Федеральное агентство связи
Сибирский государственный университет телекоммуникаций и информатики

Кафедра прикладной математики и кибернетики

Курсовой проект

по курсу

«Структуры и алгоритмы обработки данных»

Вариант 23

Выполнил: студент группы ИП-211
Степанов Алексей

Проверил: доцент кафедры ПМиК
Янченко Е.В.

Новосибирск 2023

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 10
6. РЕЗУЛЬТАТЫ 23
7. ВЫВОДЫ 26

1. ПОСТАНОВКА ЗАДАЧИ

Хранящуюся в файле базу данных загрузить в оперативную память компьютера и построить индексный массив, упорядочивающий данные **по сумме вклада и дате** , используя **метод прямого слияния** 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 дате**, и предусмотреть возможность поиска в дереве по запросу.

Закодировать файл базы данных статическим **кодом Фано**, предварительно оценив вероятности всех встречающихся в ней символов. Построенный код вывести на экран.

Структура записи:

ФИО вкладчика: текстовое поле 30 символа

формат <Фамилия>_<Имя>_<Отчество>

Сумма вклада: целое число (использовать unsigned short int)

Дата вклада: текстовое поле 10 символов

формат дд-мм-гг

ФИО адвоката: текстовое поле 22 символа

формат <Фамилия>_<буква>_<буква>

Пример записи из БД:

Петров_Иван_Федорович_____

130

15-03-46

Иванова_И_В_____

Варианты условий упорядочения и ключи поиска (К):

С = 2 - по сумме вклада и дате, К = сумма вклада;

2. ОСНОВНЫЕ ИДЕИ И ХАРАКТЕРИСТИКИ ПРИМЕНЯЕМЫХ МЕТОДОВ

2.1. МЕТОД СОРТИРОВКИ

Метод прямого слияния

В основе метода прямого слияния лежит операция слияния серий. p-серией называется упорядоченная последовательность из p элементов. Пусть имеются две упорядоченные серии a и b длины q и r соответственно. Необходимо получить упорядоченную последовательность c, которая состоит из элементов серий a и b. Сначала сравниваем первые элементы последовательностей a и b. Минимальный элемент перемещаем в последовательность c. Повторяем действия до тех пор, пока одна из последовательностей a и b не станет пустой, оставшиеся элементы из другой последовательности переносим в последовательность c. В результате получим (q+r)-серию.

Для алгоритма слияния серий с длинами q и r необходимое количество сравнений 32 и перемещений оценивается следующим образом $\min(q, r) \le C \le q+r-1$, $M=q+r$

Пусть длина списка S равна степени двойки, т.е. 2^k, для некоторого натурального k. Разобьем последовательность S на два списка a и b, записывая поочередно элементы S в списки a и b. Сливаем списки a и b с образованием двойных серий, то есть одиночные элементы сливаются в упорядоченные пары, которые записываются попеременно в очереди c0 и c1. Переписываем очередь c0 в список a, очередь c1 – в список b. Вновь сливаем a и b с образованием серий длины 4 и т. д. На каждом итерации размер серий увеличивается вдвое. Сортировка заканчивается, когда длина серии превысит общее количество элементов в обоих списках. Если длина списка S не является степенью двойки, то некоторые серии в процессе сортировки могут быть короче.

Трудоёмкость метода прямого слияния определяется сложностью операции слияния серий. На каждой итерации происходит ровно n перемещений элементов списка и не более n сравнений. Как нетрудно видеть, количество итераций равно $\lceil \log n \rceil$. Тогда

$$C < n\lceil \log n \rceil,\ M=n\lceil \log n \rceil + n.$$

Дополнительные n перемещений происходят во время начального расщепления исходного списка. Асимптотические оценки для M и C имеют следующий вид

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

Метод обеспечивает устойчивую сортировку. При реализации для массивов, метод требует наличия второго вспомогательного массива, равного по размеру исходному массиву. При реализации со списками дополнительной памяти не требуется.

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.

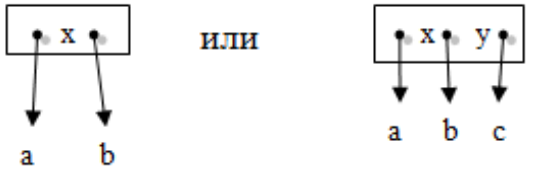
Из-за необходимости найти все элементы соответствующие заданному ключу поиска в курсовой работе использовалась вторая версия двоичного поиска, которая из необходимых элементов находит самый левый, в результате чего для поиска остальных требуется просматривать лишь оставшуюся правую часть массива.

