Федеральное агентство связи

Сибирский государственный университет телекоммуникаций и информатики

Кафедра прикладной математики и кибернетики

Курсовой проект

«Структуры и алгоритмы обработки данных»

Вариант 44

Выполнил: студент группы ИП-715

Пляскина А.Ю.

Проверил: ассистент кафедры ПМиК

Турцев А.А.

Новосибирск 2018

Оглавление

[1. Постановка задачи 2](#_Toc533614154)

[2. Основные идеи и характеристики применяемых методов 3](#_Toc533614155)

[2.1. Метод сортировки 3](#_Toc533614156)

[2.2. Двоичный поиск 3](#_Toc533614157)

[2.3. Дерево и поиск по нему 4](#_Toc533614158)

[2.4. Метод кодирования 4](#_Toc533614159)

[3. Особенности реализации алгоритмов 6](#_Toc533614160)

[4. Описание программы 8](#_Toc533614161)

[4.1. Основные переменные и структуры 8](#_Toc533614162)

[4.2. Описание подпрограмм 9](#_Toc533614163)

[5. Текст программы 12](#_Toc533614164)

[6. Результаты 25](#_Toc533614165)

[7. Выводы 29](#_Toc533614166)

# Постановка задачи

Хранящуюся в файле библиогpафическую базу данных "Жизнь замечательных людей" загрузить динамически в оперативную память компьютера в виде списка, вывести на экран по 20 записей на странице с возможностью отказа от просмотра.

Упорядочить данные с помощью цифровой сортировки (Digital sort), построить по отсортированным данным индексный массив. Упорядоченные данные вывести на экран.

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

Из записей очереди построить дерево оптимального поиска по ключу (приближенный алгоритм А2) по году издания, вывести на экран содержимое дерева и предусмотреть возможность поиска в дереве по запросу.

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

Библиогpафическая база данных "Жизнь замечательных людей"

Стpуктуpа записи:

Автоp: текстовое поле 12 символов

фоpмат <Фамилия>\_<буква>\_<буква>

Заглавие: текстовое поле 32 символа

фоpмат <Имя>\_<Отчество>\_<Фамилия>

Издательство: текстовое поле 16 символов

Год издания: целое число

Кол-во стpаниц: целое число

Пpимеp записи из БД:

Кловский\_В\_Б

Лев\_Hиколаевич\_Толстой\_\_\_\_\_\_\_\_\_

Молодая\_гваpдия\_

1963

864

Ваpианты условий упоpядочения и ключи поиска (К):

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

Ключ в дереве – год издания

# Основные идеи и характеристики применяемых методов

## Метод сортировки

Цифровая сортировка (Digital Sort)

Цифровая сортировка является одним из методов сортировки последовательностей.

Пусть дана последовательность из S чисел, представленных в *m* – ичной системе 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.

## Двоичный поиск

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

Выбранный элемент равен X. Поиск завершён.

Выбранный элемент меньше X. Продолжаем поиск в правой половине массива.

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

Из-за необходимости найти все элементы соответствующие заданному ключу поиска в курсовой работе использовалась вторая версия двоичного поиска, которая из необходимых элементов находит самый левый, в результате чего для поиска остальных требуется просматривать лишь оставшуюся правую часть массива.

Верхняя оценка трудоёмкости алгоритма двоичного поиска такова. На каждой итерации поиска необходимо два сравнение для первой версии, одно сравнение для второй версии. Количество итераций не больше, чем f_02. Таким образом, трудоёмкость двоичного поиска в обоих случаях

f_03

## Дерево и поиск по нему

ДОП А2

До сих пор предполагалось, что частота обращения ко всем вершинам дерева поиска одинакова. Однако встречаются ситуации, когда известна информация о вероятностях обращения к отдельным ключам. Обычно для таких ситуаций характерно постоянство ключей, т.е. в дерево не включаются новые вершины и не исключаются старые и структура дерева остается неизменной. Эту ситуацию иллюстрирует сканер транслятора, который определяет, является ли каждое слово программы (идентификатор) служебным. Статистические измерения на сотнях транслируемых программ могут в этом случае дать точную информацию об относительных частотах появления в тексте отдельных ключей.

Припишем каждой вершине дерева Vi вес wi, пропорциональный частоте поиска этой вершины (например, если из каждых 100 операций поиска 15 операций приходятся на вершину V1, то w1=15). Сумма весов всех вершин дает вес дерева W. Каждая вершина Vi расположена на высоте hi, корень расположен на высоте 1. Высота вершины равна количеству операций сравнения, необходимых для поиска этой вершины. Определим средневзвешенную высоту дерева с n вершинами следующим образом: hср=(w1h1+w2h2+…+wnhn)/W. Дерево поиска, имеющее минимальную средневзвешенную высоту, называется деревом оптимального поиска (ДОП).

Второй приближённый 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 суммирования 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.

## Метод кодирования

Алфавитный код Гилберта – Мура

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

Пусть символы алфавита некоторым образом упорядочены, например, *a*1*≤a*2*≤…≤an*. Код  называется *алфавитным*, если кодовые слова лексикографически упорядочены, т.е. *.*

Процесс построения кода происходит следующим образом.

1. Вычислим величины *Qi, i=*1*,n*:

*Q*1*=p*1*/*2*,*

*Q*2*=p*1*+p2/*2*,*

*Q*3*=p*1*+p*2*+p*3*/*2*,*

…

*Qn=p*1*+p*2*+…+pn-*1*+pn/*2.

1. Представим суммы *Qi* в двоичном виде.
2. В качестве кодовых слов возьмем  младших бит в двоичном представлении *Qi*, .

**Пример**. Пусть дан алфавит A={*a*1*, a*2*, a*3*, a*4*, a*5*, a*6} с вероятностями *p*1=0.36, *p*2=0.18, *p*3=0.18, p4=0.12, *p*5=0.09, *p*6=0.07. Построенный код приведен в таблице 8.

Таблица 8 Код *Гилберта-Мура*

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

Средняя длина кодового слова не превышает значения энтропии плюс 2. Действительно,

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

# 3. Особенности реализации алгоритмов

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

1. *Интерфейс программы*

Интерфейс программы реализован в бесконечном цикле с возможностью выхода из него, при нажатии клавиши «Esc». Выбор пункта меню реализован с помощью функции switch( ). Кодирование реализовано в отдельном проекте.

1. *Загрузка и вывод базы данных*

База данных открывается в int main( ), там же предусмотрена проверка на наличие файла, из которого будет производиться считывание. Считывается база данных в список структур в функции void read\_base(FILE \*f, list \*l), где struct list – список, в котором хранится база данных. Считывание производится независимо от желания пользователя, в то время как большинство остальных функций он может выбрать посредствам меню. После считывания в список структур, файл закрывается.

