Федеральное агентство связи

Сибирский государственный университет телекоммуникаций и информатики

Кафедра прикладной математики и кибернетики

Курсовой проект

«Структуры и алгоритмы обработки данных»

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 "Жизнь замечательных людей" 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 метода прямого слияния (Merge 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 (приближенный алгоритм А2) по автору, вывести на экран содержимое дерева и предусмотреть возможность поиска в дереве по запросу.

Закодировать файл базы данных кодом Фано, 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.

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):

по фамилиям(!) замечательных (!) людей, К = тpи пеpвые буквы фамилии.

Ключ в дереве – автор

# Основные идеи и характеристики применяемых методов

## Метод сортировки

Метод прямого слияния (Merge Sort)

В основе метода прямого слияния лежит операция слияния серий. *р-серией* называется упорядоченная последовательность из *р* элементов.

Пусть имеются две упорядоченные серии a и b длины q и r соответственно. Необходимо получить упорядоченную последовательность с, которая состоит из элементов серий a и b. Сначала сравниваем первые элементы последовательностей a и b. Минимальный элемент перемещаем в последовательность с. Повторяем действия до тех пор, пока одна из последовательностей a и b не станет пустой, оставшиеся элементы из другой последовательности переносим в последовательность с. В результате получим (*q+r*)-серию.

Для алгоритма слияния серий с длинами q и r необходимое количество сравнений и перемещений оценивается следующим образом

min (*q, r*) ≤ C ≤ *q+r-1*, M=*q+r*

Пусть длина списка S равна степени двойки, т.е. 2k, для некоторого натурального k. Разобьем последовательность S на два списка a и b, записывая поочередно элементы S в списки а и b. Сливаем списки a и b с образованием двойных серий, то есть одиночные элементы сливаются в упорядоченные пары, которые записываются попеременно в очереди c0 и c1. Переписываем очередь c0 в список a, очередь c1 – в список b. Вновь сливаем a и b с образованием серий длины 4 и т. д. На каждом итерации размер серий увеличивается вдвое. Сортировка заканчивается, когда длина серии превысит общее количество элементов в обоих списках. Если длина списка S не является степенью двойки, то некоторые серии в процессе сортировки могут быть короче.

При инициализации очереди обнуляются указатели, указывающие на начало и на конец очереди, т.е. очередь становится пустой.

Трудоёмкость методапрямого слияния определяется сложностью операции слияния серий. На каждой итерации происходит ровно n перемещений элементов списка и не более *n* сравнений. Как нетрудно видеть, количество итераций равно ******. Тогда

C < *n*, M=*n*+*n.*

Дополнительные *n* перемещений происходят во время начального расщепления исходного списка. Асимптотические оценки для М и С имеют следующий вид

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

Метод обеспечивает устойчивую сортировку.При реализации для массивов, метод требует наличия второго вспомогательного массива, равного по размеру исходному массиву. При реализации со списками дополнительной памяти не требуется.

## 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

ДОП А2

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 Фано

Метод Фано построения префиксного почти оптимального кода, для которого , заключается в следующем. Упорядоченный по убыванию вероятностей Cписок букв алфавита источника делится на две части так, чтобы суммы вероятностей букв, входящих в эти части, как можно меньше отличались друг от друга. Буквам первой части приписывается 0, а буквам из второй части – 1. Далее также поступают с каждой из полученных частей. Процесс продолжается до тех пор, пока весь список не разобьется на части, содержащие по одной букве.

