Министерство цифрового развития,

Связи и массовых коммуникаций российской федерации

Федеральное агентство связи

Сибирский государственный университет телекоммуникаций и информатики

Кафедра прикладной математики и кибернетики

Курсовая работа

«Структуры и алгоритмы обработки данных»

Вариант 19

Выполнила: студентка группы ИП-011

Лучинина А.О.

Проверил: доцент кафедры ПМиК

Курапова Е.В.

Новосибирск 2022

Содержание

[1. ПОСТАНОВКА ЗАДАЧИ 3](#_Toc89884116)

[2. ОСНОВНЫЕ ИДЕИ И ХАРАКТЕРИСТИКИ ПРИМЕНЯЕМЫХ МЕТОДОВ 4](#_Toc89884117)

[2.1. МЕТОД СОРТИРОВКИ 4](#_Toc89884118)

[2.2. ДВОИЧНЫЙ ПОИСК 4](#_Toc89884119)

[2.3. ДЕРЕВО И ПОИСК ПО ДЕРЕВУ 4](#_Toc89884120)

[3. ОПИСАНИЕ СТРУКТУР ДАННЫХ И ИСПОЛЬЗОВАННЫХ АЛГОРИТМОВ 5](#_Toc89884121)

[4. ОПИСАНИЕ ПРОГРАММЫ 8](#_Toc89884122)

[4.1. ОСНОВНЫЕ ПЕРЕМЕННЫЕ И СТРУКТУРЫ 8](#_Toc89884123)

[4.2. ОПИСАНИЕ ПОДПРОГРАММ 9](#_Toc89884124)

[5. ИСХОДНЫЙ ТЕКСТ ПРОГРАММЫ 10](#_Toc89884125)

[6. ВЫВОДЫ 26](#_Toc89884127)

# 

# 1. ПОСТАНОВКА ЗАДАЧИ

Хранящуюся в файле базу данных (4000 записей) загрузить динамически в оперативную память компьютера в виде списка, вывести на экран по 20 записей (строк) на странице с возможностью отказа от просмотра.

Упорядочить данные **по издательству и автору**, используя **метод прямого слияния**.

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

Из записей очереди построить **двоичное Б-дерево**, отличному от ключа сортировки, вывести на экран содержимое дерева и предусмотреть возможность поиска в дереве по запросу.

Закодировать файл базы данных статическим **кодом Шеннона**, 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

Варианты условий упорядочения и ключи поиска (К):

по издательству и автоpу, К = тpи пеpвые буквы издательства.

Ключ в дереве -год.

# 2. ОСНОВНЫЕ ИДЕИ И ХАРАКТЕРИСТИКИ ПРИМЕНЯЕМЫХ МЕТОДОВ

## 2.1. МЕТОД СОРТИРОВКИ

Метод прямого слияния

В основе метода прямого слияния лежит операция слияния серий.р-сериейназывается упорядоченная последовательность изрэлементов.Пусть имеются две упорядоченные серииaиbдлиныqиrсоответственно.Необходимо получить упорядоченную последовательность с, которая состоит изэлементов серийaиb. Сначала сравниваем первые элементы последовательно-стейaиb. Минимальный элемент перемещаем в последовательность с. Повторя-ем действия до тех пор, пока одна из последовательностейaиbне станет пустой,оставшиеся элементы из другой последовательности переносим в последователь-ность с. В результате получим (q+r)-серию.

Для алгоритма слияния серий с длинамиqиrнеобходимое количество сравнений и перемещений оценивается следующим образом min (q, r) ≤ C ≤q+r-1, M=q+r

Пусть длина спискаSравна степени двойки, т.е. 2k, для некоторого нату-ральногоk. Разобьем последовательностьSна два спискаaиb, записывая пооче-редно элементыSв списки а иb.Сливаем спискиaиbс образованием двойныхсерий, то есть одиночные элементы сливаются в упорядоченные пары, которыезаписываются попеременно в очередиc0иc1. Переписываем очередьc0в списокa, очередьc1–в списокb. Вновь сливаемaиbс образованием серий длины 4 и т.д. На каждом итерации размер серий увеличивается вдвое. Сортировка заканчи-вается, когда длина серии превысит общее количество элементов в обоих спис-ках. Если длина спискаSне является степенью двойки, то некоторые серии впроцессе сортировки могут быть короче.

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

Трудоёмкость методапрямого слияния определяется сложностью операциислияния серий. На каждой итерации происходит ровноnперемещений элементовсписка и не болееnсравнений. Как нетрудно видеть, количество итераций равно⎡logn⎤. ТогдаC<n⎡logn⎤,M=n⎡logn⎤+n.

Дополнительныеnперемещений происходят во время начального расщепленияисходного списка. Асимптотические оценки для М и С имеют следующий видС=О(nlogn), М=О(nlogn) приn→ ∞.

Метод обеспечивает устойчивую сортировку.При реализации для массивов,метод требует наличия второго вспомогательного массива, равного по размеруисходному массиву. При реализации со списками дополнительной памяти нетребуется.

