

Министерство цифрового развития, связи и массовых коммуникаций
Российской

Федерации
СибГУТИ

Кафедра ПМиК

КУРСОВОЙ ПРОЕКТ
"Структуры и алгоритмы обработки данных"
ВАРИАНТ 164

Выполнил:
студент группы ИП-217
Павлова В. А
Проверил:
Старший преподаватель
Кафедры ПМиК
Солодов П. С.

Новосибирск
2023

СОДЕРЖАНИЕ

1. ПОСТАНОВКА ЗАДАЧИ	2
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
2.4. Метод кодирования	6
3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ АЛГОРИТМОВ	7
3.1 Интерфейс программы	7
3.2 Загрузка и вывод базы данных	7
3.4 Вспомогательные функции и процедуры для сортировки данных	7
3.5 Особенности реализации бинарного поиска	7
3.6 Вспомогательные функции и процедуры для построения двоичного Б -дерева	8
3.7 Кодирование данных	8
4. ОПИСАНИЕ ПРОГРАММЫ	9
4.1. Основные переменные и структуры	9
4.2. Описание подпрограмм	11
5. ТЕКСТ ПРОГРАММЫ	14
6. РЕЗУЛЬТАТЫ	20
7. ВЫВОДЫ	25

1. ПОСТАНОВКА ЗАДАЧИ

Хранящуюся в файле базу данных (4000 записей) загрузить динамически в оперативную память компьютера в виде массива, вывести на экран по 20 записей (строк) на странице с возможностью отказа от просмотра.

Упорядочить данные по издательству и автору, используя метод цифровой сортировки. Упорядоченные данные вывести на экран. По отсортированному списку строится индексный массив.

Предусмотреть возможность быстрого поиска по первым трем буквам издательства 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 поиска по ключу, отличному от ключа сортировки, вывести на экран содержимое дерева и предусмотреть возможность поиска в дереве по запросу.

Закодировать файл базы данных 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.

Библиографическая база

данных «Жизнь замечательных

людей» Структура записи:

Автор: текстовое поле 12 символов

формат <Фамилия>_<буква>_<буква>

Заглавие: текстовое поле 32 символа

формат <Имя>_<Отчество>_<Фамилия>

Издательство: текстовое поле 16 символов

Год издания: целое число

Кол-во страниц: целое число

Пример записи из БД:

Кловский_В_Б

Лев_Николаевич_Толстой_____

Молодая_гвардия_

1963

864

Варианты условий упорядочения и ключи поиска (K):

по издательству и автору, K = три первые буквы издательства.

2. ОСНОВНЫЕ ИДЕИ И ХАРАКТЕРИСТИКИ ПРИМЕНЯЕМЫХ МЕТОДОВ

2.1. Метод сортировки

В основе метода цифровой сортировки лежит операция соединения очередей.

Пусть дана последовательность из S чисел, представленных в m – ичной системе счисления. Каждое число состоит из L цифр $d_1 d_2 \dots d_L$, $0 \leq d_i \leq m - 1$, $i=1..L$. 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 $M < \text{const } L(m+n)$. При фиксированных m и L $M=O(n)$ при $n \rightarrow \infty$, что значительно быстрее остальных рассмотренных методов. Однако если длина чисел L велика, то метод может проигрывать обычным методам сортировки. Кроме того, метод применим только, если задача сортировки сводится к задаче упорядочивания чисел, что не всегда возможно.

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

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 сравнения 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 1 показаны примеры страниц Б – дерева при $m = 1$.



Рисунок 1 – Пример страниц Б – дерева

Поэтому вновь рассмотрим задачу построения деревьев поиска в оперативной памяти компьютера. В этом случае неэффективным с точки зрения экономии памяти будет представление элементов внутри страницы в виде массива. Выход из положения – динамическое размещение на основе списочной структуры, когда внутри страницы существует список из одного или двух элементов.

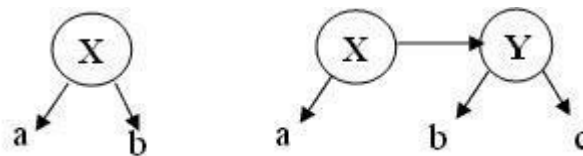


Рисунок 2 – Списочная структура

Таким образом, страницы Б-дерева теряют свою целостность и элементы списков начинают играть роль вершин в двоичном дереве. Однако остается необходимость делать различия между ссылками на потомков (вертикальными) и ссылками на одном уровне (горизонтальными), а также следить, чтобы все листья были на одном уровне.

Очевидно, двоичные Б-деревья представляют собой альтернативу AVL-деревьям. При этом поиск в двоичном Б-дереве происходит как в обычном двоичном дереве. Высота двоичного Б-дерева
$$h = \frac{\log(n+1) - 1}{\log(1+1)} + 1 = \log(n+1)$$

Если рассматривать двоичное Б-дерево как обычное двоичное дерево, то его высота может увеличиться вдвое, т.е. $h = 2 \log(n+1)$. Для сравнения, в AVL-дереве даже в самом плохом случае $h < 1.44 \log n$. Поэтому сложность поиска в двоичном Б-дереве и в AVL-дереве одинакова по порядку величины.

При построении двоичного Б-дерева реже приходится переставлять вершины, поэтому АВЛ-деревья предпочтительней в тех случаях, когда поиск ключей происходит значительно чаще, чем добавление новых элементов. Кроме того, существует зависимость от особенностей реализации, поэтому вопрос о применении того или иного типа деревьев следует решать индивидуально для каждого конкретного случая.

2.4. Метод кодирования

Процесс построения кода методом Гилберта-Мура происходит следующим образом.

1. Вычислим величины Q_i , $i = 1, \dots, n$:

$$Q_1 = p_1 / 2$$

$$Q_2 = p_1 + p_2 / 2$$

$$Q_3 = p_1 + p_2 + p_3 / 2$$

...

$$Q_n = p_1 + p_2 + \dots + p_{n-1} + p_n / 2$$

2. Представим суммы Q_i в двоичном виде.