За вывод элементов базы данных отвечает процедура void print\_base(list \*l), которая представляет возможность просмотра базы данных постранично. Процедура void print\_record(record \*l), где struct record – структура, в которой хранится один элемент базы данных, вызывается в процедуре void print\_base(list \*l) и служит для вывода структуры, в которой записан один элемент базы данных.

На странице находится двадцать элементов. Переключение между страницами осуществляется с помощью клавиш «<» и «>» на клавиатура. Прервать просмотр можно в любой момент с помощью клавиши «Esc».

1. *Вспомогательные функции и процедуры для сортировки данных*

База данных сортируется после запуска программы. Для сортировки базы данных используется процедура void DigitalSort(list \*&head, bool flag). Она сортирует список сначала по полю автора (flag = 0), а потом рекурсивно вызывает процедуру flag = 1. После завершения сортировки по публикации из нее же вызывается процедура void prior(list \*p), которая заполняет поля списка, ссылающиеся на предыдущий элемент, правильными данными. Для быстрого доступа к отсортированной и неотсортированной базе данных, перед вызовом процедуры void DigitalSort(list \*&head, bool flag), делается копия списка, хранящего элементы базы данных, с помощью процедуры void copy\_base(list \*a, list \*b).

Для просмотра отсортированной базы данных используется процедура void print\_base(list \*l).

1. *Особенности реализации бинарного поиска*

Бинарный поиск по отсортированной базе данных осуществляется в процедуре void binary\_search(char \*x, list \*l[N], queue \*&turn), где struct queue – очередь, доступ к записям ведется через индексный массив, созданный с помощью процедуры void masiving(list \*l, list \*r[N]). Сравнение массивов типа char осуществляется в функции int compare(char \*a, char \*b). Результатом работы процедуры void binary\_search(char \*x, list \*l[N], queue \*&turn) является очередь найденных элементов и номер первого и последнего элемента в индексном массиве, удовлетворяющие условию поиска. При отсутствии элементов с заданным ключом, программа выводит сообщение о том, что таких элементов нет.

Просмотреть найденную очередь можно с помощью функции void print\_stak(queue \*p).

1. *Вспомогательные функции и процедуры для построения дерева оптимального поиска (приближенный алгоритм А2).*

До построения дерева вызывается процедура void weight(list \*mas[N]), которая заполняет массив весов для найденных с помощью бинарного поиска элементов. Из нее вызывается функция void QuickSort(int \*a, list \*mas[N], int left, int right), сортирующая массив найденных элементов по возрастанию ключей, и соответсвенно преобразует массив весов. Построение дерева осуществляется в процедуре void DOP\_A2(vertex \*&q, list \*mas[N], int L, int R). Записи заносятся в дерево в процедуре void add(vertex \*&point, record \*x). Присутствует процедура и для удаления дерева void delete\_tree(vertex \*&p). Обход дерева слева направо осуществляется в процедуре void print\_tree(vertex \*&p). Поиск по дереву выполняется в процедуре void tree\_search(vertex \*&p, short int x).

1. *Кодирование данных*

Кодирование данных начинается с процедуры void reading( ), которая открывает файл базы данных для чтения, заполняет массив структур для алфавита кодовых слов всеми возможными символами, считывает из файла символы и считает их вероятности, закрывает файл, удаляет пустые символы, т.е. те, которые не встретились в файле и сортирует полученный алфавит по вероятностям. В процедуре int shanon( ) считается длина кодового слова и само кодовое слово. В процедуре void code\_file() создается бинарный файл, в который записывается закодированная база данных и считается ее размер и размер изначальной базы данных. В процедуре void print() осуществляется подсчет и вывод средней длины кодового слова, энтропии, и еще выводит символы, их вероятности, длины кодовых слов и сами кодовые слова на монитор, а так же подсчитывает и выводит коэффициент сжатия.

# Описание программы

## Основные переменные и структуры

struct record

{

char author[12];

char title[32];

char publisher[16];

short int year;

short int num\_of\_page;

};

Структура для хранения элемента базы данных. Всего таких элементов:

const int N = 4000;

struct list {

record \*data;

list \*next;

list \*prior;

};

Структура для создания списка, в котором хранятся элементы базы данных. Так же используется для хранения списка элементов с одинаковым ключом в вершине дерева.

list \*base – указатель на первый элемент списка, в котором хранятся указатели на элементы базы данных.

list \*sbase – указатель на первый элемент списка, в котором хранятся указатели на элементы отсортированной базы данных.

list \*mas\_sbase[N] – индексный массив на отсортированную базу данных.

struct queue {

list \*head;

list \*tail;

};

Структура, используемая для хранения результатов бинарного поиска.

queue \*stack – используется для хранения найденных элементов с одинаковым ключом в бинарном поиске.

int num\_h, num\_t – левая и правая граница, дающая представление о нахождение элементов стека, созданного после бинарного поиска, в массиве указателей.

struct vertex {

list \*fact;

vertex \*left;

vertex \*right;

};

Структура, представляющая дерево оптимального поиска (А2).

int \*w – вес элементов, из которых строится дерево.

В отдельном проекте реализовано кодирование:

struct code {

unsigned char a;

float q;

int w[30];

int l;

}

Структура, представляющая собой кодовое слово и всю необходимую информацию о нем.

code A[M] – кодовый алфавит

const int M = 256 – число слов в алфавите

long int n = 0 – количество элементов в итоговом алфавите, исключающем пустые символы

float entropy = (float)(0) - энтропия

float midlength = (float)(0) – средняя длина кодового слова

## Описание подпрограмм

Процедуры начальной обработки базы данных:

1. void read\_base(FILE \*f, list \*l) – считывает базу данных и создает список.

\*f – указатель на файл, из которого считываем базу данных, \*l – указатель на список, который представляет собой базу данных.

Процедуры сортировки:

1. void copy\_base(list \*a, list \*b) – делает копию списка, в котором хранится база данных.

\*a – указатель на копируемый список, \*b – указатель на список, в который копируем.

1. void prior(list \*p) – правильно заполняет поле ссылки на предыдущий элемент в списке.

\*p – указатель на список.

1. void DigitalSort(list \*&head, bool flag) – сортирует базу данных по ФИО авторов и публикации.

\*&head – ссылка указатель на список, который будем сортировать, flag – обозначает, по какому полю происходит сортировка.

Процедуры и функции для поиска в отсортированной базе данных:

1. void masiving(list \*l, list \*r[N]) – создает массив указателей на элементы списка.

\*l – указатель на список, по которому будем строить индексный массив, \*r[N] – массив указателей на элементы списка.

1. int compare(char \*a, char \*b) – сравнивает массивы символов.

\*a и \*b – указатели на массивы символов, которые сравниваем.

