Федеральное агентство связи (Россвязь).

Федеральное государственное бюджетное образовательное учреждение «Сибирский государственный университет телекоммуникаций и информатики»

(СибГУТИ)

Кафедра прикладной математики и кибернетики

КУРСОВОЙ ПРОЕКТ

По дисциплине «Структуры и алгоритмы обработки данных»

Вариант 13

Выполнил: Студент 2 курса группы ИП-811

Лупарев М.А.

Проверил: Доцент кафедры Янченко Е. В.

# **Содержание**

[**Содержание** 2](#_Toc27346479)

[**1. Постановка задачи** 3](#_Toc27346480)

[**2. Основные идеи и характеристики применяемых методов** 4](#_Toc27346481)

[**2.1. Метод сортировки** 4](#_Toc27346482)

[**2.2. Двоичный поиск** 6](#_Toc27346483)

[**2.3. Списки и очереди** 7](#_Toc27346484)

[**2.4. Вид дерева и поиск** 8](#_Toc27346485)

[**2.5. Метод кодирования** 9](#_Toc27346486)

[**3. Описание структур данных и использованных алгоритмов.** 10](#_Toc27346487)

[**3.1. Использованные структуры данных** 10](#_Toc27346488)

[**3.2. Особенности реализации алгоритмов** 11](#_Toc27346489)

[**4. Описание программы** 12](#_Toc27346490)

[**4.1. Основные переменные и структуры** 12](#_Toc27346491)

[**4.2. Описание подпрограмм** 13](#_Toc27346492)

[**5. Исходный текст программы** 17](#_Toc27346493)

[**6. Результаты** 43](#_Toc27346494)

[**7. Выводы** 45](#_Toc27346495)

# **1. Постановка задачи**

Хранящуюся в файле базу данных “Обманутые вкладчики”(4000 записей) - загрузить динамически в оперативную память с формированием индексного массива как массива указателей, вывести на экран по 25 записей (строк) на странице с возможностью отказа от просмотра.

Упорядочить данные методом сортировки Уильямса-Флойда по ФИО и сумме вклада. Упорядоченные данные вывести на экран.

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

Из записей очереди построить дерево поиска (АВЛ-дерево)по дате вклада, вывести на экран содержимое дерева 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, вычислить среднюю длину кодового слова и сравнить ее с энтропией исходного файла.

Упакованную базу данных записать в файл, вычислить коэффициент сжатия данных.

# **2. Основные идеи и характеристики применяемых методов**

## **2.1. Метод сортировки**

Метод сортировки Уильямса - Флойда основан на алгоритме построения пирамиды.

Последовательность **aL, aL+1, …, aR**называется пирамидой, если неравенство

ai≤ min (a2i, a2i+1)

выполняется для всех i, для которых хотя бы один из элементов a2i и a2i+1 существует.

Свойства пирамиды:

**1. Двустороннее усечение:**

Если последовательность aL, aL+1, ..., аR-1, aR–пирамида, то aL+1, ..., aR-1 тоже пирамида.

**2.** Если a1, a2, ..., an– пирамида, то а1 – минимальный элемент пирамиды.

**3.** Если a1, ..., an– произвольная последовательность, то an/2,.., an– пирамида.

Построение пирамиды:

Пусть aL+1, …, aR - пирамида, необходимо добавить элемент Х, чтобы получить новую пирамиду aL, …, aR.

Новый элемент **добавляем в начало**, расширяя последовательность влево.

Если aLудовлетворяет **условию пирамиды**, то пирамида построена.

Иначе найдутся такие a2L или a2L+1 , что **не будут удовлетворять** условию пирамиды.

Возьмем минимальный элемент из a2L и a2L+1, обозначим его за aj и обменяем с aL.

В результате получим a’L ≤ a2L и a’L ≤ a2L+1, что удовлетворяет **условию пирамиды**.

Идея сортировки:

**Первый этап.** Построение пирамиды из элементов массива.

В соответствии со **свойством3** правая часть массива уже пирамида. Будем добавлять по одному элементу слева, расширяя пирамиду, пока в нее не войдут все элементы массива.

**Второй этап.** Сортировка.

По **свойству 2** в пирамиде первый элемент минимальный. Производим двустороннее усечение пирамиды: уберем элементы а1 и аn. По **свойству1**a2, .., an-1 – пирамида. Поставим элемент а1 на последнее место, а элемент аn добавим к пирамиде a2,..,an-1. Отсекаем последний элемент и повторяем действия, пока пирамида не исчезнет.

Характеристики:

Оценим трудоемкость сортировки, используя уже известную оценку трудоемкости построения пирамиды:

C = 2 M = + 2

На первом этапе построение пирамиды производится n/2 раз, на втором этапе – n-1 раз.

Очевидно, трудоемкость пирамидальной сортировки имеет порядок O(nlog2n), , *n→∞*.

Количество операций сравнения и пересылки оценивается следующими неравенствами:

C < 2 n log2n + n + 2 M < n log2n + 6.5n - 4

Пирамидальная сортировка не устойчива.

Метод практически не зависит от исходной упорядоченности массива.

## **2.2. Двоичный поиск**

Идеяпоиска:

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 половине массива.
3. Выбранный элемент больше X. Продолжаем поиск в левой половине массива.

Если в массиве несколько элементов с одинаковым ключом, алгоритм двоичного поиска, используемый в курсовом проекте, находит самый левый из них. Для поиска остальных элементов с заданным ключом требуется просмотреть массив только в одном направлении – вправо от найденного элемента.

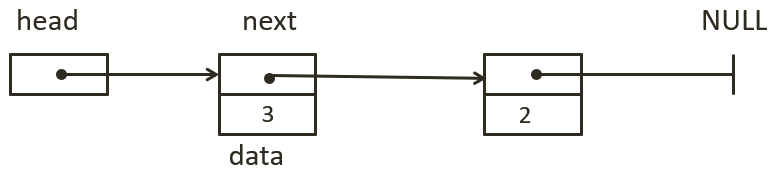
Характеристики:

На каждой итерации поиска необходимоодно сравнение. Количество итераций не больше, чем . Таким образом, трудоёмкость двоичного поиска:

С=O(log*n*), *n → ∞*.

## **2.3. Списки и очереди**

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



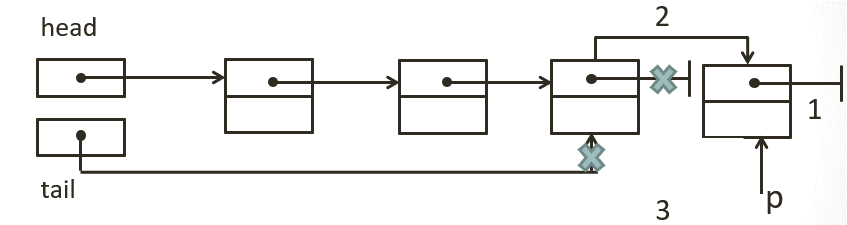
Поле **Next** может занимать произвольное место в структуре элементов списка. Однако, если оно является первым элементом структуры, то его адрес совпадает с адресом элемента списка, и это позволяет **оптимизировать** многие операции со списками.

*Очередь* – это один из двух видов списка, в котором новый элемент добавляется в конец последовательности, удаляется первый элемент последовательности.