3. В качестве кодовых слов возьмем . младших бит в двоичном представлении Q_i , $i = 1, \dots, n$

Пусть дан алфавит $A = \{a_1, a_2, a_3, a_4, a_5, a_6\}$ с вероятностями $\lceil -\log p_i \rceil + 1$
 $p_1=0.36, p_2=0.18, p_3=0.18, p_4=0.12, p_5=0.09, p_6=0.07$. Построенный код приведен в таблице 1:

Таблица 1 – Реализация кода 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.
 Действительно,

$$L_{cp} = 4 * 0.18 + 4 * 0.18 + 3 * 0.36 + 5 * 0.07 + 5 * 0.09 + 5 * 0.12$$

$$= 3.92 < 2.37 + 2$$

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ АЛГОРИТМОВ

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

3.1 Интерфейс программы

Для организации интерфейса использовалась процедура menu, которая обеспечивает корректное и понятное использование программы, предоставляет возможность многократного выбора различных вариантов обработки базы данных, в зависимости от задач пользователя.

3.2 Загрузка и вывод базы данных

Загрузка базы данных происходит при запуске программы через функцию Read_BD. Сначала проверяется наличие файла и возможность его открытия, затем, после считывания записей типа Record происходит формирование списка. После этого перед пользователем появляется меню. Один из пунктов которого отвечает за отображение данных. Функция show_records выводит по двадцать записей. Управлять сменой страниц можно нажатием управляющих кнопок «влево» и «вправо», по нажатию на знак тильды «~» предусмотрена возможность ввода определенной страницы. Прервать просмотр можно в любой момент времени с помощью клавиши «Esc».

3.4 Вспомогательные функции и процедуры для сортировки данных

Для выполнения цифровой сортировки понадобилось создать две новых структуры: List, для наложения в памяти компьютера каждого байта ключа в массив digit, который является полем, а также структура Queue, объекты которой это очереди, которые затем соединяются в упорядоченную последовательность.

Для записи данных в List была создана функция FillList, которая поочередно записывает все данные из списка. Так как отсортированные данные находятся в структуре List, то для отображения этих записей через уже созданную функцию show_records, была написана функция List_To_Record. Так же, во избежание перезаписи и лишней нагрузки проводится проверка: если массив, в котором должны быть отсортированные записи, заполнен, сортировка не будет повторяться.

3.5 Особенности реализации бинарного поиска

Для того чтобы без проблем многократно осуществлять поиск элементов, соответствующих разным ключам, требуется каждый раз создавать новую очередь, и чтобы

постоянно не выделять память(которая, как известно, не безгранична) процедура FreeQueue – очищает , ту что была распределена при предыдущем вызове функции построения очереди - MakeQueue. Новая очередь же, строится непосредственно после выполнения поиска. При его реализации была использована вторая версия двоичного поиска, так как в результате ее выполнения возвращается номер самого левого из найденных элементов, благодаря чему легко найти и вывести остальные элементы, лишь просмотрев оставшуюся правую часть массива.

3.6 Вспомогательные функции и процедуры для построения двоичного Б -дерева

Также как и для очереди, при неоднократном построении дерева требуется освобождать память, эту функцию выполняет процедура FreeTree. Для вывода дерева на экран используется процедура PrintTree, представляющая собой обход дерева слева – направо.

Аналогичная процедура PrintSearch выполняет вывод результатов поиска в дереве.

3.7 Кодирование данных

При побуквенном кодировании существует необходимость знать вероятности встречаемости символов. Для начала, происходит считывание символов из базы данных посредством функции loadFile, после которого копия каждого уникального символа записывается в массив. Затем, для подсчета вероятностей используется функция GetP. Для сортировки вероятностей применялся метод прямого включения, выделенный в функцию insertSort из-за удобства и небольшой трудоемкости. После того, как вероятности были упорядочены, вычисляются величины Q через функцию createGilbertMooreCode. По алгоритму, следующим шагом является представление вычисленных величин в двоичном виде, за что и отвечает функция calculation. К этому моменту все приготовления завершены и происходит вывод символов, вероятностей, кодовых слов и их длины, а в конце энтропию и среднюю длину слов, которые вычисляются в функции getEntropy и getAverageLength соответственно.

4. ОПИСАНИЕ ПРОГРАММЫ

4.1. Основные переменные и структуры

-

```
class Record {  
  
    char author[12];  
  
        char title[32];  
  
        char publisher[16];  
  
        short int year;  
  
        short int num_of_page;  
  
public:  
  
        void Show() { std::cout << author << "\t" << title << "\t" <<  
publisher << "\t" << year << "\t" << num_of_page << std::endl; }  
  
        char* getPub() { return publisher; }  
  
        char* getYear() { return year; }  
  
};
```

Класс, используемый для работы с базой данных. **getPub** – метод для получения издательства, которое нужно для сортировки. **getYear** – получение года издания для поиска в дереве.

-

```
struct List {  
  
    List* next;  
  
    union {  
  
        Record record;  
  
        uint8_t digit[sizeof(record)];  
  
    };  
  
};
```

Структура(список), используемая при сортировке базы данных. **next** – указатель на следующие элемент; Наложение массива **digit** на данные из структуры для сортировки через union.

•

```
struct Queue {  
  
    List* head;  
  
    List* tail;  
  
};
```

Структура(очередь), используемая при сортировке базы данных. **head** – голова очереди; **tail** – хвост очереди.

•

```
struct queue {  
    int index;  
    queue *next;  
}
```

Структура, используемая при построении очереди из элементов, полученных в результате бинарного поиска.

index – индекс элемента в базе данных;
next – указатель на следующий элемент.

•

```
struct derevo{  
    int x;  
    int balance;  
    derevo *left;  
    derevo *right;  
}
```

Структура, представляющая двоичное Б – дерево. Где **x** – индекс элемента из базы данных, а не сам элемент.

balance – баланс в вершине(больше 0 если правое поддереву на 1 выше левого, меньше 0, если выше левое и равно 0 при равных высотах левого и правого поддеревьев);
left, right – указатели на левое и правое поддерева.

•

vector<Record>sorted_record – массив с отсортированными данными.

vector<int> index – индексный массив.

vector<unsigned char> uniqSymbols – массив с уникальными символами.

double* P = new double[n] – массив с вероятностями, равный n, т.е количеству уникальных символов.

double* Q = new double[n + 1] – массив с величинами Q.

unsigned char newC** – массив куда записывается Q в двоичном виде.

