Федеральное агентство связи

Сибирский государственный университет телекоммуникаций и информатики

Кафедра прикладной математики и кибернетики

Курсовая работа

«Структуры и алгоритмы обработки данных»

Вариант 3

Выполнил: студент группы ИП-815

Захаров М.Д.

Проверил: ассистент кафедры ПМиК

Турцев А.А.

Новосибирск 2019

Содержание

[1. ПОСТАНОВКА ЗАДАЧИ 3](#_Toc230446878)

[2. ОСНОВНЫЕ ИДЕИ И ХАРАКТЕРИСТИКИ ПРИМЕНЯЕМЫХ МЕТОДОВ 4](#_Toc230446879)

[2.1. МЕТОД СОРТИРОВКИ 4](#_Toc230446880)

[2.2 ДВОИЧНЫЙ ПОИСК 4](#_Toc230446881)

[2.3 ДЕРЕВО И ПОИСК ПО ДЕРЕВУ 5](#_Toc230446882)

[3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ АЛГОРИТМОВ 7](#_Toc230446883)

[4. ОПИСАНИЕ ПРОГРАММЫ 9](#_Toc230446884)

[4.1. ОСНОВНЫЕ ПЕРЕМЕННЫЕ И СТРУКТУРЫ 9](#_Toc230446885)

[4.2. ОПИСАНИЕ ПОДПРОГРАММ 10](#_Toc230446886)

[5. ТЕКСТ ПРОГРАММЫ 12](#_Toc230446887)

[6. РЕЗУЛЬТАТЫ 25](#_Toc230446888)

[7. ВЫВОДЫ 28](#_Toc230446889)

# 1. ПОСТАНОВКА ЗАДАЧИ

Хранящуюся в файле базу данных загрузить в оперативную память компьютера и построить индексный массив, упорядочивающий данные **по дням рождения и ФИО**, используя **метод Хоара** в качестве метода сортировки.

Предусмотреть возможность 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 (приближеный алгоритм А1)**, 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, упакованную базу данных записать в файл, вычислить коэффициент сжатия данных.

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

Варианты условий упорядочения и ключи поиска (К):

по году выпуска К = год выпуска.

Ключ в дереве – Количество страниц.

# 2. ОСНОВНЫЕ ИДЕИ И ХАРАКТЕРИСТИКИ ПРИМЕНЯЕМЫХ МЕТОДОВ

## 2.1. МЕТОД СОРТИРОВКИ

Метод Хоара

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.

## 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 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

## 2.3 ДЕРЕВО И ПОИСК ПО ДЕРЕВУ

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).

*Первый алгоритм* (А1) предлагает в качестве корня использовать вершину с наибольшим весом. Затем среди оставшихся вершин снова выбирается вершина с наибольшим весом и помещается в левое или правое поддерево в зависимости от ее значения, и т.д.

2.4 МЕТОД КОДИРОВАНИЯ

Метод Фано построения префиксного почти оптимального кода, для которого H( p1<Lcp ,...,pn 1, 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.

# 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

Для организации интерфейса использовался цикл while(true) в котором рисовалось и стиралось меню и оператор switch. Который в зависимости от переменной «but»(введенной с клавиатуры кнопки) реализовывал нужный пункт меню.

2. Загрузка и вывод базы данных

Для загрузки и чтения базы данных использовались функции fopen(“testBase1.dat,”rb”) и fread((record\*)base, sizeof(record), size, fp);Данные функции вызываются независимо от желания пользователя, Здесь же создаётся массив отсортированной базы данных base, неотсортированной baseun и индексный массив index.

За вывод элементов считанной базы данных отвечает процедура *void PrintBase().* Она предоставляет возможность просмотра базы данных непрерывным потоком.

3. Вспомогательные функции и процедуры для сортировки данных

Для сортировки данных используется процедура void QuickSort(record\* base, record\* index, int l, int r). Доступ к записям базы данных осуществляется через индексный массив index, для сортировки по дням рождения и ФИО используется процедура int Less (record base, record x), возвращающая 1, если p-я запись меньше q-й, и 0 – иначе.

4. Особенности реализации бинарного поиска

Бинарный поиск по отсортированной базе осуществляется в процедуре void BSearch1 (record base[], int n, record1 \*&head), доступ к записям ведётся через индексный массив, найденные записи заносятся в очередь struct record1, которая очищается перед каждым следующим поиском. Для вывода на монитор найденных записей используется процедура void Print(record1 \*p). При реализации бинарного поиска была использована его первая версия.