1. void binary\_search(char \*x, list \*l[N], queue \*&turn) – бинарный поиск.

\*x –массив символов, корой используем как ключ для поиска, \*l[N] –массив указателей, по которому проводим поиск, \*&turn – ссылка на указатель на первый элемент очереди.

Процедуры построения дерева оптимального поиска (А2):

1. void add(vertex \*&point, record \*x) – добавление элемента в дерево.

\*&point – ссылка на указатель на корень дерева, \*x – структура для добавления в дерево.

1. void DOP\_A2(vertex \*&q, list \*mas[N], int L, int R) – построение дерева оптимального поиска, приближенный алгоритм А2.

\*&q – ссылка на указатель на корень дерева, \*mas[N] – указатель на индексный массив, L и R – левая и правая границы.

1. void weight(list \*mas[N]) – создание массива весов.

\*mas[N] – указатель на индексный массив.

1. void QuickSort(int \*a, list \*mas[N], int left, int right) – сортировка массива указателей по возрастанию ключа (года).

\*a – указатель на массив весов, \*mas[N] – указатель на массив базы данных, left и right – левая и правая границы.

1. void tree\_search(vertex \*&p, short int x) – поиск в дереве.

\*&p – ссылка на указатель на корень дерева, x – ключ поиска (год).

1. void delete\_tree(vertex \*&p) – удаление дерева.

\*&p – ссылка на указатель на корень дерева

Процедуры и функции кодирования базы данных:

1. void reading() – считывание символов базы данных, подсчет их вероятностей и преобразование алфавита.
2. int shanon() – создание кодовых слов.
3. void code\_file() – кодирование базы данных.
4. void print() – вывод статистики и алфавита с вероятностями.

Вывод на экран:

1. void print\_base(list \*l) – осуществляет просмотр базы данных постранично.

\*l – указатель на cписок, который будем выводить на экран.

1. void print\_record(record \*t) – осуществляет вывод на экран элемента базы данных постранично.

\*t – указатель на структуру, в которой хранится один элемент списка.

1. void print\_stak(queue \*p) – вывод стека постранично.

\*p – указатель на стек.

1. void print\_tree(vertex \*&p) – вывод дерева.

\*&p – ссылка на указатель на вершину дерева.

Основная программа:

1. main( ) - основная программа, в которой выводится мен., а также в зависимости от выбранного пункта меню вызываются соответствующие процедуры и функции.

# Текст программы

#include <fstream>

#include <iostream>

#include <conio.h>

#include <Windows.h>

#include <iomanip>

#include <cstdio>

#include <cstring>

#include <cstdlib>

#include <cmath>

#include <iomanip>

#include <time.h>

list \*base, \*sbase;

queue \*stack;

int \*w;

int num\_h = 0, num\_t = 0;

code A[M];

long int n = 0;

float entropy = (float)(0);

float midlength = (float)(0);

int sim\_cf = 0, sim\_f = 0;

**const** int N = 4000;

**const** int M = 256;

**struct** record

{

char author[12];

char title[32];

char publisher[16];

short int year;

short int num\_of\_page;

};

**struct** list {

record \*data;

list \*next;

list \*prior;

};

**struct** queue {

list \*head;

list \*tail;

};

**struct** vertex {

list \*fact;

vertex \*left;

vertex \*right;

};

**struct** code {

unsigned char a;

float q;

int w[30];

int l;

};

void read\_base(FILE \*f, list \*l) {

record\* tmp = **new** record();

list\* tmpl;

fread((record \*)tmp, **sizeof**(record), 1, f);

l->data = tmp;

l->prior = NULL;

l->next = NULL;

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

tmp = **new** record();

fread((record \*)tmp, **sizeof**(record), 1, f);

tmpl = l;

l = l->next = **new** list();

l->data = tmp;

l->prior = tmpl;

l->next = NULL;

}

}

void copy\_base(list \*a, list \*b) {

b->prior = NULL;

b->data = a->data;

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

a = a->next;

b->next = **new** list;

b->next->prior = b;

b = b->next;

b->data = a->data;

}

b->next = NULL;

}

void prior(list \*p) {

p->prior = NULL;

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

p->next->prior = p;

p = p->next;

}

}

void DigitalSort(list \*&head, bool flag) {

queue Q[257];

**for** (int i = 0; i < 257; i++) {

Q[i].tail = (list\*)&(Q[i].head);

}

list \*p;

SetConsoleCP(1251);

**for** (int j = 10; j >= 0; j--) {

**for** (int i = 0; i < 257; i++) {

Q[i].tail = Q[i].head = NULL;

}

**while** (head) {

int d;

**if** (flag == 0) {

**if** (head->data->author[j] == ' ')

d = 0;

**else**

d = head->data->author[j] + 129;

}

**else** {

**if** (head->data->publisher[j] == ' ')

d = 0;

**else**

d = head->data->publisher[j] + 129;

}

p = Q[d].tail;

**if** (Q[d].head == NULL)

Q[d].head = head;

**else**

p->next = head;

p = Q[d].tail = head;

head = head->next;

p->next = NULL;

}

cout << endl << endl;

head = NULL;

int i;

**for** (i = 0; i < 256; i++) {

**if** (Q[i].head != NULL)

**break**;

}

head = Q[i].head;

p = Q[i].tail;

**for** (int k = i + 1; k < 256; k++) {

**if** (Q[k].head != NULL) {

p->next = Q[k].head;

p = Q[k].tail;

}

}

}

**if** (flag == 0)

DigitalSort(head, 1);

**else**

prior(head);

}

void masiving(list \*l, list \*r[N]) {

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

r[i] = l;

l = l->next;

}

}

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

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

**if** ((int)a[i] < (int)b[i]) **return** 1;

**else**

**if** ((int)a[i] > (int)b[i]) **return** -1;

}

**return** 0;

}

void binary\_search(char \*x, list \*l[N], queue \*&turn) {

int L = 1;

int R = N;

int m, h, t;

int sravn;

turn = **new** queue();

turn->head = turn->tail;

**while** (L < R) {

m = (L + R) / 2;

sravn = compare(l[m]->data->publisher, x);

**if** (sravn == 1) {

L = m + 1;

}

**else** {

**if** (sravn == -1) {

R = m;

}

**else**

**if** (sravn == 0) {

h = m;

t = m;

**while** (compare(l[h - 1]->data->publisher, x) == 0) {

h--;

**if** (h - 1 == 0)

**break**;

}

**while** (compare(l[t + 1]->data->publisher, x) == 0) {

t++;

**if** (t + 1 == 4000)

**break**;

}

turn->head = l[h];

num\_h = h;

turn->tail = l[t];

num\_t = t;

**return**;

}

}

}

}

void add(vertex \*&point, record \*x) {

vertex \*\*p = &point;

list \*tmp;

