Федеральное агентство связи

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

Кафедра прикладной математики и кибернетики

КУРСОВОЙ ПРОЕКТ

по дисциплине «Структуры и алгоритмы обработки данных»

Вариант 18

Выполнил: студент 2 курса группы ИП-811 Мироненко К. А

Проверил: доцент кафедры ПМиК Янченко Е. В.

**Оглавление**

[1. Постановка задачи 3](#_Toc27973420)

[2. Основные идеи и характеристики применяемых методов 4](#_Toc27973421)

[2.1 Метод сортировки 4](#_Toc27973422)

[2.2 Двоичный поиск 4](#_Toc27973423)

[2.3 Списки и очереди 5](#_Toc27973424)

[2.4 Вид дерева и поиск 6](#_Toc27973425)

[2.5 Метод кодирования 7](#_Toc27973426)

[3. Описание структур данных и использованных алгоритмов 9](#_Toc27973427)

[3.1 Использованные структуры данных 9](#_Toc27973428)

[3.2 Особенности реализации алгоритмов 9](#_Toc27973429)

[4. Описание программы 11](#_Toc27973430)

[4.1 Основные переменные и структуры 11](#_Toc27973431)

[4.2 Описание подпрограмм 12](#_Toc27973432)

[5. Исходный текст пpогpаммы 20](#_Toc27973433)

[6. Результаты 36](#_Toc27973434)

[7. Выводы 43](#_Toc27973435)

# Постановка задачи

Хранящуюся в файле («testBase3.dat») базу данных «Обманутые вкладчики» (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 ФИО и сумме вклада, используя метод Цифровой сортировки. На основе упорядоченных данных построить индексный массив. Упорядоченные данные вывести на экран.

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

Из записей очереди построить АВЛ-дерево поиска по ключу, отличному от ключа сортировки (по ФИО адвоката), вывести на экран содержимое дерева и предусмотреть возможность поиска в дереве по запросу.

Закодировать файл базы данных статическим кодом Фано, предварительно оценив вероятности всех встречающихся в ней символов. Построенный код вывести на экран, вычислить среднюю длину кодового слова и сравнить ее с энтропией исходного файла.

База данных "Обманутые вкладчики"

Стpуктуpа записи:

ФИО вкладчика: текстовое поле 30 символа

фоpмат <Фамилия>\_<Имя>\_<Отчество>

Сумма вклада: целое число (использовать unsigned short int)

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 характеристики применяемых методов

## Метод сортировки

Пусть дана последовательность из S чисел, представленных в m – ичной си-стеме счисления. Каждое число состоит из L цифр d1d2…dL, 0 ≤ di ≤ m – 1, i=1..L. Сначала числа из списка S распределяются по m очередям, причём номер очереди определяется последней цифрой каждого числа. Затем полученные очереди соединяются в список, для которого все действия повторяются, но распределение по очередям производится в соответствии со следующей цифрой и т.д.

Цифровой метод может успешно использоваться не только для сортировки чисел, но и для сортировки любой информации, представленной в памяти компьютера. Необходимо лишь рассматривать каждый байт ключа сортировки как цифру, принимающую значения от 0 до 255. Тогда для сортировки потребуется m = 256 очередей. Для выделения каждого байта ключа сортировки можно использовать массив Digit, наложенный в памяти компьютера на поле элемента последовательности, по которому происходит сортировка.

Для цифровой сортировки М < const L(m + n). При фиксированных m и L М=O(n) при n → ∞, что значительно быстрее остальных рассмотренных методов. Однако если длина чисел L велика, то метод может проигрывать обычным методам сортировки. Кроме того, Метод применим только, если задача сортировки сводится к задаче упорядочивания чисел, что не всегда возможно.

Метод обеспечивает устойчивую сортировку. Чтобы изменить направление сортировки на обратное, очереди нужно присоединять в обратном порядке.

## Двоичный поиск

Идея алгоритма двоичного поиска

Алгоритм двоичного поиска в упорядоченном массиве сводится к следующему. Берём средний элемент отсортированного массива и сравниваем с ключом X.

Возможны три варианта:

1. Выбранный элемент равен X. Поиск завершён.
2. Выбранный элемент меньше X. Продолжаем поиск в правой поло-вине массива.
3. Выбранный элемент больше X. Продолжаем поиск в левой поло-вине массива.

Из-за необходимости найти все элементы, соответствующие заданному ключу поиска, в курсовой работе используется вторая версия двоичного поиска, которая находит крайний левый элемент, если массив имеет несколько элементов с одинаковым ключом. Для поиска остальных элементов требуется просмотреть оставшуюся часть массива вправо от найденного элемента.

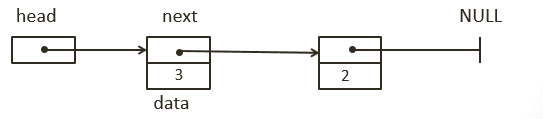
Трудоёмкость алгоритма двоичного поиска

На каждой итерации поиска необходимо одно сравнение (для второй версии). Количество итераций не больше, чем . Таким образом, трудоёмкость двоичного поиска:

С=O(log n), n → ∞

## Списки и очереди

*Списком* называется последовательность однотипных элементов, связанных между собой указателями.



Поле next является указателем на элемент списка и может занимать произвольное место в структуре элемента. Однако если оно является первым элементом структуры, то его адрес совпадает с адресом элемента списка, и это позволяет оптимизировать многие операции со списками. Поле data содержит информацию, которая будет учитываться при сортировке.

Рассмотрим два вида списков: стек и очередь. *Стек* характеризуется тем, что новый элемент добавляется в начало последовательности, а удаляться может только первый элемент списка. Стек реализует дисциплину обслуживания LIFO (Last Input, First Output). При добавлении в *очередь* новый элемент ставится в конец списка, удаляется первый элемент последовательности. Очередь реализует дисциплину обслуживания FIFO (FirstInput, FirstOutput)

## Вид дерева и поиск

*АВЛ-дерево* - дерево поиска для каждой которого вершины высоты левого и правого поддеревьев отличается не больше, чем на единицу.

Определение предложено в *1962* году *Г.М. Адельсон-Вельским и Е.М. Ландисом.* Они предложили балансировать дерево по ***высоте***, а не по размеру.

Идея алгоритма построения АВЛ-дерева

Вначале ***добавим новую вершину***в дерево так же, ***как в случайное***, т.е. проходим по пути поиска до нужного места включения в качестве листовой вершины.

***Двигаясь назад по пути поиска***, будем ***искать вершину, в которой нарушился баланс***, т.е. высота левого и правого поддерева стала отличаться больше, чем на единицу. ***Если*** **такая *вершина найдена***, то ***изменим структуру дерева для восстановления баланса***.

***При включении новой вершины её баланс равен нулю***. ***При движении назад***по пути поиска ***показатель баланса для всех вершин пересчитывается***, причем не нужно просматривать все поддеревья, только путь поиска.

***Нарушение баланса возможно только в одной вершине*** и ***один поворот полностью восстанавливает структуру АВЛ-дерева***.

Балансировка выполняется с помощью поворотов дерева: ***LL, LR, RL, RR***.

Трудоёмкость

Адельсон – Вельский и Ландис доказали теорему, гарантирующую, что АВЛ-дерево никогда не будет в среднем по высоте превышать ИСДП более, чем на 45% независимо от количества вершин:

log(*n*+1) ≤ hАВЛ(*n*) < 1,44 log(*n*+2) – 0,328 при *n→∞.*

Идея поиска элемента

Начиная с корневой вершины дерева, ***сравниваем ключ поиска с данными в текущей вершине***.

***Если ключ*** поиска ***меньше***, то ***переходим в левое поддерево***, ***если ключ*** поиска ***больше***, то ***переходим в правое поддерево***.

Действуем аналогично, ***пока не будет найден элемент с заданным ключом или листовая вершина дерева***.

***Если достигнута листовая вершина***, то ***искомого элемента нет*** в дереве.

## Метод кодирования

Идея метода Фано

Упорядоченный по убыванию вероятностей список 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.

Полученный код является префиксным и почти оптимальным.

*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.

