

Федеральное агентство связи
Сибирский государственный университет телекоммуникаций и информатики

Кафедра прикладной математики и кибернетики

Курсовой проект
по курсу
«Структуры и алгоритмы обработки данных»
Вариант

Выполнил: студент группы

Проверил: доцент кафедры ПМиК
Янченко Е.В.

Новосибирск 2018

Содержание

1. ПОСТАНОВКА ЗАДАЧИ	3
2. ОСНОВНЫЕ ИДЕИ И ХАРАКТЕРИСТИКИ ПРИМЕНЯЕМЫХ МЕТОДОВ.....	4
2.1. МЕТОД СОРТИРОВКИ.....	4
2.2 ДВОИЧНЫЙ ПОИСК	4
2.3 ДЕРЕВО И ПОИСК ПО ДЕРЕВУ	5
3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ АЛГОРИТМОВ	6
4. ОПИСАНИЕ ПРОГРАММЫ.....	7
4.1. ОСНОВНЫЕ ПЕРЕМЕННЫЕ И СТРУКТУРЫ.....	7
4.2. ОПИСАНИЕ ПОДПРОГРАММ.....	8
5. ТЕКСТ ПРОГРАММЫ	9
6. РЕЗУЛЬТАТЫ	11
7. ВЫВОДЫ	15

1. ПОСТАНОВКА ЗАДАЧИ

Хранящуюся в файле базу данных загрузить в оперативную память компьютера и построить индексный массив, упорядочивающий данные **по ФИО и названию улицы**, используя **метод Хоара** в качестве метода сортировки.

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

Из записей очереди построить **Двоичное Б-дерево по номеру квартиры**, 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.

Структура записи:

ФИО гражданина: текстовое поле 32 символа

формат <Фамилия>_<Имя>_<Отчество>

Название улицы: текстовое поле 18 символов

Номер дома: целое число

Номер квартиры: целое число

Дата поселения: текстовое поле 10 символов

формат дд-мм-гг

Пример записи из БД:

Петров_Иван_Федорович_____

Ленина_____

10

67

29-02-65

Варианты условий упорядочения и ключи поиска (К):

С = 1 - по ФИО и названию улицы, К = первые три буквы фамилии;

2. ОСНОВНЫЕ ИДЕИ И ХАРАКТЕРИСТИКИ ПРИМЕНЯЕМЫХ МЕТОДОВ

2.1. МЕТОД СОРТИРОВКИ

Метод Хоара

Метод Хоара или метод быстрой сортировки заключается в следующем. Возьмём произвольный элемент массива x . Просматривая массив слева, найдём элемент $a_i \geq x$. Просматривая массив справа, найдём $a_j \leq x$. Поменяем местами a_i и a_j . Будем продолжать процесс просмотра и обмена, до тех пор пока i не станет больше j . Тогда массив можно разбить на две части: в левой части все элементы не больше x , в правой части массива не меньше x . Затем к каждой части массива применяется тот же алгоритм.

Очевидно, трудоёмкость метода существенно зависит от выбора элемента x , который влияет на разделение массива. Максимальные значения M и C для метода быстрой сортировки достигаются при сортировке упорядоченных массивов (в прямом и обратном порядке). Тогда в этом случае в одной части остаётся только один элемент (минимальный или максимальный), а во второй – все остальные элементы. Выражения для M и C имеют следующий вид

$$M=3(n-1), C=(n^2+5n+4)/2$$

Таким образом, в случае упорядоченных массивов трудоёмкость сортировки имеет квадратичный порядок.

Элемент a_m называется *медианой* для элементов $a_L \dots a_R$, если количество элементов меньших a_m равно количеству элементов больших a_m с точностью до одного элемента (если количество элементов нечётно). В примере буква К- медиана для КУРАПОВАЕ.

Минимальная трудоёмкость метода Хоара достигается в случае, когда на каждом шаге алгоритма в качестве ведущего элемента выбирается медиана массива. Количество сравнений в этом случае $C=(n+1)\log(n+1)-(n+1)$. Количество пересылок зависит от положения элементов, но не может быть больше одного обмена на два сравнения. Поэтому количество пересылок – величина того же порядка, что и число сравнений. Асимптотические оценки для средних значений M и C имеют следующий вид

$$C=O(n \log n), M=O(n \log n) \text{ при } n \rightarrow \infty.$$

Метод Хоара неустойчив.

2.2 ДВОИЧНЫЙ ПОИСК

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

Выбранный элемент равен X . Поиск завершён.

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

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

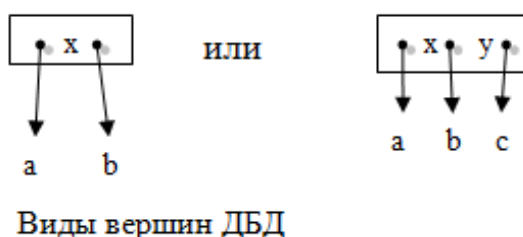
Из-за необходимости найти все элементы соответствующие заданному ключу поиска в курсовой работе использовалась вторая версия двоичного поиска, которая из необходимых элементов находит самый левый, в результате чего для поиска остальных требуется просматривать лишь оставшуюся правую часть массива.