**while** (\*p) {

**if** ((\*x).year < (\*p)->fact->data->year) {

p = &((\*p)->left);

}

**else**

**if** ((\*x).year > (\*p)->fact->data->year) {

p = &((\*p)->right);

}

**else**

**if** ((\*x).year == (\*p)->fact->data->year) {

tmp = (\*p)->fact;

**while** (tmp->next != NULL) {

tmp = tmp->next;

}

tmp->next = **new** list();

tmp->next->data = x;

tmp->next->prior = (\*p)->fact;

tmp->next->next = NULL;

**return**;

}

}

**if** ((\*p) == NULL) {

(\*p) = **new** vertex();

(\*p)->fact = **new** list();

(\*p)->fact->data = x;

(\*p)->fact->next = NULL;

(\*p)->fact->prior = NULL;

(\*p)->left = NULL;

(\*p)->right = NULL;

}

}

void DOP\_A2(vertex \*&q, list \*mas[N], int L, int R) {

int wes = 0, summa = 0;

int i;

L = L - num\_h;

R = R - num\_h;

**if** (L <= R)

{

**for** (i = L; i <= R; i++)

wes = wes + w[i];

**for** (i = L; i <= R; i++)

{

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

**break**;

summa = summa + w[i];

}

add(q, mas[i + num\_h]->data);

DOP\_A2(q, mas, L + num\_h, i - 1 + num\_h);

DOP\_A2(q, mas, i + 1 + num\_h, R + num\_h);

}

}

void weight(list \*mas[N]) {

w = **new** int[num\_t - num\_h + 1];

**for** (int i = 0; i < (num\_t - num\_h + 1); i++) {

w[i] = rand() % 999 + 1;

}

QuickSort(w, mas, 0, num\_t - num\_h);

}

void QuickSort(int \*a, list \*mas[N], int left, int right) {

int current, i, j, t;

list \*tmp;

current = mas[left + num\_h]->data->year;

i = left;

j = right;

**while** (i <= j) {

**while** (mas[i + num\_h]->data->year < current) {

i++;

}

**while** (mas[j + num\_h]->data->year > current) {

j--;

}

**if** (i <= j) {

t = a[i];

a[i] = a[j];

a[j] = t;

tmp = mas[i + num\_h];

mas[i + num\_h] = mas[j + num\_h];

mas[j + num\_h] = tmp;

i++; j--;

}

}

**if** (left < j) {

QuickSort(a, mas, left, j);

}

**if** (i < right) {

QuickSort(a, mas, i, right);

}

}

void tree\_search(vertex \*&p, short int x) {

vertex \*tmp = p;

**while** (tmp != NULL) {

**if** (tmp->fact->data->year == x) {

print\_base(tmp->fact);

**return**;

}

**else**

**if** (tmp->fact->data->year > x)

tmp = tmp->left;

**else**

**if** (tmp->fact->data->year < x)

tmp = tmp->right;

}

cout << "Книга не найдена" << endl;

}

void delete\_tree(vertex \*&p) {

**if** (p != NULL) {

delete\_tree(p->left);

delete\_tree(p->right);

**delete** p;

}

}

void reading() {

int i, j = 0, e;

FILE \*f;

errno\_t err;

err = fopen\_s(&f, "Base1.dat", "rb");

**if** (err != 0)

{

cout << "Ошибка: невозможно открыть файл" << endl;

exit(1);

}

**for** (i = 0; i < M; i++) {

A[i].a = i;

A[i].q = (float)(0);

A[i].l = 0;

}

int flag = 100;

unsigned char ac;

**while** (!feof(f)) {

fscanf\_s(f, "%c", &ac);

A[ac].q = A[ac].q + 1.0;

n++;

}

fclose(f);

**for** (i = 0; i < M; i++)

**if** (A[i].q != (float)(0)) {

A[i].q /= (float)n;

j++;

}

n = (unsigned long int)(j);

code tmp;

**for** (i = 1; i < M - 1; i++) {

e = 1;

**if** (A[i].q == (float)(0))

**for** (j = i + 1; j < M; j++)

**if** (e)

**if** (A[j].q != (float)(0)) {

tmp = A[i];

A[i] = A[j];

A[j] = tmp;

e = 0;

}

}

i = 0;

**while** (A[i].q != 0.0) i++;

n = (unsigned long int)(i);

**for** (i = 0; i < n; ++i)

**for** (j = 1; j < n; ++j) {

**if** (A[j].q > A[j - 1].q) {

tmp = A[j - 1];

A[j - 1] = A[j];

A[j] = tmp;

}

}

}

int shanon() {

float q[M];

q[0] = 0;

**for** (int i = 1; i < n; i++)

{

q[i] = q[i - 1] + A[i].q;

A[i].l = ceil(-log(A[i].q) / log(2));

}

**for** (int i = 1; i < n; i++)

{

**for** (int j = 1; j <= A[i].l; j++)

{

q[i - 1] \*= 2;

A[i].w[j] = floor(q[i - 1]);

**if** (q[i - 1] > 1) q[i - 1] -= 1;

}

}

**return** 0;

}

void code\_file() {

FILE \*f, \*cf;

errno\_t err;

err = fopen\_s(&f, "Base1.dat", "rb");

**if** (err != 0)

{

SetConsoleCP(866);

cout << "Ошибка: невозможно открыть файл" << endl;

exit(1);

}

err = fopen\_s(&cf, "CodeBase.dat", "wb");

char sim;

**while** (!feof(f)) {

fscanf\_s(f, "%c", &sim);

sim\_f++;

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

**if** (sim == A[i].a) {

**for** (int j = 0; j < A[i].l; j++) {

putc(A[i].w[j], cf);

sim\_cf++;

}

}

}

}

\_fcloseall();

}

void print() {

entropy = (float)(0);

midlength = (float)(0);

int j;

int i;

float check = 0;

**for** (i = 1; i < n; i++) {

entropy += -A[i].q \* log(A[i].q) / log(2);

midlength += A[i].l \* A[i].q;

}

**for** (i = 1; i < n; i++) {

cout << setw(1) << (char)A[i].a << setw(3) << " ";

cout << (float)A[i].q << " " << setw(2) << (int)A[i].l << " ";

**for** (j = 1; j <= A[i].l; ++j)

cout << A[i].w[j];

cout << endl;

}

SetConsoleCP(866);

cout << endl;

cout << setw(8) << "Средняя длина: " << midlength << endl;

cout << setw(8) << "Энтропия: " << entropy << endl;

cout << setw(8) << entropy + 1 << " > " << midlength << endl << endl;

cout << setw(8) << "Коэффициент сжатия: " << (float) sim\_f / sim\_cf << endl;

\_getch();

}

void print\_record(record \*t) {

SetConsoleCP(1251);