5. Вспомогательные функции и процедуры для построения дерева оптимального поиска (приближённый алгоритм А1)

Построение дерева осуществляется в процедуре void void DOP1(vertex \*&root, vertex V[]). Для этого сначала после добавления найденных записей в очередь они потом из нее добавляются в динамический массив вершин mass. Во время добавления каждой вершине приравнивается случайный вес с помощью функции rand() % 15 + 1. Cортировка происходит в процедуре void QuickSortW(vertex A[], int L, int R). Непосредственно же записи заносятся в дерево при помощи процедуры void ADD(vertex \*&p, vertex X). Обход построенного дерева выполняет процедура void obhod(vertex \*p). Поиск по дереву выполняет процедура void Search(vertex \*root).

6. Кодирование данных

Кодирование базы данных происходит в процедуры void Make(), которая сначала считает вероятности встречающихся символов и общее их количество. Затем кодирует символы и выводит символы, их вероятности, длины кодовых слов и сами кодовые слова на монитор. После этого вычисляются энтропия и средняя длина кодового слова.

# 4. ОПИСАНИЕ ПРОГРАММЫ

## 4.1. ОСНОВНЫЕ ПЕРЕМЕННЫЕ И СТРУКТУРЫ

struct record

{

char author[12];

char article[32];

char izd[16];

short int year;

short int pages;

};Запись, используемая для работы с базой данных

struct record1

{

int num;

char author[12];

char article[32];

char izd[16];

short int year;

short int pages;

record1\* next;

};Структура (список), используемая при сортировке базы данных.

next **–** указатель на следующие элемент;

struct vertex {

int num;

int W;

vertex\* left;

vertex\* right;

vertex\* up;

record record;

}

vertex \*root=NULL;

Структура, представляющая дерево оптимального поиска (А1).

int Ц – вес вершины.

vertex \*left, vertex \*right– указатели на левое и правое поддеревья.

vertex\* up- запасное «поддерево» на случай одинаковых ключей поиска в дереве.

Record record- поле данных

root – указатель на корень дерева.

## 4.2. ОПИСАНИЕ ПОДПРОГРАММ

Функции и процедуры сортировки:

7. void QuickSort(record\* base, record\* index, int l, int r)– сортирует базу данных по дням рождения и ФИО.

L – левая граница массива, R – правая граница массива,base и index массивы базы данных.

8. int Less(record base, record x), int More(record base, record x)– сравнение записей по дате рождения и ФИО.

Функции и процедуры для поиска в отсортированной базе данных:

10. void BSearch1 (record base[], int n, record1 \*&head)– осуществляет поиск в базе.

n-количество элементов в массиве.

void Pool(record1\*& head, record base[], int m,int key)-добавляет найденный элемент в список.

11. Print(head) – выводит на экран найденные записи.

Head – указатель на начало очереди.

Процедуры и функции построения дерева оптимального поиска (А1):

12. void PullMass(record1 \* &head, vertex mass[])-добаляет записи из очереди в динамический массив

13. void QuickSortW(vertex A[], int L, int R)– сортирует вершины по весу.

15. void DOP1(vertex \*&root, vertex V[])– построение дерева.

V-массив вершин.

16. void ADD(vertex \*&p, vertex X)– добавление записи в дерево.

17. void obhod(vertex \*p)– обход дерева слева направо.

P – указатель на запись в дереве.

18. void Search(vertex \*root) – поиск в дереве по ключу X.

P – указатель на запись в дереве.

Процедуры и функции кодирования базы данных:

процедура void Make(), которая сначала считает вероятности встречающихся символов и общее их количество. Затем кодирует символы и выводит символы, их вероятности, длины кодовых слов и сами кодовые слова на монитор. После этого вычисляются энтропия и средняя длина кодового слова.

int getFileSize(const char \* fileName) – возвращает размер файла для подсчета коэффициента сжатия.

Основная программа:

25. main() – основная программа, в которой, открывается и читается файл базы данных, рисуется меню и в зависимости от выбранного пункта меню вызываются соответствующие процедуры.

