Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет

телекоммуникаций и информатики»

Кафедра ПМиК

Курсовая работа

По дисциплине: «Структуры и алгоритмы обработки данных»

Вариант **15**

Выполнил: студент 2 курса

Ф. ИВТ, группа: ИП-711

Мартасов И. О.

Проверила:

доцент кафедры ПМиК

Янченко Е. В.

Новосибирск, 2018

**Содержание**

1. Постановка задачи

2. Основные идеи и хаpактеpистики пpименяемых методов

2.1. Метод сортировки

2.2. Двоичный поиск

2.3. Дерево и поиск по дереву

2.4. Метод кодирования

3. Особенности реализации алгоритмов

4. Описание программы

4.1. Основные переменные и структуры

4.2. Описание подпрограмм

5. Исходный текст программы

6. Результаты

7. Вывод

**Постановка задачи**

Хранящуюся в файле базу данных загрузить в оперативную память компьютера и построить индексный массив, упорядочивающий данные по ФИО и сумме вклада используя метод прямого слияния (Merge Sort) в качестве метода сортировки.

Предусмотреть возможность поиска по ключу, которым являются первые три буквы фамилии, 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 (ДОП, алгоритм А2) по дате вклада, 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, упакованную базу данных записать в файл, вычислить коэффициент сжатия данных.

База данных "Обманутые вкладчики"

Стpуктуpа записи:

ФИО вкладчика: текстовое поле 30 символа

фоpмат <Фамилия>\_<Имя>\_<Отчество>

Сумма вклада: целое число

Дата вклада: текстовое поле 10 символов

фоpмат дд-мм-гг

ФИО адвоката: текстовое поле 22 символа

фоpмат <Фамилия>\_<буква>\_<буква>

Пpимеp записи из БД:

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

130

15-03-46

Иванова\_И\_В\_\_\_\_\_\_\_\_\_\_\_

**Основные идеи и хаpактеpистики пpименяемых методов**

**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 нату-рального 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 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 <** *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, чемf_02. Таким образом, трудоёмкость двоичного поиска в обоих случаях:

f_03

**Дерево и поиск по дереву:**

Дерево оптимального поиска(А2):

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 сум-мирования 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.

Дерево, построенное приближенным алгоритмом А2, асимптотически приближается к оптимальному (с точки зрения средней высоты) при n→∞, т.е. алгоритм А2 является хорошим.

**Метод кодирования:**

Код Шеннона:

Код Шеннона позволяет построить почти оптимальный код с длинами ко-довых слов Li < - log pi +1. Тогда Lcp <H(p1, …,pn)+1. Код Шеннона строится следующим образом.

1.Упорядочим символы исходного алфавита А={a1,a2,…,an} по убыва-нию их вероятностей: p1≥p2≥p3≥…≥pn.

2.Составим нарастающие суммы вероятностей Qi:

Q0=0, Q1=p1, Q2=p1+p2, Q3=p1+p2+p3, … , Qn=1.

3.Представим Qi в двоичной системе счисления и возьмем в качестве кодового слова первые ⎡- log2*pi*⎤ знаков после запятой .

Для вероятностей, представленных в виде десятичных дробей, удобно оп-ределить длину кодового слова Li из соотношения

, .

Построенный код является префиксным.

**Особенности реализации алгоритмов**

В ходе выполнения курсовой работы потребовалось реализовать несколько вспомогательных, необходимых для корректной работы программы.

1. Интерфейс программы

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

С помощью булевых переменных flag и isflag исключены лишние операции по открытию файла и соблюдается корректное использование элементов меню (то есть нельзя создать очередь, пока база не отсортирована, и нельзя создать дерево, пока не создана очередь)

2. Загрузка и вывод базы данных

Для загрузки базы данных используется процедура ReadFile, в которой производится считывание записей типа contributors, а из них формируется очередь. Предусмотрены проверка на наличие файла, а также случаи некорректного выделения памяти. Процедура вызывается при запуске программы, после чего вызывается процедура FillMas, которая заполняет индексный массив и массив указателей типа contributors, с помощью которого происходит дальнейшая работа с базой.

За вывод элементов считанной базы данных отвечает процедура ShowBase и BaseHat. ShowBase предоставляет возможность постраничного просмотра базы данных, причём длина страницы выбирается пользователем при запуске процедуры(от 1 до 4000). BaseHat выводит на экран «шапку» базы – порядковый номер элемента базы и её поля – ФИО вкладчика, сумма вклада, дата вклада и ФИО адвоката. Также BaseHat используется при выводе очереди и дерева. Смена страниц осуществляется нажатием управляющих стрелок «вверх» и «вниз» на клавиатуре. Есть возможность прервать просмотр в любой момент времени нажатием клавиши «Esc».

Также, предусмотрен вариант формата отображения данных – отсортированный или исходный.