**for** (int j = 0; j < 12; j++)

cout << t->author[j];

cout << "|";

**for** (int j = 0; j < 32; j++)

cout << t->title[j];

cout << "|";

**for** (int j = 0; j < 16; j++)

cout << t->publisher[j];

cout << "| ";

cout << t->year;

cout << " | ";

cout << t->num\_of\_page << " |" << endl;

}

void print\_base(list \*l) {

int M = 20;

int i, flag = 1, num = 1;

int c = 0;

**while** (flag) {

i = 0;

system("cls");

cout << "+-----+------------+--------------------------------+----------------+------+-----+" << endl;

**while** ((i < M) && (num <= N) && (l != NULL)) {

cout << "|";

cout << setw(4) << num << ")|";

print\_record(l->data);

cout << "+-----+------------+--------------------------------+----------------+------+-----+" << endl;

num++;

i++;

**if** (l->next == NULL)

**break**;

**else**

l = l->next;

}

SetConsoleCP(866);

c = 0;

**while** ((c != 161) && (c != 27) && (c != 238)) {

**if** (\_kbhit()) {*///Пользоавтель нажал на кнопку = 1*

c = \_getch();

}

}

**if** (c == 161) {

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

**if** (l->prior == NULL) **break**;

l = l->prior;

num--;

}

}

**else**

**if** (c == 27) {

flag = 0;

**return**;

}

}

}

void print\_stak(queue \*p) {

list \*l = p->head;

int M = 20;

int i, flag = 1, num = 1;

int c = 0;

**while** (flag) {

i = 0;

system("cls");

cout << "+-----+------------+--------------------------------+----------------+------+-----+" << endl;

**while** ((i < M) && (p->tail != l)) {

cout << "|";

cout << setw(4) << num << ")|";

print\_record(l->data);

cout << "+-----+------------+--------------------------------+----------------+------+-----+" << endl;

num++;

i++;

**if** ((l->next == NULL) || (p->tail == l->next)) {

num--;

**break**;

}

**else**

l = l->next;

}

SetConsoleCP(866);

c = 0;

**while** ((c != 161) && (c != 27) && (c != 238)) {

**if** (\_kbhit()) {

c = \_getch();

}

}

**if** ((p->tail == l->next) && (c == 238)) {

**for** (int j = 0; j < i - 1; j++) {

**if** (l->prior == NULL) **break**;

l = l->prior;

num--;

}

}

**if** (c == 161) {

**for** (i = 0; i < 40; i++) {

**if** ((l->prior == NULL) || (p->head == l)) **break**;

l = l->prior;

num--;

}

}

**else**

**if** (c == 27) {

flag = 0;

**return**;

}

}

}

void print\_tree(vertex \*&p) {

**if** (p == NULL) **return**;

list \*tmp = p->fact;

print\_tree(p->left);

tmp = p->fact;

**while** (tmp != NULL) {

print\_record(tmp->data);

tmp = tmp->next;

}

cout << "---------------------------------------------------------------------------" << endl;

print\_tree(p->right);

}

int main()

{

setlocale(LC\_ALL, "Russian");

SetConsoleCP(866);

FILE \*Base1;

errno\_t err;

err = fopen\_s(&Base1, "Base1.dat", "rb");

**if** (err != 0)

{

cout << "Ошибка: невозможно открыть файл" << endl;

exit(1);

}

base = **new** list();

read\_base(Base1, base);

\_fcloseall();

sbase = **new** list();

copy\_base(base, sbase);

DigitalSort(sbase, 0);

list \*mas\_sbase[N] = {};

vertex \*root = NULL;

SetConsoleCP(1251);

reading();

shanon();

code\_file();

int enter = 0;

**while** (true) {

system("CLS");

enter = 0;

SetConsoleCP(866);

cout << "**\t**------------------------------МЕНЮ------------------------" << endl;

cout << "**\t**1. Посмотреть базу данных (< - назад, esc - выйти)" << endl;

cout << "**\t**2. Просмотр отсортированной базы данных (Digital sort)" << endl;

cout << "**\t**3. Поиск в отсортированной базе по ключу (публикация)" << endl;

cout << "**\t**4. Поиск в дереве (год)" << endl;

cout << "**\t**5. Кодирование статистика" << endl;

cout << "**\t**Esc. Выход " << endl;

cout << "**\t**----------------------------------------------------------" << endl;

**while** ((enter != 27) && (enter != 49) && (enter != 50) && (enter != 51) && (enter != 52) && (enter != 53)) {

**if** (\_kbhit()) {

enter = \_getch();

}

}

**switch** (enter) {

**case** 27: **return** 0;

**case** 49:

print\_base(base);

**break**;

**case** 50:

print\_base(sbase);

**break**;

**case** 51:

stack = NULL;

root = NULL;

char X[3];

SetConsoleCP(866);

cout << "Введите ключ (три буквы публикации): ";

cin >> X;

masiving(sbase, mas\_sbase);

binary\_search(X, mas\_sbase, stack);

**if** (stack->head != stack->tail)

print\_stak(stack);

**else**

cout << "Такой записи нет";

\_getch();

**break**;

**case** 52:

**if** (mas\_sbase[0] == NULL) **break**;

**if** (root != NULL) {

delete\_tree(root);

root = NULL;

}

short int x;

weight(mas\_sbase);

DOP\_A2(root, mas\_sbase, num\_h, num\_t);

cout << "Посмотреть дерево (1/0) ";

cin >> x;

**if** (x == 1) {

print\_tree(root);

}

**while** (true) {

SetConsoleCP(866);

cout << "Книги какого года хотите увидеть? (ext = 1)" << endl;

cin >> x;

**if** (x == 1) **break**;

tree\_search(root, x);

}

**break**;

**case** 53:

print();

**break**;

}

}

**return** 0;

}

# Результаты

Рисунок 1. Неотсортированная база данных.