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

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

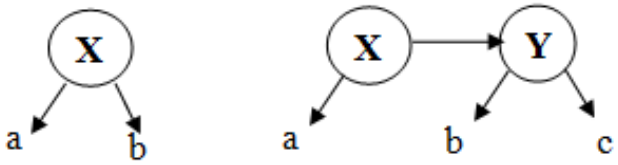
2.3 ДЕРЕВО И ПОИСК ПО ДЕРЕВУ

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 $m = 1$.



Виды вершин ДБД

. 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.



Вершины двоичного Б-дерева

2.4 МЕТОД КОДИРОВАНИЯ

Код Фано

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

Метод Фано построения префиксного почти оптимального кода, для которого Lcp < H (p1 ,...,pn) + 1, заключается в следующем. Упорядоченный по убыванию вероятностей Список букв алфавита источника делится на две части так, чтобы суммы вероятностей букв, входящих в эти части, как можно меньше отличались друг от друга. Буквам первой части приписывается 0, а буквам из второй части – 1. Далее также поступают с каждой из полученных частей. Процесс продолжается до тех пор, пока весь список не разобьется на части, содержащие по одной букве.

Пример. Пусть дан алфавит A={a1, a2, a3, a4, a5, a6} с вероятностями p1=0.36, p2=0.18, p3=0.18, p4=0.12, p5=0.09, p6=0.07.

Построенный код приведен в таблице.

Таблица 1 Код *Фано*

a_i	P_i	кодовое слово				L_i
a_1	0.36	0	0			2
a_2	0.18	0	1			2
a_3	0.18	1	0			2
a_4	0.12	1	1	0		3
a_5	0.09	1	1	1	0	3
a_6	0.07	1	1	1	1	4

Полученный код является префиксным и почти оптимальным со средней длиной кодового слова Lcp=0.36. 2+0.18. 2+0.18. 2+0.12. 3+0.09. 4+0.07. 4=2.44

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ АЛГОРИТМОВ

1. Загрузка и вывод базы данных

Загрузка базы данных происходит в функции main(). Здесь же предусмотрена проверка на наличие файла, откуда выполняется считывание и проверка на выделение памяти для считывания.

Процедура load_to_memory формирует очередь для последующей сортировки

За вывод элементов считанной базы данных отвечает процедура *Loop(),* откуда и вызываются все последующие функции. А также *show_page()* и *show_page_sor*t(), позволяющие просматривать неотсортированную и отсортированную базу по 20 элементов на странице с возможностью выхода из режима просмотра.

3. Вспомогательные функции и процедуры для сортировки данных

Для сортировки данных используется функция сортировки *merge_sort(),* которая в свою очередь используется вспомогательные процедуры *FromListToLine(), Merge()* и *Split().* Доступ к записям базы данных для сортировки без изменения изначального массива записей осуществляется через индексный массив *indarr*, для сортировки по сумме вклада и дате используется процедура *compare(),* которая включает в себя сравнение по нескольким полям структуры. При равенстве суммы вклада происходит сравнение по дате вклада, для этого год и число в дате переставляются местами (так как, год приоритетнее при сравнении) и полученная строка сравнивается с помощью стандартной функции *strcmp()*

4. Особенности реализации бинарного поиска

Бинарный поиск по отсортированной базе осуществляется в процедуре Bsearch(). Доступ к записям ведётся через индексный массив indarr. 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 при помощи процедуры BsearchAll().

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

Построение дерева осуществляется в функции add_node_tree(), которая вызывается в функции BSearchAll().Внутри нее происходит сравнение записей по дате вклада, используя функцию strcomp(). Для вывода дерева на экран используется процедура TreeRight(), которая совершает обход по дереву (ЛКП) и выводит данные на экран. Поиск в дереве осуществляется с помощью функции search_tree(), 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 осуществляется в процедуре Encode(), в которой происходит построение вектора встречаемых в базе элементов S, построение векторов P, который хранит в себе элементы из базы данных без повторения и вероятность встречи, и L, который хранит в себе элементы из базы данные без повторения и длину кодового слова. Энтропия базы данных рассчитывается в функции Entrophy(). Функции get_probability() и decrease_probab() высчитывают вероятности встречи элементов базы данных и сортируют векторы по убыванию по вероятностям соответственно. Построение кодовых слов происходит в процедуре Fano(), которая использует дополнительную для кодирования функцию MED(). Функция Fano_decode() осуществляет преобразование кодовых слов из массива с целочисленным числами в строку. Форматированный вывод происходит в процедуре Table(), включающий в себя:

- символы, встречающиеся в базе данных
- вероятность появления символа
- кодовое слово для каждого символа
- длин каждого кодового слова

Entrophy_table() выводится значение энтропии, средней длины кодового слова и избыточность.