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

Λ

0

1

00

01

10

11

110

111

1110

1111

а1

а2

а3

а4

а5

а6

# Описание структур данных и использованных алгоритмов

## Использованные структуры данных

В курсовом проекте были использованы следующие структуры данных:

Простые структуры:

* Числовые (unsigned short int; float)
* Символьные (char; unsigned char)
* Логические (bool)
* Указатели

Составные структуры:

* Массивы
* Очереди
* Списки
* Деревья

## Особенности реализации алгоритмов

Цифровая сортировка

* массив “Digit [sizeof(data)]”, наложенный в памяти компьютера на поле элемента “data”, в структуре “listDataBase”
* структура “queue”, для реализации очередей

Поиск

* Собственная функция “comparator” для корректного сравнения двух массив char

Построение АВЛ- дерева

* Рекурсия, позволяющая рекурсивно запоминать пройденные вершины по пути поиска и автоматически в них возвращаться на обратном пути рекурсии.
* Поле “balance” типа “short int” в каждой вершине дерева
* Поле “elems” типа “listDataBase \*”, содержащее элементы, соответствующие ключу
* Глобальная переменная “growth” типа “bool” отвечающая за рост дерева

Код Фано

* Функция “quickSortCoding” для упорядочивания массива символов базы в порядке убывания вероятностей

# Описание программы

## Основные переменные и структуры

Основные переменные

* char choice – переменная, отвечающая за текущий выбор элемента меню
* listDataBase \* listBase – список, состоящий из считанных из файла записей
* unsigned int sizeBase – переменная, хранящая размер БД
* itemDataBase \*\*arr – индексный массив указателей
* char keySearch[30] – массив char, используемый для хранения ключа поиска
* listDataBase \* resSearch1 – результат бинарного поиска по отсортированной БД
* treeLawyer \* treeAVL – указатель на корень АВЛ-дерева
* bool growth – логическая переменная, обозначающая произошел ли рост дерева
* treeLawyer \*resSearch2 – результат бинарного поиска в дереве;

Основные структуры (файл “struct.hpp”)

* struct itemDataBase {

char depositor[30];

unsigned short int contribution;

char date[10];

char lawyer[22];

}; – структура, хранящая информацию об элементе БД

* struct listDataBase {

listDataBase \*next;

union {

itemDataBase data;

unsigned char Digit[sizeof(data)];

};

}; – структура, используемая для чтении и сортировки БД

* struct treeLawyer {

char data[22]; // lawyer

short int balance = 0;

treeLawyer \*left = nullptr;

treeLawyer \*right = nullptr;

listDataBase \*elems = nullptr;

}; – структура, используемая для постоения АВЛ-дерева

* struct coding {

char symbol;

unsigned int quantity;

float probability;

unsigned short int lengthCW;

char \* codeword;

}; – структура, используемая при кодировании

## Описание подпрограмм

**Файл func.cpp**

* void readDataBase(listDataBase \*&p, unsigned int & size);

***процедура, производящая считывание БД;***

Параметры:

* listDataBase \*&p – ссылка на указатель, указывающий на начало списка
* unsigned int & size – ссылка на память, хранящую размер БД
* void digitalSort(listDataBase \*(&S), int reverse);

***процедура, производящая сортировку БД;***

Параметры:

* listDataBase \*&S – ссылка на указатель, указывающий на начало списка (head)
* int reverse – переменная, отвечающая за направление сортировки
* void delList(listDataBase \*&p)

***процедура, удаляющая список;***

Параметры:

* listDataBase \*&p – ссылка на указатель, указывающий на начало списка (head)
* void createIndexArr(itemDataBase\*\* &arr, listDataBase\* p ,unsigned int size)

***процедура, создающая индексный массив указателей на основе полученного списка;***

Параметры:

* itemDataBase\*\* &arr – ссылка на указатель, указывающий на начало индексного массива указателей
* listDataBase\* p – указатель на память, хранящую начало списка (head)
* unsigned int size – размер списка
* void delArr(itemDataBase\*\* &arr, unsigned int size)

***процедура, удаляющая индексный массив указателей;***

Параметры:

* itemDataBase\*\* &arr – ссылка на указатель, указывающий на начало индексного массива указателей
* unsigned int size – размер массива
* short int comparator(const char \*word1, const char \* word2)

***функция, сравнивающая две строки и на основе этого возвращающая 0 – если строки равны, -1 – если 1 строка*** ***лексикографически меньше 2, 1 – если 2 лексикографически меньше 1;***

Параметры:

* const char \*word1 – указатель на строку 1
* const char \* word2 – указатель на строку 2
* listDataBase \*binarySearch(itemDataBase \*\*arr, unsigned int size, char \*keySearch)

***функция бинарного поиска, возвращающая очередь элементов, соответствующих ключу поиска;***

Параметры:

* itemDataBase \*\*arr – указатель на память, хранящую индексный массив указателей
* unsigned int size – размер массива
* char \*keySearch – указатель на ключ поиска, не превышающий 30 символов
* void addAVL(treeLawyer \*&p, itemDataBase data, bool &growth)

***процедура добавления вершины в дерево;***

Параметры:

* treeLawyer \*&p – ссылка на указатель, указывающий на корень дерева
* itemDataBase data – добавляемые данные
* bool &growth – ссылка на переменную, отвечающцю за рост дерева
* treeLawyer \*findVertexWithKey(treeLawyer \*p, char \*key)

***функция, возвращающая найденную вершину в дереве***

Параметры:

* treeLawyer \*&p – ссылка на указатель, указывающий на корень дерева
* char \*key - указатель на ключ поиска, не превышающий 30 символов
* void delTree(treeLawyer \*&p)

***Процедура, удаляющая дерево***

Параметры:

* treeLawyer \*&p – ссылка на указатель, указывающий на корень дерева
* void quickSortCoding(coding\* A, int R, int L)

***Процедура, производящая quick-сортировку массива структуры “coding” по убыванию поля “*** ***probability”***

Параметры:

* coding\* A – указатель на начало массива
* int R – правая граница области сортировки
* int L – левая граница области сортировки
* int med(coding \*code, int borderL, int borderR)

***Функция, возвращающая медиану части массива P, т.е. такой индекс L <= m <= R, что величина между SumL и SumR минимальна***

Параметры:

* coding\* code – указатель на начало массива
* int borderL – левая граница
* int borderR – правая граница
* void codeFano(coding \* code, int borderL, int borderR, int k)

***Процедура, формирующая кодовые слова***

Параметры:

* coding\* code – указатель на начало массива
* int borderL – левая граница
* int borderR – правая граница
* int k – длина уже построенной части элементарных кодов
* void tableSymbols(coding\* &code, int &numsUnique)

***Процедура, формирующая массив кодовой таблицы и считающая кол-во уникальных символов***

Параметры:

* coding\* &code – ссылка на указатель, указывающий на начало массива
* int &numsUnique – ссылка на переменную, хранящую кол-во уникальных символов
* void pause()

***Процедура, приостанавливающая выполнение программы***

* bool selectionCheck()

***Функция, отвечающая за выбор, возвращает 1 – при положительном ответе(символ ‘y’), 0 – при отрицательном ответе(символ ‘n’)***

* void clearBuffer()

***Процедура, очищающая буфер***

**Файл funcOut.cpp**

* void printItemDB(itemDataBase item)

***Процедура, выводящая элемент БД***

Параметры:

* itemDataBase item – элемент БД
* void output(listDataBase\* p)

***Процедура, отвечающая за выбор вывода БД (постраничный или всей БД)***

Параметры:

* listDataBase\* p – указатель на начало списка (head)
* void outputDB\_PbyP(listDataBase \*head)

***Процедура, отвечающая за постраничный вывод БД***

Параметры:

* listDataBase\* head – указатель на начало списка
* void outputDB\_Full(listDataBase \*p)

***Процедура, отвечающая за полный вывод БД***

Параметры:

* listDataBase\* p – указатель на начало списка (head)
* void outputTree(treeLawyer \*p)

***Процедура, отвечающая за выбор вывода дерева (только ключи или все дерево)***

Параметры:

* treeLawyer \*p – указатель на корень дерева
* void printTableSymbols(coding \*code, int numSymbols)

***Процедура, отвечающая за вывод кодовой таблицы символов***

Параметры:

* coding \*code – указатель на начало массива
* int numSymbols – размер массива

**rotationsAVL.cpp**

* void rotateRR(treeLawyer \*&p)

***Процедура, осуществляющая RR-поворот в АВЛ-дереве***

Параметры:

* treeLawyer \*&p – ссылка на указатель, указывающий на вершину дерева в которой нарушился баланс
* void rotateLL(treeLawyer \*&p)