**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 7 и на рисунке 6.

Таблица 7 Код Фано

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *ai* | *Pi* | кодовое слово | | | | *Li* |
| *a1* | 0.36 | **0** | **0** |  |  | 2 |
| *a2* | 0.18 | **0** | **1** |  |  | 2 |
| *a3* | 0.18 | **1** | **0** |  |  | 2 |
| *a4* | 0.12 | **1** | **1** | **0** |  | 3 |
| *a5* | 0.09 | **1** | **1** | **1** | **0** | 3 |
| *a6* | 0.07 | **1** | **1** | **1** | **1** | 4 |

Λ

0

1

00

01

10

11

110

111

1110

1111

а1

а2

а3

а4

а5

а6

Рисунок 6 Кодовое дерево для кода Фано

Полученный код является префиксным и почти оптимальным со средней длиной кодового слова

*Lср*=0.36**.**2+0.18**.**2+0.18**.**2+0.12**.**3+0.09**.**4+0.07**.**4=2.44

# 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 производится с клавиатуры.

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 и формируется индексный массив.

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 MergSort(zap\* index[n], int left, int right,int sort\_ord). Она сортирует список сначала по полю заглавия (sort\_ord = 0), затем по полю автора (sort\_ord = 1). Для быстрого доступа к отсортированной и неотсортированной базе данных, сортируется индексный массив оригинальной базы данных.

1. *Особенности реализации бинарного поиска*

Бинарный поиск по отсортированной базе данных осуществляется в процедуре int Search\_Binary(zap\* arr[], int left, int right, char key[]). Результатом работы является индекс найденного элемента, удовлетворяющих условию ключа.

1. *Вспомогательные функции и процедуры для построения дерева оптимального поиска (приближенный алгоритм А2).*

До построения дерева заполняется массив вессов, для найденных в бинарном поиске элементов. Построение дерева осуществляется в процедуре void A2(int L, int R, zap\*\* mas). Записи заносятся в дерево в процедуре void add\_vertex(tree \*&p, zap\* mas, int w). Обход дерева слева направо осуществляется в процедуре void tree\_print(tree\* root). Поиск по дереву выполняется в процедуре void tree\_search(tree\* root, char \*key\_author).

1. *Кодирование данных*

Кодирование данных начинается с процедуры void BaseCoding(), 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 fano(int L, int R, int k) считается длина кодового слова и само кодовое слово. В процедуре void CodePrint() осуществляется подсчет и вывод средней длины кодового слова, энтропии, выводит символы, их вероятности, длины кодовых слов и сами кодовые слова на монитор, а так же подсчитывает и выводит коэффициент сжатия..

# Описание программы

## Основные переменные и структуры

struct zap {

char author[12];

char title[32];

char publisher[16];

short int year;

short int num\_of\_page;

};

Структура для хранения элемента базы данных. Всего таких элементов:

const int N = 4000;

struct tree {

zap\* data;

int balance;

int weight;

int height;

tree\* left, \*next, \* right;

};

Структура, представляющая дерево оптимального поиска (А2).

tree \*root = NULL; - указатель на корень дерева

struct F\_code {

float p;

int l;

char a;

};

Структура, представляющая собой кодовое слово и всю необходимую информацию о нем.

F\_code A[M]– кодовый алфавит

const int M = 256 – число символов в алфавите

int sum = 0 – количество элементов в итоговом алфавите, исключающем пустые символы

float entropy = 0 - энтропия

float midlength = 0 – средняя длина кодового слова

int fcompression = 0 – количество считанных символов с файла

int cfcompression = 0 – количество записанных символов в закодированный файл

## Описание подпрограмм

Процедуры сортировки:

1. void MergSort(zap\* index[n], int left, int right,int sort\_ord) – сортирует базу по заголовку и автору
2. void Merg(zap\* index[n], int begin, int end, int sort\_ord) – вспомогательная функция слияния серий
3. int compare(char \*a, char \*b) - сравнение полей заголовков
4. int compare\_fam(char \*a, char \*b) – сравнение полей авторов

Процедуры и функции для поиска в отсортированной базе данных:

1. int Search\_Binary(zap\* arr[], int left, int right, char key[]) – бинарный поиск.
2. int compare2(char a[], char b[]) – сравнение фамилий в поле заголовка

Процедуры построения дерева оптимального поиска (А2):

1. void add\_vertex(tree \*&p, zap\* mas, int w) – добавление элемента в дерево.
2. void A2(int L, int R, zap\*\* mas) - построение дерева оптимального поиска, приближенный алгоритм А2.
3. void count\_weight(tree \*root) – рассчёт весов вершин.
4. void count\_weight\_next(tree \*root) – вспомогательная функция для рассчёта
5. void tree\_search(tree\* root, char \*key\_author) – поиск в дереве.