# 5. ТЕКСТ ПРОГРАММЫ

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <cstdlib>

#include <conio.h>

#include <string.h>

#include <iomanip>

#include <bitset>

#include <vector>

#include <fstream>

#include <algorithm>

#include <map>

#include <sstream>

#include <stdio.h>

#include <sys/stat.h>

using namespace std;

int c = 0;

struct record

{

char author[12];

char article[32];

char izd[16];

short int year;

short int pages;

};

struct vertex {

int num;

int W;

vertex\* left;

vertex\* right;

vertex\* up;

record record;

};

vertex \*root=NULL;

struct record1

{

int num;

char author[12];

char article[32];

char izd[16];

short int year;

short int pages;

record1\* next;

};

struct Queue {

record1\* head;

record1\* tail;

};

void Pool(record1\*& head, record base[], int m,int key);

void print(record1\*& head);

int Less(record base, record x)

{

char\* auth1 = base.author;

char\* auth2 = x.author;

if (base.year < x.year)

return 1;

else if (base.year > x.year)

return 0;

else if (strcmp(auth1, auth2) < 0)

return 1;

else

return 0;

}

int More(record base, record x)

{

char\* auth1 = base.author;

char\* auth2 = x.author;

if (base.year > x.year)

return 1;

else if (base.year < x.year)

return 0;

else if (strcmp(auth1, auth2) > 0)

return 1;

else

return 0;

}

void PrintBase(record\* base, int size)

{

int n;

cout << "Num" << setw(2) << "|" << "Author" << setw(6) << "|" << "Article" << setw(24) << "|" << "Publisher" << setw(7) << "|" << "Year" << "|" << "Pages" << "|" << endl;

for (int i = 1, count = 1; i < 4000; ++i) {

int check = 0;

while (check < 20) {

cout << setw(4) << count << "|";

cout << base[i].author << "|" << base[i].article << "|" << base[i].izd << "|" << base[i].year << "|" << base[i].pages << "|" << endl;

check++;

count++;

i++;

}

cout << endl << "Press the Enter to print another 20 entries" << endl;

n = \_getch();

if (n != 13) break;

}

}

void QuickSort(record\* base, record\* index, int l, int r)

{

record x;

x = base[l];

int i = l, j = r;

while (i <= j) {

while (Less(base[i], x) == 1) {

i++;

}

while (More(base[j], x) == 1) {

j--;

}

if (i <= j)

{

record tmp = index[i];

index[i] = index[j];

index[j] = tmp;

i++;

j--;

}

}

if (l < j)

QuickSort(base, index, l, j);

if (i < r)

QuickSort(base, index, i, r);

}

void BSearch1 (record base[], int n, record1 \*&head)

{ int key;

cout<<"input key `Year`:";

cin>>key;

int L=1, R=n, Find=0, m=0;

if(1897<key&&key<1998){

while (L<R)

{

m=(L+R)/2;

if (base[m].year<key) L=m+1;

else R=m;

if (base[m].year==key)

{ L=R;

while(base[m].year==key) {

// R++;

// if(R==3999) break;

m=m+1;

}

Find=1;

m=m-1;

break;

}

}

if(Find==1){Pool(head,base,m,key);}

}

else cout<<"Wrong key"<<endl;

}

void Search(vertex \*root){

int X;

cin>>X;

vertex \*p;

p=root;

if(!(X>=0&&X<=1000)){

while(!(X>=0&&X<=1000)){

cout<<"Enter the correct key!(0-1000)"<<endl;

cin>>X;}

}

while(p!=NULL){

if(X<p->record.pages) p=p->left;

else{

if(X>p->record.pages) p=p->right;

else {

break;

}

}

}

if(p!=NULL){

while(p!=NULL){

cout << ' ' << p->num << setw(15) << p->record.author << ' ' << setw(14) << p->record.article << setw(6) << p->record.izd << setw(14) << ' ' << p->record.year << setw(18) << p->record.pages<<endl;

p=p->up;

}}

else cout<<endl<<"NOT FOUND ";

return;

}

void obhod(vertex \*p){

if(p!=NULL){

obhod(p->left);

cout << ' ' << p->num << setw(15) << p->record.author << ' ' << setw(14) << p->record.article << setw(6) << p->record.izd << setw(14) << ' ' << p->record.year << setw(18) << p->record.pages;

cout << endl;

obhod(p->right);

if(p->up!=NULL){

obhod(p->up);