## 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 . Таким образом, трудоёмкость двоичного поиска в обоих случаях



## 2.3. ДЕРЕВО И ПОИСК ПО ДЕРЕВУ

Двоичное Б-дерево

Двоичное Б-деревосостоит из вершин (страниц) с одним или двумя элемен-тами. Следовательно, каждая страница содержит две или три ссылки на поддеревья.

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

Таким образом, страницы Б-дерева теряют свою целостность и элементысписков начинаютиграть роль вершин в двоичном дереве. Однако остается необ-ходимость делать различия между ссылками на потомков (вертикальными) иссылками на одном уровне (горизонтальными), а также следить, чтобы все листьябыли на одном уровне.

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

Высота двоичного Б-дерева .Если рассматри-вать двоичное Б-дерево как обычное двоичное дерево, то его высотаможет уве-личиться вдвое, т.е.h=2log(n+1). Для сравнения, в АВЛ-дереве даже в самомплохом случаеh<1.44logn.Поэтому сложность поиска в двоичном Б-дереве и вАВЛ-дереве одинакова по порядку величины

2.4. МЕТОД КОДИРОВАНИЯ

Код Шеннона.

Построение осуществляется по следующему алгоритму:

1. Упорядочим символы исходного алфавита по убыванию их вероятностей.
2. Вычислим величины , которые называются кумулятивными вероятностями.
3. Представим в двоичной системе счисления и возьмем в качестве кодового слова первые знаков после запятой.

# 3. ОПИСАНИЕ СТРУКТУР ДАННЫХ И ИСПОЛЬЗОВАННЫХ АЛГОРИТМОВ

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

* 1. ИСПОЛЬЗОВАНИЕ СТРУКТУРЫ ДАННЫХ

**Структура ItemDb** содержит в себе одну строку, включающую переменные:

- Массив типа **char Author**, из 12 символов. Содержит информацию об авторе в формате <Фамилия>\_<буква>\_<буква>.

- Массив типа **char title**, из 32 символов. Содержит информацию о заглавии в формате <Имя>\_<Отчество>\_<Фамилия>.

- Массив типа **char publisher**, из 16 символов. Содержит название издательства.

- Переменная типа **short int year**. Содержит год издания.

- Переменная типа **short int pages**. Содержит количество страниц.

**Структура ListItem** элемент списка, содержащая в себе ссылку на следующий элемент.

**Структура CodeChar** структура для кодировки Шеннона. Содержит в себе переменные типа **char**, также типы данных **float**, **unsigned long long** для хранения вероятности и частоты

**Структура CodeTable** структура записи кодовой таблицы содержит размер и таблицу вероятностей.

3.2. ОСОБЕННОСТИ РЕАЛИЗАЦИИ АЛГОРИТМОВ

1. Загрузка и вывод базы данных

Для загрузки базы данных в main мы используем const char\* dbName = "testBase1.dat"

Для вывода базы данных разработана функция void printHeadDataBase для удобства вывода не всей базы данных используем offset и count для возможности пропустить n элементов и увидеть количество запрошенных элементов базы данных..

2. Вспомогательные функции и процедуры для сортировки данных

В качестве вспомогательной функции для сортировки MergeSort, были разработанны функции int cmpItemDB(ItemDB \* left, ItemDB \* right),которая сравнивает две записи сначала по издательству, а потом по автору (при условии, что издательства одинаковы), а также функция для сравнения строк по символьно и по длине int cmpStr(char \* str1, char \* str2, int n=-1) и функция измеряющая длину строки int getLenStr(char \* str).

3. Особенности реализации бинарного поиска

Бинарный поиск происходит в int findPosition(ItemDB\*\* data, int size, char\* key) , где мы изначально делим наш список на две части и берем средний элемент mid, мы его проверяем с ключом в данном случае это первые три буквы издательства , если совпадает, то поиск заканчивается. Если ключ меньше mid, то автоматически поиск будет идти в левой части, забыв про правую часть и граница сместится на -1, далее снова деление массива на 2. Если ключ опять не равен среднему элементу, а к примеру больше ключа, то левая граница поиска сместится на 1.

4. Вспомогательные функции и процедуры для построения двоичного Б-дерева

В качестве вспомогательных функций для построения Б-дерева мы используем TreeNode(int t1, bool leaf1) создаем узел, void insertNonFull(int k, ItemDB \* value) вставляем узел, void splitChild(int i, TreeNode\* y) разделяем потомков, void traverse() выполняем обход дерева, вводя г=его элементы на экран, идя от крайнего левого потомка к корню и к самому крайнему правому потомку, void insert(int k, ItemDB \* item) все элементы вставляем в данную функцию, вставляем целый элемент из базы данных.

5. Кодирование данных

Кодирование базы данных начинается CodeTable getTableForCharFieldWithOutCode(ListItem\* dataBase) cоздаем список вероятностей или подсчетов частоты для данного набора символов, чтобы была известна относительная частота появления каждого символа. void sortByProbability(CodeTable & table) отсортируем список символов в порядке убывания вероятности, наиболее вероятные слева и наименее вероятные справа. void shannon(int l, int h, CodeTable & table) разделим список на две части так, чтобы суммарная вероятность того, что обе части были максимально близки друг к другу, присваиваем значение 0 левой части и 1 для правой части, повторяем разделение списка и присваивание до тех пор пока, все символы не будут разделены на отдельные подгруппы и тогда код Шеннона будет считаться точным, если код символа уникален.

