Федеральное агентство связи

Сибирский государственный университет телекоммуникаций и информатики

Кафедра прикладной математики и кибернетики

Курсовая работа

«Структуры и алгоритмы обработки данных»

Вариант 23

Выполнил: студент группы ИП-713

Михеев Н.А.

Проверил: ассистент кафедры ПМиК

Турцев А.А.

Новосибирск 2018

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 динамическую память компьютера в виде списка, произвести сортировку с использованием очередей **по дате поселения и названию улицы**, используя **метод прямого слияния** 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 двоичное Б-дерево, и предусмотреть возможность поиска в дереве по запросу.

Закодировать файл базы данных статическим **кодом Гилберта-Мура**, предварительно оценив вероятности всех встречающихся в ней символов. Построенный код вывести на экран, подсчитать энтропию и среднюю длину кодового слова.

База данных "Населенный пункт"

Стpуктуpа записи:

ФИО гражданина: текстовое поле 32 символа

фоpмат <Фамилия>\_<Имя>\_<Отчество>

Название улицы: текстовое поле 18 символов

Номер дома: целое число

Номер квартиры: целое число

Дата поселения: текстовое поле 10 символов

фоpмат дд-мм-гг

Пpимеp записи из БД:

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

Ленина\_\_\_\_\_\_\_\_\_\_\_\_\_\_

10

67

29-02-65

Варианты условий упорядочения и ключи поиска (К):

По дням по дате поселения и названию улицы, К = год поселения.

Ключ в дереве – номер квартиры (целое число).

# 2. ОСНОВНЫЕ ИДЕИ И ХАРАКТЕРИСТИКИ ПРИМЕНЯЕМЫХ МЕТОДОВ

## 2.1. МЕТОД СОРТИРОВКИ

*Метод прямого слияния*

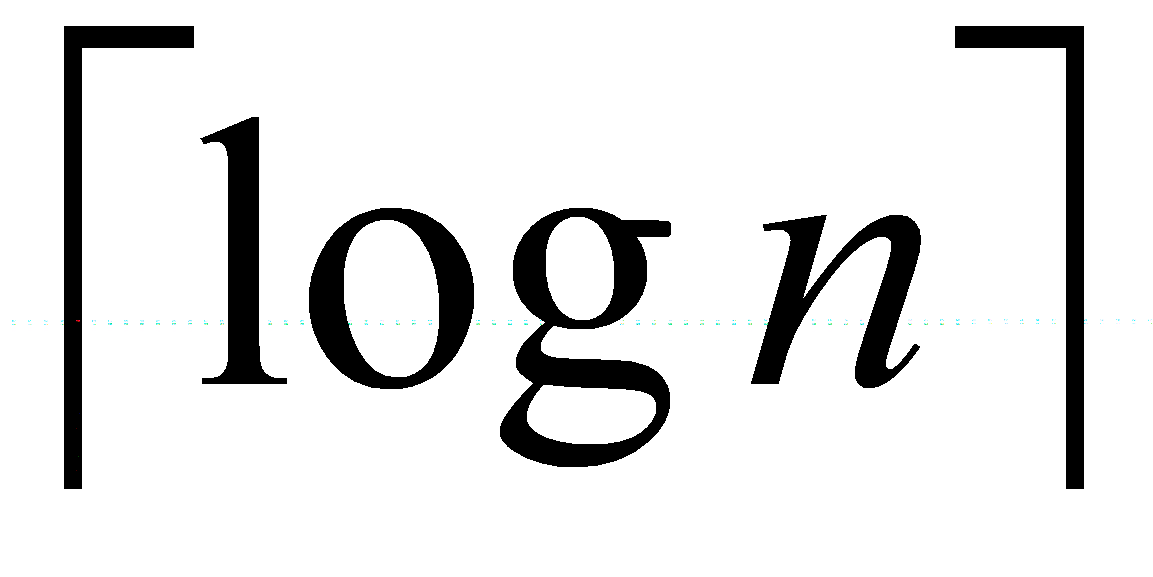
В основе метода прямого слияния лежит операция слияния серий. р-серией называется упорядоченная последовательность из р элементов.

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.

Трудоёмкость метода прямого 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:\Documents and Settings\Belida\Application Data\Microsoft\Word\СибГУТИ!!!\5_САОД\SAOD-BOOK\img\sample\f_00.GIF

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

C:\Documents and Settings\Belida\Application Data\Microsoft\Word\СибГУТИ!!!\5_САОД\SAOD-BOOK\img\sample\f_01.GIF

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.

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.

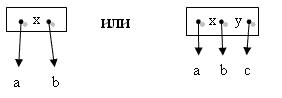
Верхняя оценка трудоёмкости алгоритма двоичного поиска такова. На каждой итерации поиска необходимо два сравнение для первой версии, одно сравнение для второй версии. Количество итераций не больше, чем C:\Documents and Settings\Belida\Application Data\Microsoft\Word\СибГУТИ!!!\5_САОД\SAOD-BOOK\img\sample\f_02.GIF. Таким образом, трудоёмкость двоичного поиска в обоих случаях

C:\Documents and Settings\Belida\Application Data\Microsoft\Word\СибГУТИ!!!\5_САОД\SAOD-BOOK\img\sample\f_03.GIF

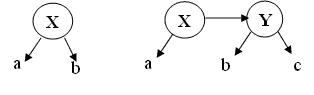
2.3. ДЕРЕВО И ПОИСК ПО ДЕРЕВУ

*Двоичное Б-дерево*

Двоичное Б-дерево состоит из вершин (страниц) с одним или двумя элементами. Следовательно, каждая страница содержит две или три ссылки на поддеревья. На рисунке показаны примеры страниц Б – дерева при m = 1.