Верхняя оценка трудоёмкости алгоритма двоичного поиска такова. На каждой итерации поиска необходимо два сравнение для первой версии, одно сравнение для второй версии. Количество итераций не больше, чем $\lceil \log_2 n \rceil$. Таким образом, трудоёмкость двоичного поиска в обоих случаях

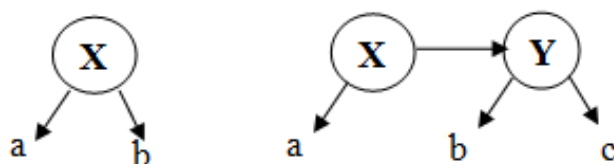
$$C=O(\log n), n \rightarrow \infty.$$

2.3 ДЕРЕВО И ПОИСК ПО ДЕРЕВУ

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



Классическое представление элементов внутри страницы в виде массива неэффективно, поэтому выбран другой способ представления – динамическое размещение на основе списочной структуры, когда внутри страницы существует список из одного или двух элементов.



2.4 МЕТОД КОДИРОВАНИЯ

Алфавитный код Гилберта – Мура

Рассмотрим источник с алфавитом $A = \{a_1, a_2, \dots, a_n\}$ и вероятностями p_1, \dots, p_n . Пусть символы алфавита некоторым образом упорядочены, например, $a_1 \leq a_2 \leq \dots \leq a_n$. Алфавитным называется код, в котором кодовые слова лексико-графически упорядочены, т.е. $\varphi(a_1) \leq \varphi(a_2) \leq \dots \leq \varphi(a_n)$.

Е.Н. Гилбертом и Э.Ф. Муром предложили метод построения алфавитного кода, для которого $L_{cp} < H + 2$. Процесс построения происходит следующим образом.

1. Составим суммы Q_i , $i = 1, n$, вычисленные следующим образом:

$$Q_1 = p_1/2, Q_2 = p_1 + p_2/2, Q_3 = p_1 + p_2 + p_3/2, \dots, Q_n = p_1 + p_2 + \dots + p_{n-1} + p_n/2.$$

2. Представим суммы Q_i в двоичном виде.

3. В качестве кодовых слов возьмем $\lceil -\log_2 p_i \rceil + 1$ младших бит в двоичном представлении Q_i .

Пример. Пусть дан алфавит $A = \{a_1, a_2, a_3, a_4, a_5, a_6\}$ с вероятностями $p_1 = 0.36$, $p_2 = 0.18$, $p_3 = 0.18$, $p_4 = 0.12$, $p_5 = 0.09$, $p_6 = 0.07$. Построенный код приведен в таблице.

Таблица 1 Код Гилберта-Мура

a_i	P_i	Q_i	L_i	кодированное слово
a_1	$1/2^3 \leq 0.18$	0.09	4	0001
a_2	$1/2^3 \leq 0.18 < 1/2^2$	0.27	4	0100
a_3	$1/2^2 \leq 0.36 < 1/2^1$	0.54	3	100
a_4	$1/2^4 \leq 0.07$	0.755	5	11000
a_5	$1/2^4 \leq 0.09$	0.835	5	11010
a_6	$1/2^4 \leq 0.12$	0.94	5	11110

Средняя длина кодового слова не превышает значения энтропии плюс 2

$$L_{cp} = 4 \cdot 0.18 + 4 \cdot 0.18 + 3 \cdot 0.36 + 5 \cdot 0.07 + 5 \cdot 0.09 + 5 \cdot 0.12 = 3.92 < 2.37 + 2$$

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ АЛГОРИТМОВ

1. Загрузка и вывод базы данных

Для загрузки базы данных разработана процедура *read_database()*, в которой производится считывание записей типа *record3*(«Населенный пункт»). Здесь же предусмотрена проверка на наличие файла, откуда выполняется считывание и проверка на выделение памяти для считывания.

За вывод элементов считанной базы данных отвечает процедура *print_database()*. Она предоставляет возможность просмотра базы данных по 20 элементов на странице с возможностью выхода из режима просмотра.

3. Вспомогательные функции и процедуры для сортировки данных

Для сортировки данных используется процедура *sort_database()*, которая очищает экран и вызывает функцию сортировки *quick_sort()*. Доступ к записям базы данных осуществляется через индексный массив *index_database*, для сортировки по ФИО и улице используется процедура *compare_records()*, которая включает в себя сравнение по нескольким полям структуры. При равенстве ФИО происходит сравнение по названию улицы. Здесь же используется процедура сравнения двух строк *compare_strings()* (т.к. не используются стандартные процедуры сравнения двух строк).

4. Особенности реализации бинарного поиска и построения очереди

Бинарный поиск по отсортированной базе осуществляется в процедуре *search_database()*, которая отвечает за ввод ключа(при необходимости повторный, с возможностью отказа) и вызов непосредственно процедуры сортировки *binary_search()*. Доступ к записям ведётся через индексный массив *index_database*, найденные записи заносятся в очередь в процедуре *add_to_queue()*. Для вывода на монитор найденных записей используется процедура *print_queue()*. При реализации бинарного поиска была использована его вторая версия, так как в результате ее выполнения возвращается номер самого левого из найденных элементов, благодаря чему легко найти и вывести остальные элементы, лишь просмотрев оставшуюся правую часть массива, пока не встретится запись, не удовлетворяющая ключу поиска.