4.2. Описание подпрограмм

Процедуры, вывода меню:

1. **bool Menu();**

Реализуют меню, с выводом возможных действий и считыванием действия пользователя. В качестве параметра передаются записи.

Процедуры начальной обработки базы данных:

2. **int Read_BD(std::vector<Record> &record);**

3. **void show_records(std::vector<Record> record);**

Read_BD – считывание базы из файла и представление ее элементов в форме вышперечисленных структур, возвращает 1, если файл нельзя открыть, и 0, если запись прошла успешно. **record** – адрес массива, в который будут записаны данные.

show_records – просмотр базы, переданной через массив **record**.

Функции и процедуры сортировки:

4. **List* FillList(std::vector<Record> record);**

5. **void digital_sort(List*& S);**

List* FillList – вспомогательная процедура для записи массива **record** в нужный, для сортировки, вид.

digital_sort – сама сортировка. В качестве параметра передается адрес созданной посредством предыдущей функции последовательности.

Функции и процедуры для поиска в отсортированной базе данных:

6. **int BinSearch(std::vector<Record> x, int N, std::vector<int> pointers, char* key);**

7. **void FreeQueue(queue *p);**

8. **void MakeQueue(char* n, queue* pq, std::vector<int> index, int pos, std::vector<Record> record);**

9. **void PrintQueue(queue *p);**

BinSearch – процедура двоичного поиска (версия 2), **x** – массив записей, в котором осуществляется поиск, **N** – количество записей (изначально – правая граница поиска),

pointers – индексный массив, через который происходит обращение к элементам, **key** – ключ поиска. Возвращает позицию найденного элемента и **-1**, в случае его отсутствия.

FreeQueue – освобождение памяти для очереди **p**, если, например, она уже создавалась.

MakeQueue – построение очереди из результатов поиска. **n** – самый левый из найденных элементов, **pq** – голова очереди, **index** – индексный массив, **pos** – позиция, в которой был найден нужный элемент, от нее просматриваем массив только вправо.

PrintQueue – вывод очереди **p** (**p** – указатель на первый элемент очереди) на экран.

Процедуры и функции построения двоичного Б-дерева:

10. **void FreeTree(derevo *p);**
11. **void CreateDBD(int x, std::vector<Record> base, derevo*& p, std::vector<int> index);**
12. **void PrintTree(std::vector<Record> x, derevo* p, std::vector<int> index);**
13. **derevo *SearchInTree(int key, std::vector<Record> x, derevo* p);**
14. **void PrintSearch(int &key, std::vector<Record> x, derevo* p);**

FreeTree – освобождение памяти для построения дерева, чтобы не возникало проблем, в случае если до этого дерево уже создавалось (**p** – указатель на корень дерева).

CreateDBD – непосредственно построение, **x** – данные, помещаемые в вершину (индекс элемента), **base** – массив структур (обращение к нему происходит при сравнении элементов через индексный массив), **p** – указатель на корень дерева, **index** – индексный массив.

PrintTree – обход дерева с корнем **p**, *используемый для вывода на экран отсортированных элементов базы данных x, в соответствии с индексным массивом – index*, элементы которого хранятся в вершинах дерева **p**.

SearchInTree – поиск в дереве с корнем **p** элементов, соответствующих ключу **key**, **x** – массив структур, к которому обращаемся при поиске, используя вершины дерева **p** в качестве индексов. Возвращает адрес вершины, в которой хранится индекс найденного элемента и **NULL**, в случае его отсутствия.

PrintSearch – вывод на экран результатов поиска (обход поддерева, начиная с вершины с адресом **p**, в которой был найден первый элемент, соответствующий ключу поиска, до того, пока не закончатся все элементы удовлетворяющие заданному условию), параметр **p** обычно изначально принимает значение, возвращаемое предыдущей функцией). Остальные параметры такие же, как и в вышеописанной функции.

Процедуры и функции кодирования базы данных:

15. **unsigned char* loadFile(int* size);**
16. **void getP(unsigned char* data, double* P, std::vector<unsigned char>& symbols, int size);**
17. **bool isInside(std::vector<unsigned char>& array, int element);**
18. **void insertSort(std::vector<unsigned char>& symbols, double* P, int n);**
19. **void createGilbertMooreCode(double* P, double* Q, int* L, int n);**
20. **unsigned char** calculation(double* Q, int* L, int n);**
21. **double getEntropy(double* P, int n);**
22. **double getAverageLength(double* P, int* L, int n);**

loadFile – функция считывания и возвращения символов из базы данных. В **size** записывается количество всех символов. При невозможности открыть или считать завершается программа с кодом 1.

getP – расчет вероятности из данных, переданных через переменную **data**, и запись полученного результат в массив **P**. В **symbols** записаны уникальные символы, количество которых подсчитывается, а затем делится на количество всех символов.

isInside – возвращает 0, если символ **element** уникален, и с досрочным выходом 1, если найдено совпадение.

insertSort – сортировка вероятностей встречаемости символов **symbols** методом прямого включения по убыванию вероятностей **P**.

createGilbertMooreCode – вычисление и запись величин **Q**, а также запись длин кодовых слов в массив **L**.

calculation – функция формирования кодовых слов на основе вычисленных величин **Q**. Возвращает двумерный массив с кодовыми словами.

getEntropy – расчёт и возвращение энтропии на основе вероятностей **P**, **n** - размер массива.

getAverageLength – расчёт и возвращение средней длины кодового слова.

Основная программа

main() – в основной программе вызывается только меню. Если пользователь захотел выйти, функция возвращает 1 и программа закрывается.