4. ОПИСАНИЕ ПРОГРАММЫ

4.1. ОСНОВНЫЕ ПЕРЕМЕННЫЕ И СТРУКТУРЫ

глобальные переменные и константы:

bool HR; - переменная типа bool, хранящая сведение о горизонтальном росте в дереве
bool VR; - переменная типа bool, хранящая сведение о вертикальном росте в дереве

struct Data - структура, используемая для работы с базой данных «Обманутые вкладчики».
{
char full_name[30]; - поле full_name типа char (используется для хранения ФИО) 30 символов
unsigned short int amount; - поле с типа unsigned short int(используется для хранения суммы вклада)
char date[10];]; - поле date типа char (используется для хранения даты вклада в формате ДД-ММ-ГГ) 10 символов
char lawyer[22];]; - поле lawyer типа char (используется для хранения ФИО юриста) 22 символа
};

struct Node - структура для работы с списком для сортировки
{
Data *memory; - поле memory типа Data(используется для хранения данных типа «Обманутые вкладчики»)
Node *next; - поле, хранящее указатель типа Node(указатель на следующий элемент в списке)
}; list – название структуры

struct queue - структура, хранящая указатели на структуру list, для слияния отсортированных очередей
{
list *head = NULL; - поле, хранящее указатель типа list (указатель на начальный (головной) элемент списка)
list *tail = NULL; - поле, хранящее указатель типа list (указатель на последний (хвостовой) элемент списка)
} line; - название структуры

struct Vertex - Структура, представляющая двоичное б-дерево.

{

Data *data; - поле данных data типа Data для хранения данных типа «Обманутые вкладчики»

Vertex* left; - поле, хранящее указатель типа Vertex на левое поддерево

Vertex* right; - поле, хранящее указатель типа Vertex на правое поддерево

Vertex *equal; - поле, хранящее указатель типа Vertex на поддерево с одинаковыми ключами

int bal; - поле bal 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.

};

4.2. ОПИСАНИЕ ПОДПРОГРАММ

Процедуры для обработки базы данных:

1. list *load_to_memory(Data *dataArr, int numStruct) - чтение базы данных с построением очереди, возвращает на указатель на поле head
2. void show_page(Data *dataArr, int numStruct) – печать неотсортированной базы по 20 элементов, в качестве параметра принимает указатель на индексный массив структур и кол-во записей
3. void show_page_sort(Data *indArr[], int numStruct) – печать отсортированной базы по 20 элементов, в качестве параметра принимает указатель на индексный массив структур и кол-во записей
4. void loop(Data *dataArr, Data *indArr[], int numStruct) – печать основного меню выбора действий с базой. В качества параметров принимает указатель на индексный массив структур, указатель типа Data на индексный массив и кол-во записей

Функции и процедуры сортировки:

5. void merge_sort(list *&S) - основная процедура сортировки методом прямого слияния, принимает в качестве параметра адрес первого элемента в списке
6. int Split(list *&S, list *&A, list *&B) - функция разделения списка на два подсписка для дальнейшей сортировки. В качестве параметров принимает адреса первого элемента списка S типа list и адреса первых элементов в списках типа list A и B для разделения основного.
7. void Merge(list *&A, int &q, list *&B, int &r, line &C) - процедура слияния отсортированных двух списков в один изачальный. В качестве параметров принимает адреса двух первых элементов двух списков типа list A и B. Длины q и r типа int. Указатель на элемент из очереди C типа line.
8. void FromListToLine(list *&list, line &queue) - процедура переписывания элементов из очереди в список. В качестве параметров принимает адрес на элемент списка list типа list и адрес элемента из очереди queue типа line.
9. int compare(Data *a, Data *b) – функция сравнения двух записей базы данных по сумме вклада. Если сумма вклада совпадает, то происходит сравнение по дате вклада. Принимает в качестве параметров указатели a и b типа Data на элементы структуры с записями базы данных.
10. void clear(list *head) - процедура очистки памяти списка для сортировки. В качестве параметров принимает указатель head типа list на первый элемент списка.

Функции и процедуры для поиска в отсортированной базе данных:

11. int BSearch(Data *indArr[], int numStruct, int *key) - функция двоичного поиска записи в базе данных по ключу. В качестве параметров принимает указатель на динамический массив indArr типа Data, количество записей в базе данных numStruct типа int и ключ поиска key типа int. Возвращает индекс на самый левый элемент с данным ключом поиска.
12. void BSearchAll(Data *intArr[], int ind, int *key, Vertex *&Root) - процедура вывода очереди с ключом поиска на экран и вызов процедуры для добавления нужных записей в двоичное б-дерево. В качестве

параметров принимает указатель на динамический массив indArr типа Data, индекс первой записи с ключом поиска ind типа int, ключ поиска key типа int и адрес корня двоичного Б-дерева Root типа Vertex;