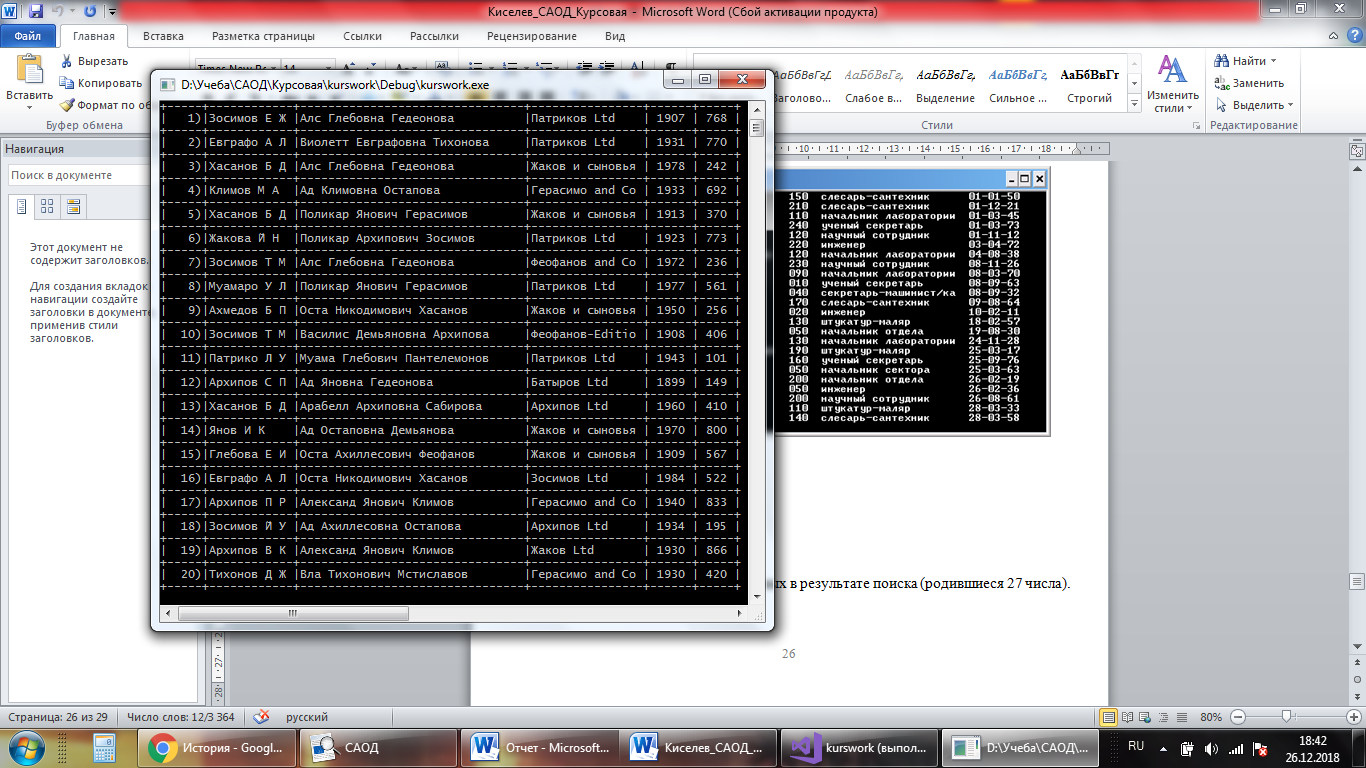


Рисунок 2. Отсортированная автору и публикации база данных.

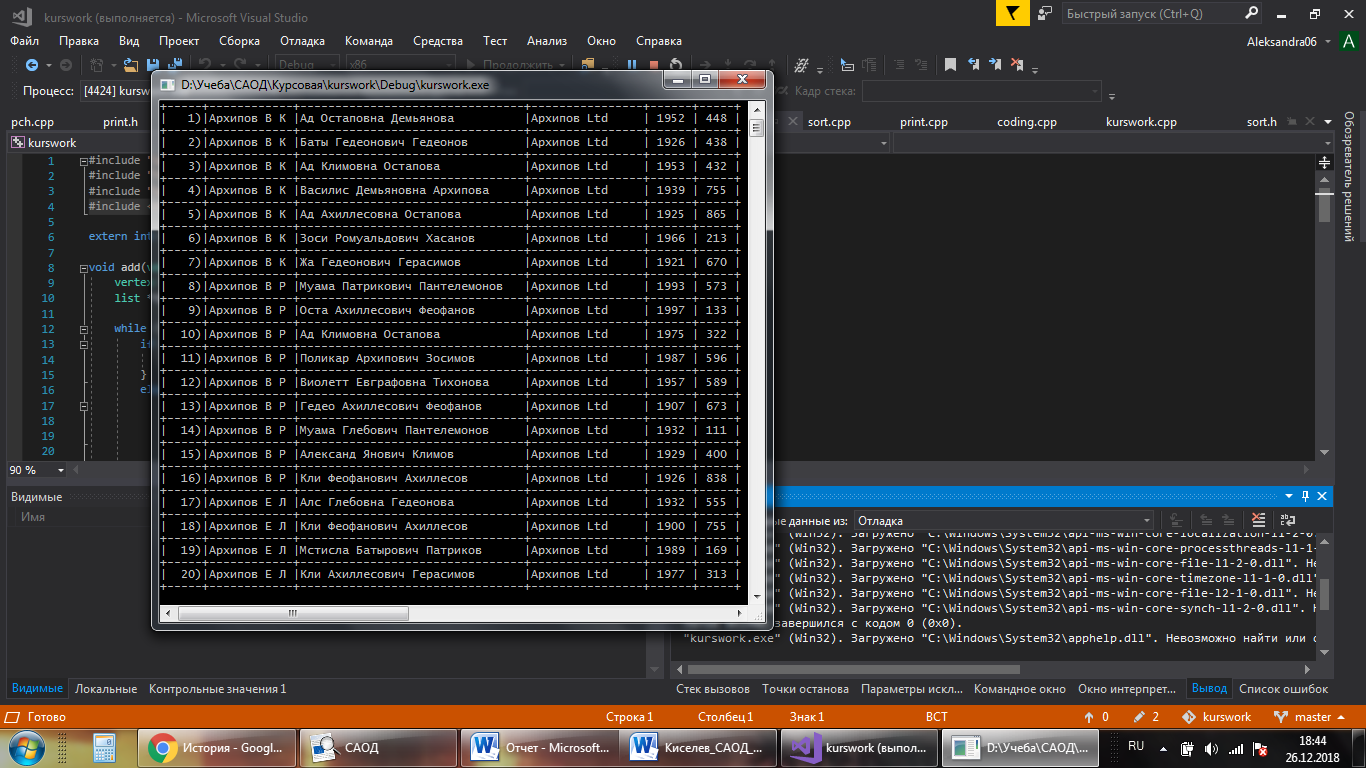


Рисунок 3. Очередь из записей, полученных в результате по первым трем буквам публикации (Зос).