3. Вспомогательные функции и процедуры для сортировки данных

Для сортировки базы данных используется функция MergeSort и вспомогательные для неё функция scrap и процедура merge. Для того, чтобы сортировка происходила по ФИО вкладчика и суммы вклада, была реализована функция Less, которая сравнивает записи x и у по этим полям, и возвращает 1, если x > y, иначе возвращается 0.

4. Особенности реализации бинарного поиска

Для того чтобы без проблем многократно осуществлять поиск элементов, соответствующих разным ключам, требуется каждый раз создавать новую очередь, и чтобы постоянно не выделять память, после выполнения поиска происходит проверка, что если данных с ключом нет, то очередь построить нельзя, иначе – можно. Также предусмотрены проверки на случай неккоректоного выделения памяти. Добавление элементов в очередь реализовано в той же функции, где и реализован поиск – BinSearch. Процедура ShowQueue выводит записи этой очереди. При реализации была использована вторая версия двоичного поиска, так как в результате ее выполнения возвращается номер самого левого из найденных элементов, благодаря чему легко найти и вывести остальные элементы, лишь просмотрев оставшуюся правую часть массива.

5. Вспомогательные функции и процедуры для построения дерева оптимального поиска

Перед построением дерева запускается процедура FillRand, которая заполняет массив весов случайными числами, а с помощью процедуры SDP данные заносятся в само дерево. При построении дерева через процедуру А2 используется индексный массив для обращения к полям базы.

Для вывода дерева на экран используется процедура LeftObhod, представляющая собой обход дерева слева – направо.

Процедура TreeSearch выполняет поиска первого элемента в дереве, который передаётся в функцию search, после чего продолжается поиск и происходит вывод результатов поиска в дереве.

6. Кодирование данных

Для подсчета вероятностей символов в файле разработана процедура probabilities, в которой вычисляются значения вероятностей и заполняется массив символов, после чего происходит их сортировка. Для сортировки массива вероятностей использовался метод Хоара (QuickSort).

В процедуре shennon вычисляются кумулятивные вероятности и длины кодовых слов, после чего строятся сами кодовые слова.

Для вычисления характеристик (энтропии, средней длины кодового слова и суммы вероятностей) разработана процедура parametrs, где вычисляются основные параметры (средняя длина кодового слова и энтропия), а также производится суммирование вероятностей для проверки их правильности.

**Описание программы**

**Основные переменные и структуры:**

struct contributors

{

char contributor\_name[30];

unsigned short int deposit;

char date\_of\_deposit[10];

char lawyer\_name[22];

}; - Структура, используемая для работы с базой данных «Обманутые вкладчики»

struct spis

{

spis\* next;

contributors\* data;

int start\_position;

}; - Структура, используемая для работы с базой данных «Обманутые вкладчики» и построения очереди

next – указатель на следующие элемент;

data – поле данных

start\_position – индекс элемента в базе

struct SortQueue

{

spis\* head;

spis\* tail;

}; - Структура, используемая при сортировке базы данных.

head – голова очереди;

tail – хвост очереди.

struct vertex

{

vertex\* left;

vertex\* right;

contributors\* data;

int ves;

}\*root, \*search\_root = NULL;

- Структура, описывающая дерево оптимального поиска

left, right – указатели на левое и правое поддерево

data – поле данных

ves – значение веса вершины

root – указатель на корень дерева

search\_root – указатель на первый найденный элемент в дереве

#define base\_fields 4000 – количество элементов в базе

int index\_mas[base\_fields] – индексный массив из 4000 элементов;

char bin\_search\_key[3] – массив для ключа поиска очереди;

char tree\_search\_key[8] – массив для ключа поиска дерева;

bool flag –вспомогательная переменная;

bool isflag – вспомогательная переменная;

int queue\_fields – количество элементов в очереди;

int ves\_mas[base\_fields] – массив весов;

spis\* head – указатель на начало базы;

spis\* queue – указатель на начало очереди;

contributors\* contributors\_mas[base\_fields] – указатель на массив структур;

double P[256] – массив вероятностей символов, встречающихся в базе;

double Q[256] – массив кумулятивынх вероятностей;

double L[256] – массив длин кодовых слов;

char a[256] – массив символов, встречающихся в базе;

string codewords[256] – массив кодовых слов;

**Описание подпрограмм:**

**Первичная обработка данных:**

**spis\* ReadFile()** – чтение базы данных из файла, возвращает голову базы

**void FillMas**(int\* index\_mas, contributors\*\* contributors\_mas, spis\* head) – заполнение индексного массива и массива указателей на структуру

index\_mas – указатель на индексный массив

contributors\_mas – указатель на массив структур

head – указатель на начало базы

**Вывод базы данных:**

**void BaseHat()** – вывод «шапки» базы

**void ShowBase**(contributors\*\* contributors\_mas) – отображение базы данных

contributors\_mas – указатель на массив структур

**Сортировка:**

**int Less**(spis\* x, spis\* y) – сравнение записей по ФИО вкладчика и сумме вклада, возвращает 0 или 1

x – указатель на 1 запись

y – указатель на 2 запись

**int scrap**(spis\*& s, spis\*& a, spis\*& b) – разделение последовательности s на два списка a и b, возвращает количество элементов в базе