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

13. void add_tree_node(Data *data, Vertex *&p) - процедура построения двоичного Б-дерева и добавления в него элементов. В качестве параметров принимает указатель на элемент структуры data типа Data и адрес корня дерева p типа Vertex.
14. int strcomp(const std::string &str1, const std::string &str2) - функция сравнения строк с датой вклада путём преобразования их в переменные целочисленного типа int. В качестве параметров принимает адреса строк с датой str1 и str2 типа std::string. В результате сравнения возвращается число, нужное для условия в какое поддерево добавлять элемент.
15. void TreeRight(Vertex *Root, int &count); - процедура для обхода двоичного Б-дерева слева направо (ЛКП) и вывода содержимого на экран. В качестве параметров принимает указатель на корень дерева Root типа Vertex и счетчик записей в дереве count типа int.
16. void search_tree(Vertex *Root, const std::string &key, int &count); - процедура для поиска записей в двоичном Б-дереве по дате вклада. В качестве параметров принимает указатель на дерево Root типа Vertex, ключ поиска key типа std::string и счетчик найденных записей для дальнейшего вывода count типа int. Если ключ поиска и дата одной записи в дереве совпали, то происходит вывод найденной и последующих записей.

Процедуры и функции кодирования базы данных:

17. void Encode(); - основная процедура кодирования.

18. std::map<char, float> get_probability(const std::vector<char> symbols); - функция вычисления вероятностей встречи элементов в базе данных. В качестве параметров принимает диманический массив символов без повторения symbols типа std::vector<char>. В результате функция возвращает упорядоченный контейнер в котором ключом является символ, а значением - его вероятности встречи.

19. std::vector<std::pair<char, float>> decrease_probab(const std::map<char, float> probab); - заполнение динамического массива типа std::vector<std::pair<char, float>> парами из контейнера и последующая сортировка массива по вероятностям. В качестве параметров принимает контейнер probab типа std::map<char, float>.

20. float Entropy(const std::vector<std::pair<char, float>> probab); - функция вычисления энтропии базы данных. В качестве параметров принимает диманический массив probab типа std::vector состоящий из пар элементов типа char и float.

21. void Fano(int L, int R, int k, const std::vector<std::pair<char, float>> P, std::vector<std::pair<char, size_t>> &Len, std::vector<std::vector<int>> &C); - основная процедура кодирования используя код Фано. В качестве параметров принимает левую границу динамического массива L типа int и правую границу R типа int; индекс для столбца в динамическом массиве для построения кодовых слов k типа int; динамический массив с вероятностями символов базы данных P типа std::vector<std::pair<char, float>>; динамический массив с длинами кодовых слов символов базы данных L типа std::vector<std::pair<char, size_t>>; динамический массив для хранения кодовых слов C типа std::vector<std::vector<float>>.

22. int MED(int L, int R, const std::vector<std::pair<char, float>> P); - функция поиска медианы в динамическом массиве. В качестве параметров принимает индекс левой границы массива L типа int и индекс правой границы R типа int, динамический массив из пар, хранящих символ и его вероятность, P типа std::vector<std::pair<char, float>>.

23. std::vector<std::pair<char, std::string>> Fano_decode(const std::vector<std::pair<char, float>> &P, const std::vector<std::vector<int>> &C); - функция преобразования кодовых слов из целочисленного типа в строку для удобного вывода. В качестве параметров принимает динамический массив из пар, хранящих символ и его вероятность, P типа std::vector<std::pair<char, float>> и динамический массив

кодовых слов C типа динамический массив из пар, хранящих символ и его вероятность, P типа std::vector<std::vector<int >>.

24. float average_length(const std::vector<std::pair<char, float>> P, const std::vector<std::pair<char, size_t>> L); - функция вычисления средней длины кодового слова. В качестве параметров принимает динамический массив из пар, хранящих символ и его вероятность, P типа std::vector<std::pair<char, float>> и динамический массив из пар, хранящих символ и длины его кодового слова, L типа std::vector<std::pair<char, size_t>>.

25. void Table(const std::vector<std::pair<char, float>> P, const std::vector<std::pair<char, size_t>> L, const std::vector<std::pair<char, std::string>> C); - процедура для вывода таблицы с символом из базы данных, вероятностью его встречи, длиной кодового слова и кодового слова. В качестве параметров принимает динамический массив из пар, хранящих символ и его вероятность, P типа std::vector<std::pair<char, float>>, динамический массив из пар, хранящих символ и длины его кодового слова, L типа std::vector<std::pair<char, size_t>> и динамический массив из пар, хранящих символ и его кодового слова, C типа std::vector<std::pair<char, std::string>>.

26. void Entrophy_table(float entrophy, float length); - процедура для вывода таблицы с энтропией базы данных, средней длиной кодового слова и избыточностью кодировки. В качестве параметров принимает значение энтропии entrophy типа float и значение средней длины кодового слова length типа float.