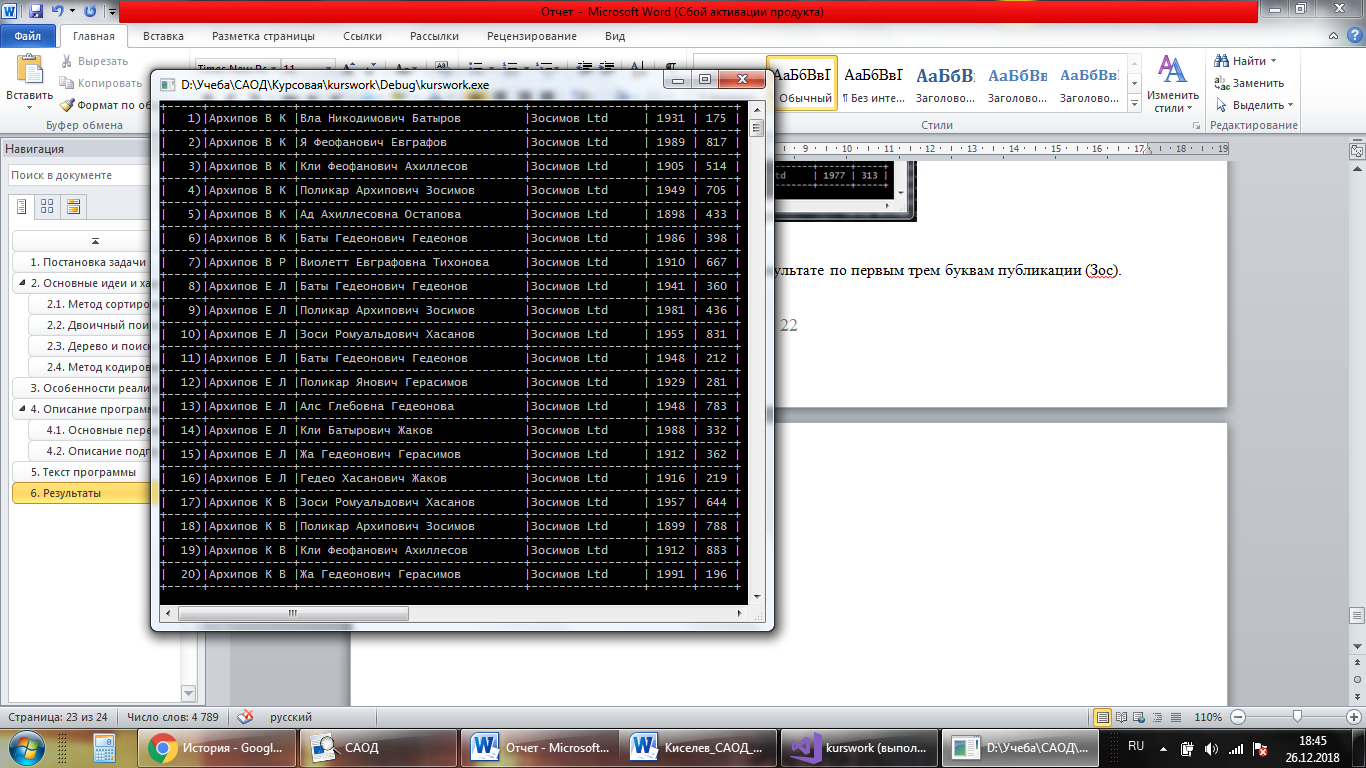


Рисунок 4. Дерево, ключ в дереве – год.

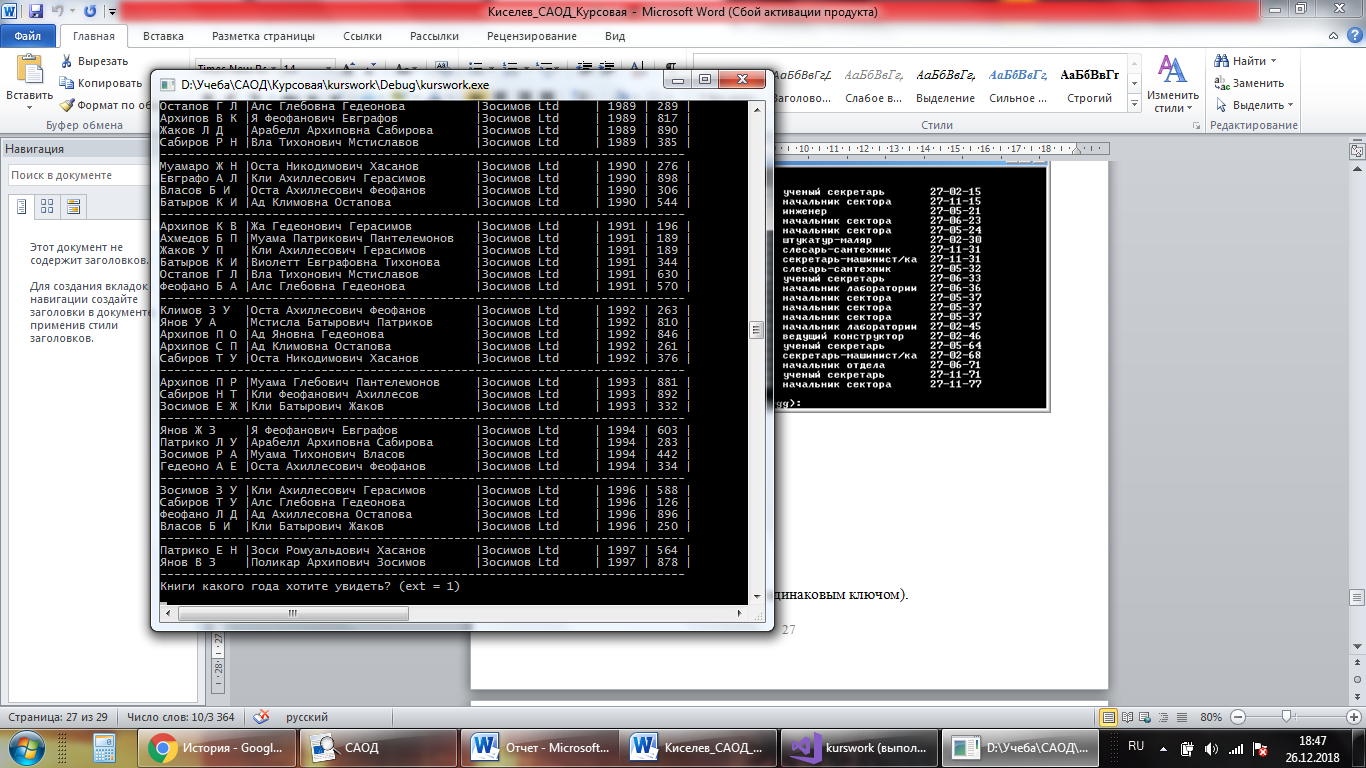


Рисунок 5. Поиск по дереву (элементы с одинаковым ключом).