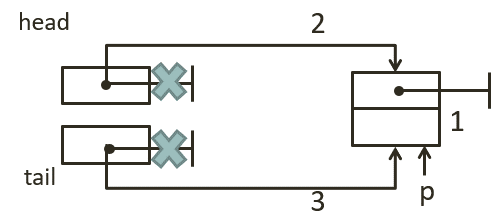
Очередь реализует дисциплину обслуживания **FIFO** (**F**irst**I**nput, **F**irst**O**utput)

Основные операции с очередью:

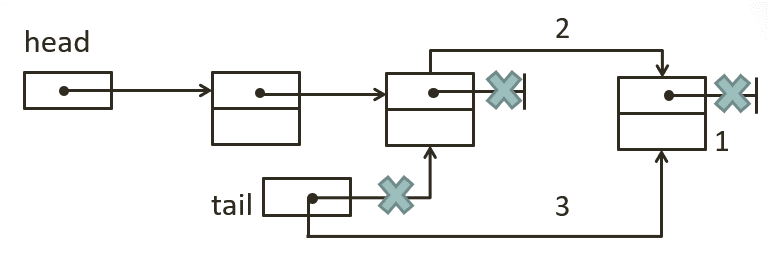
- Добавление элемента в конец очереди (непустой):



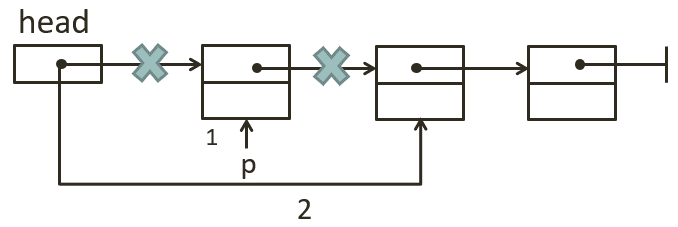
- Добавление элемента в пустую очередь:



- Добавление элемента по адресу p в очередь:



- Исключение первого элемента из очереди



## **2.4. Вид дерева и поиск**

**АВЛ-дерево** — сбалансированное по высоте двоичное дерево поиска: для каждой его вершины высота её двух поддеревьев различается не более чем на 1.

АВЛ — аббревиатура, образованная первыми буквами фамилий создателей (советских учёных) Георгия Максимовича Адельсон-Вельского и Евгения Михайловича Ландиса.

Идея построения:

Вначале **добавимновую 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**.

Характеристики:

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:

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**.

## **2.5. Метод кодирования**

Характеристики:

Пусть имеется **дискретный вероятностный источник**, порождающий символы алфавита с вероятностями .

Основной характеристикой источника является **энтропия**, которая представляет собой

**среднее значение количества информации** в сообщении источника и определяется выражением (для двоичного случая):

Энтропия характеризует **меру неопределенности выбора** для данного источника.

Код называется алфавитным, если кодовые слова лексикографически упорядочены, т.е.

Е. Н. Гилбертом и Э. Ф. Муром был предложен метод построения алфавитного кода, для которого

Идея построения:

1. Вычислим величины ,i=1, n:

…

2. Представим суммы в двоичном виде.

3. В качестве кодовых слов возьмем младших бит в двоичном представлении .

# **3. Описание структур данных и использованных алгоритмов.**

## **3.1. Использованные структуры данных**

В курсовом проекте были использованы следующие структуры данных:

Простые структуры:

* Числовые
* Символьные
* Логические
* Указатели

Составные структуры:

* Массивы - структура данных, хранящая набор значений (элементов массива), идентифицируемых по индексу или набору индексов, принимающих целые (или приводимые к целым) значения из некоторого заданного непрерывного диапазона.
* Очереди - список, операции чтения и добавления элементов в котором подвержены определенным правилам. При этом, при чтении элемента, он удаляется из очереди. Все операции проводятся по принципу «Первый пришел, первый вышел» (FIFO — first in, first out). Таким образом, для чтения в очереди доступна только голова, в то время как добавление проводится только в хвост.
* Деревья - структура данных, представляющая собой древовидную структуру в виде набора связанных узлов.

## **3.2. Особенности реализации алгоритмов**

В реализации алгоритма построения АВЛ – дерева можно выделить следующие особенности:

- Использованиерекурсии, которая позволит хранить адреса всех пройденных вершин по пути поиска и автоматически в них возвращаться на обратном пути рекурсии.

- В каждую вершину дерева вводится дополнительный показатель баланса Bal.

- Дополнительная глобальная переменная Rost обозначающая, что произошел рост дерева.

В реализации алгоритма сортировки методом Уильямса – Флойда присутствуют следующие особенности:

- Отдельная процедура для построения пирамиды.

-Использование рекурсии, при построении пирамиды.

В реализации алгоритма кодирования методом Гилберта-Мура выделю:

- Создание отдельного массива так называемых кумулятивных вероятностей для последующего построения кодовых слов.

Также при создании очереди была использована оптимизированная версия инициализации tail := (tLE\*) &head, что позволило сократить алгоритм добавления элемента в очередь.

# **4. Описание программы**

## **4.1. Основные переменные и структуры**

Основные переменные:

* constintN – количество элементов которые нужно считать из файла
* note \*db – указатель на массив считанных из файла записей
* note \*\*point\_db - указатель на массив указателей на записи, считанные из файла
* Queue\* search – указатель на очередь найденных по запросу элементов
* elem\_tree \*root – указатель на корень АВЛ-дерева

Основные структуры:

* note

Структура записи из базы данных, состоящая из полей:

* char cont\_fio[30] - ФИОвкладчика
* unsigned short int sum - Суммавклада
* chardate[10] - Дата вклада
* char adv\_fio[22] – ФИОадвоката
* elem\_queue

Структура элемента очереди, состоящая из полей:

* elem\_queue \*next – указатель на следующий элемент очереди
* note \*elem\_db – указатель на запись
* Queue

Структура очереди, состоящая из полей:

* elem\_queue \*head – указатель на головной элемент очереди
* elem\_queue \*tail – указатель на хвостовой элемент очереди
* elem\_tree

Структура элемента дерева, состоящая из полей:

* intBal – баланс элемента дерева
* queue \*data – данные элемента дерева, представленные в виде указателя на очередь, состоящую из записей
* elem\_tree \*left – указатель на левую вершину дерева
* elem\_tree \*right – указатель на правую вершину дерева

## **4.2. Описание подпрограмм**

* note\* readDb(char\* file, int num\_notes);

Возвращает указатель на созданный массив структур.

Параметры:

char\* file – имя файла

intnum\_notes – кол-во записей, которые нужно считать

Функция открывает file в бинарном режиме, создает массив структур и считывает в данный массив структур num\_notes структур из файла.

* void printDbAll(note \*\*db,const int N);

Параметры:

note \*\*db – массив указателей на структуры

constintN – размер массива db

Функция выводит на экран абсолютно все записи массива db.

* void printDbPage(note \*\*db,const int N);

Параметры:

note \*\*db – массив указателей на структуры

constintN – размер массива db

Функция выводит на экран по 25 записей массива db. Предоставляет возможность переключаться по страницами при помощи стрелок влево и вправо.

* void dbUnsort(note\*\* point\_db,note\* db,int n);

Параметры:

note \*\*point\_db – массив указателей на структуры

note \*db – массив структур

constintN – размермассиваdb

Функция возвращает массив указателей на структуры point\_db к первоначальному виду, основываясь на массиве структур db.