Основная программа:

21. main() – основная программа, в которой вызывается вспомогательная процедура loop() и остальные процедуры для работы с базой данных.

5. ТЕКСТ ПРОГРАММЫ

```
#include <iostream>
#include <stdio.h>
#include <fstream>
#include <conio.h>
#include <stdlib.h>
#include <cstring>
#include <string>
#include <vector>
#include <algorithm>
#include <math.h>
#include <map>
#include <iomanip>

bool HR;
bool VR;

struct Data
{
   char full_name[30];
   unsigned short int amount;
   char date[10];
   char lawyer[22];
};

typedef struct Node
{
   struct Node *next;
   Data *memory;
} list;

typedef struct queue
{
```

```
    list *head = NULL;
    list *tail = NULL;
} line;

struct Vertex
{
    Data *data;
    Vertex *left;
    Vertex *right;
    Vertex *equal;
    int bal;
};

list *load_to_memory(Data *dataArr, int numStruct)
{
    list *head;
    list *tail;
    head = new list;
    tail = new list;
    head->next = tail;
    delete head;
    tail->next = NULL;
    list *current;
    for (int i = 0; i < numStruct; i++)
    {
        current = new list;
        current->memory = &dataArr[i];
        tail->next = current;
        tail = current;
    }
    tail->next = NULL;
    return head;
}

void show_page(Data *dataArr, int numStruct)
{
    int ind = 0;
    char ch;
    while (true)
    {
        for (int i = ind; i < ind + 20; i++)
        {
            std::cout << i + 1 << " " << dataArr[i].full_name << "  ";
            std::cout << dataArr[i].amount << "  ";
            std::cout << dataArr[i].date << "  ";
            std::cout << dataArr[i].lawyer << "  ";
            std::cout << "\n";
        }
        std::cout << "D or d - next page, A or a - previous page, E or e - exit" << std::endl
                << std::endl;
        ch = _getch();
        switch (ch)
        {
        case 'D':
        case 'd':
            ind += 20;
            break;
        case 'A':
        case 'a':
            ind -= 20;
            break;
        case 'E':
        case 'e':
            return;
        default:
            break;
```

```
        }
        if (ind < 0)
        {
            ind = numStruct - 20;
        }
        else if (ind > numStruct - 20)
        {
            ind = 0;
        }
    }
}

void show_page_sort(Data *indArr[], int numStruct)
{
    int ind = 0;
    char ch;
    while (true)
    {
        for (int i = ind; i < ind + 20; i++)
        {
            std::cout << i + 1 << " " << indArr[i]->full_name << "  ";
            std::cout << indArr[i]->amount << "  ";
            std::cout << indArr[i]->date << "  ";
            std::cout << indArr[i]->lawyer << "  ";
            std::cout << "\n";
        }
        std::cout << "D or d - next page, A or a - previous page, E or e - exit" << std::endl
                << std::endl;
        ch = _getch();
        switch (ch)
        {
        case 'D':
        case 'd':
            ind += 20;
            break;
        case 'A':
        case 'a':
            ind -= 20;
            break;
        case 'E':
        case 'e':
            return;
        default:
            break;
        }
        if (ind < 0)
        {
            ind = numStruct - 20;
        }
        else if (ind > numStruct - 20)
        {
            ind = 0;
        }
    }
}

void clear(list *head)
{
    list *q, *current;
    q = current = head;
    while (current != NULL)
    {
        current = q->next;
        delete q;
        q = current;
    }
```

```
    head = NULL;
}

void from_list_to_line(list *&list, line &queue)
{
    queue.tail->next = list;
    queue.tail = list;
    list = list->next;
}

int compare(Data *a, Data *b)
{
    if (a->amount == b->amount)
    {
        char d[10];
        strcpy(d, a->date);
        d[0] = a->date[6];
        d[1] = a->date[7];
        d[6] = a->date[0];
        d[7] = a->date[1];

        char f[10];
        strcpy(f, b->date);
        f[0] = b->date[6];
        f[1] = b->date[7];
        f[6] = b->date[0];
        f[7] = b->date[1];

        return strcmp(d, f);
    }
    return a->amount - b->amount;
}

void Merge(list *&A, int &q, list *&B, int &r, line &C)
{
    while ((q != 0) && (r != 0))
    {
        if (compare(A->memory, B->memory) < 0)
        {
            from_list_to_line(A, C);
            q = q - 1;
        }
        else
        {
            from_list_to_line(B, C);
            r = r - 1;
        }
    }
    while (q > 0)
    {
        from_list_to_line(A, C);
        q = q - 1;
    }
    while (r > 0)