***Процедура, осуществляющая LL-поворот в АВЛ-дереве***

Параметры:

* treeLawyer \*&p – ссылка на указатель, указывающий на вершину дерева в которой нарушился баланс
* void rotateLR(treeLawyer \*&p)

***Процедура, осуществляющая LR-поворот в АВЛ-дереве***

Параметры:

* treeLawyer \*&p – ссылка на указатель, указывающий на вершину дерева в которой нарушился баланс
* void rotateRL(treeLawyer \*&p)

***Процедура, осуществляющая RL-поворот в АВЛ-дереве***

Параметры:

* treeLawyer \*&p – ссылка на указатель, указывающий на вершину дерева в которой нарушился баланс

# Исходный текст пpогpаммы

**main.cpp**

#include <conio.h>  
#include <iostream>  
  
  
#include "func.hpp"  
#include "funcOut.hpp"  
#include "struct.hpp"  
  
int main(int argc, char const \*argv[]) {  
 SetConsoleOutputCP(866);  
 SetConsoleCP(866);  
 char choice;  
 listDataBase \* listBase = nullptr;  
 unsigned int sizeBase = 0;  
 itemDataBase \*\*arr = nullptr;  
 char keySearch[30];  
 listDataBase \* resSearch1 = nullptr;  
 listDataBase \* tempList = nullptr;  
 treeLawyer \* treeAVL = nullptr;  
 bool growth = false; // Рост дерева  
 treeLawyer \*resSearch2 = nullptr;  
  
 readDataBase(listBase, sizeBase);  
 do{  
 system("CLS");  
 std::cout << "1) Вывести исходную базу данных\n";  
 std::cout << "2) Вывести отсортированную базу данных\n";  
 std::cout << "3) Бинарный поиск по базе\n";  
 std::cout << "4) Дерево\n";  
 std::cout << "5) Бинарный поиск в дереве\n";  
 std::cout << "6) Статический код\n";  
 std::cout << "\n----------------------------------------\n";  
 std::cout << "0) Выйти\n";  
 switch(choice = (char)getch()){  
 case '1':{  
 /\* Чтение и вывод исходной БД \*/  
 system("CLS");  
 readDataBase(listBase, sizeBase);  
 output(listBase);  
 pause();  
 break;  
 }  
 case '2':{  
 /\* Вывод отсортированной БД и построение массива адресов \*/  
 system("CLS");  
 digitalSort(listBase, false);  
 createIndexArr(arr, listBase, sizeBase);  
 output(listBase);  
 pause();  
 break;  
 }  
 case '3':{  
 /\* Бинарный поиск с формированием очереди и ее выводом \*/  
 system("CLS");  
 digitalSort(listBase, false);  
 createIndexArr(arr, listBase, sizeBase);  
  
 output(listBase);  
  
 std::cout<< "Ключ поиска > ";  
 clearBuffer();  
 std::cin.getline(keySearch, 30, '\n');  
 system("CLS");  
 if (resSearch1 != nullptr)  
 delList(resSearch1);  
 if ((resSearch1 = binarySearch(arr, sizeBase, keySearch)))  
 outputDB\_Full(resSearch1);  
 pause();  
 break;  
 }  
 case '4':{  
 /\* Дерево на основе бинарного поиска по ключу lawyer(адвокат) \*/  
 system("CLS");  
 if (resSearch1 != nullptr){  
 outputDB\_Full(resSearch1);  
 std::cout << "Использовать данный поиск для постоения дерева? ";  
 if(!selectionCheck())  
 delList(resSearch1);  
 system("CLS");  
 }  
 if (resSearch1 == nullptr){  
 digitalSort(listBase, false);  
 createIndexArr(arr, listBase, sizeBase);  
 output(listBase);  
 std::cout<< "Ключ поиска > ";  
 clearBuffer();  
 std::cin.getline(keySearch, 30, '\n');  
 system("CLS");  
 if ((resSearch1 = binarySearch(arr, sizeBase, keySearch)))  
 outputDB\_Full(resSearch1);  
 }  
 delTree(treeAVL);  
 if(resSearch1){  
 for (tempList = resSearch1; tempList; tempList = tempList->next) {  
 addAVL(treeAVL, tempList->data, growth);  
 }  
 outputTree(treeAVL);  
 }  
 pause();  
 break;  
 }  
 case '5':{  
 /\* Бинарный поиск по дереву \*/  
 system("CLS");  
 if (treeAVL != nullptr){  
 outputTree\_LR(treeAVL, false);  
 std::cout << "Использовать данное дерево для поика? ";  
 if(!selectionCheck()){  
 delTree(treeAVL);  
 delList(resSearch1);  
 }  
 }  
 if (treeAVL == nullptr){  
 digitalSort(listBase, false);  
 createIndexArr(arr, listBase, sizeBase);  
 system("CLS");  
 output(listBase);  
 std::cout << "Ключ поиска > ";  
 clearBuffer();  
 std::cin.getline(keySearch, 30, '\n');  
 system("CLS");  
 resSearch1 = binarySearch(arr, sizeBase, keySearch);  
 if(resSearch1){  
 for (tempList = resSearch1; tempList; tempList = tempList->next) {  
 addAVL(treeAVL, tempList->data, growth);  
 }  
 outputTree(treeAVL);  
 }  
  
 }  
 if (treeAVL){  
 std::cout << "Ищем > ";  
 std::cin.getline(keySearch, 22, '\n');  
 clearBuffer();  
 system("CLS");  
 delTree(resSearch2);  
 resSearch2 = findVertexWithKey(treeAVL, keySearch);  
 if (resSearch2) {  
 std::cout << "-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+\n";  
 std::cout << "< По ключу \"" << keySearch << "\" найдена вершина \"" << resSearch2->data << "\" по адресу " << resSearch2 << " >\n";  
 outputDB\_Full(resSearch2->elems);  
 } else  
 std::cout << "< Вершина по ключу \"" << keySearch << "\" не найдена в дереве>\n";  
 }  
 pause();  
 break;  
 }  
 case '6': {  
 /\* Статический код \*/  
 int numSymbols = 0;  
 coding \* codeFano = nullptr;  
 tableSymbols(codeFano, numSymbols);  
 printTableSymbols(codeFano, numSymbols);  
  
 pause();  
 break;  
 }  
 }  
 } while ((choice != '0') && (choice != 27));  
 return 0;  
}

**struct.hpp**

#ifndef STRUCT\_HPP  
#define STRUCT\_HPP  
  
struct itemDataBase {  
 char depositor[30];  
 unsigned short int contribution;  
 char date[10];  
 char lawyer[22];  
};  
  
struct listDataBase {  
 listDataBase \*next;  
 union {  
 itemDataBase data;  
 unsigned char Digit[sizeof(data)];  
 };  
};  
  
struct queue {  
 listDataBase \*head;  
 listDataBase \*tail;  
};  
  
struct treeLawyer {  
 char data[22]; // lawyer  
 short int balance = 0;  
 treeLawyer \*left = nullptr;  
 treeLawyer \*right = nullptr;  
 listDataBase \*elems = nullptr;  
};  
  
struct coding {  
 char symbol; // Символ  
 unsigned int quantity; // Встречаемость в текте  
 float probability; // Вероятность  
 unsigned short int lengthCW; // Длина кодового слова  
 char \* codeword; // Кодовое слово  
};  
  
#endif // STRUCT\_HPP

**funcOut.hpp**

#ifndef FUNCOUT\_HPP  
#define FUNCOUT\_HPP  
  
#include <iostream>  
#include <iomanip>  
#include <cmath>  
#include "conio.h"  
  
#include "struct.hpp"  
#include "func.hpp"  
  
const unsigned short int itemsPage = 20; // Кол-во записей на странице  
  
void printItemDB(itemDataBase item);  
  
void output(listDataBase\* p);  
void outputDB\_PbyP(listDataBase \*head);  
void outputDB\_Full(listDataBase \*p);  
  
void outputTree(treeLawyer \*p);  
void outputTree\_LR(treeLawyer \*p, bool full);  
  
void printTableSymbols(coding \*code, int numSymbols);  
  
#endif // FUNCOUT\_HPP

**funcOut.cpp**

#include "funcOut.hpp"  
  