* void makeHeap(note \*\*db, int n, int L,int R);

Параметры:

note \*\*db – массив указателей на структуры

intn – размер массива db

intL – левая граница для построения пирамиды

intR – правая граница для построения пирамиды

Функция производит построение пирамиды на интервале массива db от L до R.

* void heapSort(note \*\*db, int n);

Параметры:

note \*\*db – массив указателей на структуры

intn – размер массива db

Функция производит сортировку массива db методом Уильямса-Флойда.

* bool noteCmp(note \*a,note \*b);

Возвращает bool значение. True, если запись a больше либо равна b ,false иначе.

Параметры:

note \*a – первая запись для сравнения

note \*b – вторая запись для сравнения

Функция сравнивает две записи по ФИО и сумме вклада.

* int noteDateCmp(note \*a,note \*b);

Возвращает int значение. -1, если поле дата записи a больше, чем поле дата записи b, 1, если поле дата записи a меньше, чем поле дата записи b, 0 иначе.

Параметры:

note \*a – первая запись для сравнения

note \*b – вторая запись для сравнения

Функция сравнивает две записи по дате вклада.

* int searchCmp(char\* a,char\* b);
* Возвращает int значение. -1, если строка a лексикографически меньше, чем строка b, 1, если строка a лексикографически больше, чем строка b, 0, иначе.

Параметры:

char \*a – первая запись для сравнения

char \*b – вторая запись для сравнения

Функция производит сравнение двух строк для корректного осуществления бинарного поиска.

* void queuePrintPage(Queue\* queue);

Параметры:

Queue\* queue – очередь которую нужно вывести на экран

Функция отображает страницу вывода содержимого очереди.

* Queue\* searchPage(note\*\* db,int n);

Возвращает очередь, содержащую записи, у которых совпали значения первых трёх букв ФИО с ключом.

Параметры:

note \*\*db – массив указателей на структуры

intn – размер массива db

Функция отображает страницу поиска по базе.

* void queuePush(Queue \*queue,note \*elem\_db);

Параметры:

Queue\* queue – очередь, в которую нужно добавить запись

note \*elem\_db – указатель запись, которую нужно добавить в очередь

Функция добавляет в конец очереди queue указатель на запись elem\_db.

* void queuePrint(elem\_queue\* head);

Параметры:

elem\_queue \*head – головная запись очереди.

Функция выводит на экран все записи содержащиеся в очереди с головным элементом head.

* Queue\* bSearch(note\*\* db,intn,char\* X);

Возвращает очередь, содержащую записи, у которых совпали значения первых трёх букв ФИО с ключом X.

Параметры:

note \*\*db – массив указателей на структуры

intn – размер массива db

char\* X – строка, содержащая поисковой запрос

Функция осуществляет поиск записей подходящих под поисковой запрос, а так же создание и добавление в очередь всех этих записей.

* void treeSearchPage(elem\_tree\* root);

Параметры:

elem\_tree \*root – указатель на корень дерева

Функция отображает страницу поиска по дереву

* elem\_tree\* treeSearchDate(elem\_tree\* root, char\* search\_str);

Возвращает элемент дерева подходящий под поисковой запрос search\_str.

Параметры:

elem\_tree \*root – указатель на корень дерева

char \*search\_str – поисковой запрос

Функция осуществляет поиск по дереву с корнем root, элемента, подходящего под поисковой запросsearch\_str.

* void treePrintPage(elem\_tree\* root);

Параметры:

elem\_tree \*root – указатель на корень дерева

Функция отображает страницу для вывода содержимого дерева с корнем root.

* void treeBypassLR(elem\_tree\* root);

Параметры:

elem\_tree \*root – указатель на корень дерева

Функция осуществляет обход дерева слева направо, выводя данные на экран.

* void LL\_rotate(elem\_tree \*& p);

Параметры:

elem\_tree \*&p – ссылка на указатель на вершину дерева, в которой нарушился баланс

Функция осуществляет LL поворот в АВЛ-дереве.

* void RR\_rotate(elem\_tree \*& p);

Параметры:

elem\_tree \*&p – ссылка на указатель на вершину дерева, в которой нарушился баланс

Функция осуществляет RR поворот в АВЛ-дереве.

* void LR\_rotate(elem\_tree \*& p);

Параметры:

elem\_tree \*&p – ссылка на указатель на вершину дерева, в которой нарушился баланс

Функция осуществляет LR поворот в АВЛ-дереве.

* void RL\_rotate(elem\_tree \*& p);

Параметры:

elem\_tree \*&p – ссылка на указатель на вершину дерева, в которой нарушился баланс

Функция осуществляет RL поворот в АВЛ-дереве.

* void addAVL(note\* D,elem\_tree \*&p);

Параметры:

note \*D – указатель на запись, которую нужно добавить в дерево

elem\_tree \*&p – ссылка на указатель на корень дерева

Функция производит добавление указателя на запись D в дерево с корнем p.

* elem\_tree\* createTree(Queue\* queue);

Возвращает указатель на корень созданного АВЛ-дерева.

Параметры:

Queue \*queue – указатель на очередь содержащую записи, из которых нужно создать АВЛ-дерево.

Функция создает из записей очереди queue, АВЛ-дерево.

# **5. Исходный текст программы**

***CURS\_WORK.cpp***

#include <fstream>

#include <iostream>

#include <stdlib.h>

#include <iomanip>

#include <conio.h>

#include <math.h>

#include <string.h>

using namespace std;

bool Rost;

struct note{

char contr\_fio[30];

unsigned short int sum;

char date[10];

char adv\_fio[22];

};

struct elem\_queue{

elem\_queue\* next;

note \*elem\_db;

};

struct Queue{

elem\_queue\* head;

elem\_queue\* tail;

};

struct elem\_tree{

int Bal;

Queue\* data;

elem\_tree\* left;

elem\_tree\* right;

};

note\* readDb(char\* file, int num\_notes);

void printDbAll(note \*\*db,const int N);

void printDbPage(note \*\*db,const int N);

void dbUnsort(note\*\* point\_db,note\* db,int n);

void makeHeap(note \*\*db, int n, int L,int R);

void heapSort(note \*\*db, int n);

bool noteCmp(note \*a,note \*b);

int noteDateCmp(note \*a,note \*b);

int searchCmp(char\* a,char\* b);

void queuePrintPage(Queue\* queue);

Queue\* searchPage(note\*\* db,int n);

void queuePush(Queue \*queue,note \*elem\_db);

void queuePrint(elem\_queue\* head);

Queue\* bSearch(note\*\* db,intn,char\* X);

void treeSearchPage(elem\_tree\* root);

elem\_tree\* treeSearchDate(elem\_tree\* root, char\* search\_str);

void treePrintPage(elem\_tree\* root);

void treeBypassLR(elem\_tree\* root);

void LL\_rotate(elem\_tree \*& p);

void RR\_rotate(elem\_tree \*& p);

void LR\_rotate(elem\_tree \*& p);

void RL\_rotate(elem\_tree \*& p);

void addAVL(note\* D,elem\_tree \*&p);

elem\_tree\* createTree(Queue\* queue);

note\* readDb(char\* file, int num\_notes){

ifstreamdb\_file(file, ios\_base::binary);

note \*db = new note[num\_notes];

db\_file.read((char\*)db,sizeof(note)\*num\_notes);

return db;