}

}

return;

}

void QuickSortW(vertex A[], int L, int R){

int j=R,i=L,X=A[(L+R)/2].W;

vertex t;

while(i<=j){

while(A[i].W>X) {i++; }

while(A[j].W<X) {j--; }

if(i<=j){

t=A[j];

A[j]=A[i];

A[i]=t;

i++;

j--;

}

}

if(L<j) QuickSortW(A,L,j);

if(i<R) QuickSortW(A,i,R);

return;

}

void ADD(vertex \*&p, vertex X){

if(p==NULL){

p=new vertex;

p->num=X.num;

p->W=X.W;

strcpy(p->record.author,X.record.author);

strcpy(p->record.article,X.record.article);

strcpy(p->record.izd,X.record.izd);

p->record.year=X.record.year;

p->record.pages=X.record.pages;

p->left=p->right=NULL;

}

else if(X.record.pages<p->record.pages) ADD(p->left,X);

else if(X.record.pages>p->record.pages) ADD(p->right,X);

else ADD(p->up,X);

}

void DOP1(vertex \*&root, vertex V[]){

int nn=c;

QuickSortW(V,1,nn);

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

ADD(root,V[i]);}

//obhod(root);

// cout<<endl<<"Please input num of PAGES"<<endl;

//Search(root);

}

void PullMass(record1 \* &head, vertex mass[]) {

record1\* p = head;

for (int i = 0; i<c; p = p->next, i++) {

mass[i].num = p->num;

strcpy(mass[i].record.author, p->author);

strcpy(mass[i].record.article, p->article);

strcpy(mass[i].record.izd, p->izd);

mass[i].record.year = p->year;

mass[i].record.pages = p->pages;

mass[i].W = rand() % 15 + 1;

}

DOP1(root,mass);

}

void Pool(record1\*& head, record base[], int m,int key) {

record1\* p = head;

while (base[m].year==key) {

p = new record1;

p->num = m;

strcpy(p->author, base[m].author);

strcpy(p->article, base[m].article);

p->year = base[m].year;

p->pages = base[m].pages;

strcpy(p->izd, base[m].izd);

p->next = head;

head = p;

m = m - 1;

c++;

}

/\*p = new record1;

p->num = m;

strcpy(p->author, base[m].author);

strcpy(p->article, base[m].article);

p->year = base[m].year;

p->pages = base[m].pages;

strcpy(p->izd, base[m].izd);

p->next = head;

head = p;

c++;\*/

cout<<endl<<"Queue:"<<endl;

print(head);

vertex \*mass= new vertex[c];

PullMass(head, mass);

// cout<<mass[0].num;

}

void print(record1\*& head) {

record1\* p = head;

int code;

for (; p != NULL; p = p->next) {

cout << ' ' << p->num << setw(15) << p->author << ' ' << setw(14) << p->article << setw(6) << p->izd << setw(14) << ' ' << p->year << setw(18) << p->pages;

cout << endl;

}

// cout << endl << "DOP:"<<endl;

}

int Med(int l, int r, vector<pair<char, double>> p)

{

double sl = 0, sr;

int m = r;

for (int i = l; i < r; i++) {

sl += p[i].second;

}

sr = p[r].second;

while (sl >= sr) {

m--;

sl -= p[m].second;

sr += p[m].second;

}

if (!m) {

return m;

}

return (fabs(sl - sr) < fabs(sl + p[m].second - sr + p[m].second) ? m - 1

: m);

}

void Fano(

int l,

int r,

int k,

vector<pair<char, double>> p,

vector<vector<int>>& c)

{

if (l < r) {

k++;

int m = Med(l, r, p);

for (int i = l; i <= r; i++) {

if (i <= m) {

c[i].push\_back(0);

} else {

c[i].push\_back(1);

}

}

Fano(l, m, k, p, c);

Fano(m + 1, r, k, p, c);

}

}

void Code(vector<pair<char, double>> p, vector<vector<int>> c) {

map<char, string> code;

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

stringstream ss;

for (auto a : c[i]) {

ss << a;

}

code[p[i].first] = ss.str();

}

ifstream input("testBase1.dat");

ofstream output("code.txt");

while (true) {

char c = input.get();

if (input.eof()) {

break;

