Федеральное агентство связи

Сибирский государственный университет телекоммуникаций и информатики

Кафедра прикладной математики и кибернетики

Курсовой проект

«Структуры и алгоритмы обработки данных»

Вариант

150

Выполнил: студент Бухаркин Илья группы ИП-217

Проверил: ассистент кафедры ПМиК

Новосибирск 2021

Оглавление

[1. Постановка задачи 2](#_Toc64501734)

[2. Основные идеи и характеристики применяемых методов 3](#_Toc64501735)

[2.1. Метод сортировки 3](#_Toc64501736)

[2.2. Двоичный поиск 3](#_Toc64501737)

[2.3. Списки и очереди 4](#_Toc64501738)

[2.4. Дерево и поиск по нему 4](#_Toc64501739)

[2.5. Метод кодирования 6](#_Toc64501740)

[4. Описание программы 10](#_Toc64501741)

[4.1. Основные переменные и структуры 10](#_Toc64501742)

[4.2. Описание подпрограмм 11](#_Toc64501743)

[5. Текст программы 12](#_Toc64501744)

[6. Результаты 28](#_Toc64501745)

[7. Выводы 31](#_Toc64501746)

# Постановка задачи

Хранящуюся в файле 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 двоичное Б-дерево поиска по ключу, 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 "Обманутые вкладчики"

Стpуктуpа записи:

ФИО вкладчика: текстовое поле 30 символа

фоpмат <Фамилия>\_<Имя>\_<Отчество>

Сумма вклада: целое число (использовать unsigned short int)

Дата вклада: текстовое поле 10 символов

фоpмат дд-мм-гг

ФИО адвоката: текстовое поле 22 символа

формат <Фамилия>\_<буква>\_<буква>

Пpимеp записи из БД:

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

130

15-03-46

Иванова\_И\_В\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ваpианты условий упоpядочения и ключи поиска (К):

по фио и сумме вклада, К = первые три буквы фамилии. Ключ в дереве = ФИО адвоката

# 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

## Списки и очереди

*Списком* называется последовательность однотипных элементов, связанных между собой указателями. Будем считать, что элементы списка имеют тип tLE, указатели на элементы списка имеют тип pLE.

X: tLE p: pLE

|  |
| --- |
| Next |
| Data |

Рисунок 13 Указатель на элемент списка

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. При добавлении в *очередь* новый элемент ставится в конец списка, удаляется первый элемент последовательности.

Рассмотрим основные операции со стеком и очередью. Для работы со стеком необходимо иметь указатель на начало списка. Обозначим его Head. При работе с очередью требуется дополнительный указатель на конец очереди. Обозначим его Tail. Иногда при работе с очередью удобно объединять указатели Head и Tail в виде полей некоторой переменной Queue.

## Дерево и поиск по нему

**А2-дерево**

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).

Второй приближённый алгоритм (А2) использует предварительно упорядоченный набор вершин. В качестве корня выбирается такая вершина, что разность весов левого и правого поддеревьев была минимальна. Для этого путем последовательного суммирования весов определим вершину Vk, для которой справедливы неравенства:

 и .

Тогда в качестве "центра тяжести" может быть выбрана вершина Vk, Vk-1 или Vk+1, т. е. вершина, для которой разность весов левого и правого поддерева минимальна. Далее действия повторяются для каждого поддерева.

## 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 теле функции Menu. Для выбора пунктов меню используется функция switch( ).

1. *Загрузка и вывод базы данных*

База данных открывается в Menu и считывается в список с помощью метода ReadBase. 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 не отсортированный базы данных отвечает процедура void Print(list \*base) 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 QuickSort(list\* mas[], int left, int right, int sort). Она сортирует список по полям депозита и ФИО гражданина. Для быстрого доступа к отсортированной и неотсортированной базе данных, перед вызовом процедуры сортировки, делается индексный массив оригинальной базы данных, который и сортируется.