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



Таким образом, страницы Б-дерева теряют свою целостность и элементы списков начинают играть роль вершин в двоичном дереве. Однако остается необходимость делать различия между ссылками на потомков (вертикальными) и ссылками на одном уровне (горизонтальными), а также следить, чтобы все листья были на одном уровне.

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

Высота двоичного Б-дерева

C:\Documents and Settings\Belida\Application Data\Microsoft\Word\СибГУТИ!!!\5_САОД\SAOD-BOOK\html\img\posobie1_html_m4d5432aa.gif.

Если рассматривать двоичное Б-дерево как обычное двоичное дерево, то его высота может увеличиться вдвое, т.е. . Для сравнения, в АВЛ-дереве даже в самом плохом случае h<1.44 log n. Поэтому сложность поиска в двоичном Б-дереве и в АВЛ-дереве одинакова по порядку величины.

При построении двоичного Б-дерева реже приходится переставлять вершины, поэтому АВЛ-деревья предпочтительней в тех случаях, когда поиск ключей происходит значительно чаще, чем добавление новых элементов. Кроме того, существует зависимость от особенностей реализации, поэтому вопрос о применение того или иного тапа деревьев следует решать индивидуально для каждого конкретного случая.

2.4. МЕТОД КОДИРОВАНИЯ

Алфавитный код Гилберта – Мура

Рассмотрим источник с алфавитом А={*a1,a2,…,an*} и вероятностями *p1,…pn*. Пусть символы алфавита некоторым образом упорядочены, например, *a1≤a2≤…≤an*. *Алфавитным* называется код, в котором кодовые слова лексико-графически упорядочены, т.е. *φ(a1)≤φ(a2)≤…≤φ(an).*

Е.Н. Гилбертом и Э.Ф. Муром предложили метод построения алфавитного кода, для которого *Lср < H+2.* Процесс построения происходит следующим образом.

1. Составим суммы *Qi, i=1,n*, вычисленные следующим образом:

*Q1=p1/2, Q2=p1+p2/2, Q3=p1+p2+p3/2,…, Qn=p1+p2+…+pn-1+pn/2*.

1. Представим суммы *Qi* в двоичном виде.
2. В качестве кодовых слов возьмем ⎡-log2*pi*⎤ +1 младших бит в двоичном представлении *Qi*.

**Пример**. Пусть дан алфавит A={*a1, a2, a3, a4, a5, a6*} с вероятностями *p1*=0.36, *p2*=0.18, *p3*=0.18, *p4*=0.12, *p5*=0.09, *p6*=0.07. Построенный код приведен в таблице.

Таблица 1 Код *Гилберта-Мура*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *ai* | *Pi* | *Qi* | *Li* | кодовое слово |
| *a1*  *a2*  *a3*  *a4*  *a5*  *a6* | 1/23≤0.18  1/23≤0.18<1/22  1/22≤0.36<1/21  1/24≤0.07  1/24≤0.09  1/24≤0.12 | 0.09  0.27  0.54  0.755  0.835  0.94 | 4  4  3  5  5  5 | 0001  0100  100  11000  11010  11110 |

Средняя длина кодового слова не превышает значения энтропии плюс 2

*Lср*=4**.**0.18+4**.**0.18+3**.**0.36+5**.**0.07+5**.**0.09+5**.**0.12=3.92<2.37+2

1. ОСОБЕННОСТИ РЕАЛИЗАЦИИ АЛГОРИТМОВ

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

1. Интерфейс программы

Для организации интерфейса было реализовано меню, которое обеспечивает корректное и незатруднительное использование программы и предоставляет возможность многократного выбора различных вариантов обработки базы данных, в зависимости от задач пользователя.

1. Загрузка и вывод базы данных

Для загрузки базы данных разработана процедура void read\_file(), в которой производится считывание записей типа struct record (“запись”), а из них формируется очередь struct queue. Данная процедура вызывается независимо от желания пользователя, в то время как остальные он может выбрать посредствам меню.

За вывод элементов считанной базы данных отвечает процедура void print\_spis. Она предоставляет возможность постраничного просмотра базы данных (по 20 элементов на странице), смена страниц осуществляется нажатием управляющих клавиш «1» и «0» на клавиатуре. Есть возможность прервать просмотр в любой момент времени нажатием любой другой клавиши.

1. Вспомогательные функции и процедуры MergeSort для сортировки данных

Так как нужно было сортировать по дате, было принято решение добавить дополнительные аргументы для сортировке отдельных символов, и таким образом отсортировать сначала по дням, потом по месяцам, и только потом по годам. Все сравнения были произведены в функции-компараторе void Comparator(string str1, string str2, string str3, string str4).

1. Вспомогательные функции для построения и работы с ДБД

При добавлении вершин в дерево, может возникнуть ситуация, когда некоторые ключевые поля элементов одинаковые. Чтобы данные не терялись было решено элементы с одинаковыми ключами записывать в список, для чего была модифицирована структура дерева: struct node {record \*data; node \*left; node \*right; spis \*dbd\_head; spis \*dbd\_tail; int bal; }\*root;

1. Вспомогательные функции для кодирования

Функция void open\_file\_base() подсчитывает количество всех встречающихся в базе данных символов (в том числе и служебных). Для сортировки массив символов по убыванию вероятности встречи символов, была создана функция void sort(). Функция void print\_code() выводит таблицу, показывающую верятность встречи каждого символа, их кодовые слова и длины кодовых слов. Также выводит среднюю длину кодового слова и энтропию.

4. ОПИСАНИЕ ПРОГРАММЫ

4.1 ОСНОВНЫЕ ПЕРЕМЕННЫЕ И СТРУКТУРЫ

1. struct record{char name[32]; char street[18]; short int house; short int flat; char date[10]; } \*tt; - основная структура, куда записываются все значения полей базы данных.