}

output << code[c];

}

input.close();

output.close();

}

void Make() {

map<char, int> \_map;

ifstream input("testBase1.dat");

int count = 0;

double length = 0;

while (true) {

char c = input.get();

if (input.eof()) {

break;

}

count++;

\_map[c]++;

}

vector<pair<char, double>> vec;

for (auto i : \_map) {

vec.push\_back({i.first, i.second / (double)count});

}

double entropy = 0;

for (auto i : vec) {

entropy += i.second \* (-log2(i.second));

}

vector<vector<int>> c(vec.size());

sort(vec.begin(), vec.end(), [](pair<char, double> a, pair<char, double> b) {

return a.second > b.second;

});

vector<double> p;

for (auto i : vec) {

p.push\_back(i.second);

}

Fano(0, p.size() - 1, 0, vec, c);

int \_max = 0;

for (auto i : c) {

\_max = max(\_max, (int)i.size());

}

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

cout << bitset<sizeof(vec[i].first) \* CHAR\_BIT>(vec[i].first) << " " << setw(11)<< fixed << vec[i].second << " ";

for (auto j : c[i]) {

cout << j;

}

for (int j = 0; j < \_max - c[i].size() + 3; j++) {

cout << " ";

}

cout << setw(5) << c[i].size() << endl;

}

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

length += vec[i].second \* c[i].size();

}

cout << endl << endl << "entropiya: " << entropy << endl << "Sr dlina: " << length << endl << endl << endl;

Code(vec,c);

\_map.clear();

vec.clear();

c.clear();

p.clear();

}

void clear(void)

{

system("cls");

}

int getFileSize(const char \* fileName)

{

struct stat file\_stat;

stat(fileName, &file\_stat);

return file\_stat.st\_size;

}

int main()

{

int i = 0;

record1\* head, \* tail;

head = NULL;

tail = (record1\*)&head;

int sizebase = 0;

FILE\* fp;

int size;

size = 4000;

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

record base[4000] = { 0 };

record baseun[4000] = { 0 };

record\* index[4000] = { 0 };

i = fread((record\*)base, sizeof(record), size, fp);

cout << i << endl;

char but;

while(true){clear();

cout<<"1.Unsorted all base "<<endl<<"2.Sorted all base "<<endl<<"3.Unsorted base of 20 elements"<<endl<<"4.Sorted base of 20 elements"<<endl<<"5.Search"<<endl;

cout<<"6.Tree (DOP)"<<endl<<"7.Code"<<endl;

cin>>but;

switch(but){

case'1':{

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

i = fread((record\*)baseun, sizeof(record), size, fp);

fclose(fp);

i = 0;

int count = 4000, code;

int countMax = count / 20;

count = 1;

while (i < 4000) {

int temp = i;

cout << "Num" << setw(2) << "|" << "Author" << setw(6) << "|" << "Article" << setw(24) << "|" << "Publisher" << setw(7) << "|" << "Year" << "|" << "Pag" << endl;

for (i = temp; i < 4000; i++) {

cout << setw(4) << i+1 << "|" << baseun[i].author << "|" << setw(4) << baseun[i].article << "|" << setw(6) << baseun[i].izd << "|" << baseun[i].year << "|" << baseun[i].pages;

cout << endl;

}

cout<<"Press ESC to EXIT"<<endl;

code = getche();

//clear();

if (code == 27) {

break;

}

//else if (code == 'k') {

//cout<<search(KEY, head)<<endl;}

}

break;}

case'2':{

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

i = fread((record\*)base, sizeof(record), size, fp);

fclose(fp);

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

{

index[i] = &base[i];

}

QuickSort(base, \*index, 0, size);

i = 0;

int count = 4000, code;

int countMax = count / 20;

count = 1;

while (i < 4000) {

int temp = i;

//cout<< setw(5) <<"Num:"<< setw(12) << "Author:" << setw(18) << "Article:"<< setw(24) << "Publisher:"<< setw(7) << "Year:" << "Pages:" << endl;

cout << "Num" << setw(2) << "|" << "Author" << setw(6) << "|" << "Article" << setw(24) << "|" << "Publisher" << setw(7) << "|" << "Year" << "|" << "Pag" << endl;

for (i = temp; i < 4000; i++) {

cout << setw(4) << i+1 << "|" << base[i].author << "|" << setw(4) << base[i].article << "|" << setw(6) << base[i].izd << "|" << base[i].year << "|" << base[i].pages;

cout << endl;