s – ссылка на исходную базу

a, b – рабочие списки

**void merge**(spis\*& a, spis\*& b, SortQueue& c, int q, int r) – слияние q – серии из списка a c r – серией списка b, запись результата в очередь c

a, b – рабочие списки

c – массив из двух очередей

q – фактический размер серии в списке a

r – фактический размер серии в списке b

**spis\* MergeSort**(spis\* s) – сортировка базы данных, возвращает голову базы

s – указатель на начало базы

**Поиск:**

**spis\* BinSearch**(contributors\*\* contributors\_mas, int\* index\_mas, char\* search\_key) – бинарный поиск и добавление элементов в очередь, возвращает голову очереди или NULL, в случае её отсутствия

contributors\_mas – указатель на массив структур

index\_mas – указатель на индексный массив

search\_key – указатель на ключ поиска

**int ShowQueue**(spis\* queue) – отображение очереди, возвращает количество элементов в очереди

queue – указатель на начало очереди

**Дерево:**

**void SDP**(char\* s, contributors\*\* contributors\_mas, int c, vertex\*& p, int\* index\_mas, int i) – добавление элементов в дерево

s – указатель на поле базы, являющееся ключом дерева

contributors\_mas – указатель на массив структур

p – ссылка на корень дерева

index\_mas – указатель на индексный массив

i – индекс элемента в базе данных

**void A2**(int l, int r, int\* ves, contributors\*\* contributors\_mas, int\* index\_mas) – построение дерева

l – левая граница очереди

r – правая граница очереди

ves – указатель на массив весов

contributors\_mas – указатель на массив структур

index\_mas – указатель на индексный массив

**void FillRand**(int \*Mas, int size) – заполнение массива весов

Mas – указатель на массив весов

size – размер массива

**void LeftObhod**(vertex \*p) – отображение дерева

p – указатель на корень дерева

**vertex\* TreeSearch**(vertex \*p, char\* search\_key) – поиск первого элемента в дереве, возвращает первый найденный элемент в дереве

p – указатель на корень дерева

search\_ key – указатель на ключ поиска в дереве

**void search**(vertex \*p, char\* search\_key) – поиск остальных элементов в дереве

p – указатель на первый найденный элемент в дереве

search\_ key – указатель на ключ поиска в дереве

**Кодирование:**

**void QuickSort**(double\* a, char\* b, int l, int n) – сортировка массива вероятностей

a – указатель на массив вероятностей

b – указатель на массив символов

l – левая граница массивов

n – правая граница массивов

**void probabilities**() – подсчёт вероятностей

**void shennon**() – построение кодовых лов

**void parametrs**() – расчёт характеристик

**Заглавная функция:**

int main() – инициализация переменных и реализация меню

**Исходный текст программы**

#include <iostream>

#include <conio.h>

#include <stdlib.h>

#include <io.h>

#include <fcntl.h>

#include <math.h>

#include <iomanip>

#define base\_fields 4000

using namespace std;

double P[256], Q[256], L[256];

char a[256];

string codewords[256];

struct contributors

{

char contributor\_name[30];

unsigned short int deposit;

char date\_of\_deposit[10];

char lawyer\_name[22];

};

struct spis

{

spis\* next;

contributors\* data;

int start\_position;

};

struct SortQueue

{

spis\* head;

spis\* tail;

};

struct vertex

{

vertex\* left;

vertex\* right;

contributors\* data;

int ves;

}\*root, \*search\_root = NULL;

spis\* ReadFile()

{

int f;

spis \*head, \*p, \*tail;

f = open("testBase3.dat", O\_RDONLY | O\_BINARY);

if(f == -1)

{

cout << "ОШИБКА: Не удалось открыть файл" << endl;

system("PAUSE");

exit(0);

}

tail = (spis \*) & head;

while (!eof(f))

{

p = new spis;

if (p == NULL)

{

cout << "ОШИБКА: Не удалось выделить память под элемент списка" << endl;

system("PAUSE");

exit(0);

}

p -> data = new contributors;

if (p -> data == NULL)

{

cout << "ОШИБКА: Не удалось выделить память под данные" << endl;

system("PAUSE");

exit(0);

}

read(f, p -> data, sizeof(contributors));

tail -> next = p;

tail = p;

}

close(f);

return head;

}

void BaseHat()

{

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

cout << "Номер\t|" << "ФИО вкладчика\t\t\t|" << "Сумма вклада\t|" << "Дата вклада\t\t|" << "ФИО адвоката |" << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

}

void ShowBase(contributors\*\* contributors\_mas)

{

int base\_size, base\_index, base\_end;

char key;

system("CLS");

cout << "Введите размерность базы(1 - 4000):" << endl;

cin >> base\_size;

try

{

if(base\_size <= 0 || base\_size > 4000)

{

cout << "Неверные данные." << endl;

system("PAUSE");

system("CLS");

throw 20;