2. struct spis{spis \*next; record \*data;}\*base\_head, \*base\_tail, \*head, \*tail, \*search\_head, \*search\_tail; - для записи базового, отсортированного и после бинарного поиска базы

3. struct queue{spis \*head; \*tail; }; - структура для очередей для MergeSort

4. struct node{record \*data; node \*left; node \*right; spis \*dbd\_head; spis \*dbd\_tail; int bal;}\*root; - для построения ДБД

5. struct encoding{unsigned char symbol; p; code[20]; int length; } array\_code[256]; - для кодировки Гилберта-Мура

6. spis \*PArray[4000], \*p; - массив указателей для бинарного поиска

7. node \*q = NULL; - для поиска по дереву

8. char \*key; - ключ для бинарного поиска

9. string str; - строка для бинарного поиска

10. int size = 4000; - размер исходного файла

11. int i = 0, button, pos; - счетчик для циклов, кнопка меню, переменная для бинарного поиска

12. short int tree\_key; - ключ поиска для дерева

13. spis \*mas[4000]; - массив указателей

14. bool VR = 1, HR = 1; - вертикальный и горизонтальный рост для ДБД

4.2 ОПИСАНИЕ ПОДПРОГРАММ

Функции для сортировки

1. bool Comparator(string str1, string str2, string str3, string str4) – компаратор для сортировки
2. int Separation(spis\* &head, spis\* &a, spis\* &b, int &n) – разбиение на очереди
3. int MergeSort(spis\* &head) – сортирует базу данных по дате и улице
4. void sort() – сортировка по алфавиту

Функции для построения и работы с деревом

1. void DBD (node \*&p, record \*D) – построение двоичного Б-дерева
2. void leftright\_representation(struct node \*p) – обход дерева слева направо
3. void print\_spis(spis \*head) – вывод списка в дереве
4. node\* treeSearch(node \*p, short int tree\_key) – поиск в дереве

Функции для кодирования

1. void open\_file\_base() - подсчёт всех символов встречающихся в базе данных.

2. void shanon() - кодировка методом Шеннона.

3. void print\_code() - вывод таблицы с символами, их вероятностями и кодовыми словами.

Основная программа

Main() – вызывает все необходимые методы, создают главное меню для работы с базой данных.

5.ИСХОДНЫЙ ТЕКСТ ПРОГРАММЫ

#include <fstream>

#include <iostream>

#include <cstdio>

#include <cstdlib>

#include <string>

#include <cmath>

using namespace std;

struct record

{

char name[32];

char street[18];

short int house;

short int flat;

char date[10];

} \*tt;

struct spis

{

spis \*next;

record \*data;

}\*base\_head, \*base\_tail, \*head, \*tail, \*search\_head, \*search\_tail;

struct queue

{

spis \*head;

spis \*tail;

};

struct node

{

record \*data;

node \*left;

node \*right;

spis \*dbd\_head;

spis \*dbd\_tail;

int bal;

}\*root;

int size\_array\_of\_code = 0;

struct encoding

{

unsigned char symbol;

float p;

char code[20];

int length;

} array\_code[256];

void add\_queue(spis\* &head, spis\* &tail, record \*array)

{

spis \*p = new spis;

p -> data = array;

p -> next = NULL;

if (head == NULL)

{

head = tail = p;

head -> next = NULL;

}

else

{

tail -> next = p;

tail = p;

}

}

void read\_file()

{

fstream file("testBase4.dat", ios::binary | ios::in);

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

{

tt = new record;

file.seekp(64\*i, ios::beg);

file.read((char\*)tt, sizeof(record));

add\_queue(head, tail, tt);

add\_queue(base\_head, base\_tail, tt);

}

file.close();

}

void print\_spis(spis \*head)

{

int k = 0, i = 0, size = 4000;

char c;

spis \*p;

p = head;

do

{

system("cls");

k = i + 20;

for(i; i < k; i++)

{

if(p == NULL)

return;

cout << i + 1 << "\t";

cout << p -> data -> name << "\t ";

cout << p -> data -> street << "\t ";

cout << p -> data -> house << "\t ";

cout << p -> data -> flat << "\t ";

cout << p -> data -> date << endl;

p = p -> next;

}

cout << "\nPrint 0 to stop or any key to continue..." << endl;

cin.ignore(1);

cin >> c;

if(c == '0')

break;

}

while(i < size);

}

void printmas(spis \*mas[])

{

int i = 1, k = 0;

int move;

bool key = true;

do

{

if(key)

{

k = i + 20;

for(; i < k; i++)

{

cout << i << "\t";

cout << mas[i] -> data -> name << "\t ";

cout << mas[i] -> data -> street << "\t ";

cout << mas[i] -> data -> house << "\t ";

cout << mas[i] -> data -> flat << "\t ";

cout << mas[i] -> data -> date << endl;

}

}

else

{

k = i - 20;

for(i = i - 40; i < k; i++)

{

cout << i << "\t";

cout << mas[i] -> data -> name << "\t ";

cout << mas[i] -> data -> street << "\t ";

cout << mas[i] -> data -> house << "\t ";

cout << mas[i] -> data -> flat << "\t ";

cout << mas[i] -> data -> date << endl;

}

}

cout << "1-next\n2-back" << endl;

cin >> move;

if(move == 1)

if(i < 4000)

key = true;

else

return;

if(move == 2)

if(i > 21)

key = false;

else

return;

if(move != 1 && move != 2)

return;

}while(1);

}

bool Comparator(string str1, string str2, string str3, string str4)