    {
        from_list_to_line(B, C);
        r = r - 1;
    }
}

int Split(list *&S, list *&A, list *&B)
{
    list *k, *p;
    int n = 1;
    A = S, B = S->next;
    k = A, p = B;
```

```
    while (p != NULL)
    {
        n++;
        k->next = p->next;
        k = p;
        p = p->next;
    }
    return n;
}

void merge_sort(list *&S)
{
    int q = 0, r = 0;
    list *A, *B;
    A = B = NULL;
    line C[2];
    int length = Split(S, A, B);
    int p = 1;
    while (p < length)
    {
        for (int i = 0; i < 2; i++)
        {
            C[i].tail = (list *)&C[i].head;
        }
        int i = 0, m = length;
        while (m > 0)
        {
            if (m >= p)
                q = p;
            else
                q = m;
            m = m - q;
            if (m >= p)
                r = p;
            else
                r = m;
            m = m - r;
            Merge(A, q, B, r, C[i]);
            i = 1 - i;
        }
        A = C[0].head;
        B = C[1].head;
        p = 2 * p;
    }

    C[0].tail->next = NULL;
    S = C[0].head;
}

int strcomp(const std::string &str1, const std::string &str2)
{
    std::string temp_1;
    std::string temp_2;

    temp_1 = str1;

    temp_1[0] = str1[6];
    temp_1[1] = str1[7];
    temp_1[6] = str1[0];
    temp_1[7] = str1[1];

    temp_2 = str2;

    temp_2[0] = str2[6];
    temp_2[1] = str2[7];
    temp_2[6] = str2[0];
```

```
    temp_2[7] = str2[1];

    int year1 = atoi(strtok(const_cast<char *>(temp_1.c_str()), "-"));
    int month1 = atoi(strtok(NULL, "-"));
    int day1 = atoi(strtok(NULL, "-"));

    int year2 = atoi(strtok(const_cast<char *>(temp_2.c_str()), "-"));
    int month2 = atoi(strtok(NULL, "-"));
    int day2 = atoi(strtok(NULL, "-"));

    if (year1 < year2)
    {
        return 1;
    }
    else if (year1 > year2)
    {
        return -1;
    }
    else if (year1 == year2)
    {
        if (month1 < month2)
        {
            return 1;
        }
        else if (month1 > month2)
        {
            return -1;
        }
        else if (month1 == month2)
        {
            if (day1 < day2)
            {
                return 1;
            }
            else if (day1 > day2)
            {
                return -1;
            }
            else if (day1 == day2)
            {
                return 0;
            }
        }
    }
    return 0;
}

void add_node_tree(Data *data, Vertex *&p)
{
    Vertex *q;
    if (p == NULL)
    {
        p = new Vertex;
        p->data = data;
        p->left = p->right = p->equal = NULL;
        p->bal = 0;
        VR = true;
    }
    else if (strcomp(p->data->date, data->date) < 0)

    {
        add_node_tree(data, p->left);
        if (VR == true)
        {
            if (p->bal == 0)
            {
```

```
            q = p->left;
            p->left = q->right;
            q->right = p;
            p = q;
            q->bal = 1;
            VR = false;
            HR = true;
         }
         else
         {
            p->bal = 0;
            VR = 1;
            HR = 0;
         }
      }
      else
      {
         HR = 0;
      }
   }
   else if (strcomp(p->data->date, data->date) > 0)
   {
      add_node_tree(data, p->right);
      if (VR == true)
      {
         p->bal = 1;
         HR = true;
         VR = false;
      }
      else if (HR == true)
      {
         if (p->bal == 1)
         {
            q = p->right;
            p->bal = 0;
            q->bal = 0;
            p->right = q->left;
            q->left = p;
            p = q;
            VR = true;
            HR = false;
         }
         else
         {
            HR = false;
         }
      }
   }
   else if (strcomp(p->data->date, data->date) == 0)
   {
      add_node_tree(data, p->equal);
   }
}

int BSearch(Data *indArr[], int numStruct, int key)
{
   int L = 0, R = numStruct - 1, m, find = -1;
   while (L < R)
   {
      m = (L + R) / 2;
      if (indArr[m]->amount < key)
      {
         L = m + 1;
      }
      else
      {
```

```
            R = m;
        }
    }
    if (indArr[R]->amount == key)
    {
        std::cout << R + 1 << " " << indArr[R]->full_name << " " << indArr[R]->amount << " " << indArr[R]->date << " " <<
indArr[R]->lawyer << std::endl;
        find = R;
    }
    else
    {
        find = -1;
    }
    return find;
}

void BSearchAll(Data *indArr[], int ind, int key, Vertex *&Root)
{
    for (int i = ind + 1; indArr[i]->amount == key; i++)
    {