5. Особенности построения дерева, его вывода на экран и поиска

Построение дерева осуществляется в функции *add_to_tree()*, возвращающей указатель на корень дерева. Внутри нее происходит непосредственно добавление каждого элемента из очереди с помощью процедуры построения дерева *dbd_tree_add()*. Для вывода дерева на экран используется процедура *print_database_from_tree()*, которая совершает обход по дереву (ЛКП) и выводит(форматированно) данные на экран по 20 элементов. Поиск в дереве осуществляется с помощью функции *tree_search()*, которая, заходя в корень дерева проверяет данные на соответствие ключу, если они идентичны, то совершается поиск в левом и правом поддеревьях(так как неизвестно, где оказался элемент с одинаковым по отношению к корню ключом после поворотов в дереве), и на экран выводится корень дерева. Если ключ поиска и данные в корне дерева различны, то используется поиск в левом или правом поддеревьях в зависимости от данных. Если искомые данные меньше корня, то ищем в левом поддереве, иначе – в правом поддереве.

6. Кодирование данных

Кодирование базы данных начинается с процедуры *encode_database()*, в которой происходит построение массива встречаемых в базе символов *array_symbols*(состоит из структур, содержащих имя(порядковый номер), количества встречи символа в базе данных и вычисленной вероятности встречи), вызов построения кода *gilbert()*. А также форматированный вывод на экран результата, который включает в себя:

- символы, встречающиеся в базе
- вероятность появления символа
- длину каждого кодового слова

- кодовое слово каждого символа
- энтропию
- среднюю длину кодового слова
- общую сумму вероятностей.

Кодирование происходит в процедуре gilbert().

4. ОПИСАНИЕ ПРОГРАММЫ

4.1. ОСНОВНЫЕ ПЕРЕМЕННЫЕ И СТРУКТУРЫ

глобальные переменные и константы:

const int DATABASE_SIZE = 4000; - константа типа int, хранящая размер базы данных
 queue *head = NULL, *tail = NULL; - указатели типа queue на начало и конец списка соответственно
 record3 *database; - указатель на массив структур типа record3 («Населенный пункт»)
 bool dbd_tree_VR = true; - переменная типа bool, хранящая сведения о вертикальном росте в дереве
 bool dbd_tree_HR = true; - переменная типа bool, хранящая сведения о горизонтальном росте в дереве

int size_tree = 0; - переменная, типа int, хранящая размер дерева

struct record3 - структура, используемая для работы с базой данных «Населенный пункт».

```
{
    char a[32]; - поле a типа char (используется для хранения ФИО) 32 символа
    char b[18]; - поле b типа char (используется для хранения названия улицы) 18 символов
    short int c; - поле c типа short int (используется для хранения номера дома)
    short int d; - поле d типа short int (используется для хранения номера квартиры)
    char e[10]; - поле e типа char (используется для хранения даты поселения в формате ДД-ММ-
    ГГ) 10 символов.
};
```

struct queue - структура, используемая для работы с очередью

```
{
    record3 data; - поле data типа record3 (используется для хранения данных типа «Населенный
    пункт»)
    queue *next; - поле, хранящее указатель типа queue (указатель на следующий элемент в
    списке)
};
```

struct tree - Структура, представляющая дерево оптимального поиска (A2).

```
{
    record3 data; - поле данных data типа record3 для хранения данных типа «Населенный пункт»
    int balance; - поле balance типа int для хранения данных о дереве: 0, если у данной вершины есть
    только вертикальные ссылки (вершина одна на странице), и 1, если у данной вершины есть правая
    горизонтальная ссылка.
    tree* left; - поле, хранящее указатель типа tree на левое поддерево
    tree* right; - поле, хранящее указатель типа tree на правое поддерево
};
```

struct symbol - Структура, предназначенная для кодирования базы данных.

```
{
public:
    int probability; поле типа int, предназначенное для хранения количества раз, которое символ
    встретился в тексте
```

double all_probability; поле типа double (используется для хранения вероятности встречи символа в базе данных)

char name; поле типа char, предназначенное для хранения порядкового номера символа

symbol() : probability(0), all_probability(0), name(32) { }; - конструктор, обнуляющие все поля в структуре, а поле типа char заполняет символом «пробел»
};

4.2. ОПИСАНИЕ ПОДПРОГРАММ

Процедуры для обработки базы данных:

1. record3** read_database(); - чтение базы данных с построением индексного массива, возвращает указатель типа record3 на указатель на индексный массив структур
2. void print_database(record3** index_database); - печать базы по 20 элементов, в качестве параметра принимает указатель типа record3 на указатель на индексный массив структур
3. void exit_for_error(char* error); - выход с ошибкой, в качестве параметра принимает char* error(указатель на символьный массив – текст ошибки)
4. void delete_database(record3** index_database); - очистка памяти после базы, в качестве параметра принимает указатель типа record3 на указатель на индексный массив структур

Функции и процедуры сортировки:

5. void sort_database(record3** index_database); - сортировка базы данных, в качестве параметра принимает указатель типа record3 на указатель на индексный массив структур
6. void quick_sort(record3** index_database, int l, int r); - метод быстрой сортировки, в качестве параметров принимает указатель типа record3 на указатель на индексный массив структур, номер, с которого необходимо начать сортировку l типа int, номер, на котором нужно закончить сортировку r типа int.
7. int compare_records(record3* num1, record3* num2); - сравнение двух структур по двум полям(a и b), в качестве параметра принимает две структуры типа record3, возвращает -1 если num1 < num2, если они равны выполняется сравнение по полю b, 1 – иначе.
8. int compare_strings(char* str1, char* str2, int size); - сравнение строк, в качестве параметров принимает символьные массивы str1 и str2 – сравниваемые строки, и size – количество символов, по которым происходит сравнение, возвращает, -1 если str1 < str2, если они равны - 0, 1 – иначе.

Функции и процедуры для поиска в отсортированной базе данных:

9. void search_database(record3** index_database); - поиск в базе, в качестве параметра принимает указатель типа record3 на указатель на индексный массив структур
10. void add_to_queue(record3 data); - добавление элемента в очередь в качестве параметра принимает структуру record3
11. void print_queue(); - печать очереди

Процедуры и функции построения двоичного Б-дерева и поиска в нем

12. tree* add_to_tree(); - добавление элемента из списка в дерево, возвращает указатель на корень дерева типа tree
13. void dbd_tree_add(record3 data, tree *&root); - добавление в дерево, в качестве параметров принимает добавляемые данные(структуру типа record3) и адрес корня дерева, в которое происходит добавление
14. void print_database_from_tree(tree* root); - вывод дерева на экран, в качестве параметра принимает указатель на корень дерева.
15. void search_in_tree(tree* root); - поиск в дереве, в качестве параметра принимает указатель на корень дерева.
16. tree* tree_search(tree* root, int key); - поиск в дереве, в качестве параметров принимает указатель на корень дерева, и ключ поиска типа int

Процедуры и функции кодирования базы данных:

17. void encode_database(); - основная функция кодирования

18. void quick_sort_inverted(symbol* array_symbols, int l, int r) - сортировка вероятностей, в качестве параметров принимает массив указатель на массив символов типа symbol, l и r соответственно левая и правая границы сортировки.

19. int get_probability(symbol* array_symbols, int database_size_symbols); - вычисление вероятности встречи символа, возвращает вероятность встречи символа в базе данных, в качестве параметров принимает указатель на массив символов типа symbol и количество символов, встречающихся в базе данных соответственно.

20. void gilbert(symbol* array_symbols, int n, double *q, int *l, bool **c); функция отвечает за составление кодовых слов для каждого символа базы данных, в качестве параметров принимает:

symbol* array_symbols – указатель на массив структур символов типа symbol

int n - количество разных символов в базе данных

double *q - указатель на массив под Q

int *l - указатель на массив, в котором формируется длина кодового символа для каждого символа

bool **c - матрица - представляющая собой код для каждого символа

Основная программа:

21. main() – основная программа, в которой последовательно вызываются процедуры для работы с базой данных

5. ТЕКСТ ПРОГРАММЫ

```
#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <iostream>
#include <iomanip>
#include <algorithm>
#include <climits>
#include <utility>
#include <cmath>
```

```
using namespace std;
```

```
struct record3
```

```
{
    char a[32];
    char b[18];
    short int c;
    short int d;
    char e[10];
};
```

```
struct queue
```

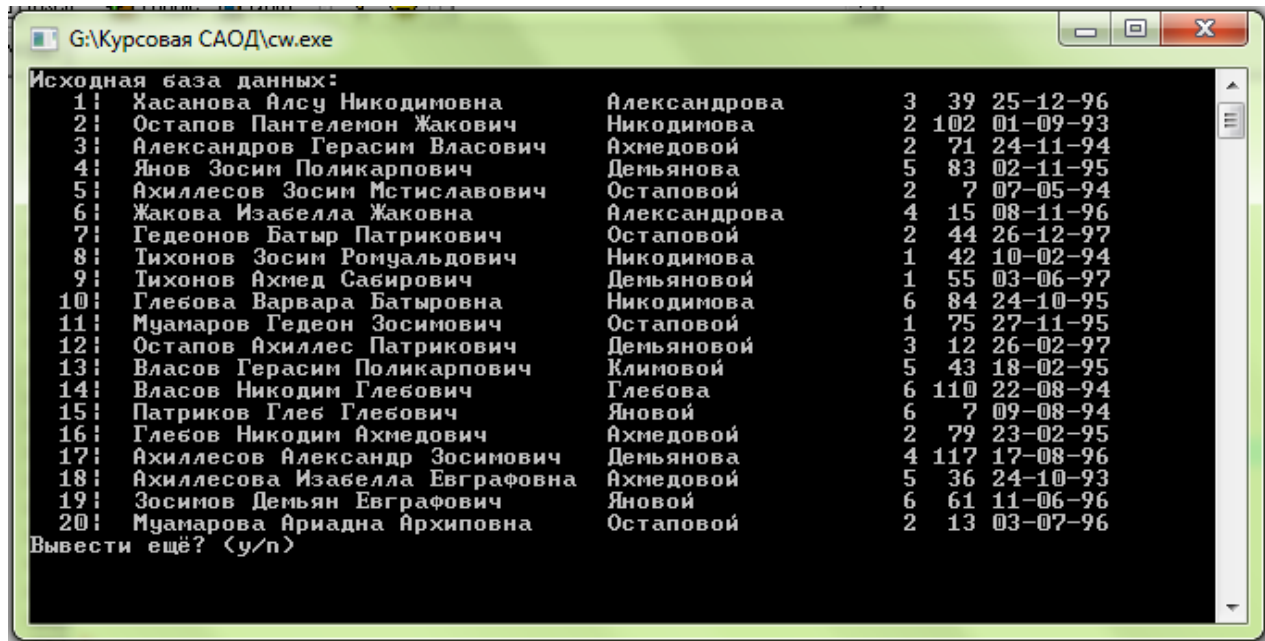
```
{
```

```
        record3 data;  
        queue *next;  
};  
  
struct tree  
{  
    record3 data;  
    int balance;  
    tree* left;  
    tree* right;  
};
```