}

elem\_tree\* treeSearchDate(elem\_tree\* root, char\* search\_str){

elem\_tree\* p = root;

while(p != NULL){

int date\_a = (p->data->head->elem\_db->date[6]-48)\*100000+(p->data->head->elem\_db->date[7]-48)\*10000+(p->data->head->elem\_db->date[3]-48)\*1000+(p->data->head->elem\_db->date[4]-48)\*100+(p->data->head->elem\_db->date[0]-48)\*10+(p->data->head->elem\_db->date[1]-48);

int date\_b = (search\_str[6]-48)\*100000+(search\_str[7]-48)\*10000+(search\_str[3]-48)\*1000+(search\_str[4]-48)\*100+(search\_str[0]-48)\*10+(search\_str[1]-48);

if(date\_a<date\_b)p=p->right;

else if(date\_a>date\_b)p=p->left;

else break;

}

if(p != NULL){

for(elem\_queue\* temp=p->data->head;temp!=NULL;temp = temp->next){

cout<<setw(30) << temp->elem\_db->contr\_fio<<setw(10) << temp->elem\_db->sum <<setw(11) << temp->elem\_db->date <<setw(22) << temp->elem\_db->adv\_fio<<endl;

}

return p;

}

else{

cout<< "Элементыненайдены" <<endl;

return NULL;

}

return p;

}

void printDbPage(note \*\*db,const int N){

system("CLS");

cout.setf(ios::left);

cout.width(70);

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

int on\_page = 25;

cout<<setw(6) << i+1 <<setw(30) << (\*db[i]).contr\_fio<<setw(10) << (\*db[i]).sum <<setw(11) << (\*db[i]).date <<setw(22) << (\*db[i]).adv\_fio<<endl;

if(!((i+1)%on\_page)){

cout<<endl<< "Страница: " << i/on\_page+1 << " из " << N/on\_page<<endl<<endl;

cout<< "<- и -> для смены страницы или ESC, чтобы выйти";

char key = (char)NULL;

do{

key = getch();

if(key == -35)key = getch();

switch(key){

case 77: if(i>=N-1){

i=-1;

}break;

case 75: if(i==on\_page-1){

i=N-on\_page-1;

break;

}else{

i-=on\_page\*2;

}break;

}

}while(key != 77 && key != 75 && key != 27);

if(key == 27)return;

system("CLS");

}

}

cout.unsetf(ios::left);

}

bool noteCmp(note \*a,note \*b){

if(strcmp(a->contr\_fio,b->contr\_fio)<0){

return false;

}else if(strcmp(a->contr\_fio,b->contr\_fio)>0){

return true;

}else if(a->sum<b->sum){

return false;

}else{

return true;

}

}

int noteDateCmp(note \*a,note \*b){

int date\_a = (a->date[6]-48)\*100000+(a->date[7]-48)\*10000+(a->date[3]-48)\*1000+(a->date[4]-48)\*100+(a->date[0]-48)\*10+(a->date[1]-48);

int date\_b = (b->date[6]-48)\*100000+(b->date[7]-48)\*10000+(b->date[3]-48)\*1000+(b->date[4]-48)\*100+(b->date[0]-48)\*10+(b->date[1]-48);

if(date\_a>date\_b){

return -1;

}else if(date\_a<date\_b){

return 1;

}else{

return 0;

}

}

void makeHeap(note \*\*db, int n, int L,int R){

note \*x=db[L];

int i=L;

while(1){

int j=2\*i;

if(j>R)break;

if((j<R) && (noteCmp(db[j+1],db[j])))j=j+1;

if(noteCmp(x,db[j]))break;

db[i]=db[j];

i=j;

}

db[i]=x;

}

void heapSort(note \*\*db, int n){

int L=floor((n-1)/2);

while(L>=0){

makeHeap(db,n,L,n-1);

L=L-1;

}

int R=n-1;

while(R>0){

note \*temp = db[0];

db[0]=db[R];

db[R]=temp;

R=R-1;

makeHeap(db,n,0,R);

}

};

void dbUnsort(note\*\* point\_db,note\* db,int n){

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

point\_db[i]=&db[i];

}

}

void queuePush(Queue \*queue,note \*elem\_db){

elem\_queue\* part = new elem\_queue;

part->elem\_db = elem\_db;

part->next = NULL;

queue->tail->next = part;

queue->tail = part;

}

void queuePrint(elem\_queue\* head){

if(head == NULL){

cout<< "Ничего не найдено!" <<endl;

return;

}

cout.setf(ios::left);

for(int i=1;head != NULL;head = head->next,i++){

cout<<setw(6) <<i<<setw(30) << head->elem\_db->contr\_fio<<setw(10) << head->elem\_db->sum <<setw(11) << head->elem\_db->date <<setw(22) << head->elem\_db->adv\_fio<<endl;

}

cout<<endl;

cout.unsetf(ios::left);

return;

}

int searchCmp(char\* a,char\* b){

int i;

for(i=0;b[i]!='\0';i++){

if(a[i]<b[i] || a[i]==' '){

return -1;

}

if(a[i]>b[i]){

return 1;

}

}

return 0;

}

Queue\* bSearch(note\*\* db,intn,char\* X){

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

bool find = false;

while(L<R){

m = floor((float)(L+R)/(float)2);

if(searchCmp(db[m]->contr\_fio,X)==-1){

L=m+1;

}else{

R=m;

}

}

if(searchCmp(db[R]->contr\_fio,X)==0){

find = true;

}else{

find = false;

}

Queue \*answer = new Queue;

answer->head = NULL;

answer->tail = (elem\_queue\*)&(answer->head);

for(int i=R;i<n &&searchCmp(db[i]->contr\_fio,X)==0;i++){

queuePush(answer,db[i]);

}

return answer;

}

void treeSearchPage(elem\_tree\* root){

char key;

Queue \*answer = NULL;

do{

system("CLS");

cout<< "Введите поисковой запрос в формате (dd-mm-yy):";

char lol[8];

cin>> lol;

if(lol[2]=='-' && lol[5]=='-'){

treeSearchDate(root,lol);

}else{

cout<< "Неверныйпоисковойзапрос" <<endl;

}

cout<<endl<< "Нажмите любую клавишу чтобы продолжить поиск..." <<endl;

cout<< "ESC) Выйти" <<endl;

elem\_tree\* root = NULL;

key = getch();

}while(key != 27);

}

void treePrintPage(elem\_tree\* root){

char key;

do{

system("CLS");

if(root!= NULL){

treeBypassLR(root);

}else{

cout<< "Деревопусто" <<endl;

}

cout<<endl<< "ESC) Выйти" <<endl;

key = getch();

}while(key != 27);

return;

}

void queuePrintPage(Queue\* queue){

char key;

do{

system("CLS");

if(queue!= NULL){

queuePrint(queue->head);

}else{

cout<< "Ничего не найдено" <<endl;

}

cout<<endl<< "ESC) Выйти" <<endl;

key = getch();

}while(key != 27);

return;

}

Queue\* searchPage(note\*\* db,int n){

char key;

Queue \*answer = NULL;

do{

system("CLS");

cout<< "Введите поисковой запрос:";

charlol[30];

cin>> lol;

cout<<endl;

answer = bSearch(db,n,lol);

queuePrint(answer->head);

cout<< "Нажмителюбуюклавишучтобыпродолжитьпоиск..." <<endl;

cout<< "ESC) Выйти" <<endl;

elem\_tree\* root = NULL;

key = getch();