1. *Особенности реализации бинарного поиска*

Бинарный поиск по отсортированной базе данных осуществляется в функции int BSearch (list \*\*A, char \*FIO). Результатом работы является индекс элемента в массиве, удовлетворяющий условию ключа.

1. *Вспомогательные функции и процедуры для построения А2-дерева.*

Построение дерева осуществляется в процедуре void A2(int L, int R, list \*\*mas). Записи заносятся в дерево в процедуре void add\_vertex(vertex \*&p, list \*\*mas, int w). Обход дерева слева направо осуществляется в процедуре void LRprint(vertex \*x). Поиск по дереву выполняется в процедуре void TreeSearch(vertex \*p, char \*data).

1. *Кодирование данных*

Кодирование данных начинается с процедуры void Decode(), 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.

Затем в процедур void Encode() база данных вновь прочитывается посимвольно и кодируется в новую, с помощью уже полученного кодового алфавита. В процедуре void CodePrint() 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 country

{

char FIO[32];

char street[18];

short int number\_home;

short int number\_appart;

char Date[10];

};

Структура для хранения элемента базы данных. Всего таких элементов:

const int N = 4000;

struct list {

country \*data;

list\* prev;

list\* next;

};

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

struct vertex {

record \*data;

vertex \*next;

int w;  
 int h;

vertex \*left;

vertex \*right;

};

Структура, представляющая дерево оптимального поиска (А2).

struct SN\_code {

float p;

float q;

int l;

char a;

};

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

SN\_code A[M]; - массив для символов

const int M = 256 – число символов в алфавите

float entropy = 0 - энтропия

float lgm = 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 Read\_base(FILE \*fp, list \*base) – прочитывание базы данных
2. void copy\_base(list\*, list\*) – дублирование прочитанной БД
3. void Print(list \*Base)- визуальный вывод списка на консоль.

Процедуры сортировки:

1. void QuickSort(list\* mas[], int left, int right, int sort) – сортирует базу данных по улице и ФИО гражданина.
2. int compare\_street(char\* a,char\* b) – сравнение полей улиц
3. int compare\_name(char\* a, char\* b) – сравнение полей имён
4. void Print\_sort\_index\_mas(list \*\*mas) – визуальный вывод отсортированного индексного массива на консоль

Процедуры и функции для поиска в отсортированной базе данных:

1. int BSearch (list \*\*A, char\* depositor) - бинарный поиск по 3 буквам фамилии.

Процедуры построения дерева оптимального поиска (А2):

1. void add\_vertex(vertex \*&point, list \*mas) – добавление элемента в дерево.
2. void A2(int L, int R, list \*\*mas) – построение дерева оптимального поиска, приближенный алгоритм А2.
3. void LRprint(vertex \*x) – вывод дерева.
4. void TreeSearch(vertex \*p, char \*data) – поиск в дереве.

Процедуры и функции кодирования базы данных:

1. void Decode()– считывание символов базы данных.
2. void shanon() – подсчет вероятностей символов, преобразование алфавита и создание кодовых слов.
3. void Encode() – закодирование базы данных
4. void CodePrint()– вывод статистики и алфавита с вероятностями.

# Текст программы

#include <fstream>

#include <iostream>

#include <conio.h>

#include <Windows.h>

#include <iomanip>

#include <cstdio>

#include <cstring>

#include <cmath>

#include <iomanip>

#include <cstdlib>

using namespace std;

const int N = 4000;

const int M = 256;

int sum = 0;

int code[M][M];

float entropy = 0, lgm = 0;

int sim\_cf = 0, sim\_f = 0, sym\_count = 0;

int fcompression = 0, cfcompression = 0;

int \*W;

struct base

{

char depositor[30];

unsigned short int deposit;

char date[10];

char lawyer[22];

};

struct list

{

base \*data;

list \*next;

list \*prev;