5. ТЕКСТ ПРОГРАММЫ

```

#include <iostream>
#include <sys/stat.h>
#include <cmath>
#include <windows.h>
#include <conio.h>
#include <vector>
#include <iomanip>

int VR = 0, HR = 0;

class Record
{
    char author[12];
    char title[32];
    char publisher[16];
    short int year;
    short int num_of_page;

public:
    void Show() {
        std::cout << author << "\t" << title << "\t" <<
        publisher << "\t" << year << "\t" << num_of_page <<
        std::endl;
    }
    char* getPub() { return publisher; }
    int getYear() { return year; }
};

struct List {
    List* next;
    union {
        Record record;
        uint8_t digit[sizeof(record)];
    };
};

struct Queue {
    List* head;
    List* tail;
};

struct queue {
    int index;
    struct queue* next;
} *headq = NULL, *tailq, *spis;

struct derevo {
    int x;
    int balance;
    derevo* left;
    derevo* right;
} *Dbd, *q;
// 1
bool Menu();
// 2 3
int Read_BD(std::vector<Record>& record);
void show_records(std::vector<Record> record);
// 4 5
List* FillList(std::vector<Record> record);
void digital_sort(List*& S);
// 6 9

int BinSearch(std::vector<Record> x, int N,
std::vector<int> pointers, char* key);
void FreeQueue(queue* p);
void MakeQueue(char* n, queue* pq, std::vector<int>
index, int pos, std::vector<Record> record);
void PrintQueue( queue* p, std::vector<Record> record);
// 10 14
void FreeTree(derevo* p);
void CreateDBD(int x, std::vector<Record> base,
derevo*& p, std::vector<int> index);
void PrintTree(std::vector<Record> x, derevo* p,
std::vector<int> index);
derevo* SearchInTree(int key, std::vector<Record> x,
derevo* p);
void PrintSearch(int &key, std::vector<Record> x,
derevo* p);
// 15 22
unsigned char* loadFile(int* size);
void getP(unsigned char* data, double* P,
std::vector<unsigned char>& symbols, int size);
bool isInside(std::vector<unsigned char>& array, int
element);
void insertSort(std::vector<unsigned char>& symbols,
double* P, int n);
void createGilbertMooreCode(double* P, double* Q, int*
L, int n);
double getEntropy(double* P, int n);
double getAverageLength(double* P, int* L, int n);
unsigned char** calculation(double* Q, int* L, int n);

unsigned char** calculation(double* Q, int* L, int n)
{
    unsigned char** C = new unsigned char* [n];
    for (int i = 0; i < n; i++)
    {
        int Li = L[i];
        unsigned char* str = C[i] = new unsigned char[Li
+ 1];
        str[Li] = 0;
        for (int j = 0; j < Li; j++)
        {
            Q[i] *= 2;
            str[j] = Q[i] >= 1 ? '1' : '0';
            if (Q[i] >= 1)
                Q[i]--;
        }
    }
    return C;
}

double getAverageLength(double* P, int* L, int n)
{
    double sum = 0;
    for (int i = 0; i < n; i++)
        sum += P[i] * L[i];
    return sum;
}

double getEntropy(double* P, int n)
{
    double sum = 0;

```

```

for (int i = 0; i < n; i++)
{
    sum += P[i] * log2f(P[i]);
}

return sum * (-1);
};

void createGilbertMooreCode(double* P, double* Q, int*
L, int n)
{
    double pSum = 0;
    for (int i = 0; i < n; i++)
    {
        Q[i] = pSum + P[i] / 2;
        pSum += P[i];
        L[i] = -floor(log2f(P[i])) + 1;
    }
}

void insertSort(std::vector<unsigned char>& symbols,
double* P, int n)
{
    for (int i = 1; i < n; i++)
    {
        double temp = P[i];
        char tempData = symbols[i];
        int j = i - 1;
        while ((j >= 0) && (temp > P[j]))
        {
            P[j + 1] = P[j];
            symbols[j + 1] = symbols[j];
            j = j - 1;
        }
        P[j + 1] = temp;
        symbols[j + 1] = tempData;
    }
}

bool isInside(std::vector<unsigned char>& array, int
element)
{
    if (array.size() == 0)
        return false;

    for (int i = 0; i < array.size(); i++)
    {
        if (array[i] == element)
            return true;
    }
    return false;
};

unsigned char* loadFile(int* size)
{
    FILE* file;
    fopen_s(&file, "testBase1.dat", "rb");
    if (file == NULL)
    {
        printf("Error opening file.\n");
        exit(1);
    }
    struct stat Stat;
    int error = fstat(_fileno(file), &Stat);
    if (error != 0)
    {
        printf("Ошибка считывания stat файла: %d\n",
error);
        exit(1);
    }
    *size = Stat.st_size;
    unsigned char* data = new unsigned char[*size];
    fread(data, *size, 1, file);
    fclose(file);
    return data;
}

void getP(unsigned char* data, double* P,
std::vector<unsigned char>& symbols, int size)
{
    for (int i = 0; i < symbols.size(); i++)
    {
        int frequency = 0;
        for (int j = 0; j < size; j++)
            if (data[j] == symbols[i])
                frequency++;

        P[i] = (double)frequency / size;
    }
}

void digital_sort(List*& S)
{
    unsigned char d;
    Queue Q[256];
    int L = 11, R = 0;
    mark:
    for (int j = L; j >= R; j--)
    {
        for (int i = 0; i < 256; i++)
        {
            Q[i].tail = (List*)&Q[i].head;
        }
        List* p = S;
        while (p != NULL)
        {
            d = p->digit[j];
            Q[(int)d].tail->next = p;
            Q[(int)d].tail = p;
            p = p->next;
        }
        p = (List*)&S;
        for (int i = 0; i < 256; i++)
        {
            if (Q[i].tail != (List*)&Q[i].head)
            {
                p->next = Q[i].head;
                p = Q[i].tail;
            }
        }
        p->next = NULL;
        if (j == 0 && L != 59) {
            L = 59;
            R = 43;
            goto mark;
        }
    }
}