}while(key != 27);

return answer;

}

void treeBypassLR(elem\_tree\* root){

cout.setf(ios::left);

if(root!=NULL){

treeBypassLR(root->left);

for(elem\_queue\* temp=root->data->head;temp!=NULL;temp = temp->next){

cout<<setw(30) << temp->elem\_db->contr\_fio<<setw(10) << temp->elem\_db->sum <<setw(11) << temp->elem\_db->date <<setw(22) << temp->elem\_db->adv\_fio<<endl;

}

treeBypassLR(root->right);

}

cout.unsetf(ios::left);

}

void LL\_rotate(elem\_tree \*& p){

elem\_tree\* q = p->left;

p->Bal = 0;

q->Bal = 0;

p->left = q->right;

q->right = p;

p = q;

}

void RR\_rotate(elem\_tree \*& p){

elem\_tree\* q = p->right;

p->Bal = 0;

q->Bal = 0;

p->right = q->left;

q->left = p;

p = q;

}

void LR\_rotate(elem\_tree \*& p){

elem\_tree\* q = p->left;

elem\_tree\* r = q->right;

if(r->Bal<0){

p->Bal = 1;

}else{

p->Bal = 0;

}

if(r->Bal>0){

q->Bal = -1;

}else{

q->Bal = 0;

}

r->Bal = 0;

q->right = r->left;

p->left = r->right;

r->left = q;

r->right = p;

p = r;

}

void RL\_rotate(elem\_tree \*& p){

elem\_tree\* q = p->right;

elem\_tree\* r = q->left;

if(r->Bal>0){

p->Bal = -1;

}else{

p->Bal = 0;

}

if(r->Bal<0){

q->Bal = 1;

}else{

q->Bal = 0;

}

r->Bal = 0;

q->left = r->right;

p->right = r->left;

r->right = q;

r->left = p;

p = r;

}

void addAVL(note\* D,elem\_tree \*&p){

if(p == NULL){

p = new elem\_tree;

p->data = new Queue;

p->data->head = NULL;

p->data->tail = (elem\_queue\*)&(p->data->head);

queuePush(p->data,D);

p->left = NULL;

p->right = NULL;

p->Bal = 0;

Rost = true;

}else if(noteDateCmp(p->data->head->elem\_db,D)==-1){

addAVL(D,p->left);

if(Rost){

if(p->Bal>0){

p->Bal = 0;

Rost = false;

}else if(p->Bal == 0){

p->Bal = -1;

Rost = true;

}else if(p->left->Bal<0){

LL\_rotate(p);

Rost = false;

}else{

LR\_rotate(p);

Rost = false;

}

}

}else if(noteDateCmp(p->data->head->elem\_db,D)==1){

addAVL(D,p->right);

if(Rost){

if(p->Bal<0){

p->Bal=0;

Rost = false;

}else if(p->Bal == 0){

p->Bal = 1;

Rost = true;

}else if(p->right->Bal>0){

RR\_rotate(p);

Rost = false;

}else{

RL\_rotate(p);

Rost = false;

}

}

}else{

queuePush(p->data,D);

}

}

elem\_tree\* createTree(Queue\* queue){

elem\_tree\* root = NULL;

Rost = true;

elem\_queue\* temp;

if(queue != NULL){

for(elem\_queue\* temp=queue->head;temp!=NULL;temp = temp->next){

addAVL(temp->elem\_db,root);

}

}

returnroot;

}

void printDbAll(note \*\*db,const int N){

system("CLS");

cout.setf(ios::left);

cout.width(70);

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

cout<<setw(6) << i+1 <<setw(30) << (\*db[i]).contr\_fio<<setw(10) << (\*db[i]).sum <<setw(11) << (\*db[i]).date <<setw(22) << (\*db[i]).adv\_fio<<endl;

}

char key = (char)NULL;

do{

cout<< "Нажмите ESC чтобывыйти";

cout<<"\b\b\b\b\b\b\b\b\b\b\b";

key = getch();

}while(key != 27);

cout.unsetf(ios::left);

}

int main(){

const int N = 4000;

note \*db = readDb((char\*)"db/testBase3.dat",N);

note \*\*point\_db = new note\*[N];

Queue\* search = NULL;

elem\_tree\* root = NULL;

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

point\_db[i]=&db[i];

}

char key = (char)NULL;

bool dbSorted = false;

bool avlCreated = false;

do{

system("CLS");

cout<< "Выберитеопцию..." <<endl;

cout<< "Отображение:" <<endl;

cout<< "1) Постраничное" <<endl;

cout<< "2) Всё" <<endl;

cout<<endl<< "Сортировка/Сброс:" <<endl;

cout<< "3) Сортировать" <<endl;

cout<< "4) Сбросить" <<endl;

if(dbSorted){

cout<<endl<< "Поиск (Первые три буквы фамилии):" <<endl;

cout<< "5) Поиск" <<endl;

cout<< "6) Вывести результаты последнего поискового запроса" <<endl;

cout<<endl<< "АВЛ дерево (Дата):" <<endl;

cout<< "7) Построить АВЛ дерево по результатам последнего поискового запроса" <<endl;

}

if(avlCreated){

cout<< "8) Вывести построенное АВЛ дерево" <<endl;

cout<< "9) Поиск в АВЛ дереве" <<endl;

}

cout<<endl<< "Нажмите ESC чтобы выйти" <<endl;

cout<<endl<< "Статусбар: ";

key = getch();

switch(key){

case 49:printDbPage(point\_db,N);break;

case 50:printDbAll(point\_db,N);break;

case 51:cout<< "Сортировка...";heapSort(point\_db,N);dbSorted=true;break;

case 52:cout<< "Сброс...";dbUnsort(point\_db,db,N);dbSorted=false;avlCreated=false;break;

}

if(dbSorted){

switch(key){

case 53:search = searchPage(point\_db,N);avlCreated=false;break;

case 54:queuePrintPage(search);break;

case 55:root = createTree(search);avlCreated=true;break;

}

}

if(avlCreated){

switch(key){

case 56:treePrintPage(root);break;

case 57:treeSearchPage(root);break;

}

}

}while(key != 27);

return 0;

}

***CURS\_WORK\_CODING.cpp***

#include <iostream>

#include <iomanip>

#include <algorithm>

#include <fstream>

#include <conio.h>

using namespace std;

struct treeElem {

treeElem \*right;

treeElem \*left;

short int symbol;

};

void QuickSortV2\_symbols(char \*Symbols, float \*Chances, int n, int L, int R) {

while (L < R) {

int mid = (L + R) / 2;

char x = Symbols[mid];

int i = L;

int j = R;

while (i <= j) {

while (Symbols[i] < x) {

i++;

}

while (Symbols[j] > x) {

j--;

}

if (i <= j) {

char temp = Symbols[i];

Symbols[i] = Symbols[j];

Symbols[j] = temp;

float temp\_2 = Chances[i];

Chances[i] = Chances[j];

Chances[j] = temp\_2;

i++;

j--;

}

}

if (j - L < R - i) {

QuickSortV2\_symbols(Symbols, Chances, n, L, j);

L = i;

}

else {

QuickSortV2\_symbols(Symbols, Chances, n, i, R);

R = j;

}

}

}

int BSearchAll2(char \*textSymbols, int L\_2, int R\_2, char X) {

int L = L\_2, R = R\_2, m;

bool find = false;

while (L < R) {

m = floor((float)(L + R) / (float)2);

if (textSymbols[m] < X) {

L = m + 1;