};

struct Vertex

{

base \*data;

int w;

int h;

Vertex \*Next;

Vertex \*Left;

Vertex \*Right;

};

Vertex \*root = NULL;

struct SN\_code

{

float p;

float q;

int l;

char a;

};

SN\_code A[M];

int Menu();

void Read\_base(FILE \*fp, list \*base);

void copy\_base(list \*, list \*);

void Print(list \*Base);

void Print\_sort\_index\_mas(list \*\*mas);

void quick\_sort(list \*&head);

void perseption(list \*h1, list \*t1, list \*h2, list \*t2, list \*&hr, list \*&tr, int sort);

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

int compare\_name(char \*a, char \*b);

int BSearch(list \*\*A, char \*date);

void A2(int L, int R, list \*\*mas);

void seth(Vertex \*p);

void LR\_print(Vertex \*p, int &count);

void TreeSearch(Vertex \*p, char \*data);

int size(Vertex \*);

int height(Vertex \*);

int max(int, int);

int sdp(Vertex \*, int);

void shanon();

void Code();

void code\_file();

void CodePrint();

int Menu()

{

system("cls");

setlocale(LC\_ALL, "Russian");

FILE \*fp;

int search\_start = 0;

int search = N - 1;

fp = fopen("testBase3.dat", "rb");

list \*OriginBase = new list;

list \*SortBase = new list;

Read\_base(fp, OriginBase);

fclose(fp);

copy\_base(OriginBase, SortBase);

quick\_sort(SortBase);

list \*mas[N];

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

{

mas[i] = SortBase;

SortBase = SortBase->next;

}

W = new int[N];

Code();

code\_file();

int SoD;

cout << "1.Standard list\n2.Sorted list\n3.BSearch\n4.TreePrint\n5.Coding\n6.Exit" << endl;

cin >> SoD;

switch (SoD)

{

case 1:

{

Print(OriginBase);

break;

}

case 2:

{

Print\_sort\_index\_mas(mas);

break;

}

case 3:

{

cout << endl;

cout << "Enter Fio" << endl;

char date[22];

SetConsoleCP(866);

cin >> date;

SetConsoleCP(1251);

if (strcmp(date, "0") != 0)

{

search = BSearch(mas, date);

int fam\_pos = 0;

if (search == -1)

{

cout << "This Fio doesn't exists''" << endl;

}

else

{

do

{

if (search == 0)

{

break;

}

else

{

search--;

}

if (strncmp(mas[search]->data->depositor, date, 3) != 0)

{

search++;

break;

}

} while (1);

search\_start = search;

do

{

search++;

if (search == N)

{

search--;

break;

}

if (strncmp(mas[search]->data->depositor, date, 3) != 0)

{

break;

}

} while (1);

cout << endl

<< endl

<< "Founded " << search - search\_start << " pozitions (" << search\_start << " - " << search - 1 << ")" << endl;

for (int i = search\_start; i < search; i++)

{

cout << i << " " << mas[i]->data->depositor << "\t" << mas[i]->data->deposit << "\t" << mas[i]->data->date

<< "\t" << mas[i]->data->lawyer << endl;

W[i] = rand() % 99 + 1;

}

A2(search\_start, search, mas);

root->h = 1;

seth(root);

}

}

break;

}

case 4:

{

int count = 0;

char pos[22];

char \*spc = " ";

SetConsoleCP(1251);

LR\_print(root, count);

SetConsoleCP(866);

cout << endl;

printf("+------+----------+----------+----------------+\n");

printf("|%6d| UniqSize | Height | Mid. height |\n", N);

printf("+------+----------+----------+----------------+\n");

printf("| A2 |%10d|%10d|%16.2f|\n", size(root), height(root), (double)sdp(root, 1) / size(root));

printf("+------+----------+----------+----------------+\n");

cout << endl

<< "input element fo find: " << endl;