{

int year = str1.compare(6, 2, str2, 6, 2);

int month = str1.compare(3, 2, str2, 3, 2);

int day = str1.compare(0, 2, str2, 0, 2);

if((year <= 0) && (month <= 0) && (day <= 0) && (str3 < str4))

{

return true;

}

else if((year <= 0) && (month <= 0) && (day < 0))

{

return true;

}

else if((year <= 0) && (month < 0))

{

return true;

}

else if(year < 0)

{

return true;

}

return false;

}

int Separation(spis\* &head, spis\* &a, spis\* &b, int &n)

{

spis \*p,\*q;

a = head;

b = head -> next;

n = 1;

p = a;

q = b;

while (q)

{

n++;

p -> next = q -> next;

p = q;

q = q -> next;

}

}

int MergeSort(spis\* &head)

{

int m, n;

spis \*a, \*b, \*s, \*d;

queue c[2];

int p = 1, q, r;

if(head == 0)

return 0;

Separation(head, a, b, n);

while(p < n)

{

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

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

int 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;

while((q > 0) && (r > 0))

{

string str1 = a -> data -> date;

string str2 = b -> data -> date;

string str3 = a -> data -> street;

string str4 = b -> data -> street;

if(Comparator(str1, str2, str3, str4))

{

c[i].tail -> next = a;

c[i].tail = a;

a = a -> next;

q--;

}

else

{

c[i].tail -> next = b;

c[i].tail = b;

b = b -> next;

r--;

}

}

while(q > 0)

{

c[i].tail -> next = a;

c[i].tail = a;

a = a -> next;

q--;

}

while(r > 0)

{

c[i].tail->next = b;

c[i].tail = b;

b = b->next;

r--;

}

i = 1-i;

}

a = c[0].head;

b = c[1].head;

p\*= 2;

}

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

head = c[0].head;

}

int Binary\_Search(int n, spis\* A[], char \*key)

{

int m = 0;

int l = 0, r = n - 1;

string str;

while(l < r)

{

m = (l + r) / 2;

str = A[m] -> data -> date;

if(str.compare(6, 2, key, 0, 2) < 0)

l = m + 1;

else

r = m;

}

str = A[r] -> data -> date;

if(str.compare(6, 2, key, 0, 2) == 0)

return r;

else

return -1;

}

bool VR = 1, HR = 1;

void DBD (node \*&p, record \*D)

{

node \*q = NULL;

if (p == NULL)

{

p = new node;

p -> data = D;

p -> dbd\_head = NULL;

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

p -> bal = 0;

VR = true;

}

else if((p -> data -> flat) > (D -> flat))

{

DBD(p -> left, D);

if(VR == true)

{

if(p -> bal == 0)

{

q = p -> left;

p -> left = q -> right;

q -> right = p;

p = q;

q -> bal = 1;

VR = false;

HR = true;

}

else

{

p -> bal = 0;

VR = true;

HR = false;

}

}

else

HR = false;

}

else if((p -> data -> flat) < (D -> flat))

{

DBD(p -> right, D);

if(VR == true)

{

p -> bal = 1;

HR = true;

VR = false;

}

else if(HR == true)

{

if(p -> bal == 1)

{

q = p -> right;

p -> bal = 0;

q -> bal = 0;

p -> right = q -> left;

q -> left = p;

p = q;

VR = true;

HR = false;

}

else

HR = false;

}

}

else if((p -> data -> flat) == (D -> flat))

{

add\_queue(p -> dbd\_head, p -> dbd\_tail, D);

}

}

void leftright\_representation(struct node \*p) //обход слева направо

{

if (p == NULL) return;

else

{

leftright\_representation(p->left);

cout << p -> data -> name << "\t ";

cout << p -> data -> street << "\t ";

cout << p -> data -> house << "\t ";

cout << p -> data -> flat << "\t ";

cout << p -> data -> date << endl;

leftright\_representation(p->right);

}

}

int count\_of\_symbols = 0;

void open\_file\_base()

{

unsigned char ch;

FILE \*file = fopen("testBase4.dat", "rb");

if (file == NULL)

{

printf("Database can't open. Press any key for return to menu.");

return;

}

for (int i = 0; i < 256; i++)

{

array\_code[i].symbol = 1;

array\_code[i].p = 0;

array\_code[i].length = 0;

}

while (!feof(file))

{

fscanf(file, "%c", &ch);

array\_code[ch].p += 1.0;

array\_code[ch].symbol = ch;

count\_of\_symbols++;

}

fclose(file);

for (int i = 0; i < 256; i++)

{

if (array\_code[i].p != (float)0)

{

array\_code[i].p /= (float)count\_of\_symbols;

size\_array\_of\_code++;

}

}

}

void sort()

{

encoding temp;

for(int i = 0; i < 256; i++)

for(int j = 255; j > i; j--)

{

if(array\_code[j].p > array\_code[j - 1].p)

{

temp = array\_code[j];

array\_code[j] = array\_code[j-1];

array\_code[j-1] = temp;

}

}

}

void encoding\_gilbert\_mur()

{

float Q[size\_array\_of\_code], sum = 0;

char ch;

Q[0] = array\_code[0].p / 2; array\_code[0].length = (int)ceilf(-log2f(array\_code[0].p)) + 1;

sum = array\_code[0].p;

for (int i = 1; i < size\_array\_of\_code; i++)

{

Q[i] = sum + array\_code[i].p / 2;

array\_code[i].length = (int)ceilf(-log2f(array\_code[i].p)) + 1;

sum += array\_code[i].p;

}

for (int i = 0; i < size\_array\_of\_code; i++)

{

for (int j = 0; j < array\_code[i].length; j++)

{

Q[i] \*= 2;

int digit = (int)Q[i];

if (digit == 0)

ch = '0';

if (digit == 1)

ch = '1';

array\_code[i].code[j] = ch;

if (Q[i] > 1)