# 4. ОПИСАНИЕ ПРОГРАММЫ

## 4.1. ОСНОВНЫЕ ПЕРЕМЕННЫЕ И СТРУКТУРЫ

struct stack

{

char Author[12];

char FamPeople[32];

char izdat[16];

short int year;

short int page;

};

Запись, используемая для работы с базой данных.

struct Que

{

Que\* next;

stack\* data;

int W;

};

Структура (список), используемая при добавлении записи после поиска в очередь.

Que \*next **–** указатель на следующие элемент;

stack \*data **–** поле данных;

int W – вес записи.

struct Queue {

Que \*head;

Que \*tail;

};

Структура, которая используется для MergeSort.

struct Vertex

{

stack\* data;

Vertex \*left;

Vertex \*right;

Vertex \*next;

};

Структура, предназначенная для построения дерева А1.

struct VertexChar{

unsigned char data;

int count=0;

VertexChar\* left=NULL;

VertexChar\* right=NULL;

};

Структура, предназначенная для построения СДП-дерева для кодирования.

## 4.2. ОПИСАНИЕ ПОДПРОГРАММ

Запись и вывод:

1. void Record(stack \*Records[]) – запись в массив указателей.

2. void CreateInd(stack \*Records[], stack \*Ind[]) – формирование копии массива указателей.

3. void Print(stack\* Ind[]) – вывод записей из массива.

4. void PrintQueQue(Que \*head) – вывод записей из очереди.

5. void ObhodRight(Vertex \*p,int &i,int &x) – обход по дереву А1.

6. void PrintNext(Vertex \*p,int &i,int &x) – вывод, вызывающийся во время обхода по дереву А1.

7. void ObhodRight(VertexChar \*p) – обход по дереву с одновременным вывод по дереву хранящее в себе байты.

Сортировка:

8. int less(stack \*A,stack \*B) – сравнение двух записей для сортировки массива.

9. void QuickSort(int L, int R,stack\* Ind[]) – сортировка массива.

10. void StackToQueue (Que \*\*head, Que \*\*tail) – перенос из списка в очередь.

11. void Split (Que \*\*head, Que \*\*head\_a, Que \*\*head\_b) – разделение очереди.

12. int less(Que \*\*a, Que \*\*b) – сравнение двух элементов из очереди.

13. void MergeS (Que \*\*head\_a, Que \*\*head\_b, Que \*\*tail, int q, int r) – полное разделение списка.

14. void MergeSort (Que \*\*head, Que \*\*tail,int count) – сортировка очереди.

15. void QuickSort(int L, int R,unsigned char \*mas,float \*P) – сортировка массива для кодирования.

Поиск:

16. int Sravn(stack\* b,char a[]) – сравнение записей и ключа поиска.

17. int BSearch2(char x[],int n,stack\* Ind[]) – двоичный поиск второго типа.

18. void CreateQueQue(stack \*Ind[],int R,Que \*&head, Que \*&tail,char ch[],int &count) – запись из массива в очередь.

19. void FindTree(Vertex \*p,int D) – поиск в дереве.

Построение дерева:

20. void DOPNext(Vertex \*&p,stack\* D) – добавление записи в поле next вершины.

21. void AddDOP(Vertex\* &p, stack\* a) – добавление вершины в дерево.

22. void A1(Vertex \*&Root,Que\* p) – построение дерева А1.

Кодирование:

23. void ReadText(VertexChar \*&p,unsigned char ch,int &count) – добавление байта в дерево.

24. int ReadFromTXT(FILE \*pf,VertexChar \*&head,int &count, int &size) – считывание байтов из файла.

25. void VtoM(VertexChar \*p,unsigned char \*mas,float \*P,int count,int &k) – перенос байтов из дерева в массивы.

26. void Shennon(int size,float \*P,float \*Q,float \*Qc,int \*L,char \*\*C) – кодирование методом Шеннона.

Основная программа:

27. int main() – основная программа, в которой вызываются основные процедуры, а также выводится меню.

# 5. ИСХОДНЫЙ ТЕКСТ ПРОГРАММЫ

#include <stdlib.h>

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

#include <stdio.h>

#include <conio.h>

#include <cstring>

#include <iostream>

struct stack

{

char Author[12];

char FamPeople[32];

char izdat[16];

short int year;

short int page;

};

struct Que

{

Que\* next;

stack\* data;

int W;

};

struct Queue {

Que \*head;

Que \*tail;

};

struct Vertex

{

stack\* data;

Vertex \*left;

Vertex \*right;

Vertex \*next;

};

void Record(stack \*Records[]){

FILE \*pf;

stack \*p;

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

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

Records[i]=new stack;

fread(Records[i]->Author,sizeof(char),12,pf);

fread(Records[i]->FamPeople,sizeof(char),32,pf);

fread(Records[i]->izdat,sizeof(char),16,pf);

fread(&Records[i]->year,sizeof(short int),1,pf);

fread(&Records[i]->page,sizeof(short int),1,pf);