        std::cout << i + 1 << " " << indArr[i]->full_name << "  " << indArr[i]->amount << "  " << indArr[i]->date << "  " << indArr[i]-
>lawyer << std::endl;
        add_node_tree(indArr[i], Root);
    }
}

void TreeRight(Vertex *Root, int &count)
{

    if (Root != NULL)
    {
        TreeRight(Root->left, count);
        TreeRight(Root->equal, count);
        std::cout << count++ << " " << Root->data->full_name << "  " << Root->data->amount << "  " << Root->data->date << "  " <<
Root->data->lawyer << std::endl;
        TreeRight(Root->right, count);
    }
}

void search_tree(Vertex *Root, const std::string &key, int &count)
{

    if (Root)
    {

        if (strcomp(Root->data->date, key) == -1)
        {
            search_tree(Root->left, key, count);
        }
        else if (strcomp(Root->data->date, key) == 1)
        {
            search_tree(Root->right, key, count);
        }
        else if (strcomp(Root->data->date, key) == 0)
        {
            while (Root)
            {
                std::cout << count++ << " " << Root->data->full_name << "  " << Root->data->amount << "  " << Root->data->date << "  "
<< Root->data->lawyer << std::endl;
                Root = Root->equal;
            }
        }
    }
}

void Sort_probability(std::vector<char> &A, std::vector<float> &P, int L, int R)
```

```
{
    float x = P[(L + R) / 2];
    int i = L;
    int j = R;
    while (i <= j)
    {
        while (P[i] > x)
        {
            i++;
        }
        while (P[j] < x)
        {
            j--;
        }
        if (i <= j)
        {
            std::swap(A[i], A[j]);
            std::swap(P[i], P[j]);
            i++;
            j--;
        }
    }
    if (L < j)
        Sort_probability(A, P, L, j);
    if (i < R)
        Sort_probability(A, P, i, R);
}

float Entrophy(const std::vector<float> probab)
{
    float entrophy = 0;
    for (size_t i = 0; i < probab.size(); i++)
    {
        entrophy += -probab[i] * log2f(probab[i]);
    }
    return entrophy;
}

std::vector<int> calc_length(const std::vector<float> P)
{
    std::vector<int> L(P.size());
    for (size_t i = 0; i < P.size(); i++)
    {
        L[i] = 0;
    }
    return L;
}

int MED(int L, int R, const std::vector<float> P)
{
    int med = 0;
    float Sl = 0;
    float Sr = 0;

    for (int i = L; i <= R; i++)
    {
        Sl = Sl + P[i];
    }
    Sr = P[R];
    med = R;
    while (Sl >= Sr)
    {
        med = med - 1;
        Sl = Sl - P[med];
        Sr = Sr + P[med];
    }
```

```
    return med;
}

void Fano(int L, int R, int k, const std::vector<float> P, std::vector<int> &Len, std::vector<std::vector<int>> &C)
{
    if (L < R)
    {
        ++k;
        int m = MED(L, R, P);
        for (int i = L; i <= R; i++)
        {
            if (i <= m)
            {
                C[i][k] = 0;
            }
            else
            {
                C[i][k] = 1;
            }
            ++Len[i];
        }
        Fano(L, m, k, P, Len, C);
        Fano(m + 1, R, k, P, Len, C);
    }
}

std::vector<std::string> Fano_decode(const std::vector<float> &P, const std::vector<std::vector<int>> &C)
{
    std::vector<std::string> decode(P.size());
    std::string decodeStr = "";
    int iter = 0;

    for (auto word : C)
    {
        for (int i = 0; i < word[word.size() - 1]; i++)
        {
            decodeStr += std::to_string(word[i]);
        }
        decode[iter] = decodeStr;
        decodeStr = "";
        iter++;
    }
    return decode;
}

float avarage_length(const std::vector<float> P, const std::vector<int> L)
{
    float length = 0;
    for (size_t i = 0; i < P.size(); i++)
    {
        length += P[i] * L[i];
    }
    return length;
}

void Table(const std::vector<char> A, const std::vector<float> P, const std::vector<int> L, const std::vector<std::string> C)
{
    std::cout << "Symbol: |" << std::setw(10) << "Probability of a symbol: |" << std::setw(15) << "Code word:  |" << std::setw(10) <<
"Length of the codeword" << std::endl;
    for (size_t i = 0; i < P.size(); i++)
    {
        std::cout << "    " << A[i] << "     |" << std::setw(18) << P[i] << "       |" << std::setw(10) << C[i] << "     |" << std::setw(10) <<
L[i] << std::endl;
    }
}
```

```
void Entrophy_table(float entrophy, float length)
{
   std::cout << "Entrophy: |" << std::setw(10) << "Avarage length: |" << std::setw(10) << "Code redundancy " << std::endl;
   std::cout << entrophy << "   |" << std::setw(10) << length << "      |" << std::setw(15) << std::setw(15) << length - entrophy <<