Q[i]--;

}

}

}

void print\_code()

{

cout << "Symbol Chance Code word Length of code word\n" <<endl;

for (int i = 0; i < size\_array\_of\_code; i++)

{

printf(" %c ", array\_code[i].symbol);

printf(" ");

printf("%f", array\_code[i].p);

printf(" ");

printf("%s\t\t", array\_code[i].code);

printf(" ");

printf("%d\n", array\_code[i].length);

}

float entropy = 0, average\_length = 0;

float wes = 0;

for (int i = 0; i < size\_array\_of\_code; i++)

{

average\_length += array\_code[i].length \* array\_code[i].p;

entropy += array\_code[i].p \* -log2f(array\_code[i].p);

}

wes = average\_length \* count\_of\_symbols;

printf("\n\nAverage length of code word: %f\n", average\_length);

printf("Entropy of source file: %f\n", entropy);

printf("Code mass: %.3f KB\n", wes/8/1024);

}

node\* treeSearch(node \*p, short int tree\_key)

{

short int house = p -> data -> flat;

short int a = house - tree\_key;

if (p != NULL)

{

if(a > 0)

{

treeSearch(p -> left, tree\_key);

}

if(a < 0)

{

treeSearch(p -> right, tree\_key);

}

if(a == 0)

{

return p;

}

}

else

{

cout <<"Not found"<< endl;

system("PAUSE");

return NULL;

}

}

int main()

{

spis \*PArray[4000], \*p;

node \*q = NULL;

char \*key;

string str;

int size = 4000;

int i = 0, button, pos;

short int tree\_key;

spis \*mas[4000];

read\_file();

MergeSort(head); //merge sort

for(p = head, i = 0; p; p = p -> next, i++) //Points array

{

PArray[i] = p;

}

for(p = base\_head, i = 0; p; p = p->next, i++)

{

mas[i] = p;

}

open\_file\_base();

sort();

encoding\_gilbert\_mur();

while(1)

{

system("cls");

cout << "1. Main Base" << endl;

cout << "2. Sorted Base" << endl;

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

cout << "4. DBD-Tree" << endl;

cout << "5. Base Coding" <<endl;

cout << "0. Exit" << endl;

cin >> button;

system("cls");

switch(button)

{

case 1:

printmas(mas); //start base

break;

case 2:

print\_spis(head); //sorted base

break;

case 3:

cout << "Print a key:";

cin >> key;

pos = Binary\_Search(size, PArray, key);

if(pos != -1)

{

str = PArray[pos] -> data -> date;

while(str.compare(6, 2, key, 0, 2) == 0)

{

add\_queue(search\_head, search\_tail, PArray[pos] -> data);

pos++;

str = PArray[pos] -> data -> date;

}

}

print\_spis(search\_head); //search base

break;

case 4:

{

if (search\_head == NULL)

{

cout << "Make search first!" << endl;

}

else

{

for(p = search\_head; p; p = p -> next)

{

DBD(root, p -> data);

}

leftright\_representation(root);

cout << "What house would you like to find?" << endl;

cin >> tree\_key;

node \*q = treeSearch(root, tree\_key);

if(q != NULL)

{

cout << q -> data -> name << "\t ";

cout << q -> data -> street << "\t ";

cout << q -> data -> house << "\t ";

cout << q -> data -> flat << "\t ";

cout << q -> data -> date << endl;

print\_spis(q -> dbd\_head);

}

}

system("PAUSE");

}

break;

case 5:

print\_code();

system("PAUSE");

break;

default:

exit(0);

}

}

return 0;