#include <cmath>  
  
/\* Вывод элемента БД \*/  
void printItemDB(itemDataBase item) {  
 std::cout << item.depositor << " ";  
 std::cout.setf(std::ios::left);  
 std::cout.width(5);  
 std::cout << item.contribution << " ";  
 std::cout << item.date << " ";  
 std::cout << item.lawyer << "\n";  
}  
  
/\* Вывод \*/  
void output(listDataBase\* p){  
 std::cout << " 1) Постранично\n";  
 std::cout << " 2) Вся база\n";  
 char c;  
 do {  
 c = (char)getch();  
 } while ((c != '1') && (c != '2'));  
 system("CLS");  
 if (c == '1')  
 outputDB\_PbyP(p);  
 else  
 outputDB\_Full(p);  
}  
  
/\* Постраничный вывод БД \*/  
void outputDB\_PbyP(listDataBase \*head) {  
 if (head == nullptr){  
 std::cout << "-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+"  
 << "\n"  
 << " < Список пуст >"  
 << "\n"  
 << "-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+"  
 << "\n";  
 return;  
 }  
 listDataBase \*p = head;  
 int j = 0;  
 int key;  
 int i = itemsPage;  
 do {  
 system("CLS");  
 std::cout << "-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+\n";  
 for (; (j < i) && (p != nullptr); j++, p = p->next) {  
 std::cout.setf(std::ios::right);  
 std::cout.width(4);  
 std::cout << (j + 1) << ")"  
 << " ";  
 printItemDB(p->data);  
 }  
 std::cout << "-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+\n";  
 do {  
 switch (key = getch()) {  
 case 75: // left  
 if (j != itemsPage) {  
 i = i - itemsPage;  
 j = i - itemsPage;  
 p = head;  
 for (int f = 0; f < j; f++)  
 p = p->next;  
 } else  
 key = 0;  
 break;  
 case 77: // right  
 if (p != nullptr) {  
 i = i + itemsPage;  
 } else  
 key = 0;  
 break;  
 }  
 } while ((key != 75) && (key != 77) && (key != 27));  
  
 } while (key != 27);  
}  
  
/\* Полный вывод БД \*/  
void outputDB\_Full(listDataBase \*p) {  
 if (p == nullptr){  
 std::cout << "-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+"  
 << "\n"  
 << " < Список пуст >"  
 << "\n"  
 << "-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+"  
 << "\n";  
 return;  
 }  
 std::cout << "-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+\n";  
 int i = 0;  
 for (; p; p = p->next) {  
 std::cout.setf(std::ios::right);  
 std::cout.width(4);  
 std::cout << (i + 1) << ")"  
 << " ";  
 printItemDB(p->data);  
 i++;  
  
 if(kbhit()){  
 std::cout << "\nContinue?\n";  
 if (!(selectionCheck()))  
 break;  
 }  
 }  
 std::cout << "-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+\n";  
}  
/\* Вывод дерева \*/  
void outputTree(treeLawyer \*p){  
 std::cout << " 1) Только ключи\n";  
 std::cout << " 2) Всё дерево\n";  
 char c;  
 do {  
 c = (char)getch();  
 } while ((c != '1') && (c != '2'));  
// system("CLS");  
 outputTree\_LR(p, c != '1');  
}  
  
/\* Вывод слева направо \*/  
void outputTree\_LR(treeLawyer \*p, bool full) {  
 if (p != nullptr) {  
 outputTree\_LR(p->left, full);  
 if (full) {  
 std::cout << " > " << p->data << "\n";  
 int i = 0;  
 for (listDataBase \*temp = p->elems; temp; temp = temp->next) {  
 std::cout << " ";  
 std::cout.setf(std::ios::right);  
 std::cout.width(4);  
 std::cout << (i + 1) << ")"  
 << " ";  
 printItemDB(temp->data);  
 i++;  
 }  
 } else  
 std::cout << " > " << p->data << "\n";  
 outputTree\_LR(p->right, full);  
 }  
}  
  
void printTableSymbols(coding \*code, int numSymbols){  
 system("CLS");  
 std::cout << "╔═══════════╦══════════════════╦══════════════════════════╦══════════════════════════╦════════════════════════════════╗" << "\n";  
 std::cout << "║" << std::setw(11) << "" << "║" << std::setw(18) << "" << "║" << std::setw(26) << "" << "║" << std::setw(26) << "" << "║" << std::setw(32) << "" << "║" << "\n";  
 std::cout << "║" << "Код символа" << "║" << " Кол-во в тексте " << "║" << " Вероятность в тексте " << "║" << " Длина кодового слова " << "║" << " Кодовое слово " << "║" << "\n";  
 std::cout << "║" << std::setw(11) << "" << "║" << std::setw(18) << "" << "║" << std::setw(26) << "" << "║" << std::setw(26) << "" << "║" << std::setw(32) << "" << "║" << "\n";  
 std::cout << "╠═══════════╬══════════════════╬══════════════════════════╬══════════════════════════╬════════════════════════════════╣" << "\n";  
  
 float entropy = 0;  
 float averageLength = 0;  
 for (int i = 0; i < numSymbols; i++) {  
 entropy += code[i].probability \* std::log2(code[i].probability);  
 averageLength += (float)code[i].lengthCW \* code[i].probability;  
 std::cout << "║"  
 << std::setw(7) << (int)(unsigned char)code[i].symbol << std::setw(4) << "" << "║"  
 << std::setw(15) << code[i].quantity << std::setw(3) << "" << "║"  
 << std::setw(23) << std::fixed << code[i].probability << std::setw(3) << "" << "║"  
 << std::setw(23) << code[i].lengthCW << std::setw(3) << "" << "║"  
 << std::setw(29) << code[i].codeword << std::setw(3) << "" << "║" << "\n";  
 }  
 std::cout << "╚═══════════╩══════════════════╩══════════════════════════╩══════════════════════════╩════════════════════════════════╝" << "\n";  
  
 std::cout << " Энтропия: " << -entropy << "\n";  
 std::cout << " Средняя длина код. слова: " << averageLength << "\n";  
  
}

**func.hpp**

#ifndef FUNC\_HPP  
#define FUNC\_HPP  
#include <fstream>  
#include <iostream>  
#include <iomanip>  
#include <conio.h>  
#include <afxres.h> // STD\_INPUT\_HANDLE (Извлекает дескриптор указанного стандартного устройства)  
  
#include "struct.hpp"  
#include "rotationsAVL.hpp"  
  
void readDataBase(listDataBase \*&p, unsigned int & size);  
  
void digitalSort(listDataBase \*(&S), int reverse);  
  
void delList(listDataBase \*&p);  
  
void createIndexArr(itemDataBase\*\* &arr, listDataBase\* p ,unsigned int size);  
void delArr(itemDataBase\*\* &arr, unsigned int size);  
  
short int comparator(const char \*word1, const char \* word2);  
listDataBase \*binarySearch(itemDataBase \*\*arr, unsigned int size, char \*keySearch);  
  
void addAVL(treeLawyer \*&p, itemDataBase data, bool &growth);  
treeLawyer \*findVertexWithKey(treeLawyer \*p, char \*key);  
void delTree(treeLawyer \*&p);  
  
void tableSymbols(coding\* &code, int &numsUnique);  
  
void pause();  
bool selectionCheck();  
void clearBuffer();  
#endif // FUNC\_HPP

**func.cpp**

#include "func.hpp"  
  
/\* Считывание БД и формирование исходного списка \*/  
void readDataBase(listDataBase \*&p, unsigned int & size){  
 if (p != nullptr){  
 delList(p);  
 size = 0;  
 }  
 std::fstream \*file = new std::fstream;  
 file->open("testBase3.dat", std::ios::in | std::ios::binary);  
 if (!(\*file)){  
 std::cout << "Error#1. Файл \"testBase3.dat\" не найден!";  
 pause();  
 exit(1);  
 }  
  
 itemDataBase\* data = new itemDataBase;  
 listDataBase\* temp = p = new listDataBase;  
 file->read((char \*)data, sizeof(\*data)).eof();  
 temp->data = \*data;  
 size++;  
 while (!file->read((char \*)data, sizeof(\*data)).eof()) {  
 temp->next = new listDataBase;  
 temp = temp->next;  
 temp->data = \*data;  
 size++;  
 }  
 temp->next = nullptr;  
 file->close();  
}  
  