}

}

catch(int i)

{

base\_size = i;

}

base\_index = 0;

base\_end = base\_size;

while(key != 27)

{

system("CLS");

BaseHat();

for (int i = base\_size \* base\_index; i < base\_end; i++)

{

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

cout << contributors\_mas[i] -> contributor\_name << "\t|";

cout << contributors\_mas[i] -> deposit << "\t\t|";

cout << contributors\_mas[i] -> date\_of\_deposit << "\t\t|";

cout << contributors\_mas[i] -> lawyer\_name << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|" << endl;

}

cout << endl << "Выход из базы данных - ESC" << endl;

cout << "Идти ниже по базе - стрелка вниз" << endl;

cout << "Идти выше по базе - стрелка вверх" << endl;

key = \_getch();

switch(key)

{

case 80:

if(base\_end == base\_fields)

{

cout << "Вы достигли дна. Дальше только наверх" << endl;

\_getch();

break;

}

if(base\_end + base\_size > base\_fields)

{

base\_index++;

base\_end = base\_end + (base\_fields - base\_end);

break;

}

else

{

base\_index++;

base\_end += base\_size;

break;

}

break;

case 72:

if(base\_end == base\_size)

{

cout << "Вы достигли потолка. Дальше только вниз" << endl;

\_getch();

break;

}

if(base\_size \* (base\_index - 1) < base\_end - base\_size)

{

base\_end = base\_size;

base\_index--;

system("cls");

break;

}

else

{

base\_index--;

base\_end -= base\_size;

break;

}

default:

break;

}

}

}

void FillMas(int\* index\_mas, contributors\*\* contributors\_mas, spis\* head)

{

for(int i = 0; i < base\_fields; i++)

{

index\_mas[i] = i;

}

spis\* p;

int i;

for(p = head, i = 0; p != NULL; p = p -> next, i++)

{

contributors\_mas[i] = p -> data;

}

}

int Less(spis\* x, spis\* y)

{

if(strncmp(x -> data -> contributor\_name, y -> data -> contributor\_name, 30) < 0)

{

return 1;

}

else if(strncmp(x -> data -> contributor\_name, y -> data -> contributor\_name, 30) > 0)

{

return 0;

}

else if(strncmp(x -> data -> contributor\_name, y -> data -> contributor\_name, 30) == 0)

{

if(x -> data -> deposit < y -> data -> deposit)

{

return 1;

}

else if(x -> data -> deposit > y -> data -> deposit)

{

return 0;

}

}

}

int scrap(spis\*& s, spis\*& a, spis\*& b)

{

spis \*k;

spis \*p;

int n;

a = s;

b = s -> next;

n = 1;

k = a;

p = b;

while(p != NULL)

{

n++;

k -> next = p -> next;

k = p;

p = p -> next;

}

return n;

}

void merge(spis\*& a, spis\*& b, SortQueue& c, int q, int r)

{

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

{

if(Less(a, b) == 1)

{

c.tail -> next = a;

c.tail = a;

a = a -> next;

q--;

}

else

{

c.tail -> next = b;

c.tail = b;

b = b -> next;

r--;

}

}

while(q > 0)

{

c.tail -> next = a;

c.tail = a;

a = a -> next;

q--;

}

while(r > 0)

{

c.tail -> next = b;

c.tail = b;

b = b -> next;

r--;

}

}

spis\* MergeSort(spis\* s)

{

int q, r;

spis \*a, \*b;

SortQueue c[2];

int n = scrap(s, a, b);

int p = 1;

int i, m;

while(p < n)

{

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

c[1].tail = (spis \*) & 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 = m - r;

merge(a, b, c[i], q, r);

i = 1 - i;

}

a = c[0].head;

b = c[1].head;

p = 2 \* p;

}

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

s = c[0].head;

return s;

}

spis\* BinSearch(contributors\*\* contributors\_mas, int\* index\_mas, char\* search\_key)

{

int mid\_mas;

int left\_mas = 0;

int right\_mas = base\_fields - 1;

char buf\_search\_key[3];

spis\* head;

spis\* tail;

while(left\_mas < right\_mas)

{

mid\_mas = (left\_mas + right\_mas) / 2;

for(int i = 0; i < 3; i++)

{

buf\_search\_key[i] = contributors\_mas[index\_mas[mid\_mas]] -> contributor\_name[i];

}

if (strncmp(search\_key, buf\_search\_key, 3) > 0)

{

left\_mas = mid\_mas + 1;

}

else

{

right\_mas = mid\_mas;

}

}

for(int i = 0; i < 3; i++)

{

buf\_search\_key[i] = contributors\_mas[index\_mas[left\_mas]] -> contributor\_name[i];

}

if(strncmp(search\_key, buf\_search\_key, 3) == 0)

{

tail = (spis\* ) & head;

}

else

{

return NULL;

}

while(strncmp(search\_key, buf\_search\_key, 3) == 0)

{

spis\* p = new spis;

if (p == NULL)