}

cout<<"Press ESC to EXIT"<<endl;

code = getche();

//clear();

if (code == 27) {

break;

}

break;

}

break;}

case'3':{

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

i = fread((record\*)baseun, sizeof(record), size, fp);

fclose(fp);

i = 0;

int count = 4000, code;

int countMax = count / 20;

count = 1;

while (i < 4000) {

int temp = i;

//cout<< setw(5) <<"Num:"<< setw(12) << "Author:" << setw(18) << "Article:"<< setw(24) << "Publisher:"<< setw(7) << "Year:" << "Pages:" << endl;

cout << "Num" << setw(2) << "|" << "Author" << setw(6) << "|" << "Article" << setw(24) << "|" << "Publisher" << setw(7) << "|" << "Year" << "|" << "Pag" << endl;

for (i = temp; i < temp +20; i++) {

cout << setw(4) << i+1 << "|" << baseun[i].author << "|" << setw(4) << baseun[i].article << "|" << setw(6) << baseun[i].izd << "|" << baseun[i].year << "|" << baseun[i].pages;

cout << endl;

}

cout << "Page " << temp/20+1 << ' ' << "of " << countMax << endl << "To view the next page press any key, exit - esc " << endl;

code = getche();

cout<<"Press ESC to EXIT"<<endl;

//clear();

if (code == 27) {

break;

}

//else if (code == 'k') {

//cout<<search(KEY, head)<<endl;}

}

if (code != 27) {

while(code !=27){

code=getche();

if(code ==27)

break;

}}

else break;

break;}

case'4':{

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

i = fread((record\*)base, sizeof(record), size, fp);

fclose(fp);

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

{

index[i] = &base[i];

}

QuickSort(base, \*index, 0, size);

i = 0;

int count = 4000, code;

int countMax = count / 20;

count = 1;

while (i < 4000) {

int temp = i;

//cout<< setw(5) <<"Num:"<< setw(12) << "Author:" << setw(18) << "Article:"<< setw(24) << "Publisher:"<< setw(7) << "Year:" << "Pages:" << endl;

cout << "Num" << setw(2) << "|" << "Author" << setw(6) << "|" << "Article" << setw(24) << "|" << "Publisher" << setw(7) << "|" << "Year" << "|" << "Pag" << endl;

for (i = temp; i < temp + 20; i++) {

cout << setw(4) << i+1 << "|" << base[i].author << "|" << setw(4) << base[i].article << "|" << setw(6) << base[i].izd << "|" << base[i].year << "|" << base[i].pages;

cout << endl;

}

cout << "Page " << temp/20+1 << ' ' << "of " << countMax << endl << "To view the next page press any key, exit - esc " << endl;

code = getche();

cout<<"Press ESC to EXIT"<<endl;

if (code == 27) {

break;

}

}

if (code != 27) {

while(code !=27){

code=getche();

if(code ==27)

break;

}}

else break;

break;

}

case'5':{

int count = 4000, code;

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

i = fread((record\*)base, sizeof(record), size, fp);

fclose(fp);

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

{

index[i] = &base[i];

}

QuickSort(base, \*index, 0, size);

//cout<<"Input KEY:";

//if(1897<key<1998){

BSearch1(base, 4000, head);//}

cout<<"Press ESC to EXIT"<<endl;

code = getche();

if (code == 27|code == 13) {

break;

}

break;

}

case'6':{

int count = 4000, code;

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

i = fread((record\*)base, sizeof(record), size, fp);

fclose(fp);

if(head!=NULL){

cout<<"DOP"<<endl;

obhod(root);

cout<<endl<<"Please input num of PAGES"<<endl;

Search(root);}

else cout<<"Queue not a build"<<endl;

cout<<"Press ESC to EXIT"<<endl;

code = getche();

if (code == 27|code == 13) {

break;}

break;

}

case'7':{

int count = 4000, code;

Make();

int sizecode=0;

sizecode=getFileSize("code.txt");

sizebase=getFileSize("testBase1.dat");

cout<<"The weight of the original file ="<<sizebase<<" byte"<<endl;

cout<<"The weight of the encoded file ="<<sizecode<<" byte"<<endl;

cout<<"The compression ratio ="<<(float)sizebase/sizecode<<endl;

cout<<"Press ESC to EXIT"<<endl;

code = getche();

if (code == 27) {

break;}

}

}

}

return 0;

}

# 

# 6. РЕЗУЛЬТАТЫ

Рисунок 1. Меню



Рисунок 2. Отсортированная по году выпуска база данных.