SetConsoleCP(866);

cin.sync();

cin.getline(pos, 22);

SetConsoleCP(1251);

for (int i = strlen(pos); i < 21; i++)

{

strcat(pos, spc);

}

if (strcmp(pos, "0") != 0)

{

TreeSearch(root, pos);

}

\_getch();

break;

}

case 5:

{

CodePrint();

break;

}

case 6:

{

system("PAUSE");

return 0;

}

default:

{

cout << "Invalid number entered\nPlease enter numder again" << endl;

system("PAUSE");

system("cls");

break;

}

}

\_getch();

Menu();

return 0;

}

int main()

{

Menu();

return 0;

}

void CodeBase()

{

FILE \*fp, \*fcoded;

fp = fopen("testBase3.dat", "rb");

fcoded = fopen("BaseCoded.dat", "wb");

char buffer;

while (!feof(fp))

{

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

fcompression++;

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

{

if (buffer == (char)(i - 128))

{

for (int j = 0; j < A[i].l; j++)

{

putc(code[i][j], fcoded);

cfcompression++;

}

}

}

}

fclose(fp);

fclose(fcoded);

}

void CodePrint()

{

lgm = 0;

SetConsoleCP(866);

printf("\n\nshenon: \n");

printf("-------------------------------------------------------------------------------\n");

printf("| num element | symbol | veroyatnost | binary code| dlina coda |\n");

printf("| | | | | |\n");

SetConsoleCP(1251);

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

{

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

for (int j = 0; j < A[i].l + 1; j++)

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

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

printf(" ");

printf(" | %7d |\n", A[i].l + 1);

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

}

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("| entropy | avarage dlina coda | koof sjatia |\n");

printf("|\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\n");

printf("| %10f | %10.5f | %10.5f |\n", entropy, lgm, (float)sim\_f / sim\_cf);

printf("|\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\n");

cout << endl

<< endl

<< entropy + 1 << " > " << lgm << endl

<< endl;

SetConsoleCP(1251);

}

void Code()

{

int i, j;

FILE \*f;

f = fopen("testBase3.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(f))

{

char c;

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

if (feof(f))

break;

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

sum++;

}

fclose(f);

printf("\n");

bool b = true;

while (b)

{

b = false;

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

{

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

{

SN\_code B = A[i - 1];

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

A[i] = B;

b = true;

}

}

}

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

{

A[i].p /= sum;

A[i].q = A[i].p;

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

sym\_count++;

}

shanon();

}

void code\_file()

{

FILE \*f, \*cf;

f = fopen("testBase3.dat", "rb");

cf = fopen("CodeBase.dat", "wb");

char sim;

while (!feof(f))

{

fscanf(f, "%c", &sim);

sim\_f++;

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

{

if (sim == A[i].a)

{

for (int j = 0; j < A[i].l; j++)

{

putc(code[i][j], cf);

sim\_cf++;

}

}

}

}

\_fcloseall();

}

void shanon()

{

for (int i = 0; i < M && A[i].p != 0; 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 < M && A[i].p != 0; 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 LR\_print(Vertex \*p, int &count)

{

if (p != NULL)

{

LR\_print(p->Left, count);

cout << count + 1 << ") " << p->data->depositor << "\t" << p->data->deposit << "\t" << p->data->date << "\t" << p->data->lawyer << endl;

count++;

LR\_print(p->Next, count);

LR\_print(p->Right, count);

}

}

void add\_vertex(Vertex \*&p, list \*mas, int w)

{

if (p == NULL)

{

p = new Vertex;

p->data = mas->data;

p->w = w;

p->Next = NULL;

p->Left = NULL;

p->Right = NULL;

}

else if (strncmp(p->data->depositor, mas->data->depositor, 3) == 0)

{

add\_vertex(p->Next, mas, w);

}

else if (strncmp(p->data->depositor, mas->data->depositor, 3) > 0)

{

add\_vertex(p->Left, mas, w);