Процедуры и функции кодирования базы данных:

1. void BaseCoding() – считывание символов базы данных, подсчет их вероятностей и преобразование алфавита.
2. void fano(int L, int R, int k) – создание кодовых слов.
3. int med(int L, int R) – рассчёт медианы для Фано
4. void CodeBase() – кодирование базы данных.

Вывод на экран:

1. void tree\_print(tree\* root) – вывод дерева
2. void tree\_print\_next(tree\* root) – вывод повторяющихся элементов в дереве
3. void CodePrint() – 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>

#include <algorithm>

#define n 4000

using namespace std;

const int M=256;

const int N=4000;

int sum=0;

int code[M][M];

float entropy = 0, midlength = 0;

int fcompression = 0, cfcompression = 0;

int \*W;

int wgh = 0;

struct zap {

char author[12];

char title[32];

char publisher[16];

short int year;

short int num\_of\_page;

};

struct tree {

zap\* data;

int balance;

int weight;

int height;

tree\* left, \*next, \* right;

};

tree \*root = NULL;

struct F\_code {

float p;

int l;

char a;

};

F\_code A[M];

void fano(int L, int R, int k);

int med(int L, int R);

void BaseCoding();

void CodePrint();

void CodeBase();

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

int ind1 = 0, ind2 = 0;

while (a[ind1] != ' ') {

ind1++;

}

while (b[ind2] != ' ') {

ind2++;

}

ind1++;

ind2++;

while (a[ind1] != ' ') {

ind1++;

}

while (b[ind2] != ' ') {

ind2++;

}

return strcmp(&a[ind1], &b[ind2]);

}

int compare2(char a[], char b[]) {

int ind1 = 0, ind2 = 0;

while (a[ind1] != ' ') {

ind1++;

}

while (b[ind2] != ' ') {

ind2++;

}

ind1++;

ind2++;

while (a[ind1] != ' ') {

ind1++;

}

while (b[ind2] != ' ') {

ind2++;

}

return strncmp(&a[ind1], &b[ind2],3);

}

int compare\_fam(char \*a, char \*b){

char bufferA[12];

char bufferB[12];

strcpy(bufferA, a);

strcpy(bufferB, b);

//находим позицию первого пробела

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

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

//минимальное количество символов в фамилии 3

int count =4;

//если сравниваемые фамилии разных размеров выбираем меньший размер для сравнений

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

count = space\_pos\_a;

}

else{

count = space\_pos\_b;

}

count -= 1;

//сравниваем фамилии

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

return 1;

}

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

return -1;

}

//если сравниваемые фамилии разных размеров выбираем меньший

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

return 1;

}

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

return -1;

}

//сравниваем фамилии

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

return 1;

}

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

return -1;

}

//если сравниваемые фамилии разных размеров выбираем меньший

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

return 1;

}

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

return -1;

}

return 0;

}

void Merg(zap\* index[n], int begin, int end, int sort\_ord)

{

int i = begin,

t = 0,

mid = begin + (end - begin) / 2,

j = mid + 1;

zap\* d[n];

while (i <= mid && j <= end) {

int comp\_res = 0;

if (sort\_ord == 0){

comp\_res = compare(index[i]->title, index[j]->title);

}

else{

comp\_res = compare\_fam(index[i]->author, index[j]->author);

}

if (comp\_res <= 0) {

d[t] = index[i]; i++;