6. РЕЗУЛЬТАТЫ

Рисунок подписывается снизу!

Рисунок 1. Неотсортированная база данных.

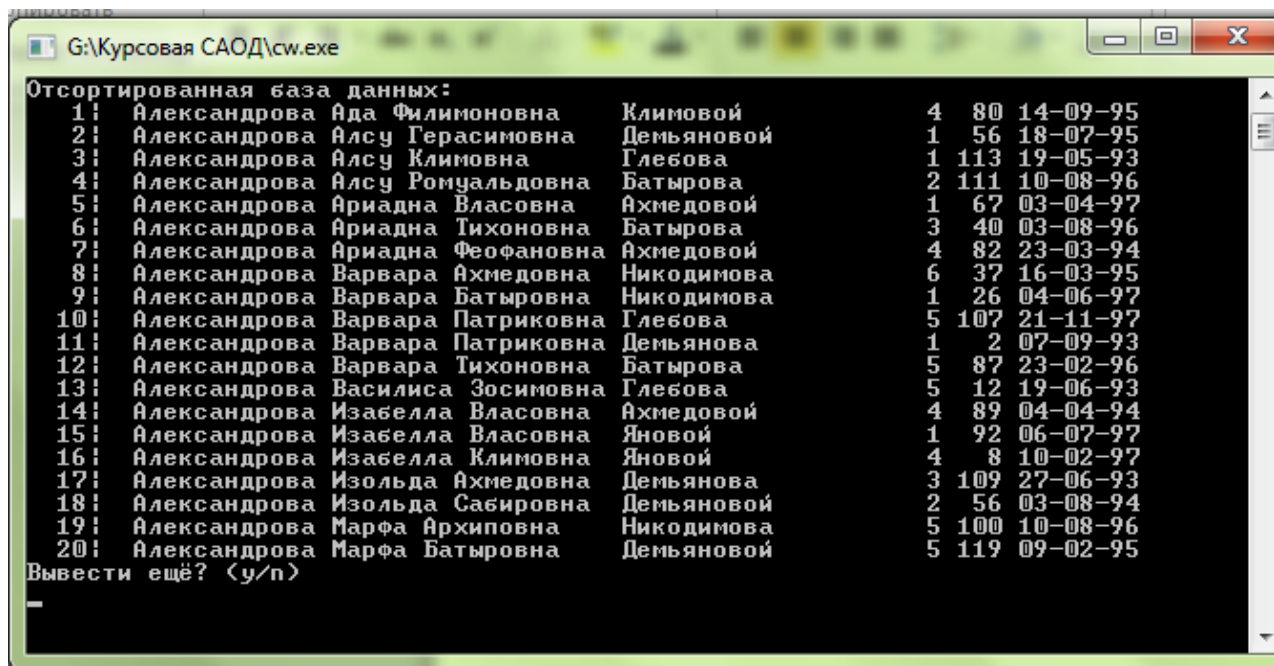


Исходная база данных:

1:	Хасанова Алсу Никодимовна	Александрова	3	39	25-12-96
2:	Остапов Пантелемон Жакович	Никодимова	2	102	01-09-93
3:	Александров Герасим Власович	Ахмедовой	2	71	24-11-94
4:	Янов Зосим Поликарпович	Демьянова	5	83	02-11-95
5:	Ахиллесов Зосим Мстиславович	Остаповой	2	7	07-05-94
6:	Жакова Изабелла Жаковна	Александрова	4	15	08-11-96
7:	Геденов Батыр Патрикovich	Остаповой	2	44	26-12-97
8:	Тихонов Зосим Ромуальдович	Никодимова	1	42	10-02-94
9:	Тихонов Ахмед Сабирович	Демьяновой	1	55	03-06-97
10:	Глебова Варвара Батыровна	Никодимова	6	84	24-10-95
11:	Муамаров Геден Зосимович	Остаповой	1	75	27-11-95
12:	Остапов Ахиллес Патрикovich	Демьяновой	3	12	26-02-97
13:	Власов Герасим Поликарпович	Климовой	5	43	18-02-95
14:	Власов Никодим Глебович	Глебова	6	110	22-08-94
15:	Патриков Глеб Глебович	Яновой	6	7	09-08-94
16:	Глебов Никодим Ахмедович	Ахмедовой	2	79	23-02-95
17:	Ахиллесов Александр Зосимович	Демьянова	4	117	17-08-96
18:	Ахиллесова Изабелла Евграфовна	Ахмедовой	5	36	24-10-93
19:	Зосимов Демьян Евграфович	Яновой	6	61	11-06-96
20:	Муамарова Ариадна Архиповна	Остаповой	2	13	03-07-96

Вывести ещё? (y/n)

Рисунок 2. Отсортированная по ФИО и улице база данных.