}

else {

R = m;

}

}

if (textSymbols[R] == X) {

find = true;

}

else {

find = false;

}

return R;

}

float enthropy(int uniqueSymbols, float \*textChances) {

float answer = 0;

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

answer += (textChances[i] \* log2(textChances[i]));

}

return -answer;

}

float averageWordLength(int uniqueSymbols, int \*lengthWords, float \*textChances) {

float answer = 0;

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

answer += lengthWords[i] \* textChances[i];

}

return answer;

}

int getFileSize(const char\* filename) {

ifstreamf(filename, ios::binary);

f.seekg(0, ios::end);

int fileSize = f.tellg();

return fileSize;

}

int getChances(int \*asciiChances, const char\* filename) {

ifstream in;

in.open(filename, ios::binary);

if (!in.is\_open())

{

cout<< "Неудалосьоткрытьфайл \"" <<filename<< "\""<<endl;

exit(1);

}

int uniqueSymbols = 0;

char ch;

while (!in.read((char\*)&ch, sizeof(ch)).eof()) {

if (asciiChances[(int)ch + 128] == 0) {

uniqueSymbols++;

}

asciiChances[(int)ch + 128]++;

}

return uniqueSymbols;

}

void codeGilbert(int uniqueSymbols, float \*textChances, int \*lengthWords, float \*qiMas, char \*\*codeTable) {

float pr = 0;

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

qiMas[i] = pr + textChances[i] / 2;

pr = pr + textChances[i];

lengthWords[i] = ceil(-log2(textChances[i])) + 1;

}

for (int i = 1, j = 0; i<= uniqueSymbols; i++, j = 0) {

for (; j <lengthWords[i]; j++) {

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

codeTable[i][j] = (char)(floor(qiMas[i]) + 48);

if (qiMas[i] >= 1) {

qiMas[i] = qiMas[i] - 1;

}

}

codeTable[i][j] = '\0';

}

};

void addTree(treeElem \*&root, char symbol, const char\* code, int k = 0) {

if (root == NULL) {

root = new treeElem;

root->left = NULL;

root->right = NULL;

root->symbol = -129;

}

if (code[k] == '\0') {

root->symbol = symbol;

return;

}

if (code[k] == '0') {

addTree(root->left, symbol, code, k + 1);

}

else if (code[k] == '1') {

addTree(root->right, symbol, code, k + 1);

}

}

void decodeFile(treeElem \*root, const char\* in\_filename, const char \*out\_filename) {

ofstream out;

ifstream in;

in.open(in\_filename, ios::binary);

out.open(out\_filename, ios::binary);

if (!out.is\_open()) {

cout<< "Неудалосьсоздатьфайл \"" <<out\_filename<< "\""<<endl;

return;

}

if(!in.is\_open()){

cout<< "Неудалосьоткрытьфайл \"" <<in\_filename<< "\""<<endl;

return;

}

char ch;

treeElem \*temp = root;

while (!in.read((char\*)&ch, sizeof(ch)).eof()) {

for (int i = 7; i >= 0; i--) {

if ((int)(temp->symbol) != -129) {

char temp\_c = temp->symbol;

out.write((char\*)&temp\_c,sizeof(temp\_c));

temp = root;

}

if ((ch&(1 <<i)) != 0) {

temp = temp->right;

}

else {

temp = temp->left;

}

if(temp == NULL){

cout<< "Что то не работает" <<endl;

return;

}

}

}

}

void encodeFile(int uniqueSymbols, char \*textSymbols, char \*\*codeTable, const char\* in\_filename, const char\* out\_filename) {

ofstream out;

ifstream in;

in.open(in\_filename, ios::binary);

out.open(out\_filename, ios::binary);

if (!out.is\_open()) {

cout<< "Неудалосьсоздатьфайл \"" <<out\_filename<< "\""<<endl;

return;

}

if(!in.is\_open()){

cout<< "Неудалосьоткрытьфайл \"" <<in\_filename<< "\""<<endl;

return;

}

char ch;

char currentCode = 0;

short int currentCode\_pos = 7;

while (!in.read((char\*)&ch, sizeof(ch)).eof()) {

int j = BSearchAll2(textSymbols, 1, uniqueSymbols, ch);

char \*code = codeTable[j];

for (int i = 0; code[i] != '\0'; i++) {

if (code[i] == '1') {

currentCode = (currentCode | (1 <<currentCode\_pos));

}

currentCode\_pos--;

if (currentCode\_pos< 0) {

out.write((char\*)&currentCode,sizeof(currentCode));

currentCode\_pos = 7;

currentCode = 0;

}

}

}

if (currentCode != 0) {

out.write((char\*)&currentCode,sizeof(currentCode));

}

}

float compressRatio(const char\* sourceFileName, const char\* encodedFileName){

return ((float)getFileSize(sourceFileName)/getFileSize(encodedFileName));

}

int main() {

char sourcefile[100];

cout<< "Введитеимяфайла: ";

cin>>sourcefile;

cout<<endl;

int fileSize = getFileSize(sourcefile);

cout<<fileSize<<endl;

int \*asciiChances = new int[256];

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

asciiChances[i] = 0;

int \*uniqueSymbols = new int;

\*uniqueSymbols = getChances(asciiChances,sourcefile);

char \*textSymbols = new char[\*uniqueSymbols + 1];

float \*textChances = new float[\*uniqueSymbols + 1];

int \*lengthWords = new int[\*uniqueSymbols + 1];

for (int i = 0; i<= \*uniqueSymbols; i++) {

lengthWords[i] = 0;

}

float \*qiMas = new float[\*uniqueSymbols + 1];

char \*\*codeTable = new char\*[\*uniqueSymbols + 1];

for (int i = 0; i<= \*uniqueSymbols; i++) {

codeTable[i] = new char[\*uniqueSymbols + 1];

}

for (int i = 0, j = 1; i < 256; i++) {

if (asciiChances[i] != 0) {

textSymbols[j] = i - 128;

textChances[j] = (float)asciiChances[i] / fileSize;

j++;

}

}

QuickSortV2\_symbols(textSymbols, textChances, \*uniqueSymbols, 1, \*uniqueSymbols);

codeGilbert(\*uniqueSymbols, textChances, lengthWords, qiMas, codeTable);

cout.setf(ios::left);

cout.width(70);

cout<<setw(10) << "Символ" <<setw(15) << "Вероятность" <<setw(20) << "Кодовое слово" <<setw(20) << "Длина кодового слова" <<endl;

for (int i = 1; i<= \*uniqueSymbols; i++) {

cout<<i<< ") " <<setw(10) << (int)textSymbols[i] <<setw(15) <<textChances[i] <<setw(20) <<codeTable[i] <<setw(20) <<lengthWords[i] <<endl;

}

cout<<endl<<setw(10) << "Энтропия" <<setw(15) << "LсрГилберт-Мур" <<endl;

cout<<setw(10) <<enthropy(\*uniqueSymbols, textChances) <<setw(15) <<averageWordLength(\*uniqueSymbols, lengthWords, textChances) <<endl<<endl;

treeElem \*root = NULL;

for (int i = 1; i<= \*uniqueSymbols; i++) {

addTree(root, textSymbols[i], codeTable[i]);