```



```

void show_records(std::vector<Record> record) {
    system("cls");
    int records_on_list = 20;
    int size = record.size();
    int ch;
    for (int i = 0; i < size / records_on_list; i++) {
        for (int j = 0; j < records_on_list; j++) {
            record[i * records_on_list + j].Show();
        }
        std::cout << std::setw(20) << "Page " << i + 1 << "
Records " << i * records_on_list << " - " << (i + 1) *
records_on_list << std::endl;
        std::cout << "  <-- Previous      Next -->" << std::endl;
        std::cout << "    Press Esc to return to the menu
OR\n";
        std::cout << "    Press ~ to input page: ";
        ch = _getch();
        int kd;
        switch (ch) {
            case 224:
                kd = _getch();
                switch (kd) {
                    {
                        case 77:
                            if (i + 1 >= (size /
records_on_list)) {
                                std::cout << "\n
Error! It's the last sheet\n";
                                system("pause");
                                i -= 1;
                            }
                            break;
                        case 75:
                            if (i - 1 < 0) {
                                std::cout << "\n
Error! It's the first sheet\n";
                                system("pause");
                                i += 1;
                            }
                            else {
                                i -= 2;
                            }
                            break;
                        default:
                            i = i - 1;
                            break;
                    }
                    break;
                }
            case 96:
                std::cout << "\n    Input sheet number: ";
                std::cin >> i;
                i -= 2;
                break;
            case 27:
                i = (size / records_on_list) + 1;
                break;
            default:
                i = i - 1;
                break;
        }
        system("cls");
    }
}

std::vector<Record> List_To_Record(struct List* head) {
    struct List* r = head;
    std::vector<Record> buf;
    while (r != NULL) {
        buf.push_back(r->record);
        r = r->next;
    }
    return buf;
}

List* FillList(std::vector<Record> record) {
    List* p = NULL, * head = NULL;
    for (int i = 0; i < record.size(); i++) {
        p = new List;
        p->record = record[i];
        p->next = head;
        head = p;
    }
    return head;
}

int Read_BD(std::vector<Record> &record) {
    FILE* file;
    file = new FILE;
    if (fopen_s(&file, "testBase1.dat", "rb")) {
        fprintf(stderr, "Cannot open input file.\n");
        return 1;
    }
    Record* tt = new Record[4000];
    int i = fread((Record*)tt, sizeof(Record), 4000, file);
    fclose(file);
    for (int i = 0; i < 4000; i++) {
        record.push_back(tt[i]);
    }
    return 0;
}

bool Menu() {
    std::vector<Record> record;
    if (Read_BD(record)) { return 1; }
    List* list = FillList(record);
    std::vector<Record> sorted_record(1);
    std::vector<int> index;
    int size = -1;
    unsigned char* data = loadFile(&size);
    std::vector<unsigned char> uniqSymbols;
    double entropy = 0;
    unsigned char** newC;
    for (int i = 0; i < size; i++) {
        if (!isInside(uniqSymbols, data[i]))
            uniqSymbols.push_back(data[i]);
    }
    int n = uniqSymbols.size();
    double* Q = new double[n + 1];
    int* LGilbertMoore = new int[n];
    double* P = new double[n];
    for (int i = 0; i < 4000; i++) {
        index.push_back(i);
    }
    while (1) {

```

<pre>system("cls"); std::cout << "\tMenu!\n\t1 - Show records\n\t2 - Sorting\n\t3 - Queue\n\t4 - Tree\n\t5 - Search\n\t6 - Coding\n\t6 - Exit\n\n"; int ch = _getch(); int var; switch (ch) { case 49: system("cls"); std::cout << "\tShowing menu\n\t1 - sorted_record); Show unsorted records\n\t2 - Show sorted records\n\t3 - Exit\n\n\t"; var = _getch(); switch (var) { case 49: show_records(record); break; case 50: if (sorted_record.size() == 1) { std::cout << "Error! Records aren't sorted!" << std::endl << "\t"; not created\n\t"; system("pause"); } else show_records(sorted_record); break; case 51: break; default: std::cout << "Error! Incorrect input!" << std::endl << "\t"; system("pause"); break; } break; case 50: system("cls"); if (sorted_record.size() != 1) { std::cout << "\tSorting already has done!" << std::endl << "\t"; system("pause"); break; } digital_sort(list); sorted_record = List_To_Record(list); std::cout << "\tSorting is done!" << "\nError!Element std::endl << "\t"; system("pause"); break; case 51: if (sorted_record.size() == 1) { std::cout << "Error! Records aren't sorted!" << std::endl << "\t"; LGilbertMoore, n); system("pause"); break; } std::cout << "Input Key (first 3 letters of Publisher): "; char x[3]; int y; std::cin >> x; y = BinSearch(sorted_record, 4000, index, x);</pre>	<pre>}; if (y == -1) { std::cout << "\tError!Element system("pause"); break; } FreeQueue(spis); MakeQueue(x, spis, index, y, spis = headq; PrintQueue(spis, sorted_record); system("pause"); break; case 52: spis = headq; VR = 0, HR = 0; FreeTree(Dbd); Dbd = NULL; if (spis == NULL) { std::cout << "\n\tError!Tree is system("pause"); break; } while (spis != NULL) { CreateDBD(spis->index, sorted_record, Dbd, index); spis = spis->next; } derevo * temp; temp = Dbd; PrintTree(sorted_record, temp, index); system("pause"); break; case 53: derevo * tmp, * searchSubTree; std::cout << "\tInput year:"; std::fflush(stdin); int b; std::cin >> b; tmp = Dbd; searchSubTree = SearchInTree(b, sorted_record, tmp); PrintSearch(b, sorted_record, tmp); if (b == 0) std::cout << not found!\n"; system("pause"); break; case 54: getP(data, P, uniqSymbols, size); insertSort(uniqSymbols, P, n); createGilbertMooreCode(P, Q, newC = calculation(Q, LGilbertMoore, n); std::cout << "Gilbert-Mure Code: " << std::cout << "Char\t\tP[i]\t\t" Code\t\tL[i]" << std::endl; for (int i = 0; i < n; i++) { printf_s("%c\t", uniqSymbols[i]);</pre>
---	---

```

std::cout << std::setw(15) <<
P[i] << "\t";
std::string string;
for (int j = 0; j <
LGilbertMoore[i]; j++)
string.push_back(newC[i][j]);
std::cout << std::setw(17) <<
string << "\t";
std::cout << std::setw(3) << }
LGilbertMoore[i] << std::endl;
}
entropy = getEntropy(P, n);
printf("Entropy: %.5f\nL average: ", while (p != NULL) {
entropy);
std::cout << getAverageLength(P, record[p->index].Show();
LGilbertMoore, n) << std::endl;
system("pause");
break;
}
case 55:
return 0;
default:
std::cout << "Error! Incorrect input!" <<
std::endl << "\t";
system("pause");
break;
}
}
}
void PrintQueue( queue* p, std::vector<Record> record) {
while (p != NULL) {
record[p->index].Show();
p = p->next;
}
}
void FreeTree(derevo* p) {
if (p != NULL) {
FreeTree(p->left);
FreeTree(p->right);
free(p);
p = NULL;
}
}
void CreateDBD(int x, std::vector<Record> base, derevo*
&p, std::vector<int> index) {
if (!p) {
p = new(struct derevo);
p->x = x; p->left = NULL; p->right = NULL;
p->balance = 0; VR = 1;
}
else {
if (strcmp(base[x].getPub(), base[p->x].getPub())
CreateDBD(x, base, p->left, index);
if (VR == 1) {
if (p->balance == 0) {
q = p->left;
p->left = q->right;
q->right = p;
p = q;
p->balance = 1;
VR = 0; HR = 1;
}
else p->balance = 0; HR = 0;
}
else HR = 0;
}
else {
if (strcmp(base[x].getPub(),
base[p->x].getPub()) > 0) {
CreateDBD(x, base, p->right,
index);
if (VR == 1) {
p->balance = 1;
VR = 0;
HR = 1;
}
else {
int BinSearch(std::vector<Record> x, int N,
std::vector<int> pointers, char* key) {
int m, L, R;
L = 0;
R = N - 1;
while (L < R) {
m = (L + R) / 2;
if (strcmp(x[pointers[m]].getPub(), key, 3) < 0) <= 0) {
L = m + 1;
else R = m;
}
if (strcmp(x[pointers[R]].getPub(), key, 3) == 0) return
R;
else return -1;
}
}
void FreeQueue(queue* p) {
queue* q;
while (p != NULL) {
q = p;
p = p->next;
free(q);
}
p = NULL;
}
}
void MakeQueue(char* n, queue* pq, std::vector<int>
index, int pos, std::vector<Record> record) {
char y[3];
headq = NULL;
while (1) {
for (int i = 0; i < 3; i++) y[i] =
(record[index[pos]].getPub())[i];
if (strcmp(y, n, 3) != 0) break;
}
}
}