Отсортированная база данных:

1:	Александрова Ада Филимоновна	Климовой	4	80	14-09-95
2:	Александрова Алсу Герасимовна	Демьяновой	1	56	18-07-95
3:	Александрова Алсу Климовна	Глебова	1	113	19-05-93
4:	Александрова Алсу Ромуальдовна	Батырова	2	111	10-08-96
5:	Александрова Ариадна Власовна	Ахмедовой	1	67	03-04-97
6:	Александрова Ариадна Тихоновна	Батырова	3	40	03-08-96
7:	Александрова Ариадна Феофановна	Ахмедовой	4	82	23-03-94
8:	Александрова Варвара Ахмедовна	Никодимова	6	37	16-03-95
9:	Александрова Варвара Батыровна	Никодимова	1	26	04-06-97
10:	Александрова Варвара Патрикovich	Глебова	5	107	21-11-97
11:	Александрова Варвара Патрикovich	Демьянова	1	2	07-09-93
12:	Александрова Варвара Тихоновна	Батырова	5	87	23-02-96
13:	Александрова Василиса Зосимовна	Глебова	5	12	19-06-93
14:	Александрова Изабелла Власовна	Ахмедовой	4	89	04-04-94
15:	Александрова Изабелла Власовна	Яновой	1	92	06-07-97
16:	Александрова Изабелла Климовна	Яновой	4	8	10-02-97
17:	Александрова Изольда Ахмедовна	Демьянова	3	109	27-06-93
18:	Александрова Изольда Сабировна	Демьяновой	2	56	03-08-94
19:	Александрова Марфа Архиповна	Никодимова	5	100	10-08-96
20:	Александрова Марфа Батыровна	Демьяновой	5	119	09-02-95

Вывести ещё? (y/n)

Рисунок 3. Очередь из записей, полученных в результате поиска (Фамилии начинаются на «Хас»).

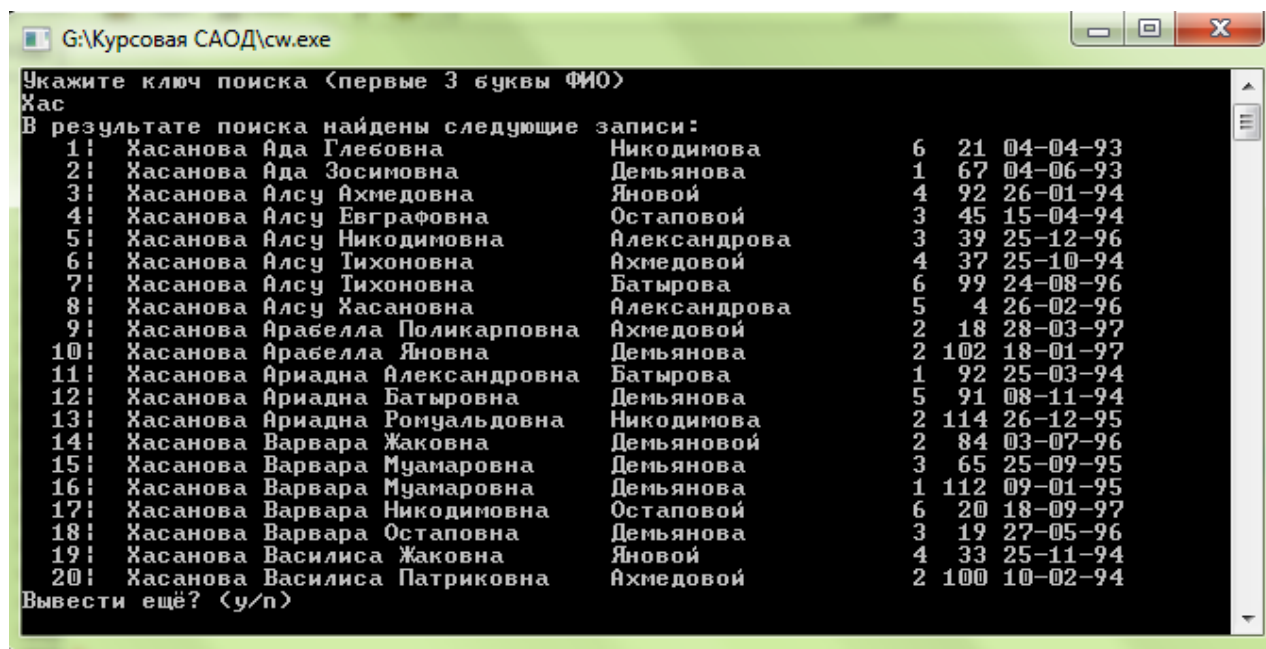


Рисунок 4. Дерево, ключ в дереве – номер квартиры(4 поле).