}

else if (strncmp(p->data->depositor, mas->data->depositor, 3) < 0)

{

add\_vertex(p->Right, mas, w);

}

}

void A2(int L, int R, list \*\*mas)

{

int wes = 0, sum = 0;

int i;

if (L <= R)

{

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

{

wes = wes + W[i];

}

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

{

if ((sum < (wes / 2)) && (sum + W[i]) > (wes / 2))

{

break;

}

sum = sum + W[i];

}

add\_vertex(root, mas[i], W[i]);

A2(L, i - 1, mas);

A2(i + 1, R, mas);

}

}

void TreeSearch(Vertex \*p, char \*name)

{

if (p != NULL)

{

TreeSearch(p->Left, name);

int comparison = strcmp(p->data->lawyer, name);

if (comparison == 0)

{

cout << p->data->depositor << "\t" << p->data->deposit << "\t" << p->data->date << "\t" << p->data->lawyer << endl;

}

TreeSearch(p->Next, name);

TreeSearch(p->Right, name);

}

}

void seth(Vertex \*p)

{

if (p)

{

if (p->Next)

{

p->Next->h = p->h + 1;

}

if (p->Left)

{

p->Left->h = p->h + 1;

}

if (p->Right)

{

p->Right->h = p->h + 1;

}

seth(p->Left);

seth(p->Right);

seth(p->Next);

}

}

void Read\_base(FILE \*fp, list \*rbase)

{

base \*struk = new base();

fread((base \*)struk, sizeof(base), 1, fp);

rbase->data = struk;

rbase->prev = NULL;

rbase->next = NULL;

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

{

struk = new base();

list \*rbase\_prev;

fread((base \*)struk, sizeof(base), 1, fp);

rbase\_prev = rbase;

rbase = rbase->next = new list();

rbase->data = struk;

rbase->prev = rbase\_prev;

rbase->next = NULL;

}

}

int BSearch(list \*\*A, char \*FIO\_prefix)

{

int l = 0, r = N, m = 0;

while (l < r)

{

m = (l + r) / 2;

if (strncmp(A[m]->data->depositor, FIO\_prefix, 3) == 0)

{

return m;

}

if (strncmp(A[m]->data->depositor, FIO\_prefix, 3) < 0)

l = m + 1;

else

r = m - 1;

}

return -1;

}

int compare\_name(char \*a, char \*b)

{

char bufferA[30];

char bufferB[30];

strcpy(bufferA, a);

strcpy(bufferB, b);

int space\_pos\_a = strchr(a, ' ') - a;

int space\_pos\_b = strchr(b, ' ') - b;

int count = 4;

if (space\_pos\_a < space\_pos\_b)

{

count = space\_pos\_a;

}

else

{

count = space\_pos\_b;

}

count -= 1;

if (strncmp(bufferA, bufferB, count) > 0)

{

return 1;

}

if (strncmp(bufferA, bufferB, count) < 0)

{

return -1;

}

if (space\_pos\_a > space\_pos\_b)

{

return 1;

}

if (space\_pos\_a < space\_pos\_b)

{

return -1;

}

strcpy(bufferA, a);

strcpy(bufferB, b);

bufferA[space\_pos\_a] = 'a';

bufferB[space\_pos\_b] = 'a';

space\_pos\_a = strchr(bufferA, ' ') - bufferA;

space\_pos\_b = strchr(bufferB, ' ') - bufferB;

count = 4;

if (space\_pos\_a < space\_pos\_b)

{

count = space\_pos\_a;

}

else

{

count = space\_pos\_b;

}

count -= 1;

if (strncmp(bufferA, bufferB, count) > 0)

{

return 1;

}

if (strncmp(bufferA, bufferB, count) < 0)

{

return -1;

}

if (space\_pos\_a > space\_pos\_b)

{

return 1;

}

if (space\_pos\_a < space\_pos\_b)