```

```

        if (HR == 1) {
            if
                derevo* SearchInTree(int key, std::vector<Record> x,
                derevo* p) {
                    q = while (p != NULL) {
                        if (key < x[p->x].getYear())
                            p = p->left;
                        else if (key < x[p->x].getYear())
                            p = p->right;
                        else return p;
                    }
                    return NULL;
                }

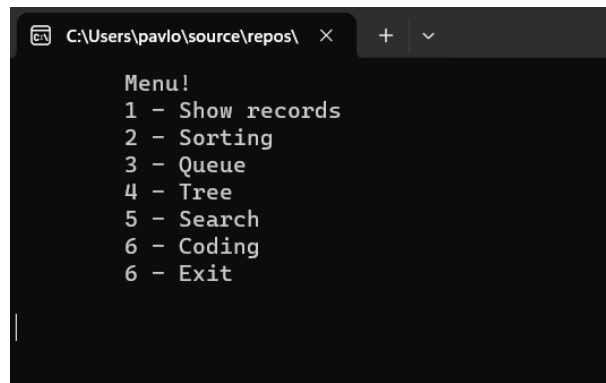
            p = void PrintSearch(int &key, std::vector<Record> x,
            derevo* p) {
                VR int flag = 0;
                if (p != NULL) {
                    HR PrintSearch(key, x, p->left);
                    if (key == x[p->x].getYear()) {
                        x[p->x].Show(); flag = 1;
                    }
                    else HR = 0;
                }
                PrintSearch(key, x, p->right);
            }
            if (flag == 1) { key = -1; }
        }
    }
}

void PrintTree(std::vector<Record> x, derevo* p,
std::vector<int> index) {
    if (p != NULL) {
        PrintTree(x, p->left, index);
        x[p->x].Show();
        PrintTree(x, p->right, index);
    }
}

int main()
{
    bool exit = 1;
    while (exit) {
        exit = Menu();
    }
}

```

6. РЕЗУЛЬТАТЫ



```
C:\Users\pavlo\source\repos\  ×  +  ▾  
Menu!  
1 - Show records  
2 - Sorting  
3 - Queue  
4 - Tree  
5 - Search  
6 - Coding  
6 - Exit
```

Рисунок 3 – Основное меню



```
C:\Users\pavlo\source\repos\  ×  +  ▾  
Showing menu  
1 - Show unsorted records  
2 - Show sorted records  
3 - Exit
```

Рисунок 4 – Меню показа записей

C:\Users\pavlo\source\repos\ X + v

Зосимов Е Ж	Алс Глебовна Гедеонова	Патриков Ltd	1907	768
Евграфо А Л	Виолетт Евграфовна Тихонова	Патриков Ltd	1931	770
Хасанов Б Д	Алс Глебовна Гедеонова	Жаков и сыновья	1978	242
Климов М А	Ад Климовна Остапова	Герасимо and Co	1933	692
Хасанов Б Д	Поликар Янович Герасимов	Жаков и сыновья	1913	370
Жакова Й Н	Поликар Архипович Зосимов	Патриков Ltd	1923	773
Зосимов Т М	Алс Глебовна Гедеонова	Феофанов and Co	1972	236
Муамаро У Л	Поликар Янович Герасимов	Патриков Ltd	1977	561
Ахмедов Б П	Оста Никодимович Хасанов	Жаков и сыновья	1950	256
Зосимов Т М	Василис Демьяновна Архипова	Феофанов-Editio	1908	406
Патрико Л У	Муама Глебович Пантелемонов	Патриков Ltd	1943	101
Архипов С П	Ад Яновна Гедеонова	Батыров Ltd	1899	149
Хасанов Б Д	Арабелл Архиповна Сабирова	Архипов Ltd	1960	410
Янов И К	Ад Остаповна Демьянова	Жаков и сыновья	1970	800
Глебова Е И	Оста Ахиллесович Феофанов	Жаков и сыновья	1909	567
Евграфо А Л	Оста Никодимович Хасанов	Зосимов Ltd	1984	522
Архипов П Р	Александ Янович Климов	Герасимо and Co	1940	833
Зосимов Й У	Ад Ахиллесовна Остапова	Архипов Ltd	1934	195
Архипов В К	Александ Янович Климов	Жаков Ltd	1930	866
Тихонов Д Ж	Вла Тихонович Мстиславов	Герасимо and Co	1930	420