{

cout << "ОШИБКА: Не удалось выделить память под элемент очереди" << endl;

system("PAUSE");

exit(0);

}

p -> data = new contributors;

if (p -> data == NULL)

{

cout << "ОШИБКА: Не удалось выделить память под данные" << endl;

system("PAUSE");

exit(0);

}

p -> data = contributors\_mas[index\_mas[left\_mas]];

p -> start\_position = index\_mas[left\_mas];

tail -> next = p;

tail = p;

left\_mas++;

if(left\_mas == base\_fields)

{

break;

}

for(int i = 0; i < 3; i++)

{

buf\_search\_key[i] = contributors\_mas[index\_mas[left\_mas]] -> contributor\_name[i];

}

}

return head;

}

int ShowQueue(spis\* queue)

{

spis\* temp;

int i;

if(queue == NULL)

{

cout << "Данные не обнаружены" << endl;

system("PAUSE");

}

else

{

BaseHat();

for(temp = queue, i = 0; temp != NULL; i++, temp = temp -> next)

{

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

cout << temp -> data -> contributor\_name << "\t|";

cout << temp -> data -> deposit << "\t\t|";

cout << temp -> data -> date\_of\_deposit << "\t\t|";

cout << temp -> data -> lawyer\_name << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|" << endl;

}

cout << "Количество найденных элементов:" << i << endl;

system("PAUSE");

}

return i;

}

void SDP(char\* s, contributors\*\* contributors\_mas, int c, vertex\*& p, int\* index\_mas, int i)

{

if(p == NULL)

{

p = new vertex;

p -> data = new contributors;

p -> data = contributors\_mas[index\_mas[i]];

p -> left = NULL;

p -> right = NULL;

p -> ves = c;

}

else if(strncmp(s, p -> data -> date\_of\_deposit, strlen(p -> data -> date\_of\_deposit)) < 0)

{

SDP(s, contributors\_mas, c, p -> left, index\_mas, i);

}

else if(strncmp(s, p -> data -> date\_of\_deposit, strlen(p -> data -> date\_of\_deposit)) > 0)

{

SDP(s, contributors\_mas, c, p -> right, index\_mas, i);

}

else if(strncmp(s, p -> data -> date\_of\_deposit, strlen(p -> data -> date\_of\_deposit)) == 0)

{

SDP(s, contributors\_mas, c, p -> right, index\_mas, i);

}

}

void A2(int l, int r, int\* ves, contributors\*\* contributors\_mas, int\* index\_mas)

{

int i;

int wes = 0;

int summa = 0;

if(l <= r)

{

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

{

wes = wes + ves[i];

}

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

{

if((summa < wes / 2) && (summa + ves[i] >= wes / 2))

{

break;

}

summa = summa + ves[i];

}

SDP(contributors\_mas[index\_mas[i]] -> date\_of\_deposit, contributors\_mas, ves[i], root, index\_mas, i);

A2(l, i - 1, ves, contributors\_mas, index\_mas);

A2(i + 1, r, ves, contributors\_mas, index\_mas);

}

}

void FillRand(int \*Mas, int size)

{

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

Mas[i] = rand() % 100;

}

int o = 1;

void LeftObhod(vertex \*p)

{

if(p != NULL)

{

LeftObhod(p -> left);

cout << o++ << "\t|";

cout << p -> data -> contributor\_name << "\t|";

cout << p -> data -> deposit << "\t\t|";

cout << p -> data -> date\_of\_deposit << "\t\t|";

cout << p -> data -> lawyer\_name << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|" << endl;

LeftObhod(p -> right);

}

}

vertex\* TreeSearch(vertex \*p, char\* search\_key)

{

while(p != NULL)

{

if(strncmp(search\_key, p -> data -> date\_of\_deposit, 8) < 0)

{

p = p -> left;

}

if(strncmp(search\_key, p -> data -> date\_of\_deposit, 8) > 0)

{

p = p -> right;

}

else

{

return p;

}

}

return NULL;

}

void search(vertex \*p, char\* search\_key)

{

if (p != NULL)

{

search(p -> left, search\_key);

if(strncmp(search\_key, p -> data -> date\_of\_deposit, 8) == 0)

{

cout << o++ << "\t|";

cout << p -> data -> contributor\_name << "\t|";

cout << p -> data -> deposit << "\t\t|";

cout << p -> data -> date\_of\_deposit << "\t\t|";

cout << p -> data -> lawyer\_name << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|" << endl;

}

search(p -> right, search\_key);

}

}

void QuickSort(double\* a, char\* b, int l, int n)

{

double x = a[(l+n) / 2];

int i = l;

int j = n;

while(i <= j)

{

while(a[i] > x)

{

i++;

}

while(a[j] < x)

{

j--;

}

if(i <= j)

{

double temp = a[i];

a[i] = a[j];

a[j] = temp;

char buf = b[i];

b[i] = b[j];

b[j] = buf;

i++;

j--;

}

}

if(l < j)

{

QuickSort(a, b, l, j);

}

if(i < n)

