);

void push\_queue(spis \*c, spis \*\*b);

int compare(basa \*a, basa \*b);

void ll(vertex\* &point);

void rr(vertex\* &point);

void lr(vertex\* &point);

void rl(vertex\* &point);

void addAVL(spis \*p, vertex \*&point);

void Obhod(vertex \*v);

int compare\_date(basa \*a,basa \*b);

void searchAVL (vertex \*root, char \*data);

void shanon();

void FPrint();

void FStart();

int main()

{

setlocale(LC\_ALL, "Russian");

const int num = 4000;

FILE\* fp;

fp = fopen("testbase4.dat", "rb");

basa\* mas[num];

basa\* mas\_orig[num];

basa mas2[num];

fread(mas2, sizeof(basa), 4000, fp);

fclose(fp);

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

mas[i] = &mas2[i];

mas\_orig[i] = &mas2[i];

}

qs(mas,0,num-1);

while(true){

int v;

system("cls");

SetConsoleCP(866);

cout << "1.Оригинальная база" << endl;

cout << "2.Отсортированная" << endl;

cout << "3.Бинарный поиск" << endl;

cout << "4.Кодирование" << endl;

cout << "5.Выход" << endl;

cin >> v;

if (v == 1 || v == 2 || v == 3 || v==4 || v==5) {

if (v == 1) {

print(mas\_orig);

}

if (v == 2){

print(mas);

}

if (v == 3){

char key[10];

SetConsoleCP(1251);

search(mas,0,num-1);

system("pause");

system("cls");

cout << endl << "TREE:" << endl << endl;

system("pause");

Obhod(tree);

cout << endl << "Type key for search:" << endl;

SetConsoleCP(866);

cin >> key;

SetConsoleCP(1251);

char \*spc = " ";

for(int i = strlen(key); i<9;i++){

strcat(key, spc);

}

if(strcmp(key,"0")!=0){

searchAVL(tree,key);

}

}

if(v==4){

FStart();

FPrint();

}

if(v==5){

return 0;

}

}

system("pause");

}

return 0;

}

void push\_queue(spis \*c, spis \*\*b)

{

if((\*b) != NULL)

{

spis \*temp = new spis;

temp = c;

temp->next = NULL;

(\*b)->next = temp;

(\*b) = temp;

}

else

{

(\*b) = new spis;

(\*b) = c;

(\*b)->next = NULL;

head = (\*b);

}

}

int compare(basa \*a, basa \*b)

{

if(strcmp(a->b, b->b) != 0){

return strcmp(a->b, b->b);

}

else if (strncmp(a->a, b->a,3) != 0) {

return strncmp(a->a, b->a,3);

}

return 0;

}

int compare\_date(basa \*a, char \*b)

{

if(strcmp(&a->e[6], &b[6]) != 0){

return strcmp(&a->e[6], &b[6]);

}

else if (a->e[3] - b[3] != 0) {

return a->e[3] - b[3];

}

else if (a->e[4] - b[4] != 0) {

return a->e[4] - b[4];

}

else if (a->e[0] - b[0] != 0) {

return a->e[0] - b[0];

}

else if (a->e[1] - b[1] != 0) {

return a->e[1] - b[1];

}

return 0;

}

void ll(vertex\* &point)

{

vertex \*current = point->left;

point->bal = 0;

current->bal = 0;

point->left = current->right;

current->right = point;

point = current;

}

void rr(vertex\* &point)

{

vertex \*current = point->right;

point->bal = 0;

current->bal = 0;

point->right = current->left;

current->left = point;

point = current;

}

void lr(vertex\* &point)

{

vertex \*current = point->left;

vertex \*reva = current->right;

if (reva->bal < 0) {

point->bal = 1;

}

else {

point->bal = 0;

}

if (reva->bal > 0) {

current->bal = -1;

}

else {

current->bal = 0;

}

reva->bal = 0;

current->right = reva->left;

point->left = reva->right;

reva->left = current;

reva->right = point;

point = reva;

}

void rl(vertex\* &point)

{

vertex \*current = point->right;

vertex \*reva = current->left;

if (reva->bal < 0) {

point->bal = 1;

}

else {

point->bal = 0;

}

if (reva->bal > 0) {

current->bal = -1;

}

else {

current->bal = 0;

}

reva->bal = 0;

current->left = reva->right;

point->right = reva->left;

reva->right = current;

reva->left = point;

point = reva;

}

void addAVL(spis \*p, vertex \*&point)

{

if (point == NULL) {

point = new vertex;

point->pole = p->data;

point->bal = 0;

point->left = NULL;

point->right = NULL;

point->next = NULL;

up = true;

}

else{

if (compare\_date(point->pole, p->data->e) > 0) {

addAVL(p, point->left);

if (up == true) {

if (point->bal > 0) {

point->bal = 0;

up = false;