{

return -1;

}

bufferA[space\_pos\_a] = 'a';

bufferB[space\_pos\_b] = 'a';

space\_pos\_a = strchr(bufferA, ' ') - bufferA;

space\_pos\_b = strchr(bufferB, ' ') - bufferB;

count = 4;

if (space\_pos\_a < space\_pos\_b)

{

count = space\_pos\_a;

}

else

{

count = space\_pos\_b;

}

count -= 1;

if (strncmp(bufferA, bufferB, count) > 0)

{

return 1;

}

if (strncmp(bufferA, bufferB, count) < 0)

{

return -1;

}

if (space\_pos\_a >= space\_pos\_b)

{

return 1;

}

return -1;

}

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

{

int res = strcmp(a->depositor, b->depositor);

if (res > 0)

{

return 1;

}

else if (res < 0)

{

return -1;

}

else

{

if (a->deposit > b->deposit)

{

return 1;

}

else if (a->deposit < b->deposit)

{

return -1;

}

else

{

return 0;

}

}

}

void perseption(list \*h1, list \*t1, list \*h2, list \*t2, list \*&hr, list \*&tr)

{

if (compare(h1->data, h2->data) < 0)

{

hr = h1;

h1 = h1->next;

}

else

{

hr = h2;

h2 = h2->next;

}

tr = hr;

while (t1->next != h1 && t2->next != h2)

{

if (compare(h1->data, h2->data) < 0)

{

tr->next = h1;

h1 = h1->next;

tr = tr->next;

}

else

{

tr->next = h2;

h2 = h2->next;

tr = tr->next;

}

}

if (t1->next != h1)

{

tr->next = h1;

tr = t1;

}

else if (t2->next != h2)

{

tr->next = h2;

tr = t2;

}

tr->next = NULL;

}

void quick\_sort(list \*&head)

{

list \*t = new list;

t->data = NULL;

t->next = head;

int k = 1;

int e = 0;

list \*hp, \*tp, \*h1, \*t1, \*h2, \*t2, \*hr, \*tr;

while (k < 4000)

{

hp = t;

while (hp != NULL)

{

if (hp->next == NULL)

{

break;

}

h1 = hp->next;

t1 = h1;

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

{

if (t1->next == NULL)

{

break;

}

t1 = t1->next;

}

if (t1->next == NULL)

{

break;

}

h2 = t1->next;

t2 = h2;

t1->next = NULL;

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

{

if (t2->next == NULL)

{

break;

}

t2 = t2->next;

}

tp = t2->next;

t1->next = NULL;

t2->next = NULL;

perseption(h1, t1, h2, t2, hr, tr);

hp->next = hr;

head = hr;

tr->next = tp;

hp = tr;

}

k \*= 2;

}

}

void Print(list \*Base)

{

int i;

char n;

SetConsoleCP(1251);

cout << "1 for 4000 / 2 for 20 per click/ 3 for nothing" << endl;

cin >> n;

if (n == '1')

{

i = 0;

while (i < N)

{

cout << i + 1 << "\t" << Base->data->depositor << "\t" << Base->data->deposit << "\t" << Base->data->date << "\t" << Base->data->lawyer << endl;

i++;

if (int result = \_kbhit())

{

printf("\nKey '%c' was pressed. Press 'e' key to exit.\n", \_getch());

char something;

cin >> something;

if (something == 'e')

{

return;

}

}

Base = Base->next;

}

}

if (n == '2')

{

i = 0;

int g = 0;

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

{

while ((g++) < 20)

{

cout << i + g << "\t" << Base->data->depositor << "\t" << Base->data->deposit << "\t" << Base->data->date << "\t" << Base->data->lawyer << endl;

Base = Base->next;

}

if (g >= 20)

{

cout << "y for continue" << endl;

cin >> n;

if (n == 'y')

g = 0;

else

return;