/\* Цифровая сортировка \*/  
void digitalSort(listDataBase \*(&S), int reverse) {  
 int KDI[32];  
 for (int i = 0; i < 30; i++)  
 KDI[i] = i;  
 KDI[30] = 31;  
 KDI[31] = 30;  
 int L = 32;  
  
 queue q[256];  
 listDataBase \*p;  
 unsigned char d;  
 int k;  
  
 for (int j = L - 1; j >= 0; j--) {  
 for (int i = 0; i <= 255; i++) {  
 q[i].tail = (listDataBase \*)&(q[i].head);  
 }  
 p = S;  
 k = KDI[j];  
 while (p != nullptr) {  
 d = p->Digit[k];  
 q[d].tail->next = p;  
 q[d].tail = p;  
 p = p->next;  
 }  
  
 p = (listDataBase \*)&S;  
  
 int i = 0;  
 int sign = 1;  
 if (reverse == 1) {  
 i = 255;  
 sign = -1;  
 }  
  
 while ((i > -1) && (i < 256)) {  
 if (q[i].tail != (listDataBase \*)&(q[i].head)) {  
 p->next = q[i].head;  
 p = q[i].tail;  
 }  
 i += sign;  
 }  
 p->next = nullptr;  
 }  
}  
  
/\* Удаление списка \*/  
void delList(listDataBase \*&p){  
 while(p){  
 listDataBase \* temp = p->next;  
 delete p;  
 p = temp;  
 }  
}  
  
/\* Инициализация массива \*/  
void createIndexArr(itemDataBase\*\* &arr, listDataBase\* p ,unsigned int size){  
 if (arr != nullptr)  
 delArr(arr, size);  
 arr = new itemDataBase \*[size] { nullptr };  
 for (int i = 0; p != nullptr; p = p->next, i++)  
 arr[i] = &(p->data);  
}  
  
/\* Удаление массива \*/  
void delArr(itemDataBase\*\* &arr, unsigned int size){  
 for (unsigned int i = 0; i < size; i++)  
 delete arr[i];  
 arr = nullptr;  
}  
  
short int comparator(const char \*word1, const char \* word2) {  
 int i = 0;  
 while (word1[i] != '\0' && word2[i] != '\0') {  
 if ((word1[i] == ' ' && word2[i] != ' '))  
 return -1;  
 else if ((word2[i] == ' ' && word1[i] != ' '))  
 return 1;  
 else if ((word1[i] < word2[i]))  
 return -1;  
 else if ((word1[i] > word2[i]) || (word2[i] == ' ' && word1[i] != ' '))  
 return 1;  
 i++;  
 }  
 return 0;  
}  
/\* Бинарный поиск (ver#2) \*/  
listDataBase \*binarySearch(itemDataBase \*\*arr, unsigned int size, char \*keySearch) {  
 listDataBase \*p = nullptr, \*head = nullptr;  
 int L = 0, R = (int)size - 1, m;  
 while (L < R) {  
 m = (L + R) / 2;  
 if (comparator(arr[m]->depositor, keySearch) == -1)  
 L = m + 1;  
 else  
 R = m;  
 }  
 if (comparator(arr[R]->depositor, keySearch) == 0) {  
 p = head = new listDataBase;  
 p->data = \*(arr[R]);  
 R++;  
  
 while (R < size && (comparator(arr[R]->depositor, keySearch) == 0)) {  
 p->next = new listDataBase;  
 p = p->next;  
 p->data = \*(arr[R]);  
 R++;  
 }  
 p->next = nullptr;  
 }  
 else  
 std::cout << "-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+"  
 << "\n"  
 << " < Значения для данного ключа в базе данных не найдены >"  
 << "\n"  
 << "-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+"  
 << "\n";  
 return head;  
}  
  
  
void addAVL(treeLawyer \*&p, itemDataBase data, bool &growth) {  
 if (p == nullptr) {  
 p = new treeLawyer;  
 strcpy(p->data, data.lawyer);  
 p->elems = new listDataBase;  
 p->elems->data = data;  
 p->elems->next = nullptr;  
 growth = true;  
 } else if (comparator(p->data, data.lawyer) == 1) {  
 addAVL(p->left, data, growth);  
 if (growth) {  
 if (p->balance > 0) {  
 p->balance = 0;  
 growth = false;  
 } else if (p->balance == 0) {  
 p->balance = -1;  
 growth = true;  
 } else if ((p->left)->balance < 0) {  
 rotateLL(p);  
 growth = false;  
 } else {  
 rotateLR(p);  
 growth = false;  
 }  
 }  
 }  
 else if (comparator(p->data, data.lawyer) == -1) {  
 addAVL(p->right, data, growth);  
 if (growth) {  
 if (p->balance < 0) {  
 p->balance = 0;  
 growth = false;  
 } else if (p->balance == 0) {  
 p->balance = 1;  
 growth = true;  
 } else if ((p->right)->balance > 0) {  
 rotateRR(p);  
 growth = false;  
 } else {  
 rotateRL(p);  
 growth = false;  
 }  
 }  
 }  
 else {  
 listDataBase \*q = p->elems;  
 while (q->next != nullptr)  
 q = q->next;  
 q->next = new listDataBase;  
 q = q->next;  
 q->data = data;  
 q->next = nullptr;  
 growth = false;  
 }  
}  
  
/\* Поиск вершины по ключу \*/  
treeLawyer \*findVertexWithKey(treeLawyer \*p, char \*key) {  
 treeLawyer \*q = p;  
 while (q != nullptr) {  
 if (comparator(key, q->data) == -1)  
 q = q->left;  
 else if (comparator(key, q->data) == 1)  
 q = q->right;  
 else  
 break;  
 }  
 return q;  
}  
  
void delTree(treeLawyer \*&p){  
 if(p != nullptr){  
 delTree(p->left);  
 delTree(p->right);  
  
 delList(p->elems);  
 delete p;  
 p = nullptr;  
 }  
}  
  
void quickSortCoding(coding\* A, int R, int L) // Сортировка для кодировки (сортировка по убыванию вероятности)  
{  
 while (L < R) {  
 float x = A[L].probability;  
 int i = L;  
 int j = R;  
 while (i <= j) {  
 while (A[i].probability > x)  
 i++;  
 while (A[j].probability < x)  
 j--;  
  
 if (i <= j) {  
 char tmp\_ch;  
 tmp\_ch = A[i].symbol;  
 A[i].symbol = A[j].symbol;  
 A[j].symbol = tmp\_ch;  
  
 unsigned int tmp\_q;  
 tmp\_q = A[i].quantity;  
 A[i].quantity = A[j].quantity;  
 A[j].quantity = tmp\_q;  
  
 float tmp\_prop;  
 tmp\_prop = A[i].probability;  
 A[i].probability = A[j].probability;  
 A[j].probability = tmp\_prop;  
 i++;  
 j--;  
 }  
 }  
  
 if (j - L > R - i) {  
 quickSortCoding(A, R, i);  
 R = j;  
 }  
 else {  
 quickSortCoding(A, j, L);  
 L = i;  
 }  
 }  
}  
  
//Находит медиану части массива P, т.е. такой индекс L <= m <= R, что минимальна величина  
int med(coding \*code, int borderL, int borderR) {  
 float SumL = 0;  
 for (int i = borderL; i < borderR; i++)  
 SumL = SumL + code[i].probability;  
  
 float SumR = code[borderR].probability;  
 int m = borderR;  
  
 while (SumL >= SumR) {  
 m = m - 1;  
 SumL = SumL - code[m].probability;  
 SumR = SumR + code[m].probability;  
 }  
  
 return m;  
}  
  
void codeFano(coding \* code, int borderL, int borderR, int k) {  
 //k - длина уже построенной части элементарных кодов  
 if (borderL < borderR) {  
 k = k + 1;  
 int m = med(code, borderL, borderR);  
 for (int i = borderL; i <= borderR; i++) {  
 if (code[i].codeword != nullptr){  
 char \*temp = new char[k + 1];  
 for(int j = 0; j < k - 1; j++)  
 temp[j] = code[i].codeword[j];  
 delete[] code[i].codeword;  
 code[i].codeword = temp;  
 }  
 else  
 code[i].codeword = new char[k + 1];  
  
 if (i <= m)  
 code[i].codeword[k - 1] = '0';  
 else  
 code[i].codeword[k - 1] = '1';  
  
 code[i].codeword[k] = '\0';  
 code[i].lengthCW = code[i].lengthCW+ 1;  
 }  
 codeFano(code, borderL, m, k);  
 codeFano(code, m + 1, borderR, k);  
 }  
}  
  
void tableSymbols(coding\* &code, int &numsUnique){  
 int windows866[256] = {0};  
 int totalNums = 0;  
 char ch;  
  
 std::fstream file("testBase3.dat", std::ios::in | std::ios::binary);  
  
 if (!(file.is\_open())){  
 std::cout << "Error#1. Файл \"testBase3.dat\" не найден!";  
 exit(1);  
 }  
  
 while (!file.read((char\*)&ch, sizeof(ch)).eof()){  
 totalNums++;  
 if (int(ch) < 0)  
 windows866[int(ch) + 256]++;  
 else  
 windows866[int(ch)]++;  
 }  
 file.close();  
  
 for (int i = 0; i < 256; i++)  
 if (windows866[i] != 0 )  
 numsUnique++;  
  
 code = new coding[numsUnique];  
 unsigned short int temp = 0;  
 for (int i = 0; i < 256; i++) {  
 if(windows866[i] != 0){  
 code[temp].symbol = char(i);  
 code[temp].quantity = windows866[i];  
 code[temp].probability = (float)windows866[i] / (float)totalNums;  
 temp++;  
 }  
 }  
  
 quickSortCoding(code, numsUnique - 1, 0);  
 codeFano(code, 0, numsUnique - 1, 0);  
  