}

6.РЕЗУЛЬТАТЫ

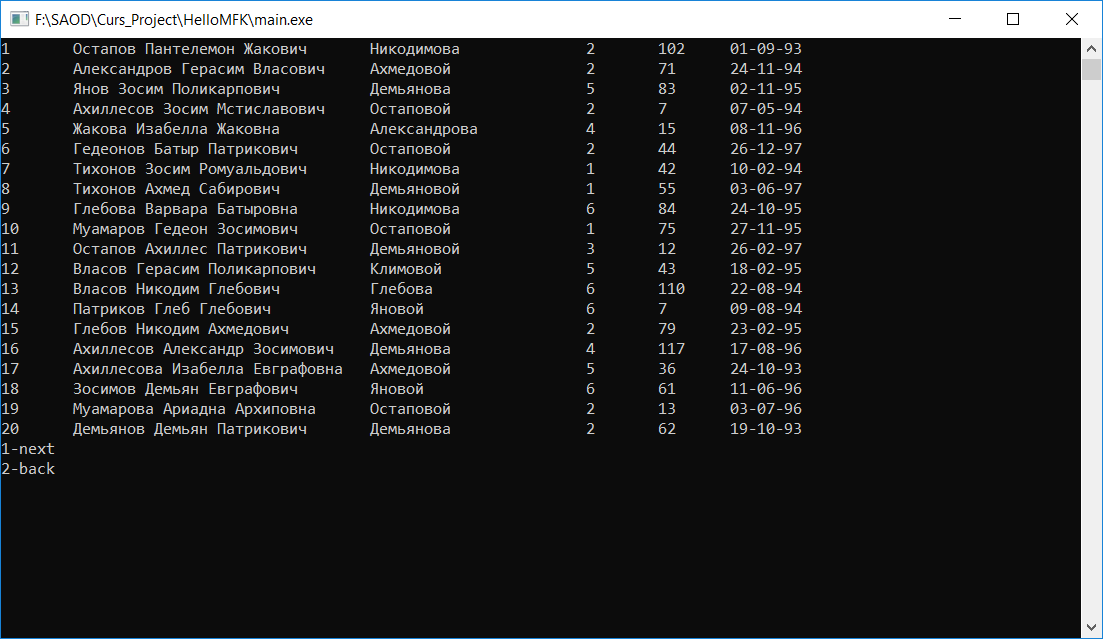


Рисунок 1. Исходная база данных

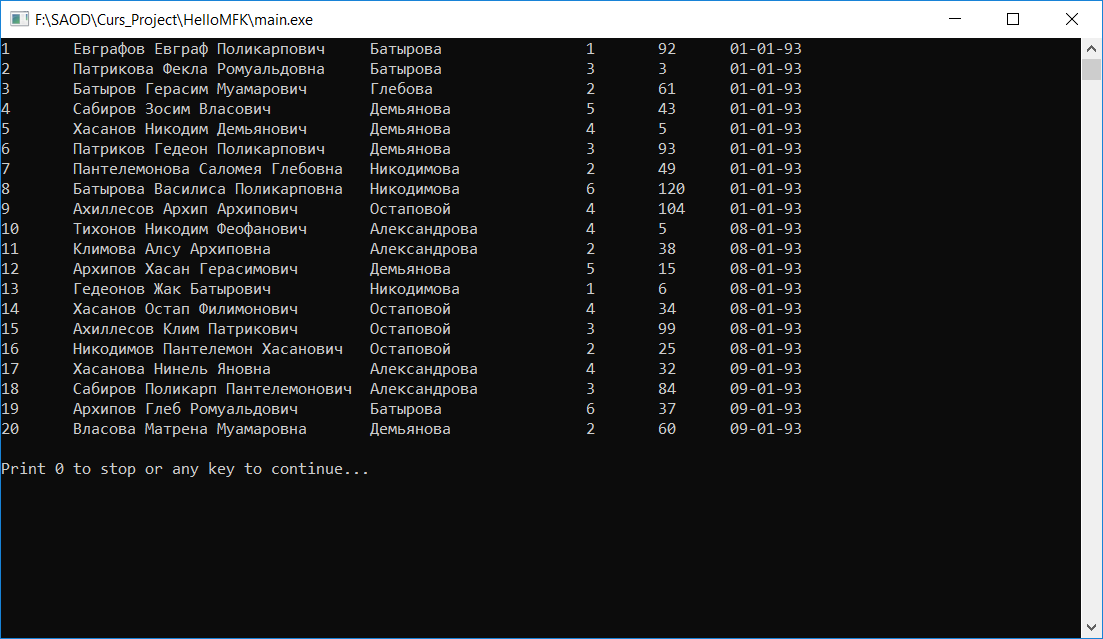


Рисунок 2. Отсортированная база данных

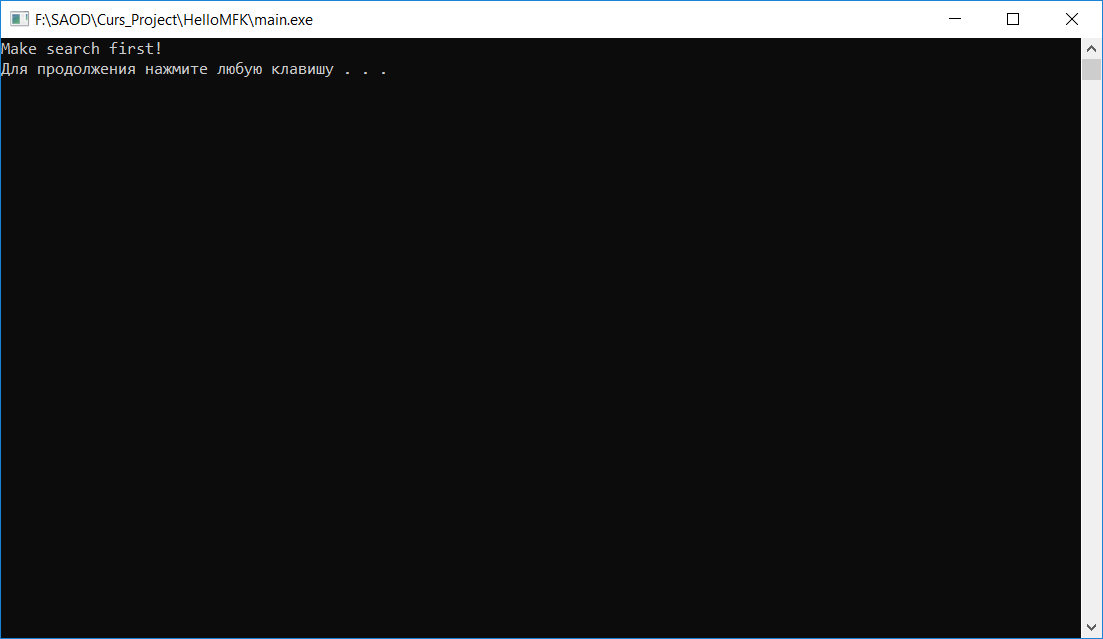


Рисунок 3. Попытка вызова дерева без предварительного поиска