}

}

}

}

void Print\_sort\_index\_mas(list \*\*mas)

{

int i;

char n;

SetConsoleCP(1251);

cout << "1 for 4000 / 2 for 20 per click/ 3 for nothing" << endl;

cin >> n;

if (n == '1')

{

i = 0;

while (i < N)

{

cout << i + 1 << "\t" << mas[i]->data->depositor << "\t" << mas[i]->data->deposit << "\t" << mas[i]->data->date << "\t" << mas[i]->data->lawyer << endl;

i++;

if (int result = \_kbhit())

{

printf("\nKey '%c' was pressed. Press 'e' key to exit.\n", \_getch());

char something;

cin >> something;

if (something == 'e')

{

return;

}

}

}

}

if (n == '2')

{

i = 0;

int g = -1;

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

{

while ((g++) < 20)

{

cout << i + g << "\t" << mas[i + g]->data->depositor << "\t" << mas[i + g]->data->deposit << "\t" << mas[i + g]->data->date << "\t" << mas[i + g]->data->lawyer << endl;

}

if (g >= 20)

{

cout << "y for continue" << endl;

cin >> n;

if (n == 'y')

g = 0;

else

return;

}

}

}

}

void copy\_base(list \*a, list \*b)

{

b->prev = NULL;

b->data = a->data;

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

{

a = a->next;

b->next = new list;

b->next->prev = b;

b = b->next;

b->data = a->data;

}

b->next = NULL;

}

int size(Vertex \*x)

{

if (x == NULL)

{

return 0;

}

else

{

return 1 + size(x->Left) + size(x->Right);

}

}

int max(int x, int y)

{

if (x > y)

return x;

return y;

}

int height(Vertex \*x)

{

if (x == NULL)

{

return 0;

}

else

{

return 1 + max(height(x->Left), height(x->Right));

}

}

int sdp(Vertex \*x, int l)

{

if (x == NULL)

{

return 0;

}

else

{

return l + sdp(x->Left, l + 1) + sdp(x->Right, l + 1);

}

}

# Результаты

Рисунок 1. Неотсортированная база данных.

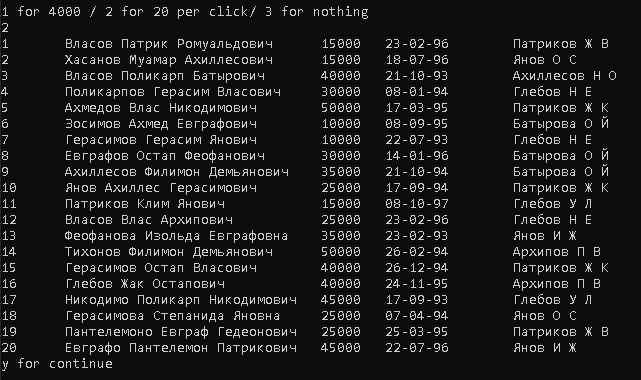


Рисунок 2. Отсортированная база данных по улице и ФИО.