}  
//  
  
/\* Пауза \*/  
void pause() {  
 std::cout << "\nPress any key to continue!\n";  
 getch();  
}  
  
/\* Да | Нет \*/  
bool selectionCheck() {  
 int key;  
 short int selection = -1;  
 clearBuffer();  
 while(selection == -1) {  
 key = getch();  
 if (key == 13) // Enter  
 selection = 1;  
 else if (key == 27) // Esc  
 selection = 0;  
 else if ((key == 110) || key == (78)) // n|N  
 selection = 0;  
 else if ((key == 121) || key == (89)) // y|Y  
 selection = 1;  
 else if ((key == 173) || key == (141)) // н|Н  
 selection = 1;  
 else if ((key == 226) || key == (146)) // т|Т  
 selection = 0;  
 }  
 std::cout << "\n";  
 return selection;  
}  
  
/\* Очистка буфера \*/  
void clearBuffer(){  
 FlushConsoleInputBuffer(GetStdHandle(STD\_INPUT\_HANDLE)); // Очищает буфер консоли  
 if (std::cin.fail()) { // Очищает поток ввода  
 std::cin.clear();  
 while (std::cin.get() != '\n')  
 ;  
 }  
}

**rotationsAVL.hpp**

#ifndef ROTATIONSAVL\_HPP  
#define ROTATIONSAVL\_HPP  
#include "struct.hpp"  
  
void rotateRR(treeLawyer \*&p);  
void rotateLL(treeLawyer \*&p);  
void rotateLR(treeLawyer \*&p);  
void rotateRL(treeLawyer \*&p);  
#endif // ROTATIONSAVL\_HPP

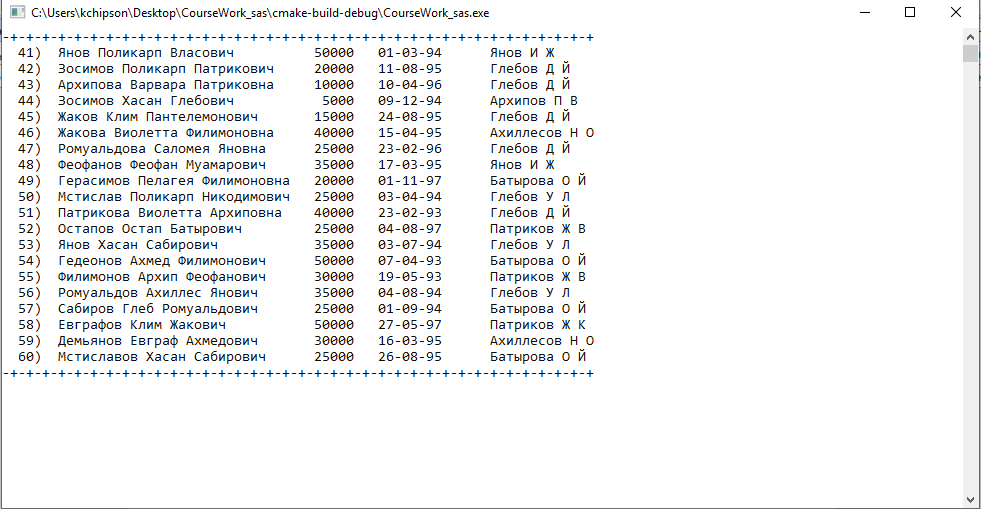
**rotationsAVL.cpp**

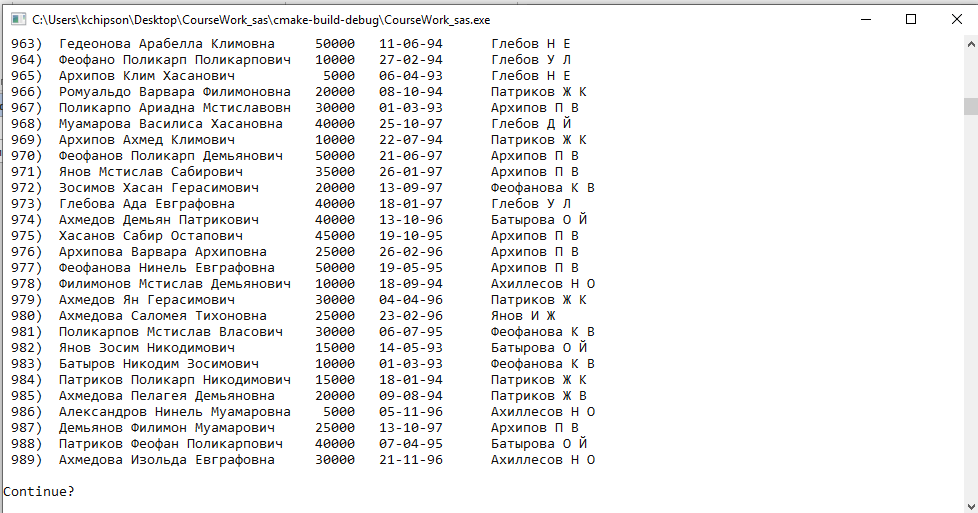
#include "rotationsAVL.hpp"  
  
void rotateRR(treeLawyer \*&p) {  
 treeLawyer \*q = p->right;  
 p->balance = 0;  
 q->balance = 0;  
 p->right = q->left;  
 q->left = p;  
 p = q;  
}  
  
void rotateLL(treeLawyer \*&p) {  
 treeLawyer \*q = p->left;  
 p->balance = 0;  
 q->balance = 0;  
 p->left = q->right;  
 q->right = p;  
 p = q;  
}  
  
void rotateLR(treeLawyer \*&p) {  
 treeLawyer \*q = p->left;  
 treeLawyer \*r = q->right;  
 if (r->balance < 0)  
 p->balance = 1;  
 else  
 p->balance = 0;  
  
 if (r->balance > 0)  
 q->balance = -1;  
 else  
 q->balance = 0;  
  
 r->balance = 0;  
 q->right = r->left;  
 p->left = r->right;  
 r->left = q;  
 r->right = p;  
 p = r;  
}  
  
void rotateRL(treeLawyer \*&p) {  
 treeLawyer \*q = p->right;  
 treeLawyer \*r = q->left;  
 if (r->balance > 0)  
 p->balance = -1;  
 else  
 p->balance = 0;  
  
 if (r->balance < 0)  
 q->balance = 1;  
 else  
 q->balance = 0;  
  
 r->balance = 0;  
 q->left = r->right;  
 p->right = r->left;  
 r->right = q;  
 r->left = p;  
 p = r;  
}