}

else {

d[t] = index[j]; j++;

}

t++;

}

while (i <= mid) {

d[t] = index[i]; i++; t++;

}

while (j <= end) {

d[t] = index[j]; j++; t++;

}

for (i = 0; i < t; i++)

index[begin + i] = d[i];

}

void MergSort(zap\* index[n], int left, int right,int sort\_ord)

{

zap\* temp;

if (left < right)

if (right - left == 1) {

int comp\_res = 0;

if (sort\_ord == 0){

comp\_res = compare(index[right]->title, index[left]->title);

}

else{

comp\_res = compare\_fam(index[right]->author, index[left]->author);

}

if (comp\_res < 0) {

temp = index[left];

index[left] = index[right];

index[right] = temp;

}

}

else {

MergSort(index, left, left + (right - left) / 2,sort\_ord);

MergSort(index, left + (right - left) / 2 + 1, right,sort\_ord);

Merg(index, left, right,sort\_ord);

}

}

int Search\_Binary(zap\* arr[], int left, int right, char key[])

{

int midd = 0;

while (1)

{

midd = (left + right) / 2;

if (compare2(arr[midd]->title, key) >= 0)

right = midd;

else if (compare2(arr[midd]->title, key))

left = midd + 1;

if (left >= right)

break;

}

return midd;

}

void add\_vertex(tree \*&p, zap\* mas, int w) {

if (p == NULL) {

p = new tree;

p->data = mas;

p->weight = w;

p->left = NULL;

p->right = NULL;

p->next = NULL;

}

else if (compare\_fam(p->data->author, mas->author)==0) {

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

}

else if (compare\_fam(p->data->author, mas->author)>0) {

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

}

else if (compare\_fam(p->data->author, mas->author)<0) {

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

}

}

void A2(int L, int R, zap\*\* 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];

}

//cout << L << " - " << R << " - " << wes << " - " << i << " - " << W[i] << endl;

//cout << L << " - " << R << " - "<< i<< endl<< mas[i]->author << " - "<< mas[i]->num\_of\_page<< " - "<< mas[i]->publisher<< " - "<< mas[i]->title<< " - "<< mas[i]->year<< endl;

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

A2(L, i - 1,mas);

A2(i + 1, R,mas);

}

}

void count\_weight\_next(tree \*root){

if(root!=NULL){

wgh=wgh+1;

count\_weight\_next(root->next);

}

}

void count\_weight(tree \*root){

if(root!=NULL){

wgh=0;

count\_weight\_next(root->next);

root->weight = wgh;

count\_weight(root->left);

count\_weight(root->right);

}

}

void tree\_print\_next(tree\* root) {

if(root!=NULL)

{

cout << root->data->author << "\t" << root->data->title << "\t\t" << root->data->publisher << "\t" << root->data->year << "\t\t" << root->data->num\_of\_page << endl;

tree\_print\_next(root->next);

}

}

void tree\_search(tree\* root, char \*key\_author) {

if(root!=NULL){

if (compare\_fam(root->data->author, key\_author)>0) {

tree\_search(root->left, key\_author);

}

else{

if (compare\_fam(root->data->author, key\_author)<0) {

tree\_search(root->right, key\_author);

}

else{

if (compare\_fam(root->data->author, key\_author)==0) {

cout << root->weight <<"\t" << root->data->author << "\t" << root->data->title << "\t\t" << root->data->publisher << "\t" << root->data->year << "\t\t" << root->data->num\_of\_page << endl;

tree\_print\_next(root->next);

}

}

}

}

}

void tree\_print(tree\* root) {

if(root!=NULL)

{

tree\_print(root->left);

cout << root->weight <<"\t" << root->data->author << "\t" << root->data->title << "\t\t" << root->data->publisher << "\t" << root->data->year << "\t\t" << root->data->num\_of\_page << endl;

tree\_print\_next(root->next);

tree\_print(root->right);