}

fclose(pf);

}

void CreateInd(stack \*Records[], stack \*Ind[]){

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

Ind[i]=Records[i];

}

}

void Print(stack\* Ind[])

{

int x;

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

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

printf("%4d %12s%32s%16s %4d%4d\n",i+1,Ind[i]->Author,Ind[i]->FamPeople,Ind[i]->izdat,Ind[i]->year,Ind[i]->page);

i++;

}

x=getch();

if(x==110) break;

}

}

int less(stack \*A,stack \*B){

for(int i=0;i<16;i++){

if(A->izdat[i]==' ' && B->izdat[i]!= ' ') return 1;

if(A->izdat[i]<B->izdat[i]) return 1;

if(A->izdat[i]>B->izdat[i]) return -1;

}

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

if(A->FamPeople[i]<B->FamPeople[i]) return 1;

if(A->FamPeople[i]>B->FamPeople[i]) return -1;

}

return 0;

}

void QuickSort(int L, int R,stack\* Ind[]){

int i,j;

int t;

stack \*x;

x=Ind[(R+L)/2];

i=L;

j=R;

while(i<j){

while(less(Ind[i],x)==1){

i++;

}

while(less(Ind[j],x)==-1){

j--;

}

if(i<=j){

stack\* temp;

temp=Ind[i];

Ind[i]=Ind[j];

Ind[j]=temp;

i++;

j--;

}

}

if(L<j) QuickSort(L,j,Ind);

if(i<R) QuickSort(i,R,Ind);

}

int Sravn(stack\* b,char a[]){

if(strncmp(b->izdat,a,3)==0) return 0;

if(strncmp(b->izdat,a,3)<0) return 1;

if(strncmp(b->izdat,a,3)>0) return -1;

}

int BSearch2(char x[],int n,stack\* Ind[]){

int L,R,m;

char ch;

L=0;

R=n-1;

while(L<R){

m=(L+R)/2;

if(Sravn(Ind[m],x)==1) L=m+1;

else R=m;

}

if(Sravn(Ind[R],x)==0) return R;

else return -1;

}

void CreateQueQue(stack \*Ind[],int R,Que \*&head, Que \*&tail,char ch[],int &count){

Que \*p;

while(Sravn(Ind[R],ch)==0 && R<=3999){

p=new Que;

p->data=Ind[R];

p->W=rand()%100;

p->next=NULL;

if(head!=NULL) tail->next=p;

else head=p;

tail=p;

R++;

count++;

}

}

void PrintQueQue(Que \*head){

int i=1,x;

while(head!=NULL){

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

if(head==NULL) break;

printf("%4d %12s%32s%16s %4d%4d %d\n",i,head->data->Author,head->data->FamPeople,head->data->izdat,head->data->year,head->data->page,head->W);

head=head->next;

i++;

}

x=getch();

if(x==110) break;

}

}

void StackToQueue (Que \*\*head, Que \*\*tail) {

(\*tail)->next=\*head;

\*tail=\*head;

\*head=(\*head)->next;

(\*tail)->next=NULL;

}

void Split (Que \*\*head, Que \*\*head\_a, Que \*\*head\_b) {

Que \*k, \*p;

\*head\_a=\*head;

\*head\_b=(\*head)->next;

k=\*head\_a;

p=\*head\_b;

while (p!=NULL) {

k->next=p->next;

k=p;

p=p->next;

}

}

int less(Que \*\*a, Que \*\*b){

if((\*a)->W<(\*b)->W) return 1;

if((\*a)->W>(\*b)->W) return -1;

return 0;

}

void MergeS (Que \*\*head\_a, Que \*\*head\_b, Que \*\*tail, int q, int r) {

while (q!=0 && r!=0) {

if (less(head\_a,head\_b)>=0) {

StackToQueue (head\_a, tail);

q--;

}

else {

StackToQueue (head\_b, tail);

r--;

}

}

while (q>0) {

StackToQueue (head\_a, tail);

q--;

}

while (r>0) {

StackToQueue (head\_b, tail);

r--;

}

}

void MergeSort (Que \*\*head, Que \*\*tail,int count) {

int i, m, q, p=1, r;

Que \*a=NULL, \*b=NULL;

Queue c[2];

Split (head, &a, &b);

int n=count;

while (p<n) {

c[0].tail=(Que\*)&c[0].head;

c[1].tail=(Que\*)&c[1].head;

i=0;

m=n;

while (m>0) {

if (m>=p) q=p;

else q=m;

m-=q;

if (m>=p) r=p;

else r=m;

m-=r;

MergeS (&a, &b, &c[i].tail, q, r);

i=1-i;

}

a=c[0].head;

b=c[1].head;

p\*=2;

}

c[0].tail->next=NULL;

\*head=c[0].head;

}

void DOPNext(Vertex \*&p,stack\* D){

if(p==NULL){

p=new Vertex;

p->data=D;

p->next=NULL;

}

else DOPNext(p->next,D);