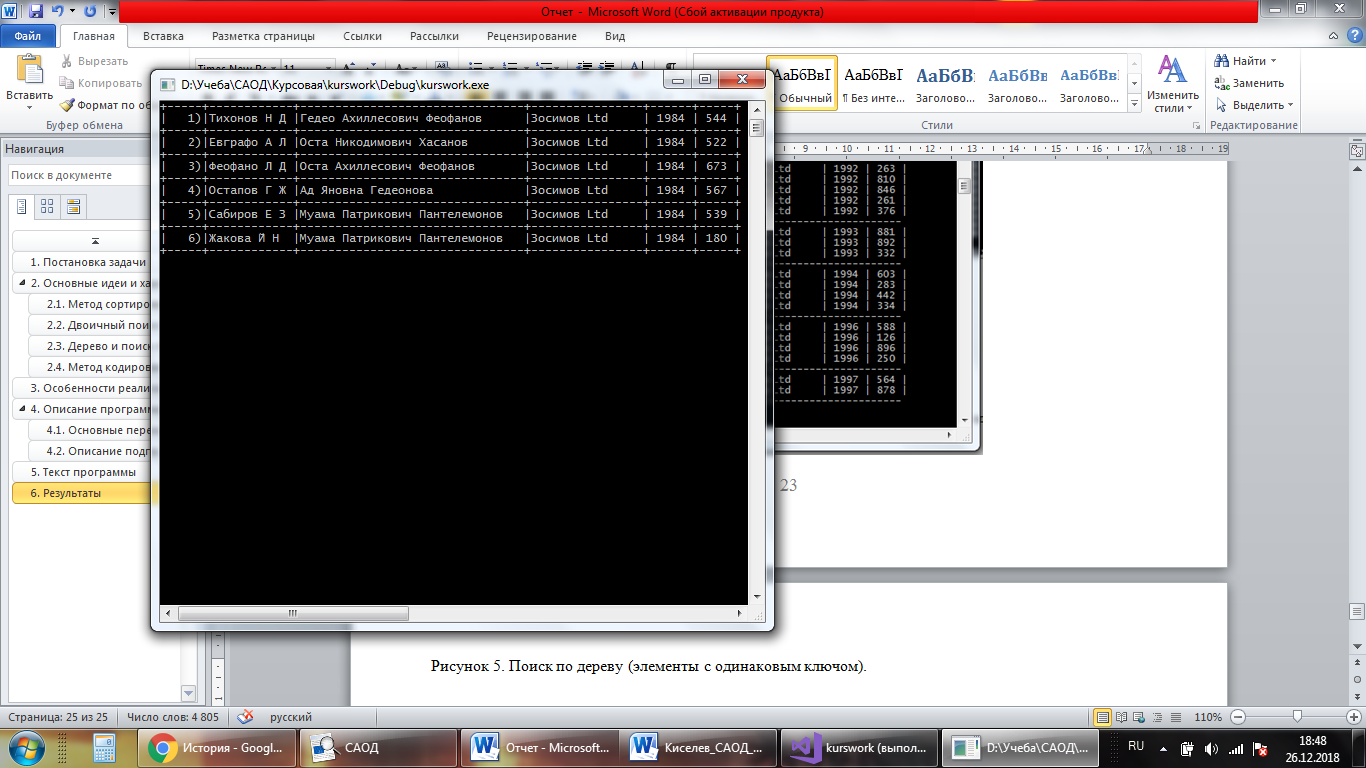
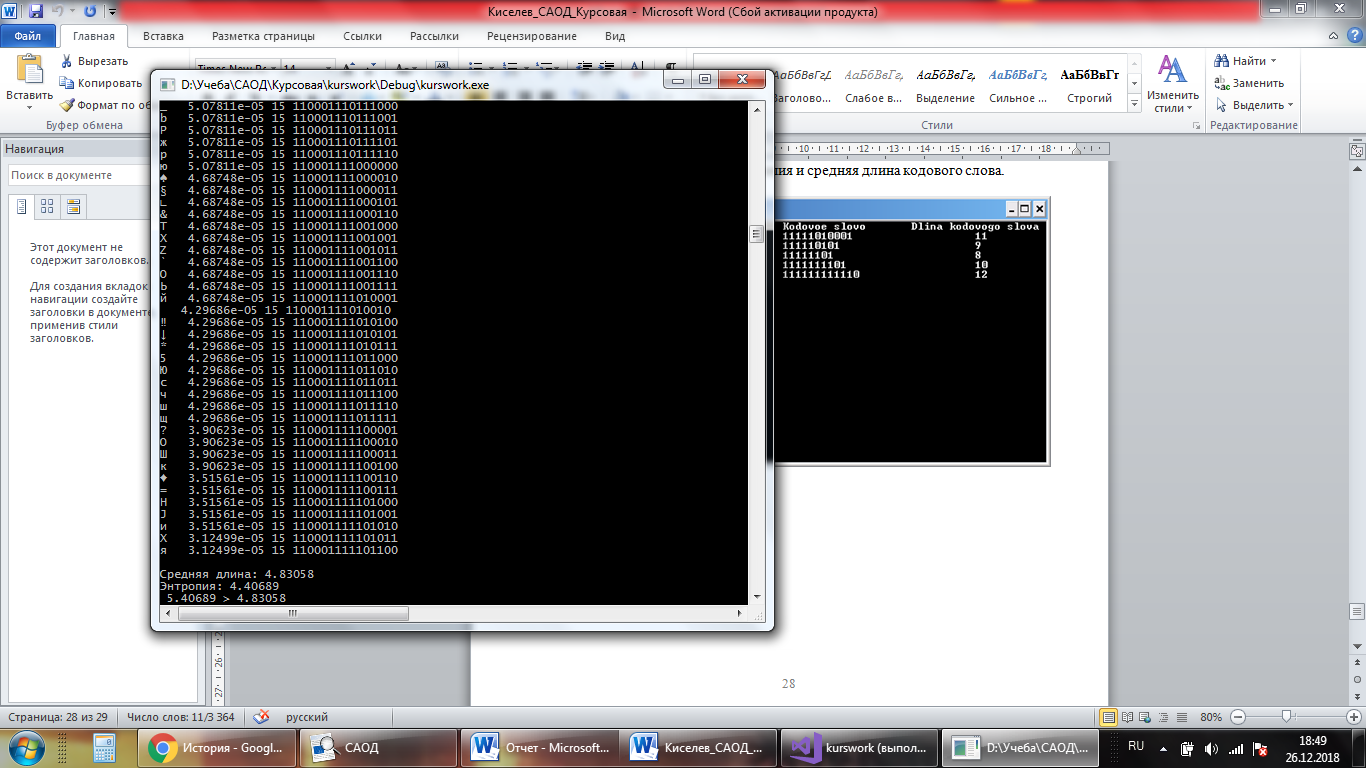


Рисунок 6. Кодовые слова, энтропия и средняя длина кодового слова.



# Выводы

В ходе выполнения курсовой работы были выполнены все поставленные задачи и реализованы необходимые алгоритмы: сортировки, поиска, построения дерева оптимального поиска (приближённый алгоритм А2), 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.