Page 1 Records 0 - 20

<-- Previous Next -->

Press Esc to return to the menu OR

Press ~ to input page: |

Рисунок 5 – Не сортированная база данных

C:\Users\pavlo\source\repos\ X + v

Тихонов Н Д	Вла Никодимович Батыров	Патриков Ltd	1908	831
Глебова Е И	Муама Патрикович Пантелемонов	Архипов Ltd	1951	568
Архипов Е Л	Ад Яновна Гедеонова	Жаков Ltd	1935	726
Глебов М В	Кли Феофанович Ахиллесов	Архипов Ltd	1944	842
Сабиров Т У	Ад Яновна Гедеонова	Батыров Ltd	1979	522
Зосимов Т М	Василис Демьяновна Архипова	Жаков Ltd	1944	351
Жакова П Ж	Кли Батырович Жаков	Феофанов-Editio	1967	861
Архипов П Р	Муама Глебович Пантелемонов	Зосимов Ltd	1993	881
Сабиров Р А	Вла Тихонович Мстиславов	Феофанов and Co	1940	155
Жакова С В	Александ Янович Климов	Патриков Ltd	1949	374
Муамаро И Л	Кли Ахиллесович Герасимов	Архипов Ltd	1946	273
Зосимов Е Ж	Муама Тихонович Власов	Феофанов-Editio	1983	875
Жакова У Н	Оста Никодимович Хасанов	Герасимо and Co	1933	340
Климова Б С	Кли Феофанович Ахиллесов	Герасимо and Co	1950	222
Жакова С В	Муама Патрикович Пантелемонов	Феофанов and Co	1960	377
Зосимов Й У	Мстисла Батырович Патриков	Патриков Ltd	1962	897
Батыров К И	Ад Ахиллесовна Остапова	Жаков и сыновья	1940	808
Феофано Л Ж	Поликар Архипович Зосимов	Феофанов and Co	1917	185
Патрико И Г	Арабелл Архиповна Сабирова	Архипов Ltd	1983	378
Муамаро У Л	Ад Ахиллесовна Остапова	Жаков Ltd	1899	451

Page 20 Records 380 - 400

<-- Previous

Next -->

Press Esc to return to the menu OR

Press ~ to input page: |

Рисунок 6 – Переход на 20 страницу записей

Архипов В К	Жа Гедеонович Герасимов	Архипов Ltd	1921	670
Архипов В К	Зоси Ромуальдович Хасанов	Архипов Ltd	1966	213
Архипов В К	Ад Ахиллесовна Остапова	Архипов Ltd	1925	865
Архипов В К	Василис Демьяновна Архипова	Архипов Ltd	1939	755
Архипов В К	Ад Климовна Остапова	Архипов Ltd	1953	432
Архипов В К	Баты Гедеонович Гедеонов	Архипов Ltd	1926	438
Архипов В К	Ад Остаповна Демьянова	Архипов Ltd	1952	448
Архипов В Р	Кли Феофанович Ахиллесов	Архипов Ltd	1926	838
Архипов В Р	Александ Янович Климов	Архипов Ltd	1929	400
Архипов В Р	Муама Глебович Пантелемонов	Архипов Ltd	1932	111
Архипов В Р	Гедео Ахиллесович Феофанов	Архипов Ltd	1907	673
Архипов В Р	Виолетт Евграфовна Тихонова	Архипов Ltd	1957	589
Архипов В Р	Поликар Архипович Зосимов	Архипов Ltd	1987	596
Архипов В Р	Ад Климовна Остапова	Архипов Ltd	1975	322
Архипов В Р	Оста Ахиллесович Феофанов	Архипов Ltd	1997	133
Архипов В Р	Муама Патрикович Пантелемонов	Архипов Ltd	1993	573
Архипов Е Л	Ад Остаповна Демьянова	Архипов Ltd	1960	417
Архипов Е Л	Александ Янович Климов	Архипов Ltd	1934	625
Архипов Е Л	Кли Батырович Жаков	Архипов Ltd	1900	870
Архипов Е Л	Баты Гедеонович Гедеонов	Архипов Ltd	1972	231

Page 1 Records 0 - 20

<-- Previous Next -->

Press Esc to return to the menu OR

Press ~ to input page: |

Рисунок 7 – Сортированная база данных

Хасанов П Б	Оста Ахиллесович Феофанов	Жаков Ltd	1903	680
Янов В З	Зоси Ромуальдович Хасанов	Жаков Ltd	1899	707
Янов В З	Ад Климовна Остапова	Жаков Ltd	1930	108
Янов В З	Поликар Янович Герасимов	Жаков Ltd	1924	802
Янов В З	Гедео Ахиллесович Феофанов	Жаков Ltd	1975	738
Янов В З	Арабелл Архиповна Сабирова	Жаков Ltd	1978	629
Янов Ж З	Арабелл Архиповна Сабирова	Жаков Ltd	1900	118
Янов Ж З	Ахилле Поликарпович Александро	Жаков Ltd	1966	463
Янов И К	Изольд Ромуальдовна Герасимова	Жаков Ltd	1921	897
Янов И К	Зоси Ромуальдович Хасанов	Жаков Ltd	1922	713
Янов И К	Александ Янович Климов	Жаков Ltd	1951	761
Янов И К	Ад Яновна Гедеонова	Жаков Ltd	1924	215
Янов Л П	Кли Ахиллесович Герасимов	Жаков Ltd	1985	483
Янов Л П	Ад Яновна Гедеонова	Жаков Ltd	1945	826
Янов Л П	Вла Тихонович Мстиславов	Жаков Ltd	1973	844
Янов Л П	Кли Ахиллесович Герасимов	Жаков Ltd	1989	894
Янов У А	Ад Остаповна Демьянова	Жаков Ltd	1911	493
Янов У А	Оста Никодимович Хасанов	Жаков Ltd	1948	812
Янов У А	Вла Никодимович Батыров	Жаков Ltd	1945	514
Архипов В К	Оста Никодимович Хасанов	Жаков и сыновья	1990	837
Архипов В К	Алс Глебовна Гедеонова	Жаков и сыновья	1911	527
Архипов В К	Муама Глебович Пантелемонов	Жаков и сыновья	1936	196
Архипов В К	Алс Глебовна Гедеонова	Жаков и сыновья	1954	535
Архипов В К	Гедео Хасанович Жаков	Жаков и сыновья	1919	297
Архипов В К	Кли Батырович Жаков	Жаков и сыновья	1972	217
Архипов В К	Ад Остаповна Демьянова	Жаков и сыновья	1941	164
Архипов В Р	Оста Ахиллесович Феофанов	Жаков и сыновья	1962	671
Архипов В Р	Кли Феофанович Ахиллесов	Жаков и сыновья	1975	277
Архипов В Р	Алс Глебовна Гедеонова	Жаков и сыновья	1991	783
Архипов В Р	Поликар Янович Герасимов	Жаков и сыновья	1965	471