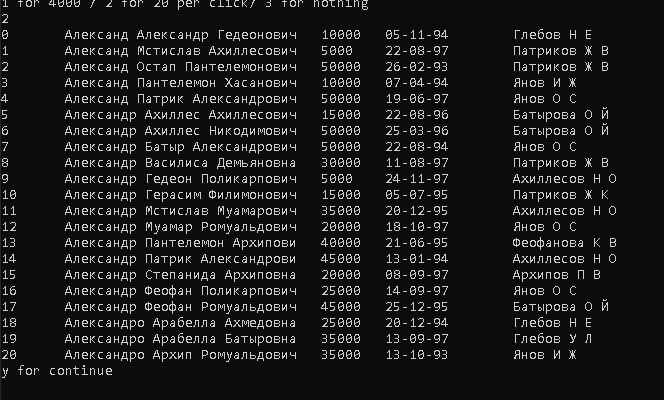


Рисунок 3. Бинарный поиск по ФИО

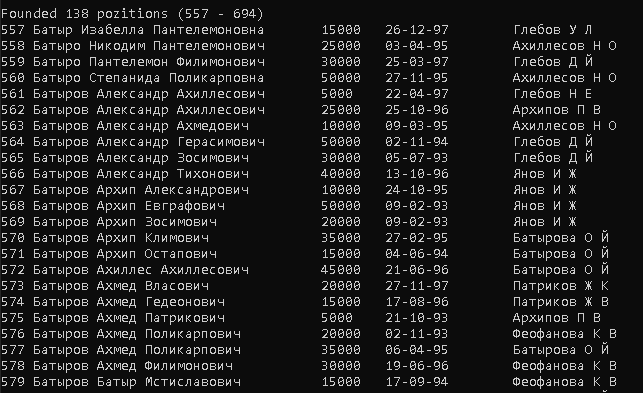


Рисунок 4. Дерево, ключ в дереве – первые 3 буквы ФИО.

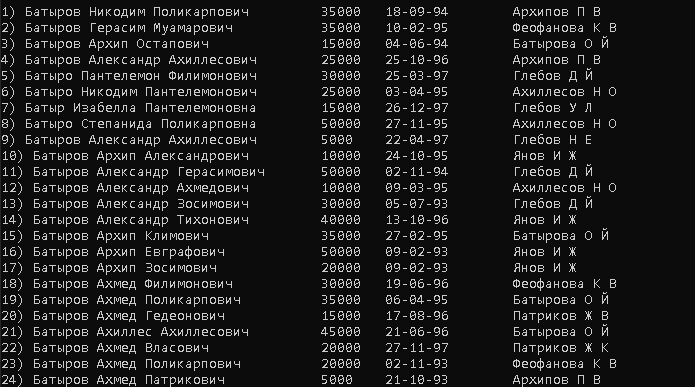


Рисунок 5. Поиск по дереву (элементы с одинаковым ключом).

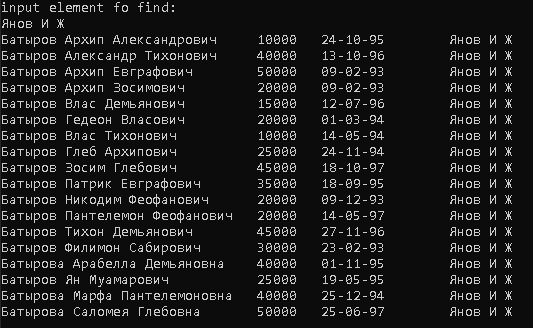
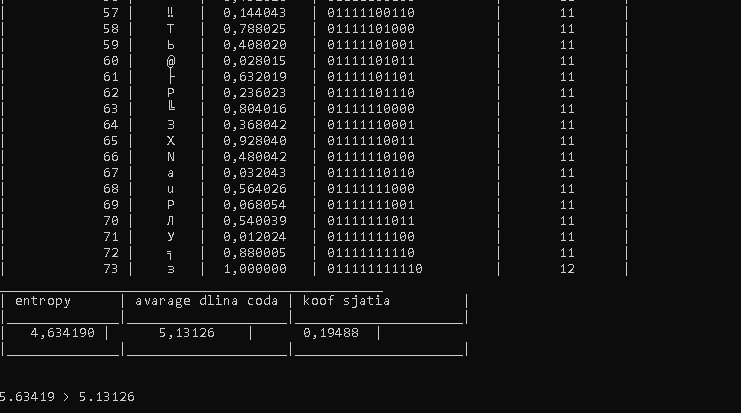


Рисунок 6. Кодовые слова, энтропия и средняя длина кодового слова.



# Выводы

В ходе выполнения курсовой работы были выполнены все поставленные задачи и реализованы необходимые алгоритмы: сортировки, поиска, построения А2-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.