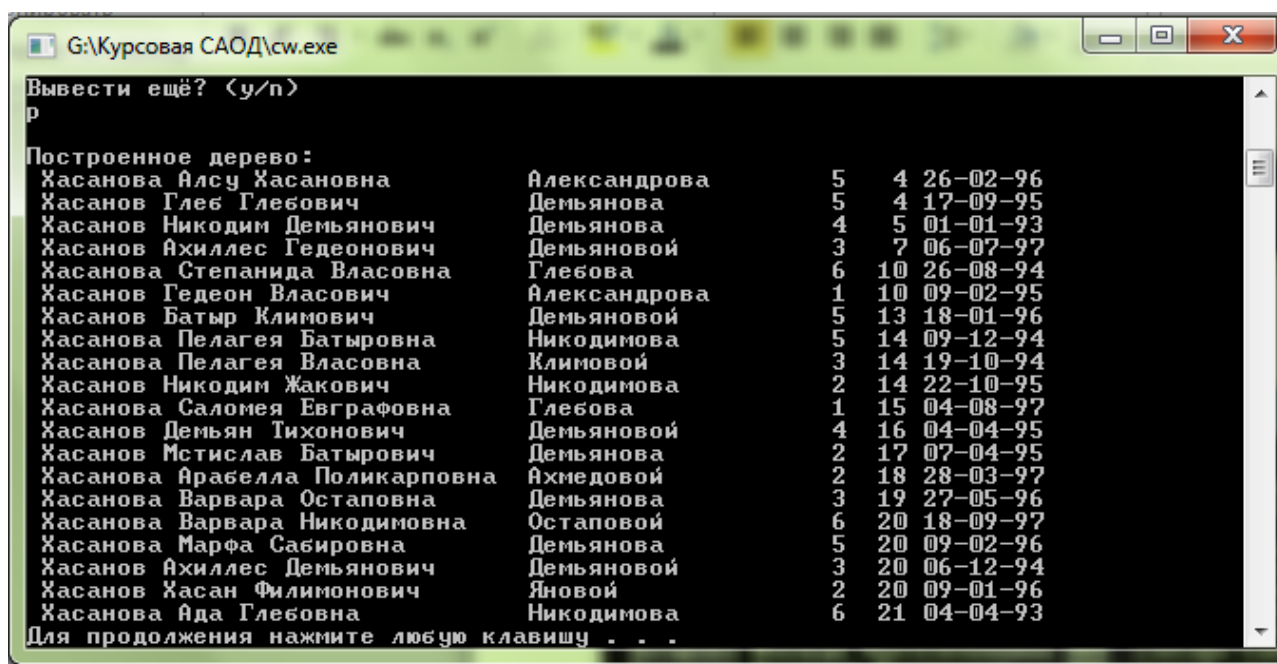
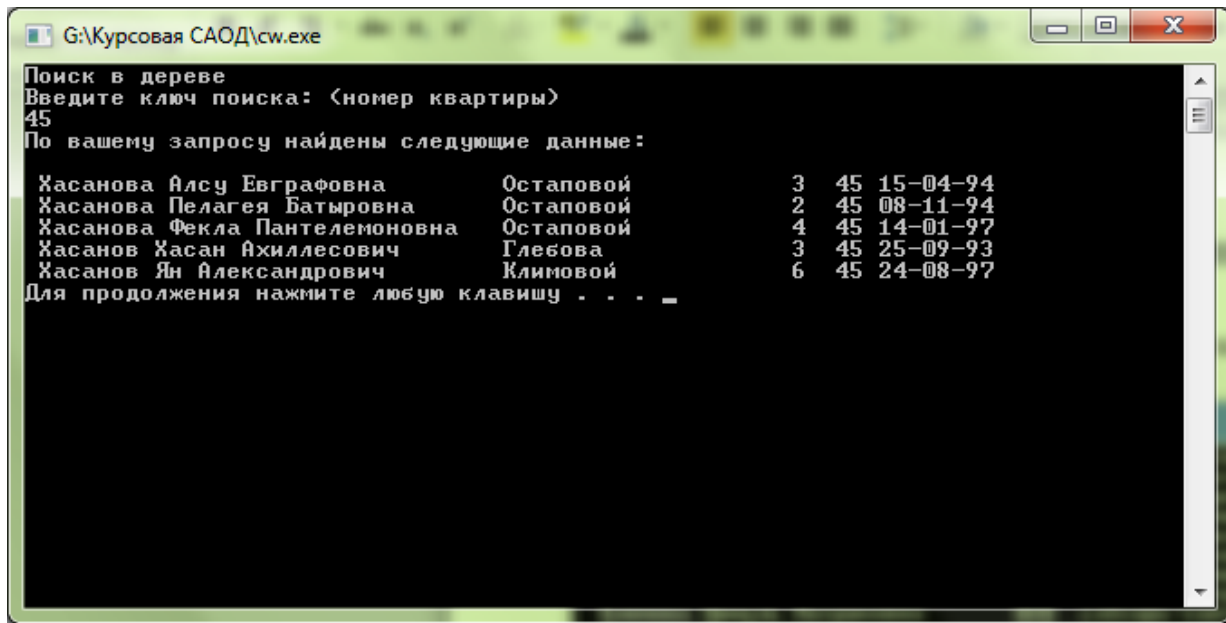


Рисунок 5. Поиск по дереву (элементы с одинаковым ключом).