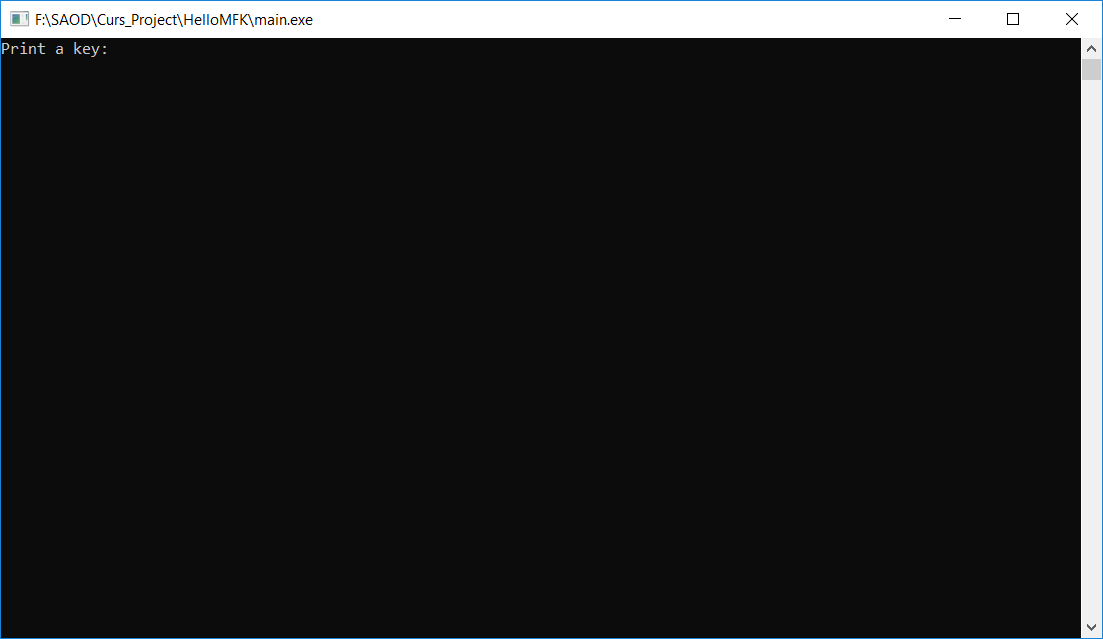


Рисунок 4. Ожидание ввода ключа поиска

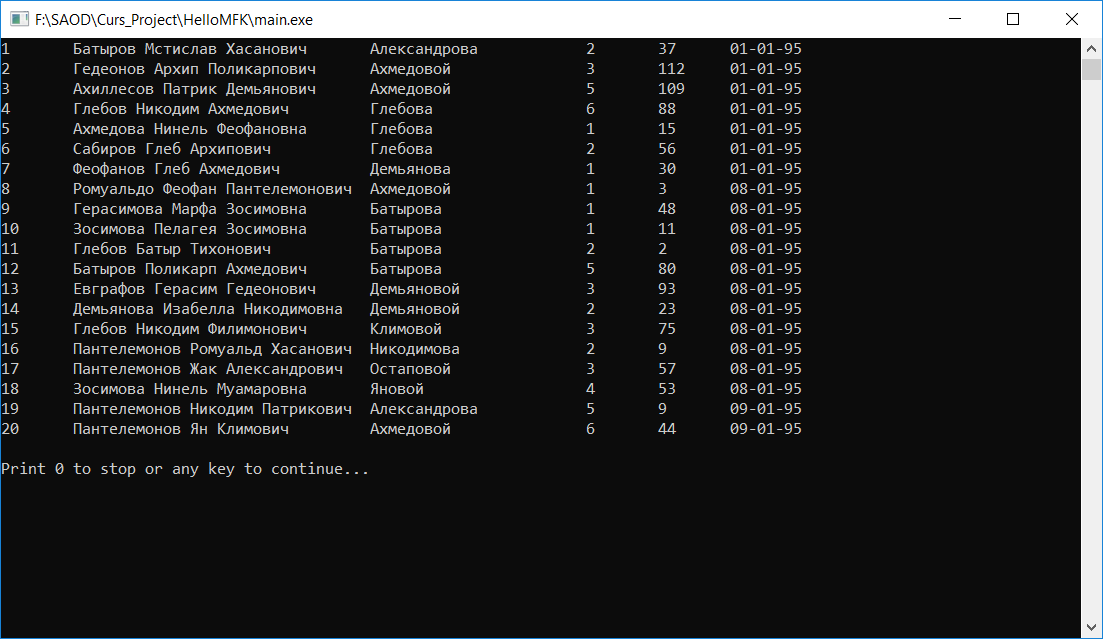


Рисунок 5. Результат бинарного поиска

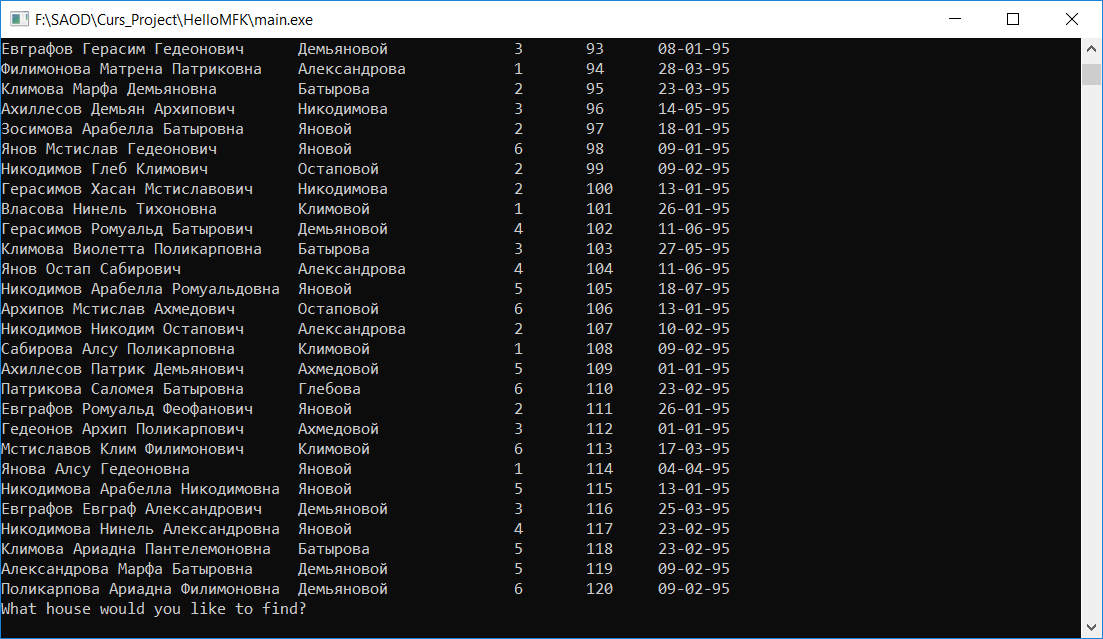


Рисунок 6. Вывод построенного ДБД дерева и ожидание ввода ключа для поиска по дереву