Рисунок 3. Очередь из записей, полученных в результате поиска (1992 года выпуска).



Рисунок 4. Дерево, ключ в дереве – кол-во страниц.



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

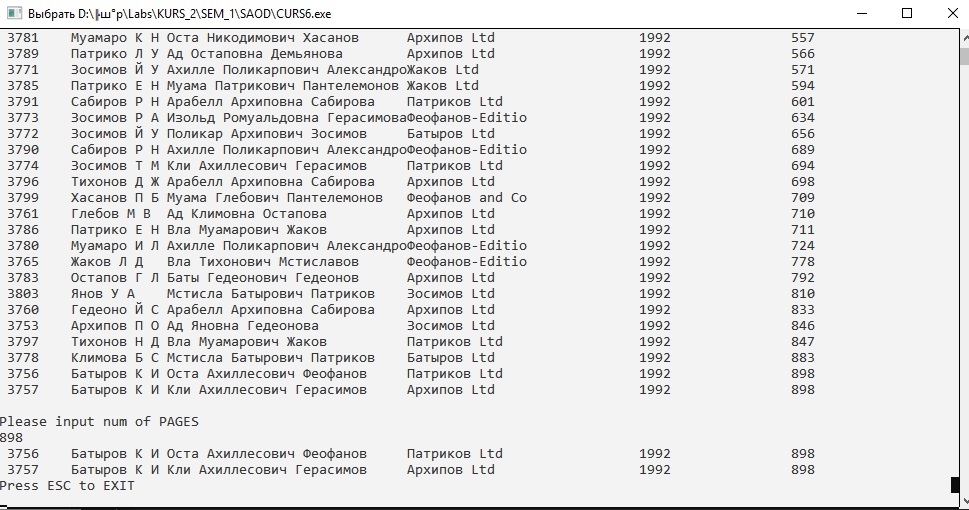
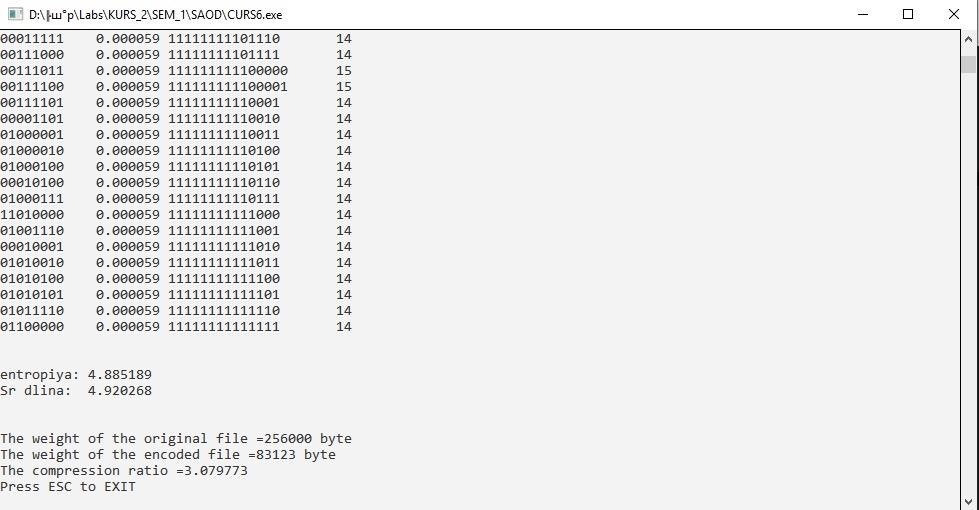


Рисунок 6. Примеры кодовых слов, энтропия , средняя длина кодового слова, размеры файлов и коэффициент сжатия.



# 7. ВЫВОДЫ

В ходе выполнения курсовой работы были выполнены все поставленные задачи и реализованы необходимые алгоритмы: сортировки, поиска, построения дерева оптимального поиска (приближённый алгоритм А1), 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.