}

char key = (char)NULL;

do{

char filename1[100];

char filename2[100];

cout<< "Выберитеопцию:" <<endl;

cout<< "1) Закодировать кодом Гилберта-Мура" <<endl;

cout<< "2) Декодировать код Гилберта-Мура" <<endl;

cout<< "3) Вычислить коэффициент сжатия" <<endl<<endl;

cout<< "9) Очиститьэкран" <<endl;

key = getch();

if(key == 49){

cout<< "Введитеимяисходногофайла: ";

cin>> filename1;

cout<< "Введите имя файла с результатом: ";

cin>> filename2;

encodeFile(\*uniqueSymbols, textSymbols, codeTable, filename1, filename2);

}elseif (key == 50){

cout<< "Введите имя исходного файла: ";

cin>> filename1;

cout<< "Введите имя файла с результатом: ";

cin>> filename2;

decodeFile(root, filename1, filename2);

}else if(key == 51){

cout<< "Введите имя исходного файла: ";

cin>> filename1;

cout<< "Введите имя сжатого файла: ";

cin>> filename2;

cout<< "Коэффициентсжатия = " <<compressRatio(filename1,filename2) <<endl;

}else if (key == 57){

system("CLS");

}

cout<<endl;

}while(key != 27);

system("PAUSE");

return 0;