}

else

if (point->bal == 0) {

point->bal = -1;

up = true;

}

else

if (point->left->bal < 0) {

ll(point);

up = false;

}

else {

lr(point);

up = false;

}

}

}

else if (compare\_date(point->pole, p->data->e) < 0) {

addAVL(p, point->right);

if (up == true) {

if (point->bal < 0) {

point->bal = 0;

up = false;

}

else {

if (point->bal == 0) {

point->bal = 1;

up = true;

}

else {

if (point->right->bal > 0) {

rr(point);

up = false;

}

else {

rl(point);

up = false;

}

}

}

}

}

else if (compare\_date(point->pole, p->data->e) == 0){

addAVL(p, point->next);

up = false;

}

}

}

void Obhod(vertex \*v) {

if(v==NULL){

return;

}

else {

Obhod(v->left);

cout << " " << v->pole->a << "\t" << v->pole->b << "\t" << v->pole->c << "\t" << v->pole->d << "\t" << v->pole->e << endl;

Obhod(v->next);

Obhod(v->right);

}

}

void searchAVL (vertex \*root, char \*data) {

if (root == NULL) {

return;

}

else if (compare\_date(root->pole, data) > 0){

searchAVL(root->left, data);

}

else if (compare\_date(root->pole, data) < 0){

searchAVL(root->right, data);

}

else{

cout << root->pole->a << "\t" << root->pole->b << "\t" << root->pole->c << "\t" << root->pole->d << "\t" << root->pole->e << endl;

searchAVL(root->next, data);

}

}

void print(basa\* mas[])

{

int v;

system("cls");

SetConsoleCP(1251);

cout << "1.Printout by 20" << endl;

cout << "2.4000 for 1 click" << endl;

cout << "3.No print" << endl;

cin >> v;

if (v == 1 or v == 2 or v == 3) {

if (v == 1) {

cout << "no 20" << endl;

int temp = 0;

int answer;

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

cout << i << "\t" << mas[i]->a << "\t" << mas[i]->b << "\t" << mas[i]->c << "\t" << mas[i]->d << "\t" << mas[i]->e << endl;

temp++;

if (temp % 20 == 0) {

cout << "Dalshe?" << endl << "1=da, 0=net" << endl;

cin >> answer;

if (answer == 1) {

continue;

}

if (answer == 0) {

cout << "Closing prog" << endl;

system("PAUSE");

break;

}

if ((answer != 1) && (answer != 0)) {

int s;

cout << "Missclick or u really wanna exit? " << "1 fo true, 0 fo no" << endl;

cin >> s;

if (s == 1){

break;

}

if (s == 0){

cout << "kk, np, moving next" << endl;

continue;

}

if ((s != 1 ) & (s != 0)){

cout << "u missed again. Closing." << endl;

break;

}

}

}

}

}

if (v == 2) {

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

cout << i << "\t" << mas[i]->a << "\t" << mas[i]->b << "\t\t" << mas[i]->c << "\t" << mas[i]->d << "\t" << mas[i]->e << endl;

if (kbhit())

if (getch() == 13) {

cout << "Stopped " << endl << "Moving next? (Double enter for true)" << endl;

cout << "To exit press esc" << endl;

if (getch() == 27) {

break;

}

if (getch() == 13) {

continue;

}

if ((getch() != 13) && (getch() != 27)) {

int s;

cout << "Missclick or u really wanna exit? " << "1 fo true, 0 fo no" << endl;

cin >> s;

if (s == 1){

break;

}

if (s == 0){

cout << "kk, np, moving next" << endl;

continue;

}

if ((s != 1) && (s != 0)){

break;

}

}

}

if (i == 3999) {

cout << "End of file" << endl;

}

}

}

}

SetConsoleCP(866);

}

void qs(basa\* mas[], int left, int right)

{

register int i, j;

basa \*x;

basa \*temp;

i = left; j = right;

x = mas[(left+right)/2];

do {

while((compare(mas[i],x) < 0) && (i < right)) i++;

while((compare(mas[j],x) > 0) && (j > left)) j--;

if(i <= j) {

temp = mas[i];

mas[i] = mas[j];

mas[j] = temp;

i++; j--;

}

} while(i <= j);

if(left < j) qs(mas, left, j);

if(i < right) qs(mas, i, right);

}

void shanon()

{

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

{

A[i].q = A[i - 1].q + A[i].p;

A[i].l = ceil(-log(A[i].p) / log(2));

}

for (int i = 1; i < v; i++)

{

for (int j = 1; j <= A[i].l; j++)

{

A[i - 1].q \*= 2;

code[i][j] = floor(A[i - 1].q);

while (A[i - 1].q >= 1)

A[i - 1].q -= 1;