}

void PrintNext(Vertex \*p,int &i,int &x){

if(p!=NULL){

if(i%20==0 && x!=110 && i!=0) x=getch();

if(x==110) return;

i++;

printf("%4d %12s%32s%16s %4d%4d\n",i,p->data->Author,p->data->FamPeople,p->data->izdat,p->data->year,p->data->page);

PrintNext(p->next,i,x);

}

}

void AddDOP(Vertex\* &p, stack\* a){

if(p==NULL){

p=new Vertex;

p->data=a;

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

}

else if(a->page<p->data->page) AddDOP(p->left,a);

else if(a->page>p->data->page) AddDOP(p->right,a);

else if(a->page==p->data->page) DOPNext(p,a);

}

void A1(Vertex \*&Root,Que\* p){

while(p!=NULL){

AddDOP(Root,p->data);

p=p->next;

}

}

void ObhodRight(Vertex \*p,int &i,int &x){

if(p!=NULL){

ObhodRight(p->left,i,x);

PrintNext(p,i,x);

ObhodRight(p->right,i,x);

}

}

void FindTree(Vertex \*p,int D){

int i=0,x=0;

if(p==NULL) printf("Nothing to find");

else if(p->data->page==D) PrintNext(p,i,x);

else{

if(p->data->page<D) FindTree(p->right,D);

if(p->data->page>D) FindTree(p->left,D);

}

}

struct VertexChar{

unsigned char data;

int count=0;

VertexChar\* left=NULL;

VertexChar\* right=NULL;

};

void QuickSort(int L, int R,unsigned char \*mas,float \*P){

int i,j;

int t;

float x;

x=P[(R+L)/2];

i=L;

j=R;

while(i<j){

while(P[i]>x){

i++;

}

while(P[j]<x){

j--;

}

if(i<=j){

char temp;

temp=mas[i];

mas[i]=mas[j];

mas[j]=temp;

float tempf;

tempf=P[i];

P[i]=P[j];

P[j]=tempf;

i++;

j--;

}

}

if(L<j) QuickSort(L,j,mas,P);

if(i<R) QuickSort(i,R,mas,P);

}

void ReadText(VertexChar \*&p,unsigned char ch,int &count){

if(p==NULL){

p=new VertexChar;

p->data=ch;

p->count++;

count++;

}

else if(p->data==ch){

p->count++;

}

else if(ch<p->data) { ReadText(p->left,ch,count);}

else if(ch>p->data) { ReadText(p->right,ch,count);}

}

int ReadFromTXT(FILE \*pf,VertexChar \*&head,int &count, int &size){

unsigned char ch;

if(pf==NULL){

printf("Не удалось открыть файл");

return -1;

}

printf("%c\n",ch);

while (fread(&ch,sizeof(unsigned char),1,pf)!=0)

{

count++;

ReadText(head,ch,size);

}

return 1;

}

void ObhodRight(VertexChar \*p){

if(p!=NULL){

ObhodRight(p->left);

printf("\n %3u |%4d",p->data,p->count);

ObhodRight(p->right);

}

}

void VtoM(VertexChar \*p,unsigned char \*mas,float \*P,int count,int &k){

if(p!=NULL){

VtoM(p->left,mas,P,count,k);

mas[k]=p->data;

P[k]=(float)p->count/count;

k++;

VtoM(p->right,mas,P,count,k);

}

}

void Shennon(int size,float \*P,float \*Q,float \*Qc,int \*L,char \*\*C){

Q[0]=0;

L[0]=-log2(P[0])+1;

for(int i=1;i<size;i++){

Q[i]=Q[i-1]+P[i-1];

L[i]=ceil(-log2f(P[i]));

}

for(int i=0;i<size;i++) Qc[i]=Q[i];

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

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

Q[i]=Q[i]\*2;

if(Q[i]>=1){

C[i][j]='1';

Q[i]-=1;

}

else C[i][j]='0';

}

}

}

int main()

{

stack\* Records[4000];

stack\* Ind[4000];

stack\* UnsortInd[4000];

int fin,count=0,i=0,x=0,cr,ttt;

Vertex \*Root=NULL;

Que \*head=NULL;

Que \*tail=NULL;

Record(Records);

CreateInd(Records,Ind);

CreateInd(Records,UnsortInd);

QuickSort(0,3999,Ind);

int j=0;

float Lsr=0,H=0;

float Psum=0;

int size\_char=0;

int count\_char=0;

VertexChar \*RootChar=NULL;

FILE \*pf=fopen("testBase1.dat","rb");

if(ReadFromTXT(pf,RootChar,count\_char,size\_char)==0) return 0;

fclose(pf);

unsigned char \*mas=new unsigned char[size\_char];

float \*P=new float[size\_char];

float \*Q=new float[size\_char];

float \*Qc=new float[size\_char];

int \*L=new int[size\_char];

char \*\*C=new char\*[size\_char];

for(int i=0;i<size\_char;i++) C[i]=new char[size\_char];

VtoM(RootChar,mas,P,count\_char,j);

QuickSort(0,size\_char-1,mas,P);

while(1){

system("cls");

printf("1 - UnSort | 2 - Sorted | 3 - Search | 4 - Tree A1 | 5 - Coding ");

int choice;

std:: cin >> choice;

switch(choice){

case 1:

Print(UnsortInd);

printf("\n 1 - exit ");

std:: cin >> ttt;

break;

case 2:

Print(Ind);

printf("\n 1 - exit ");

std ::cin >> ttt;

break;

case 3:

char ch[150];

printf("What do you want to find: ");

scanf("%s",&ch);

fin=BSearch2(ch,4000,Ind);

if(fin !=-1){

CreateQueQue(Ind,fin,head,tail,ch,count);

PrintQueQue(head);

}

printf("\n 1 - exit ");

std :: cin >> ttt;

break;

case 4:

Root=NULL;

A1(Root,head);

ObhodRight(Root,i,x);

printf("\n\nWhat do you want to find: ");

scanf("%d",&cr);

FindTree(Root,cr);

printf("\n 1 - exit ");

std :: cin >> ttt;

break;

case 5:

printf("Unsort\n");

printf(" Byte |Count ");

ObhodRight(RootChar);

printf("\nSum Count: %d\n",count\_char);

printf("\n\n\nSort\n");

printf(" I | Byte | Ver");

for(int i=0;i<size\_char;i++){

printf("\n%3d | %3u | %f",i,mas[i],P[i]);

Psum+=P[i];

}

Shennon(size\_char,P,Q,Qc,L,C);

printf("\n\n\n i| Byte | P | Q | L | Code");

for(int i=0;i<size\_char;i++){

printf("\n%3d| %3u | %f | %f | %2d | ",i,mas[i],P[i],Qc[i],L[i]);

for(int j=0;j<L[i];j++) printf("%c",C[i][j]);

}

for(int i=0;i<size\_char;i++){

Lsr+=P[i]\*L[i];

H+=-P[i]\*log2f(P[i]);

}

printf("\n Lsr = %.2f\n H = %.2f\n Psum = %.2f",Lsr,H,Psum);

printf("\n 1 - exit ");

std ::cin >> ttt;

break;

}

}

return 0;

}

# 6. РЕЗУЛЬТАТЫ

Рисунок 1. Меню.

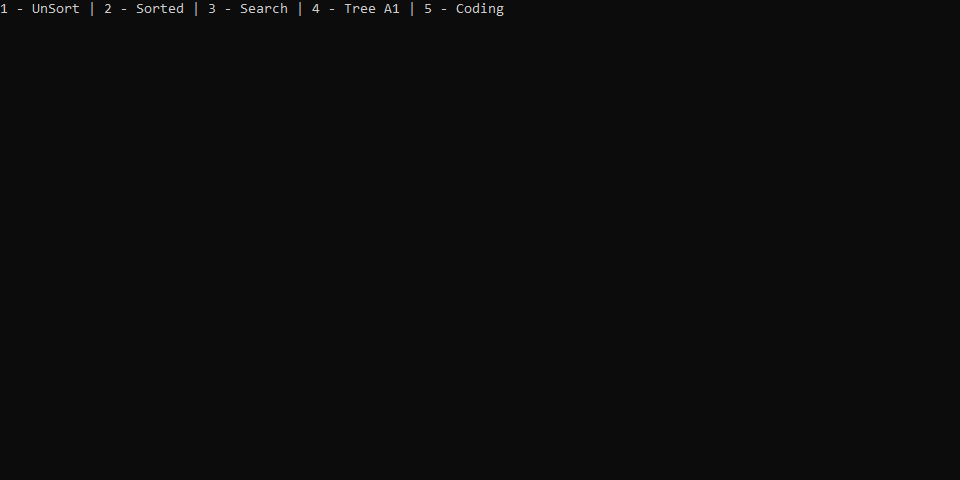


Рисунок 2. Неотсортированная база данных.



Рисунок 3. Отсортированная база данных.

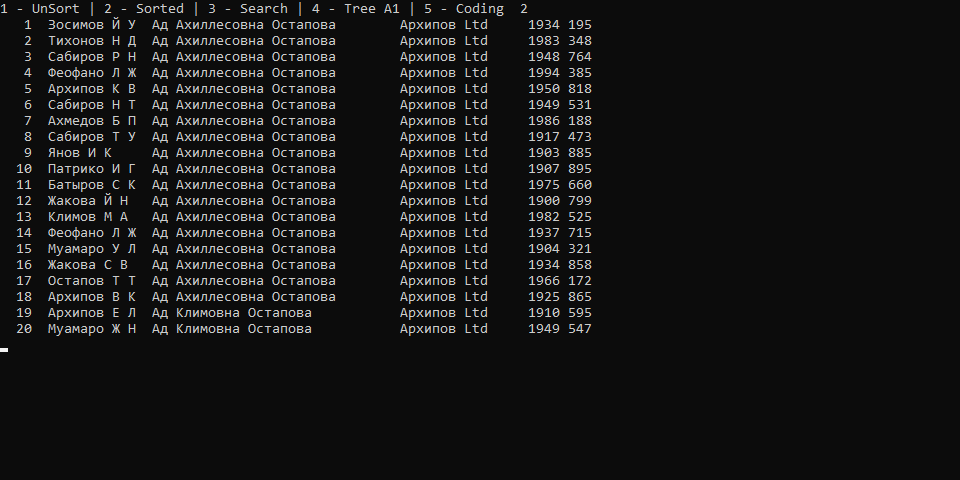


Рисунок 4. Запрос ключа поиска.