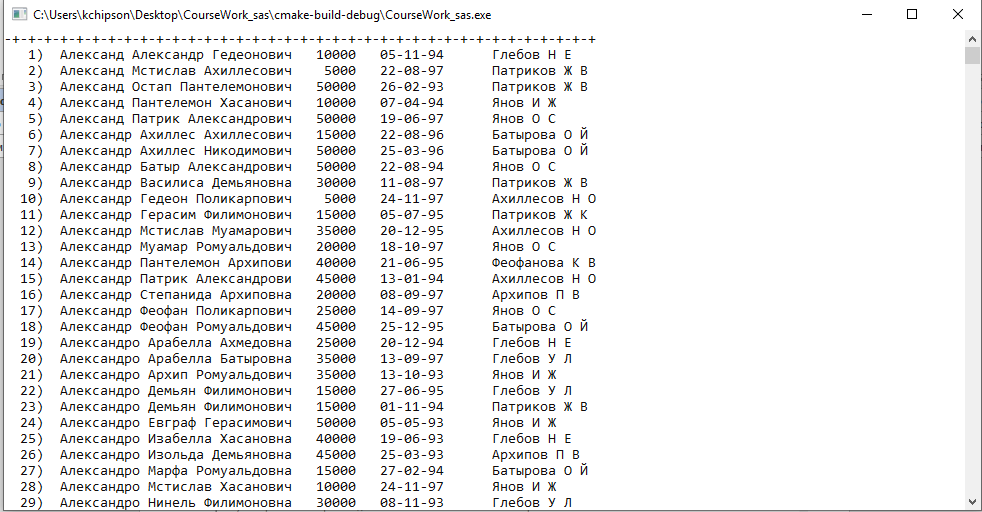
# Результаты

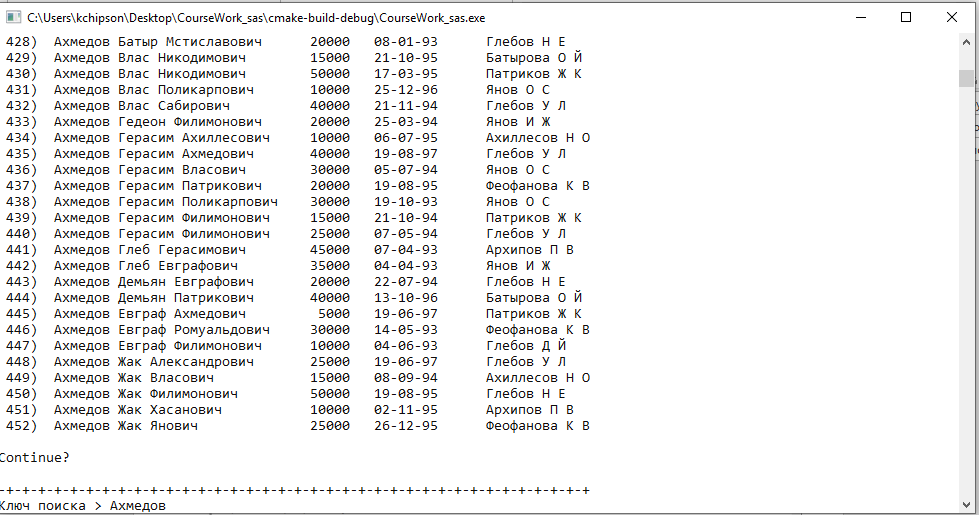


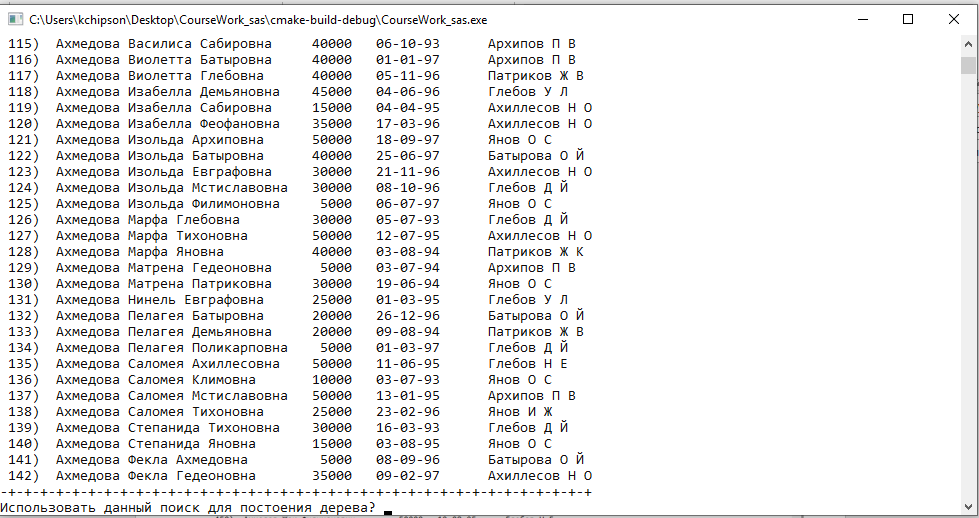


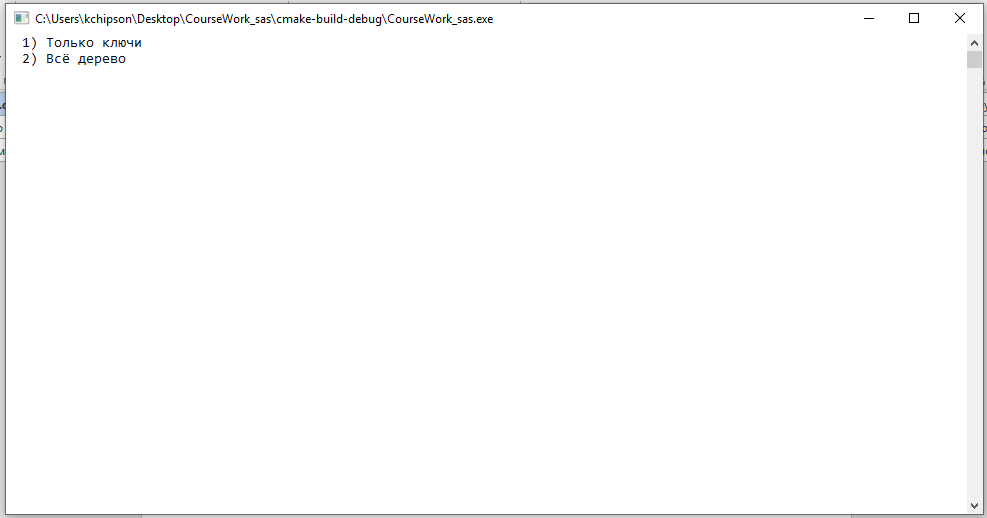


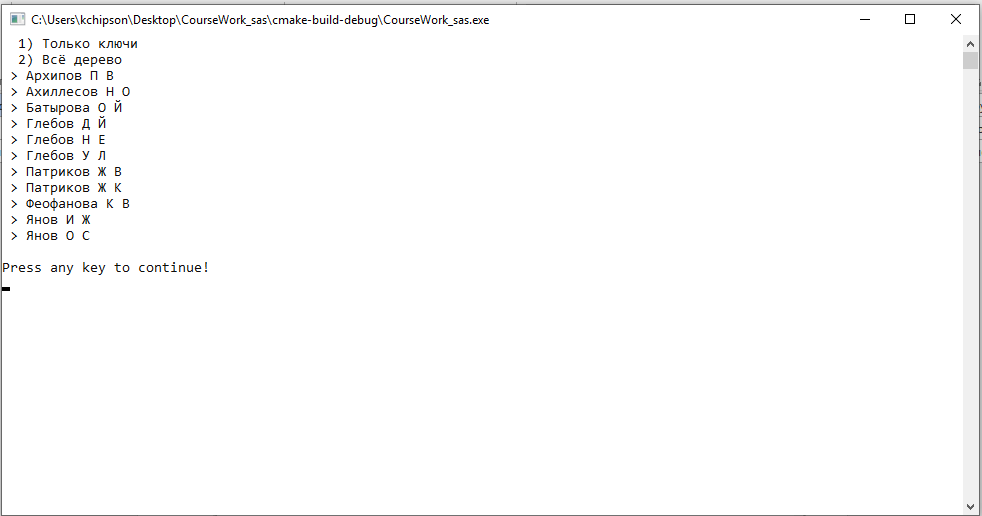


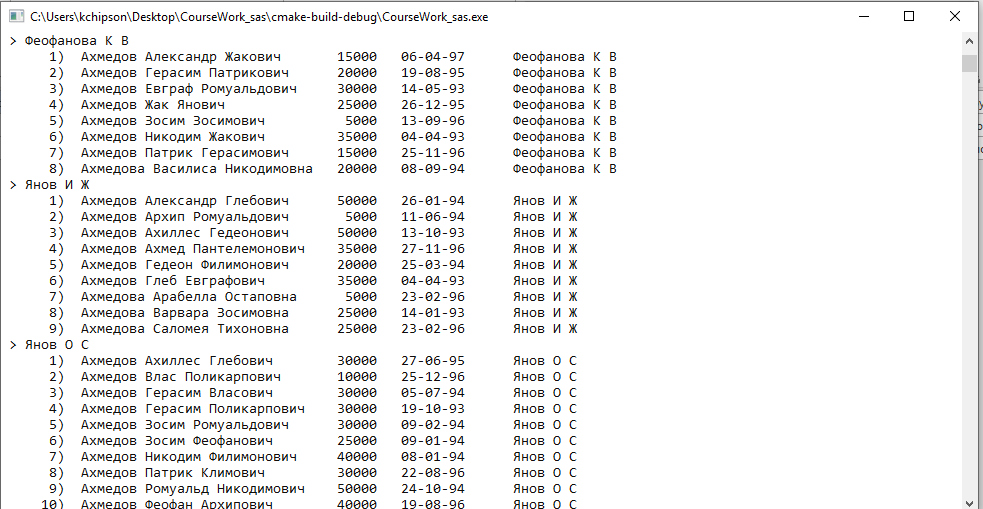


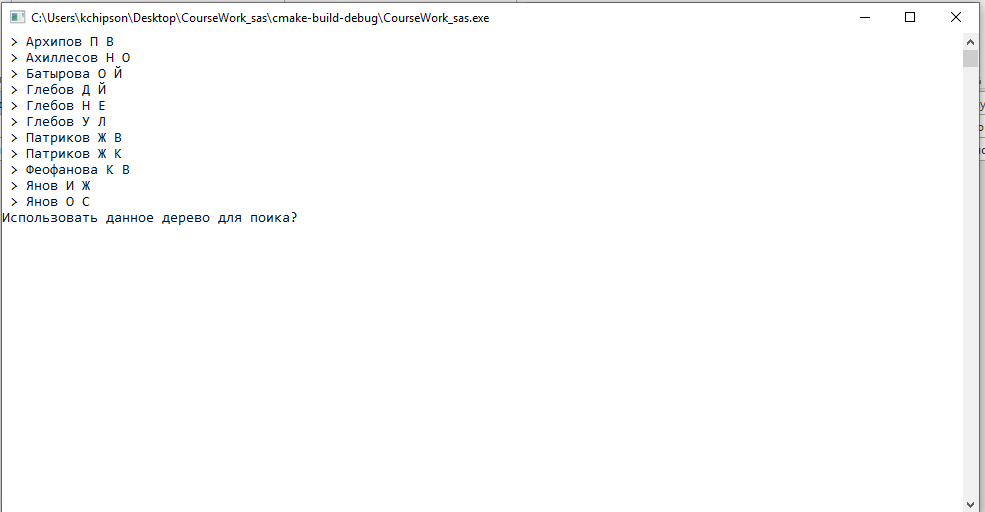


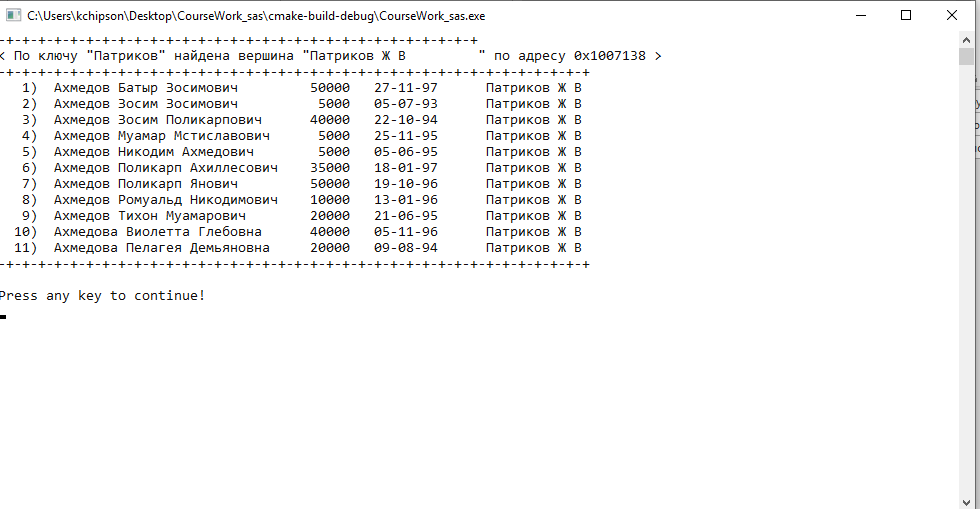


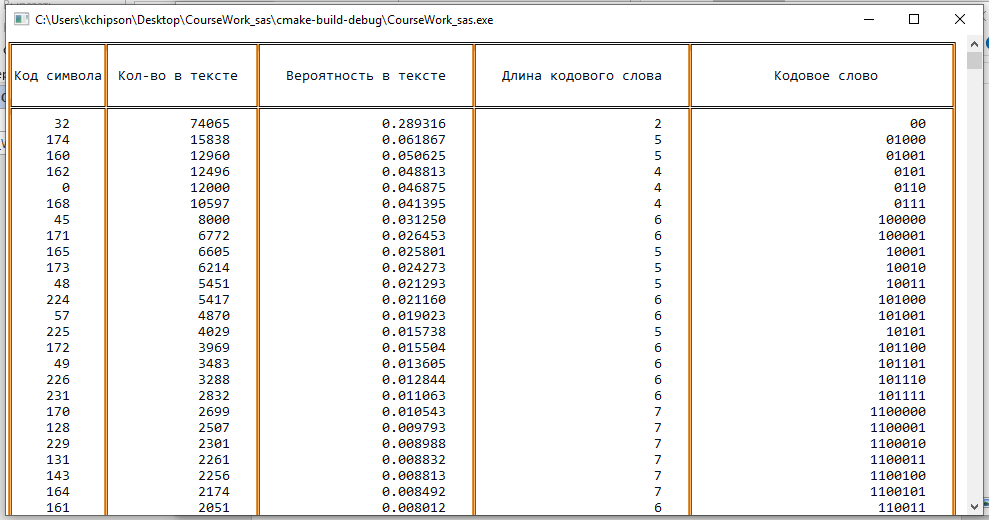


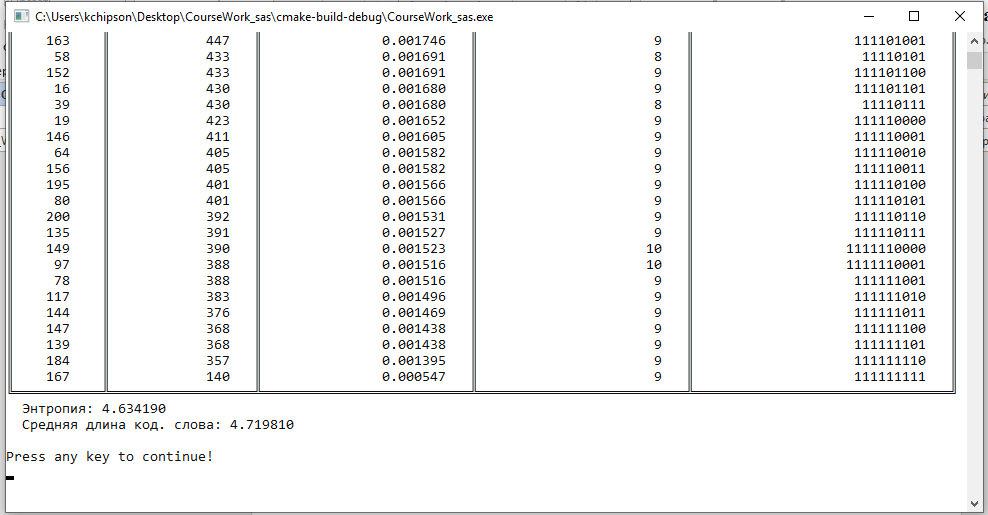












# Выводы

В ходе выполнения данной работы была создана программа, предоставляющая функционал по обработке базы данных “Обманутые вкладчики”:

1) Вывести исходную базу данных

2) Вывести отсортированную базу данных

3) Бинарный поиск по базе

4) Дерево

5) Бинарный поиск в дереве

6) Статический код

В ходе выполнения курсовой работы были выполнены все поставленные задачи и реализованы необходимые алгоритмы: сортировки, поиск, построение АВЛ-дерева, поиск 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.