Рисунок 8 – Очередь по ключу «Жак»

Архипов К В	Александ Янович Климов	Жаков и сыновья	1957	862
Архипов К В	Гедео Ахиллесович Феофанов	Жаков и сыновья	1937	343
Архипов К В	Муама Патрикович Пантелемонов	Жаков и сыновья	1950	412
Архипов Е Л	Кли Ахиллесович Герасимов	Жаков и сыновья	1918	416
Архипов Е Л	Баты Гедеонович Гедеонов	Жаков и сыновья	1942	622
Архипов Е Л	Баты Гедеонович Гедеонов	Жаков и сыновья	1942	723
Архипов Е Л	Оста Ахиллесович Феофанов	Жаков и сыновья	1945	655
Архипов Е Л	Оста Никодимович Хасанов	Жаков и сыновья	1995	163
Архипов Е Л	Кли Батырович Жаков	Жаков и сыновья	1918	320
Архипов Е Л	Кли Ахиллесович Герасимов	Жаков и сыновья	1970	657
Архипов Е Л	Ахилле Поликарпович Александро	Жаков и сыновья	1980	764
Архипов В Р	Ад Остаповна Демьянова	Жаков и сыновья	1943	419
Архипов В Р	Кли Феофанович Ахиллесов	Жаков и сыновья	1908	283
Архипов В Р	Баты Гедеонович Гедеонов	Жаков и сыновья	1955	671
Архипов В Р	Кли Батырович Жаков	Жаков и сыновья	1927	553
Архипов В Р	Муама Патрикович Пантелемонов	Жаков и сыновья	1906	481
Архипов В Р	Виолетт Евграфовна Тихонова	Жаков и сыновья	1982	623
Архипов В Р	Поликар Архипович Зосимов	Жаков и сыновья	1985	434
Архипов В Р	Поликар Янович Герасимов	Жаков и сыновья	1965	471
Архипов В Р	Алс Глебовна Гедеонова	Жаков и сыновья	1991	783
Архипов В Р	Кли Феофанович Ахиллесов	Жаков и сыновья	1975	277
Архипов В Р	Оста Ахиллесович Феофанов	Жаков и сыновья	1962	671
Архипов В К	Ад Остаповна Демьянова	Жаков и сыновья	1941	164
Архипов В К	Кли Батырович Жаков	Жаков и сыновья	1972	217
Архипов В К	Гедео Хасанович Жаков	Жаков и сыновья	1919	297
Архипов В К	Алс Глебовна Гедеонова	Жаков и сыновья	1954	535
Архипов В К	Муама Глебович Пантелемонов	Жаков и сыновья	1936	196
Архипов В К	Алс Глебовна Гедеонова	Жаков и сыновья	1911	527
Архипов В К	Оста Никодимович Хасанов	Жаков и сыновья	1990	837

Для продолжения нажмите любую клавишу . . . |

Рисунок 9 – ДБД по полученной очереди

Input year:1942				
Янов У А	Муама Патрикович Пантелемонов	Жаков и сыновья	1942	191
Патрико И Б	Вла Никодимович Батыров	Жаков и сыновья	1942	679
Остапов М Г	Вла Тихонович Мстиславов	Жаков и сыновья	1942	522

Для продолжения нажмите любую клавишу . . . |

Рисунок 10 – Поиск в дереве по ключу «1942»

X	4.6875e-05	1111111110111001	16
■	4.6875e-05	1111111110111100	16
Z	4.6875e-05	1111111110111111	16
	4.6875e-05	1111111111000010	16
щ	4.6875e-05	1111111111000101	16
л	4.6875e-05	1111111111001000	16
т	4.29687e-05	1111111111001011	16
ё	4.29687e-05	1111111111001110	16
*	4.29687e-05	1111111111010001	16
	4.29687e-05	1111111111010011	16
	4.29687e-05	1111111111010110	16
	4.29687e-05	1111111111011001	16
о	4.29687e-05	1111111111011100	16
ў	4.29687e-05	1111111111011111	16
5	4.29687e-05	1111111111100001	16
	4.29687e-05	1111111111100100	16
•	4.29687e-05	1111111111100111	16
?	3.90625e-05	1111111111101001	16
0	3.90625e-05	1111111111101100	16
†	3.90625e-05	1111111111101111	16
ь	3.90625e-05	1111111111110001	16
	3.51562e-05	1111111111110011	16
H	3.51562e-05	1111111111110110	16
=	3.51562e-05	1111111111111000	16
J	3.51562e-05	1111111111111010	16
ш	3.51562e-05	1111111111111100	16
F	3.125e-05	1111111111111110	16
	3.125e-05	1111111111111111	16
Entropy: 4.88692			
L average: 6.48848			
Для продолжения нажмите любую клавишу . . .			

Рисунок 11 – Пример кодовых слов и вывод энтропии и средней длины

7. ВЫВОДЫ

В ходе выполнения курсовой работы были выполнены все поставленные задачи и реализованы необходимые алгоритмы: сортировки, поиска, построения двоичного Б – дерева, поиска по дереву и кодирования базы данных.

В результате кодирования были получены данные подтверждающие теоретические сведения. К таковым относятся: величины средней длины кодового слова и энтропии ($L_{cp} \leq H + 2$).

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