}

}

int main() {

srand(time(NULL));

int a, c = 0;

W = new int[N];

int w[M];

FILE\* fp;

fp=fopen("testBase1.dat", "rb");

zap\* arr = new zap[n];

fread(arr, sizeof(zap), n, fp);

fclose(fp);

zap\*\* index = new zap \* [n];

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

index[i] = &arr[i];

}

MergSort(index, 0, n - 1,1);

MergSort(index, 0, n - 1,0);

while(true){

int v;

system("cls");

cout << "1.Without" << endl;

cout << "2.Sorting" << endl;

cout << "3.Search" << endl;

cout << "4.Code" << endl;

cout << "5.Exit" << endl;

cin >> v;

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

if(v==1){

int b =0;

cout << "output of 20 entries/full list? (1/2) : ";

cin >> b;

if (b == 1) {

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

while (c < 20) {

cout << c + i << "\t" << arr[c + i].author << "\t" << arr[c + i].title << "\t\t" << arr[c + i].publisher << "\t" << arr[c + i].year << "\t\t" << arr[c + i].num\_of\_page << endl;

c++;

} c = 0;

cout << "continue?(1/2)";

cin >> a;

if (a == 2) {

break;

}

}

}

else if (b == 2) {

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

cout << i + 1 << "\t" << arr[i].author << "\t" << arr[i].title << "\t\t" << arr[i].publisher << "\t" << arr[i].year << "\t\t" << arr[i].num\_of\_page << endl;

if (\_kbhit()) {

cout << "continue?(%/0) : ";

cin >> a;

if (!a) {

break;

}

}

}

}

}

if (v==2){

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

cout << i + 1 << "\t" << index[i]->author << "\t" << index[i]->title << "\t\t" << index[i]->publisher << "\t" << index[i]->year << "\t\t" << index[i]->num\_of\_page << endl;

}

}

if (v==3){

char key[6];

cout << endl << "Enter key: ";

cin >> &key[2];

key[0] = ' ';

key[1] = ' ';

cout << endl << endl;

int index\_element = Search\_Binary(index, 0, n, key);

char\* mass\_char = index[index\_element]->title;

if(index\_element!=0){

while (compare2(index[index\_element-1]->title, mass\_char) == 0) {

index\_element-=1;

if(index\_element==0){

break;

}

}

}

int L = index\_element;

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

W[i] = 0;

}

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

w[i] = 0;

}

int tmp\_queue = 0;

while (compare2(index[index\_element]->title, mass\_char) == 0) {

tmp\_queue++;

cout << index[index\_element]->author << "\t" << index[index\_element]->title << "\t\t" << index[index\_element]->publisher << "\t" << index[index\_element]->year << "\t\t" << index[index\_element]->num\_of\_page << endl;

index\_element++;

if(index\_element==N){

index\_element--;

break;

}

w[index[index\_element]->author[0]+128] +=1;

}

for(int i = L; i <= index\_element;i++){

W[i] = w[index[i]->author[0]+128];

// cout << W[i] << endl;

}

system("pause");

system("CLS");

int i = 0;

A2(L,index\_element,index);

cout <<"LRprint: " << endl<< endl;

count\_weight(root);

tree\_print(root);

char key\_auth[12];

cout << "Enter key\_author" << endl;

SetConsoleCP(866);

cin >> key\_auth;

SetConsoleCP(1251);

char \*spc = " ";

strcat(key\_auth, spc);

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

tree\_search(root,key\_auth);

}

}

if(v==4){

BaseCoding();

CodeBase();

CodePrint();

}

if (v==5){

return 0;

}

}

system("pause");

}

}

void CodeBase() {

FILE \*fp, \*fcoded;

fp = fopen("testBase1.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(){

setlocale(LC\_ALL, "Russian");

SetConsoleCP(866);

printf("\n\nКод Фано: \n");