}

}

}

void FPrint(){

SetConsoleCP(866);

printf("\n\nКод Фано: \n");

printf("| Номер Символа | Символ | Вероятность | Кодовое слово | Длина кодового|\n");

printf("| | | | | слова |\n");

SetConsoleCP(1251);

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

{

printf("| %12d | %c | %2.6f | ",i, A[i].a, A[i].q);

for (int j = 1; j <= A[i].l; j++)

printf("%d", code[i][j]);

for (int j = A[i].l + 1; j < 18; j++)

printf(" ");

printf(" | %7d |\n", A[i].l);

length += A[i].p \* A[i].l;

}

SetConsoleCP(866);

printf("\n| Энтропия | Средняя длина |\n");

printf("| %12f | %10.5f |\n", entropy, length);

SetConsoleCP(1251);

}

void FStart(){

int i,j;

FILE \*fp;

fp=fopen("testBase4.dat", "rb");

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

A[i].p = 0;

A[i].l = 0;

A[i].q = 0;

A[i].a = (char)(i-128);

}

while (!feof(fp)) {

char c;

fscanf(fp, "%c", &c);

if (feof(fp))

break;

A[c+128].p +=1;

sum++;

}

printf("\n");

fclose(fp);

bool flag = true;

while (flag)

{

flag = false;

for (int i = 1; i < M; i++)

{

if (A[i - 1].p < A[i].p)

{

SHANON Buf = A[i-1];

A[i-1]=A[i];

A[i]=Buf;

flag = true;

}

}

}

for (i = 0; i < M, A[i].p != 0; i++){

A[i].p /=sum;

v++;

entropy += A[i].p \* abs(log(A[i].p) / log(2));

}

shanon();

}

void search(basa \*mas[], int left, int right){

basa \*Que[4000];

spis \*qu = new spis;

int v = 0, mid = 0, find = 0;

cout << "Biinary search (street): ";

char b[3];

SetConsoleCP(866);

cin >> b;

SetConsoleCP(1251);

for (v = 0; v < 4000; v++) {

while ( left < right ) {

mid = (left + right) / 2;

if (strncmp(b, mas[v]->b,3) < 0) left = mid + 1;

else right = mid;

}

if (strncmp(b, mas[v]->b,3) == 0) {

Que[find] = mas[v];

find +=1;

qu->data = mas[v];

push\_queue(qu, &tail);

qu->next = new spis;

qu = qu->next;

}

}

cout << "Founded " << find << " positions" << endl;

system("pause");

tree = NULL;

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

if (head != NULL){

cout << head->data->a << "\t" << head->data->b << "\t" << head->data->c << "\t" << head->data->d << "\t" << head->data->e << endl;

addAVL(head, tree);

head = head->next;

}

else cout << "kkk" << endl;

}

}

# Результаты

Рисунок 1. Неотсортированная база данных.

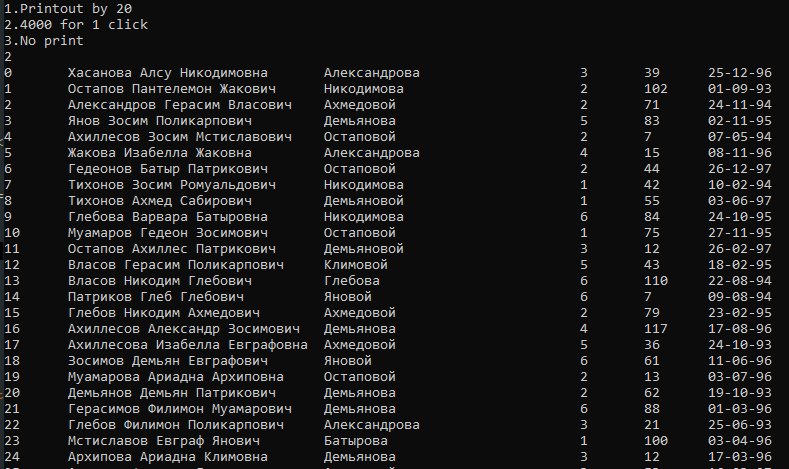


Рисунок 2. Отсортированная база данных по улице и ФИО.