{

QuickSort(a, b, i, n);

}

}

void probabilities()

{

long count = 0;

FILE \*fin = fopen("testBase3.dat", "r+");

while (!feof(fin))

{

int j = fgetc(fin);

P[j]++;

count++;

}

count--;

fclose(fin);

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

{

if(P[i] > 0)

{

a[i] = i;

P[i] /= (float)count;

}

}

QuickSort(P, a, 0, 255);

}

void shennon()

{

Q[0] = 0;

if(P[0] > 0)

{

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

}

for(int i = 1; i < 256; i++)

{

if(P[i] > 0)

{

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

L[i] = (int)(-1 \* log2(P[i]) + 1);

}

}

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

{

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

{

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

if((char)(Q[i] + 48) == '2')

{

codewords[i] += "0";

}

else

{

codewords[i] += (char)(Q[i] + 48);

}

if(Q[i] > 1)

{

Q[i] -= 1;

}

}

}

}

void parametrs()

{

double S = 0;

double M = 0;

double psum = 0;

int cou = 1;

cout << "Кодовая таблица" << endl;

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

{

if(P[i] > 0)

{

cout << cou++ << "\t" << a[i] << "\t" << P[i] << "\t " << codewords[i].length() << "\t" << codewords[i] << endl;

S += P[i] \* (float)L[i];

M += -1 \* P[i] \* log2(P[i]);

psum += P[i];

}

}

cout << "Сумма вероятностей: " << psum << endl;

cout << "Энтропия: " << M << endl;

cout << "Средняя длина кодового слова: " << S << endl;

}

int main()

{

int index\_mas[base\_fields];

int i;

char ch, bh;

char bin\_search\_key[3];

char tree\_search\_key[8];

bool flag = true;

bool isflag = true;

int queue\_fields;

int ves\_mas[base\_fields];

FillRand(ves\_mas, base\_fields);

spis\* head;

spis\* queue;

spis\* p;

contributors\* contributors\_mas[base\_fields];

head = ReadFile();

FillMas(index\_mas, contributors\_mas, head);

while(1)

{

system("CLS");

cout << "1.Формат отображения данных" << endl;

cout << "2.Вывод базы данных" << endl;

cout << "3.Поиск" << endl;

cout << "4.Дерево" << endl;

cout << "5.Кодирование" << endl;

cout << "6.Выход" << endl;

ch = \_getch();

switch(ch)

{

case '1':

system("CLS");

cout << "1.Исходный" << endl;

cout << "2.Отсортированный" << endl;

bh = \_getch();

switch(bh)

{

case '1':

system("CLS");

if(!flag)

{

head = ReadFile();

for(p = head, i = 0; p != NULL; p = p -> next, i++)

{

contributors\_mas[i] = p -> data;

}

flag = true;

}

else if(flag)

{

cout << "База уже в исходном виде" << endl;

system("PAUSE");

}

break;

case '2':

system("CLS");

if(flag)

{

head = MergeSort(head);

for(p = head, i = 0; p != NULL; p = p -> next, i++)

{

contributors\_mas[i] = p -> data;

}

flag = false;

}

else if(!flag)

{

cout << "База уже отсортирована" << endl;

system("PAUSE");

}

break;

default:

break;

}

break;

case '2':

ShowBase(contributors\_mas);

break;

case '3':

system("CLS");

if(flag)

{

cout << "Сначала отсортируйте базу!" << endl;

system("PAUSE");

break;

}

cout << "Введите первые три буквы фамилии (формат Абв)" << endl;

cin >> bin\_search\_key;

system("CLS");

queue = BinSearch(contributors\_mas, index\_mas, bin\_search\_key);

queue\_fields = ShowQueue(queue);

break;

case '4':

system("CLS");

if(flag || queue == NULL)

{

cout << "Невозможно создать дерево" << endl;

}

else

{

int last\_position = queue -> start\_position + queue\_fields;

A2(queue -> start\_position, last\_position - 1, ves\_mas, contributors\_mas, index\_mas);

BaseHat();

LeftObhod(root);

o = 1;

system("PAUSE");

}

if(root != NULL)

{

system("CLS");

cout << "Введите дату вложения (формат ДД-ММ-ГГ)" << endl;

cin >> tree\_search\_key;

system("CLS");

search\_root = TreeSearch(root, tree\_search\_key);

if(search\_root == NULL)

{

cout << "Данные не обнаружены" << endl;

}

else

{

BaseHat();

search(search\_root, tree\_search\_key);

o = 1;

}

}

search\_root = NULL;

root = NULL;

system("PAUSE");

break;

case '5':

if(isflag)

{

probabilities();

shennon();

isflag = false;

}

system("CLS");

parametrs();

system("PAUSE");

break;

case '6':

exit(0);

default:

break;

}

}

return 0;