Рисунок 5. Результаты поиска.

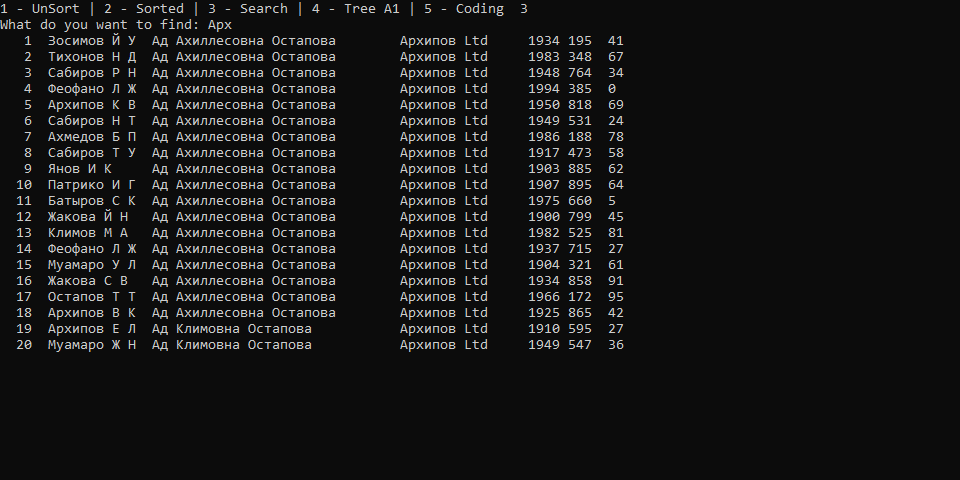


Рисунок 6. Вывод построенного дерева.

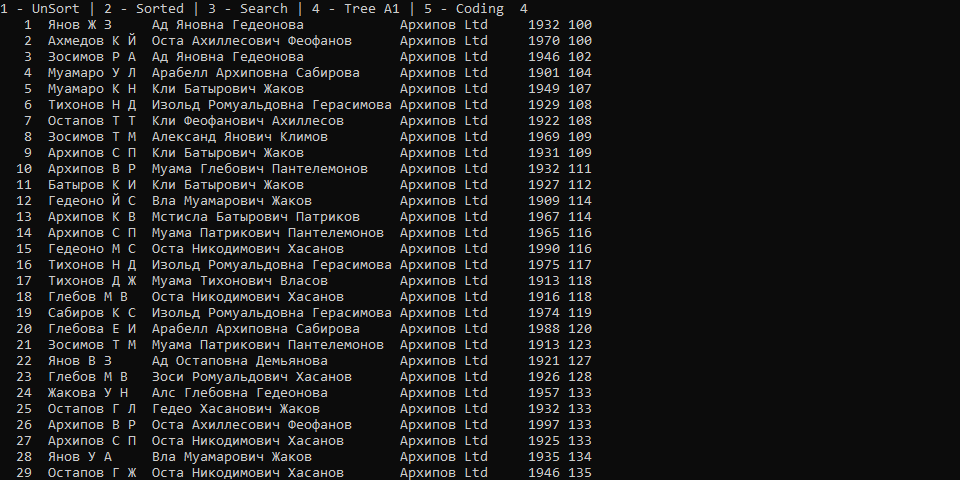


Рисунок 7. Запрос для поиска внутри дерева.

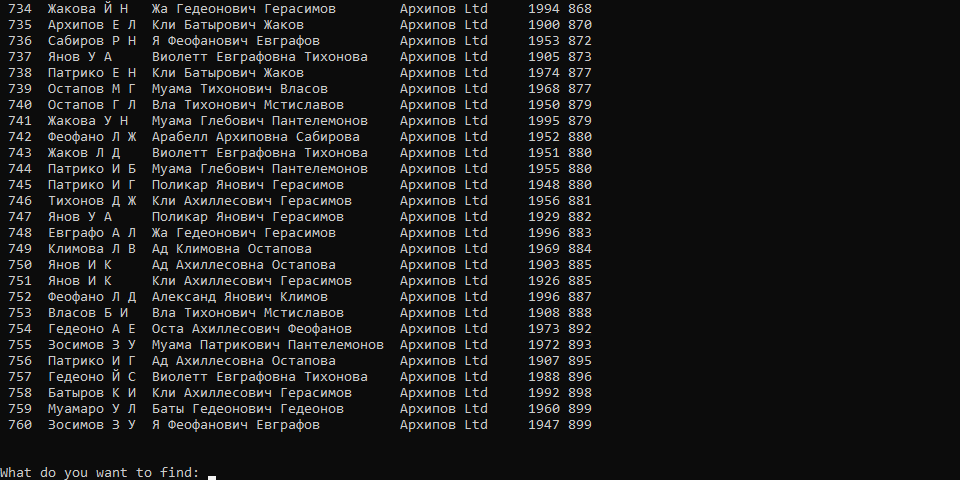


Рисунок 8. Результаты поиска внутри дерева.