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

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

printf("| | | | | слова |\n");

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

SetConsoleCP(1251);

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

{

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

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);

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

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

}

SetConsoleCP(866);

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

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

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

printf("| %10f | %10.5f | %10.5f |\n", entropy, midlength, (float)fcompression/cfcompression);

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

}

void BaseCoding(){

int i,j;

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

A[i].p = 0;

A[i].l = 0;

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

}

FILE \*fp;

fp=fopen("testBase1.dat", "rb");

while (!feof(fp)) {

char c;

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

if (feof(fp))

break;

//cout << c<<" - " << (int)c <<endl;

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

sum++;

}

printf("\n");

fclose(fp);

bool b = true;

while (b)

{

b = false;

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

{

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

{

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

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

A[i]=B;

b = true;

}

}

}

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

A[i].p /=sum;

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

}

fano(0, M - 1, 0);

}

void fano(int L, int R, int k)

{

if (L < R)

{

k++;

int m = med(L, R);

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

{

if (i <= m)

code[i][k] = 0;

else

code[i][k] = 1;

A[i].l++;

}

fano(L, m, k);

fano(m + 1, R, k);

}

}

int med(int L, int R)

{

float sl = 0, sr;

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

sl += A[i].p;

sr = A[R].p;

int m = R;

while (sl >= sr)

{

m--;

sl -= A[m].p;

sr += A[m].p;

}

return m;

}

# Результаты

Рисунок 1. Неотсортированная база данных.



Рисунок 2. Отсортированная база данных по автору и фамилии в заглавии.

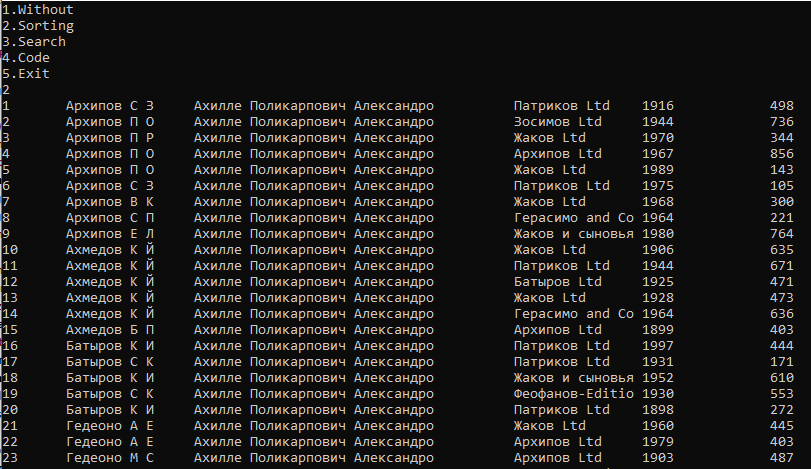


Рисунок 3. Очередь из записей, полученных в результате по первым трем буквам фамилии в заглавии(Хас).

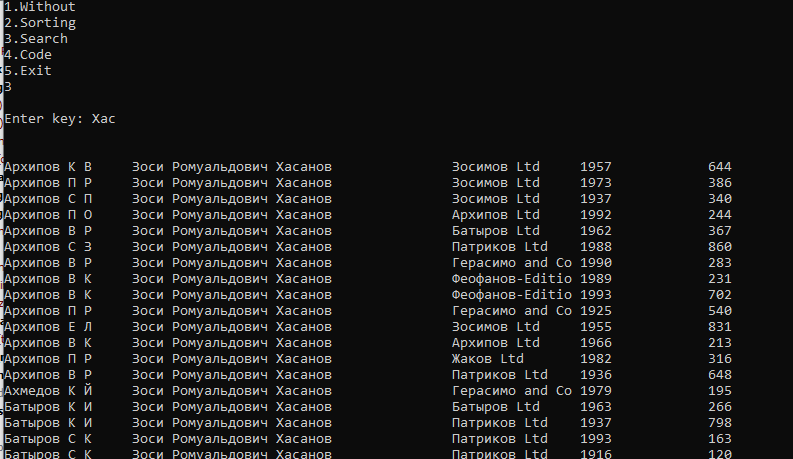


Рисунок 4. Дерево, ключ в дереве – фамилия автора.