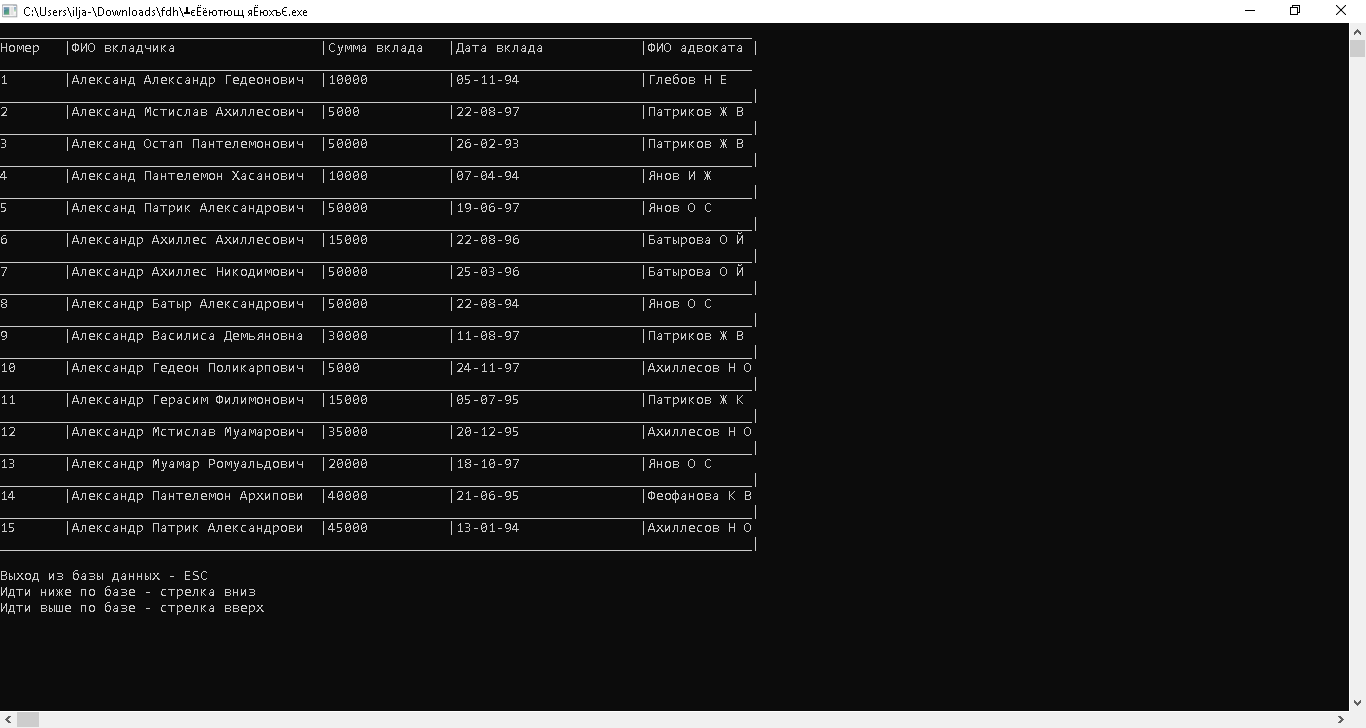
}

**Результаты**

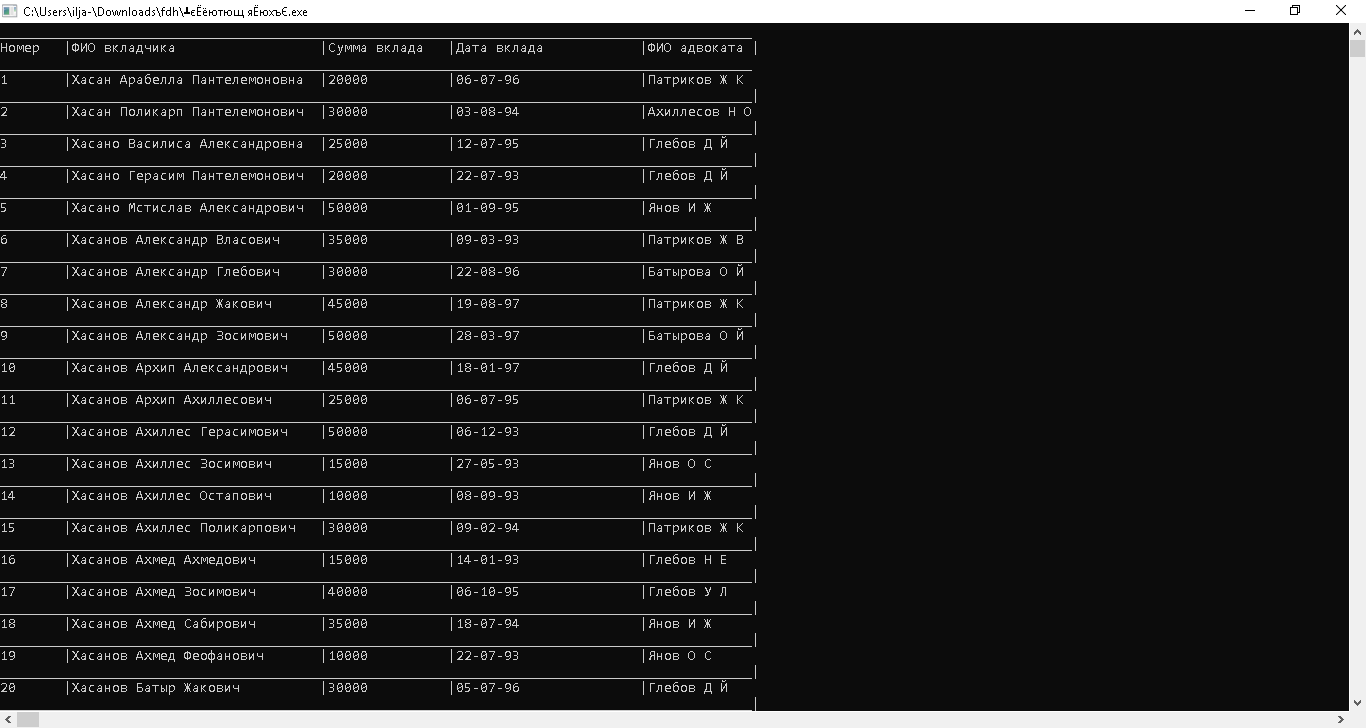
**1.Исходная база данных**

****

**2.Отсортированная база данных по ФИО вкладчика и сумме вклада**

****

**3.Очередь из элементов, полученных в результате поиска (Хас)**

****

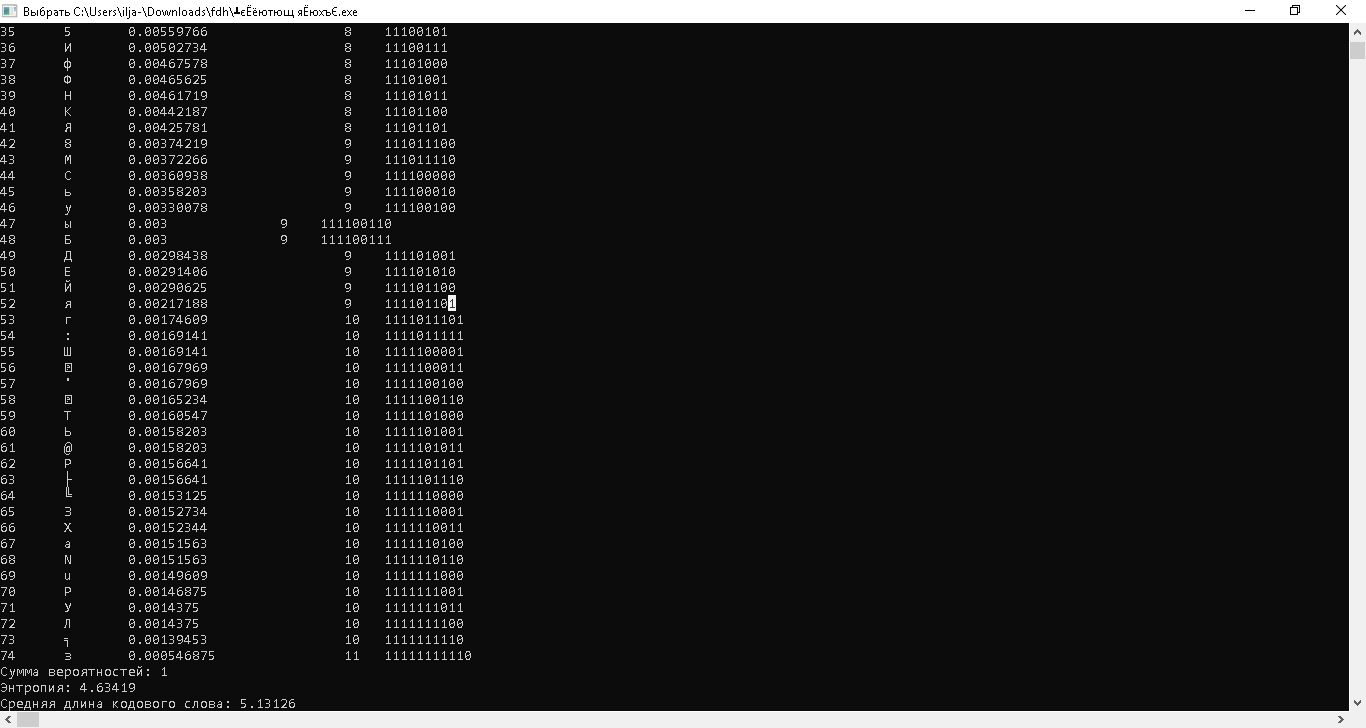
**4.Дерево поиска (ключ – дата вклада)**

****

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

****

**6.Кодовая таблица и характеристики**



**Вывод**

В ходе выполнения курсовой работы были выполнены все поставленные задачи и реализованы необходимые алгоритмы: сортировка методом прямого слияния, бинарного поиска, построения дерева оптимального поиска, 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.