Рисунок 9. Кодирование. Вывод дерева.

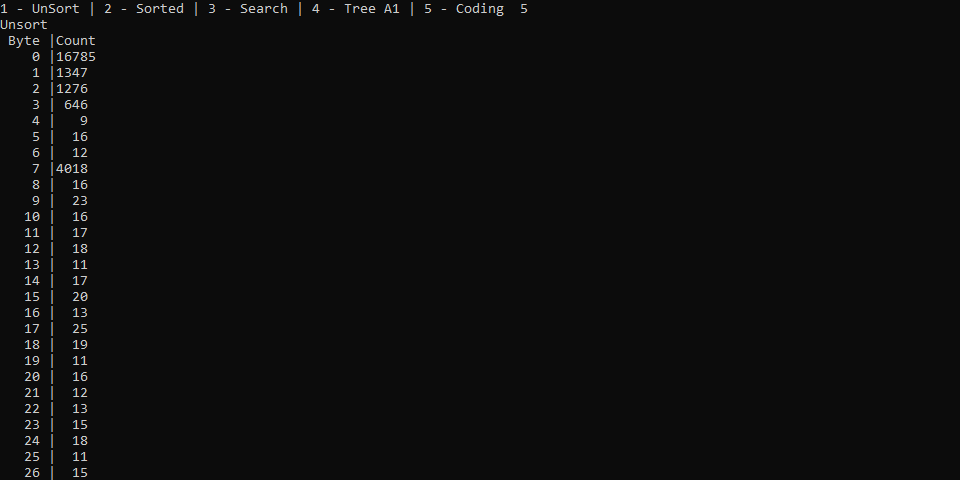


Рисунок 10. Кодирование. Вывод массива с вероятностями.



Рисунок 11. Кодирование. Вывод финальной таблицы.

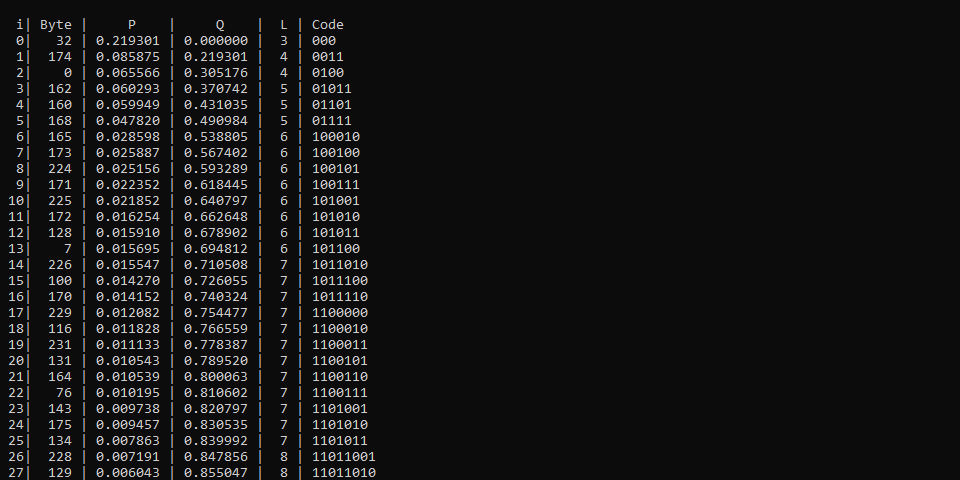
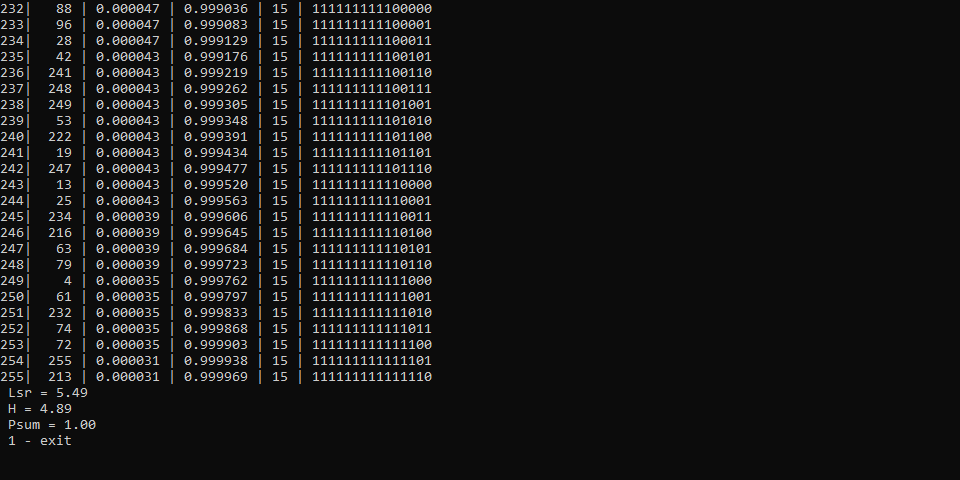


Рисунок 12. Кодирование. Вывод энтропии и средней длины.



# 7. ВЫВОДЫ

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