Результаты кодирования базы данных(начальный и конечный фрагменты):

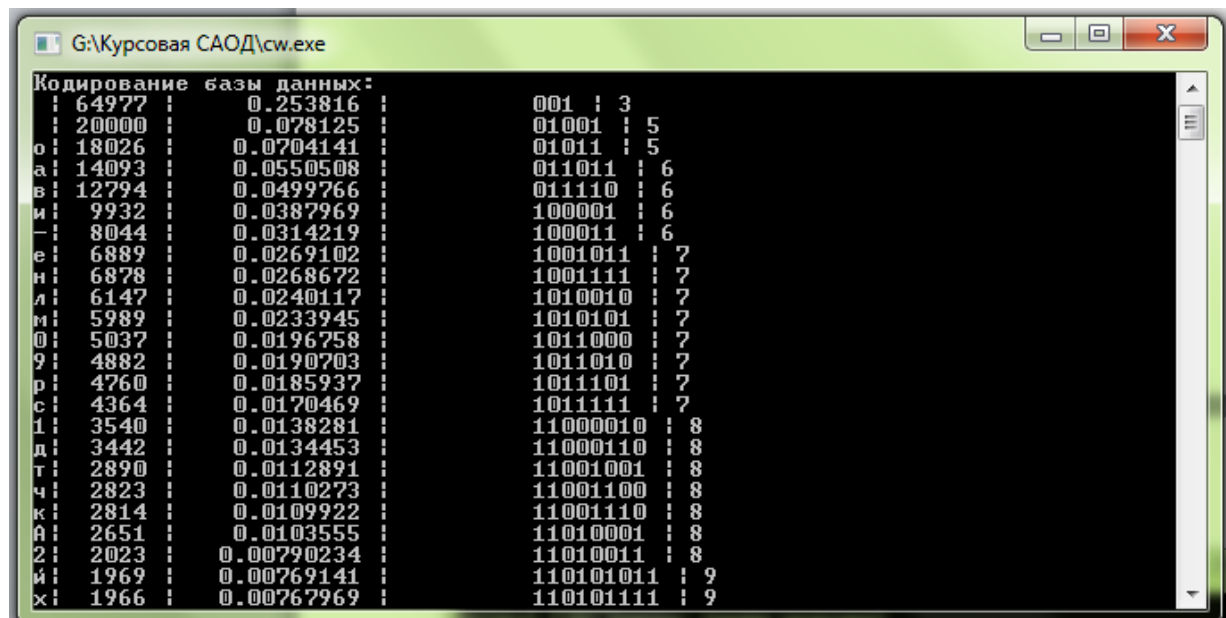


Рисунок 6. Примеры кодовых слов для наиболее вероятных символов в базе

```

G:\Курсовая САОД\sw.exe
.: 28 : 0.000109375 : 111111111000111 : 15
T: 28 : 0.000109375 : 111111111001010 : 15
/: 28 : 0.000109375 : 111111111001110 : 15
E: 28 : 0.000109375 : 111111111010001 : 15
>: 27 : 0.000105469 : 111111111010101 : 15
R: 27 : 0.000105469 : 111111111011000 : 15
.: 27 : 0.000105469 : 111111111011100 : 15
<: 27 : 0.000105469 : 111111111011111 : 15
8: 26 : 0.000101563 : 111111111100011 : 15
U: 26 : 0.000101563 : 111111111100110 : 15
l: 25 : 9.76563e-005 : 111111111101001 : 15
': 25 : 9.76563e-005 : 111111111101100 : 15
: 24 : 9.375e-005 : 111111111101111 : 15
*: 24 : 9.375e-005 : 111111111110011 : 15
+: 24 : 9.375e-005 : 111111111110110 : 15
л: 22 : 8.59375e-005 : 111111111111001 : 15
↓: 22 : 8.59375e-005 : 111111111111011 : 15
a: 21 : 8.20313e-005 : 111111111111110 : 15
Сумма вероятностей: 1
Средняя длина кодового слова: 6.08058
Энтропия: 4.70718
Для продолжения нажмите любую клавишу . . .

```

Рисунок 7. Примеры кодовых слов для минимально вероятных символов в базе, а также вычисленная средняя длина кодового слова и энтропия источника

7. ВЫВОДЫ

В ходе выполнения курсового проекта были выполнены все поставленные задачи и реализованы необходимые алгоритмы: сортировки, двоичного поиска, создания очереди, построения двоичного бинарного дерева, поиска по дереву, кодирования данных.

Четкая структуризация кода и грамотно подобранные имена переменных, структур данных, функций и процедур способствуют удобочитаемости программы.

Все разработанные алгоритмы расширяют возможности работы с данными и способствуют улучшению эффективности анализа и обработки данных и представляют собой минимальный набор процедур для представления и обработки базы данных.