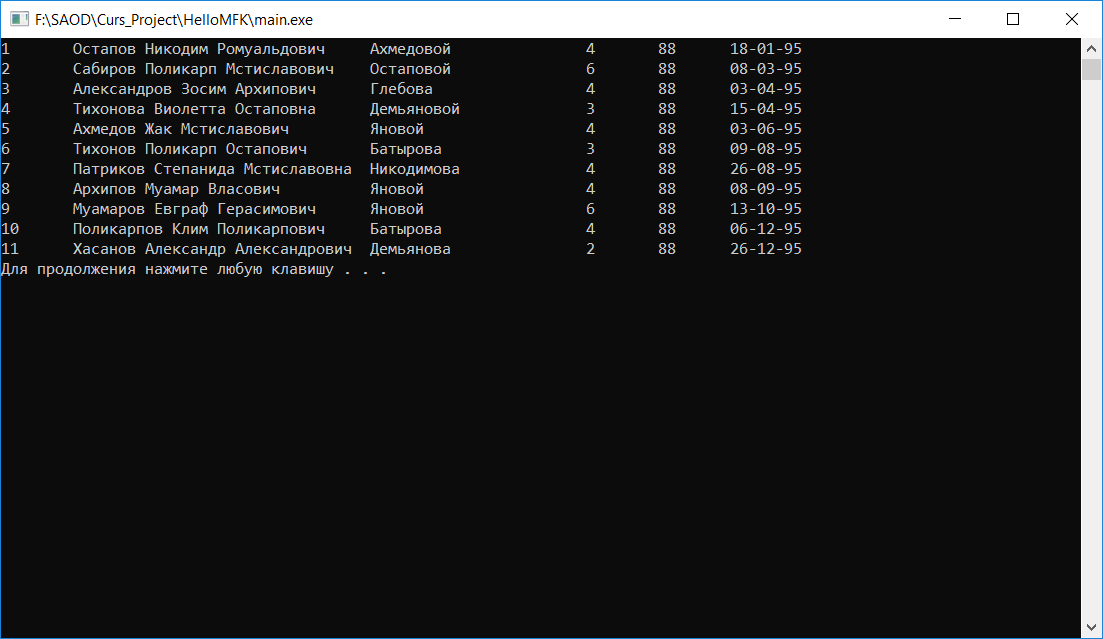


Рисунок 7. Вывод результата поиска по дереву

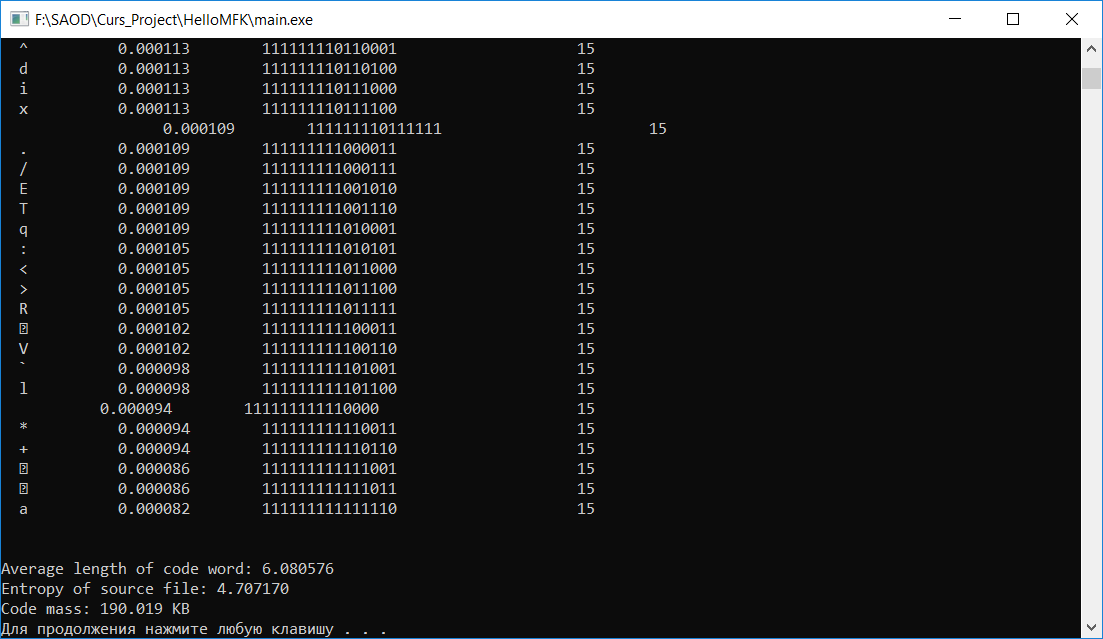


Рисунок 8. Коды символов, энтропия и средняя длина

7.ВЫВОДЫ

В ходе выполнения курсовой работы были выполнены все поставленные задачи и реализованы необходимые алгоритмы: сортировки методоп прямого слияния, поиска, построения бинарного Б-дерева, поиск по дереву и кодирование базы данных методом Гилберта- Мура.

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