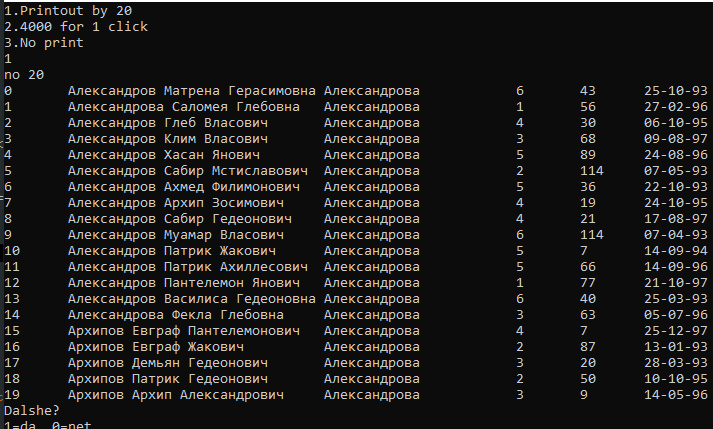


Рисунок 3. Бинарный поиск по улице

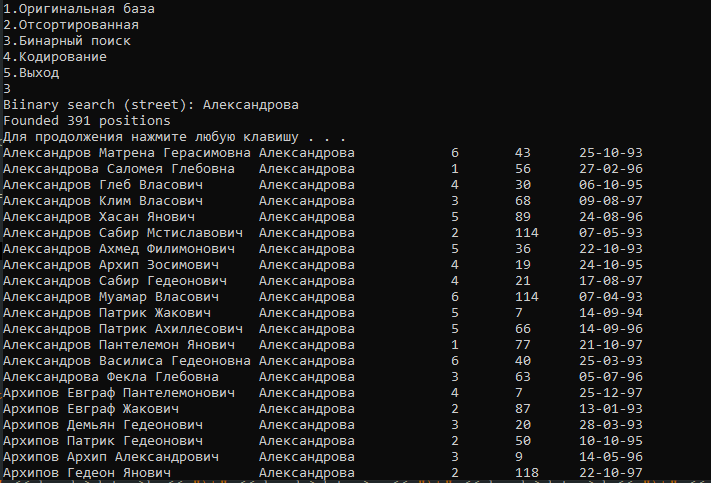


Рисунок 4. Дерево, ключ в дереве – дата.

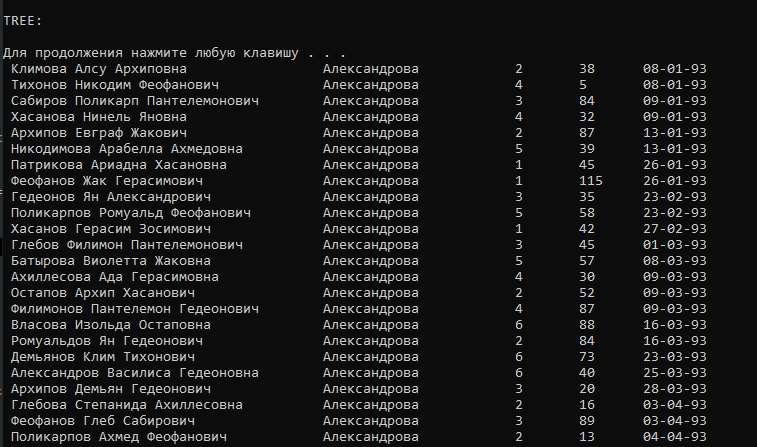


Рисунок 5. Поиск по дереву (элементы с одинаковым ключом).

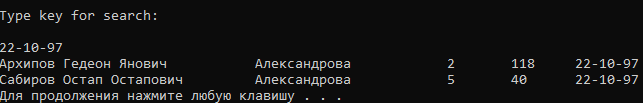
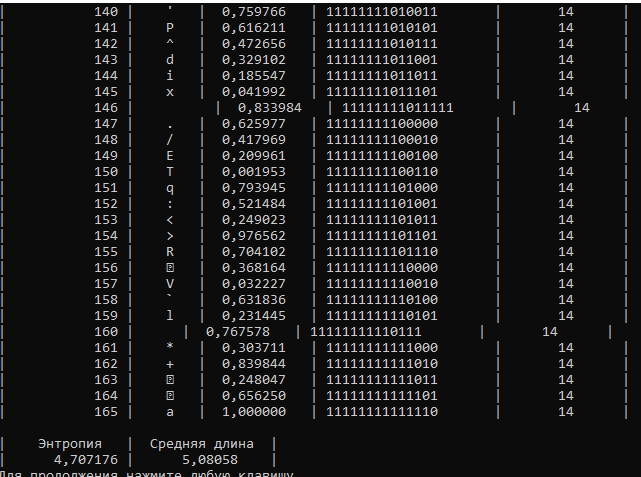


Рисунок 6. Кодовые слова, энтропия и средняя длина кодового слова.



# Выводы

В ходе выполнения курсовой работы были выполнены все поставленные задачи и реализованы необходимые алгоритмы: сортировки, поиска, построения АВЛ-дерева, поиска по дереву и кодирование базы данных.

В результате кодирования были получены данные, подтверждающие теоретические сведения. К таковым относятся: величины средней длины кодового слова и энтропии (Lср ≤ H + 1).

Четкая структуризация кода и грамотно подобранные имена переменных, структур данных, функций и процедур способствуют удобочитаемости программы.

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