}

# **6. Результаты**

База данных по умолчанию

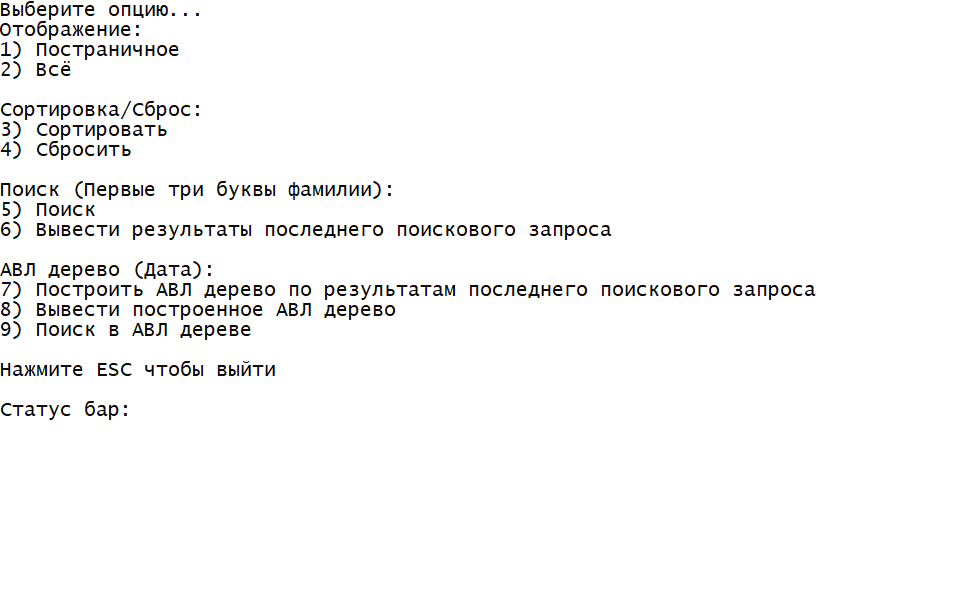
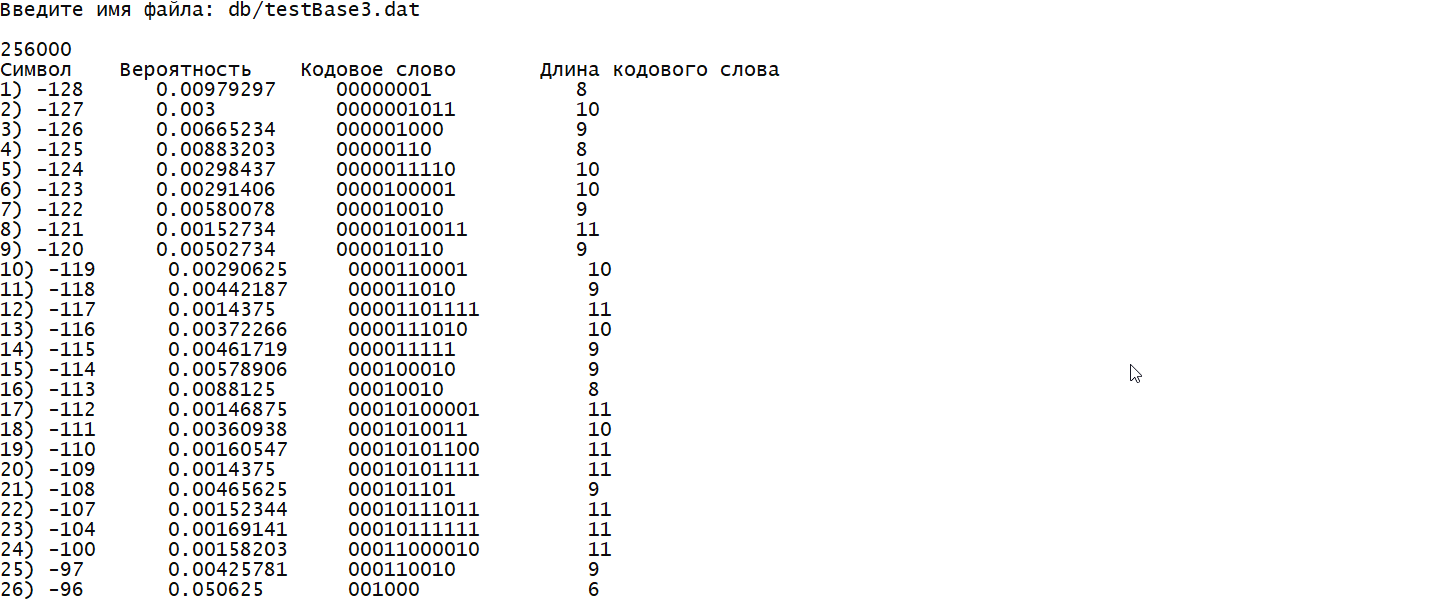
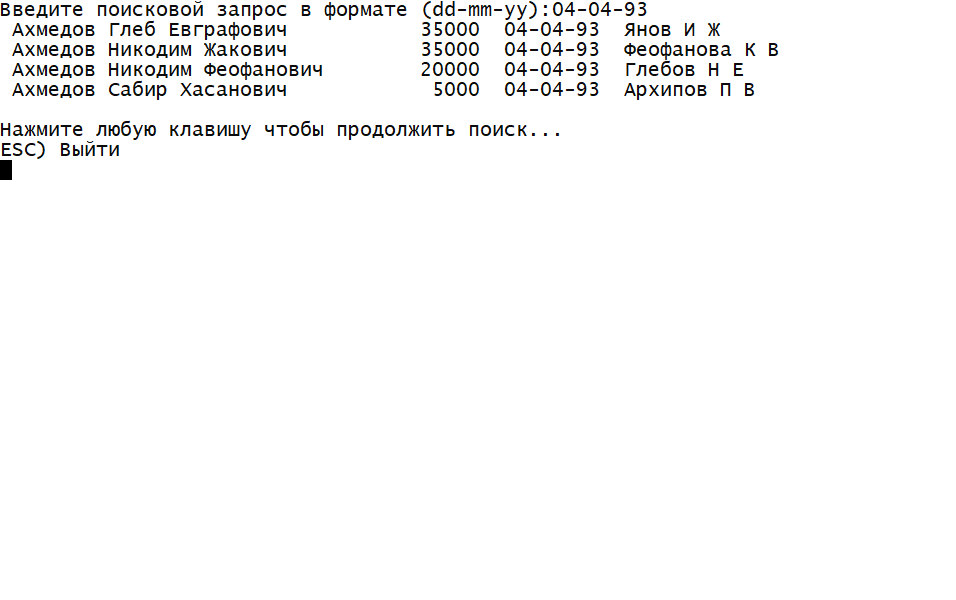
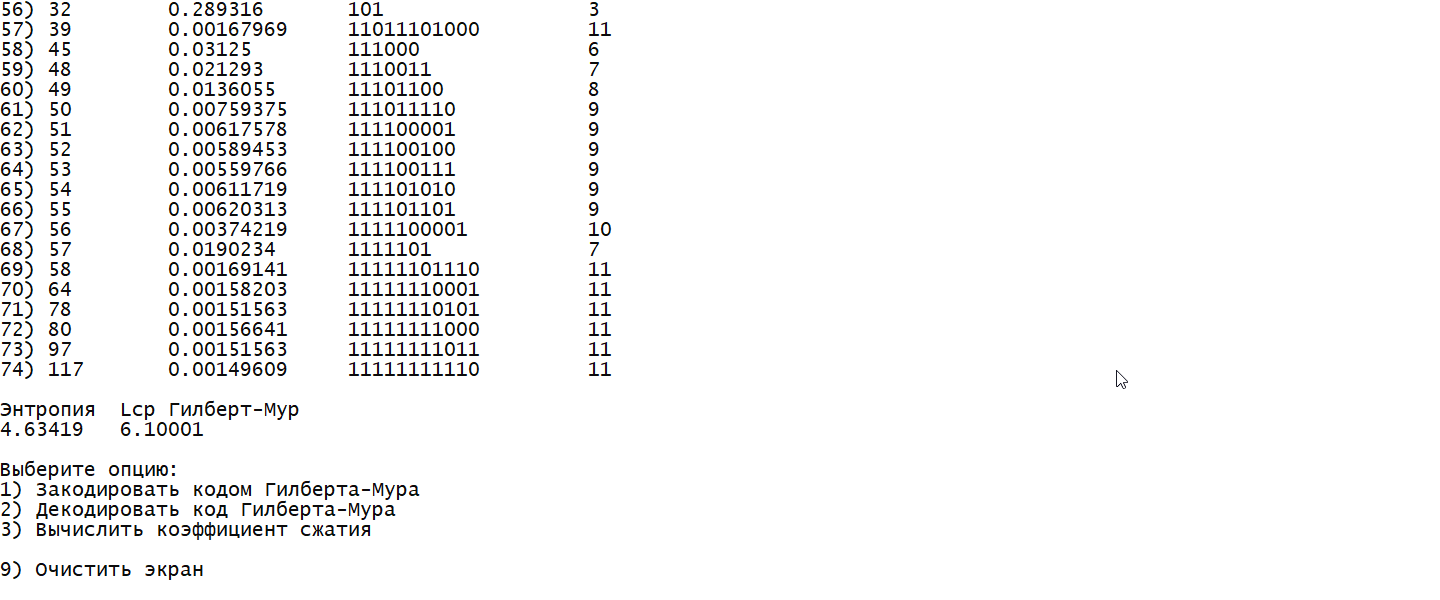
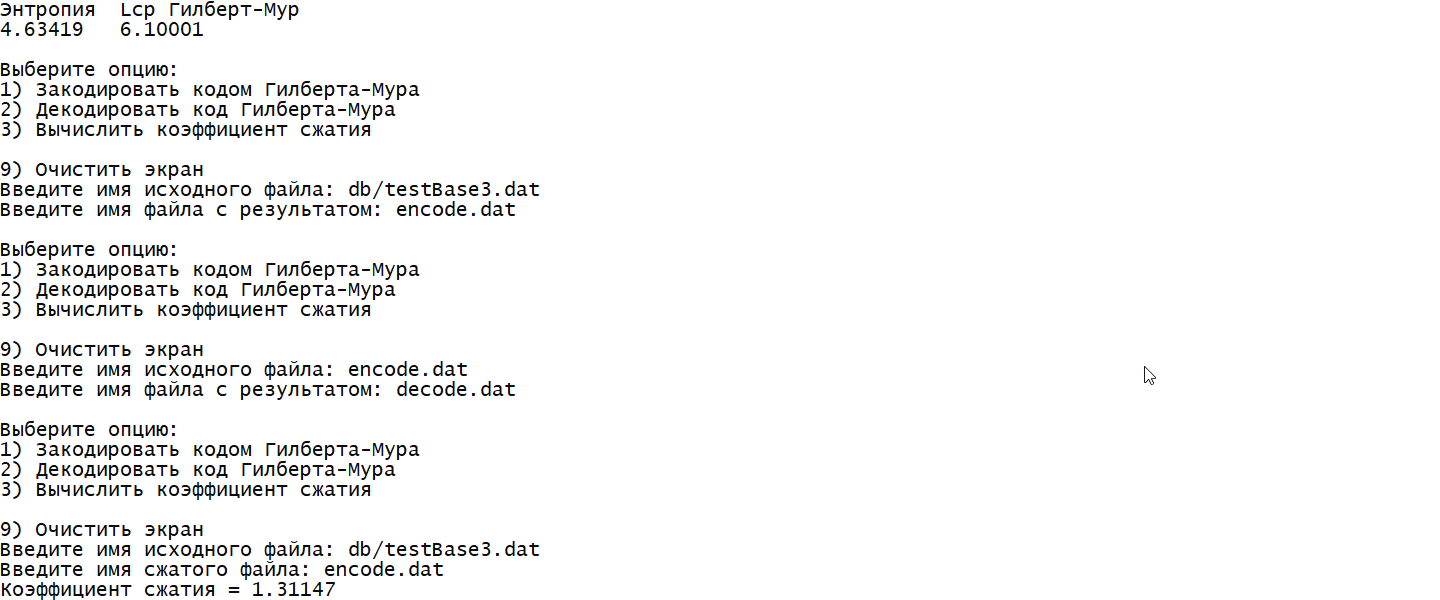
Стартовый экран

Окно вывода результата последнего поиска

Окно поиска

База данных после сортировки

Окно программы после сортировки



Окно кодирования файла (кодирование, декодирование, коэффициент сжатия)

Окно кодирования файла (энтропия и средняя длина)

Окно кодирования файла (таблица частот и кодов)

Окно поиска по дереву

Обход построенного дерева

Окно программы после построения дерева

# **7. Выводы**

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

А именно использование данной программы позволяет:

1. Вывести содержимое БД
2. Отсортировать записи методом Уильямса-Флойда по ФИО и сумме вклада
3. Выполнить двоичный поиск по первым трём буквам фамилии
4. Построить дерево поиска по ключу (АВЛ-дерево)
5. Выполнить поиск в дереве по дате вклада
6. Закодировать базу кодом Гилберта-Мура

По моему мнению, задача, поставленная в начале работы над курсовым проектом, была успешно выполнена.

Благодаря данному проекту я научился производить обработку объемных данных, а также, пройдя весь путь написания программы для курсового проекта, я познал все тонкости предоставленных алгоритмов.