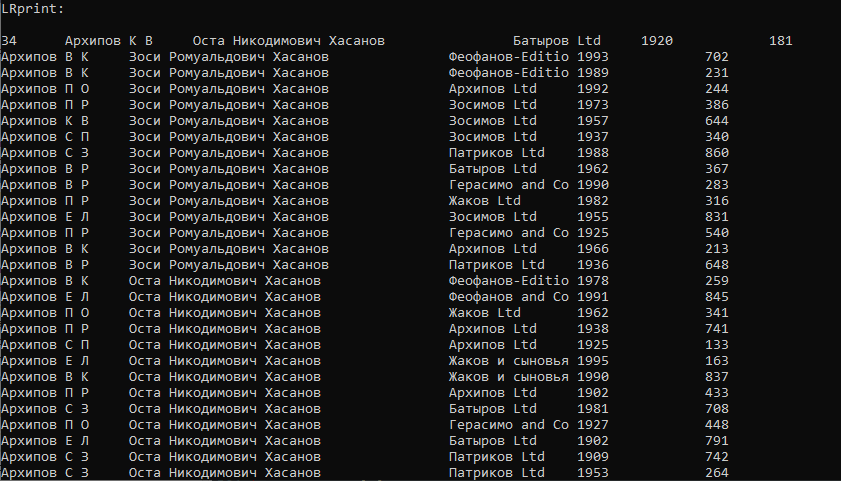


Рисунок 5. Поиск по дереву (элементы с одинаковым ключом).

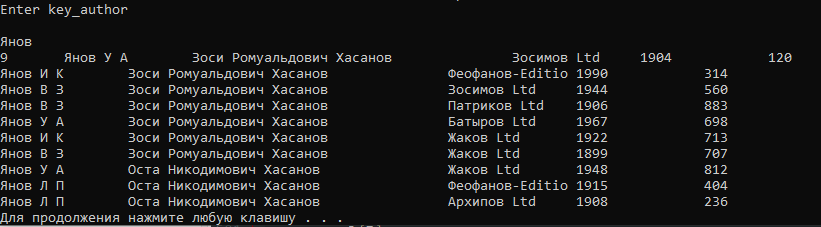
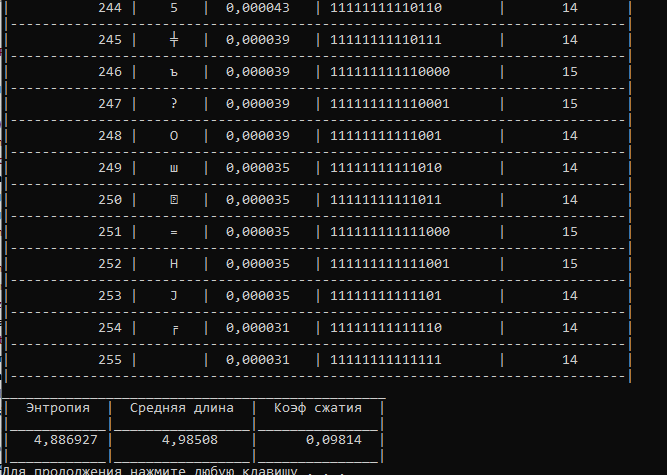


Рисунок 6. Кодовые слова, энтропия и средняя длина кодового слова.



# Выводы

В ходе выполнения курсовой работы были выполнены все поставленные задачи и реализованы необходимые алгоритмы: сортировки, поиска, построения дерева оптимального поиска (приближённый алгоритм А2), 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.