std::endl
            << std::endl;
}

void Encode()
{
   std::vector<char> S;
   std::vector<char> A;
   std::vector<float> P;
   std::vector<int> L;

   int size = 0;

   std::ifstream file("testBase3.dat", std::ios::binary);

   char *buf = NULL;

   file.seekg(0, std::ios::end);
   size = file.tellg();
   file.seekg(0, std::ios::beg);

   buf = new char[size];
   auto i = 0;
   while (file.read(&buf[i], 1))
   {
      ++i;
   }
   file.close();

   for (int i = 0; i < size; i++)
   {
      S.push_back(buf[i]);
      // std::cout << buf[i];
      if (std::find(A.begin(), A.end(), buf[i]) == A.end())
      {
         A.push_back(buf[i]);
      }
   }

   std::cout << S.size() << " " << A.size() << std::endl;

   for (size_t i = 0; i < A.size(); i++)
   {
      P.push_back(static_cast<float>(std::count(S.begin(), S.end(), A[i])) / S.size());
   }

   Sort_probability(A, P, 0, A.size() - 1);

   float entrophy = Entrophy(P);
   L = calc_length(P);
   std::vector<std::vector<int>> C(P.size(), std::vector<int>(P.size()));
   std::vector<std::string> Code;

   Fano(0, P.size() - 1, -1, P, L, C);

   for (size_t i = 0; i < P.size(); i++)
   {
      C[i][P.size() - 1] = L[i];
   }

   Code = Fano_decode(P, C);
```

```
    float avarage = avarage_length(P, L);

    Table(A, P, L, Code);
    Entrophy_table(entrophy, avarage);
}

void loop(Data *dataArr, Data *indArr[], int numStruct)
{
    int key;
    int ind = -1, count = 1;
    Vertex *Root = NULL;
    std::string temp;
    while (true)
    {
        std::string chose;
        std::cout << "1. Show unsorted list\n"
                  "2. Show sorted list\n"
                  "3. Search\n"
                  "4. Show b-Tree\n"
                  "5. Fano Encode DB\n"
                  "0 or Any key: EXIT\n\n> ";
        std::cin >> chose;
        switch (chose[0])
        {
        case '1':
            show_page(dataArr, numStruct);
            break;
        case '2':
            show_page_sort(indArr, numStruct);
            break;
        case '3':
        {

            std::cout << "Enter amount: ";
            std::cin >> key;
            ind = BSearch(indArr, numStruct, key);
            if (ind == -1)
            {
                std::cout << "Wrong key!!" << std::endl;
                break;
            }
            BSearchAll(indArr, ind, key, Root);
        }
        break;
        case '4':
            if (ind == -1)
            {
                std::cout << "Search first!";
                break;
            }
            else
            {
                TreeRight(Root, count);
                std::cout << std::endl;
                std::cout << "Enter date in format: 'dd-mm-yy': ";
                std::cin >> temp;
                std::cout << std::endl;
                count = 1;
                search_tree(Root, temp, count);
                std::cout << std::endl;
            }
            break;
        case '5':
            Encode();
            break;
        case '0':
```

```
            return;
        default:
            return;
        }
    }
}

int main()
{

    std::ifstream file("testBase3.dat", std::ios::binary);
    if (!file.is_open())
    {
        std::cout << "File isn't open!" << std::endl;
        return 1;
    }
    file.seekg(0, std::ios::end);
    int fileSize = file.tellg();
    file.seekg(0, std::ios::beg);

    int numStruct = fileSize / sizeof(Data);

    Data *dataArr = new Data[numStruct];
    Data *indArr[numStruct];

    file.read(reinterpret_cast<char *>(dataArr), fileSize);

    list *head = load_to_memory(dataArr, numStruct);

    merge_sort(head);

    list *current = head;

    for (int i = 0; current; i++)
    {
        indArr[i] = current->memory;
        current = current->next;
    }

    loop(dataArr, indArr, numStruct);

    clear(head);
    delete[] dataArr;

    file.close();
    return 0;
}
```

```
1. Show unsorted list
2. Show sorted list
3. Search
4. Show b-Tree
5. Fano Encode DB
0 or Any key: EXIT

>
```

Рисунок 1. Меню программы

```
 1 Власов Патрик Ромуальдович      15000  23-02-96   Патриков Ж В
 2 Хасанов Муамар Ахиллесович      15000  18-07-96   Янов О С
 3 Власов Поликарп Батырович       40000  21-10-93   Ахиллесов Н О
 4 Поликарпов Герасим Власович     30000  08-01-94   Глебов Н Е
 5 Ахмедов Влас Никодимович        50000  17-03-95   Патриков Ж К
 6 Зосимов Ахмед Евграфович        10000  08-09-95   Батырова О Й
 7 Герасимов Герасим Янович        10000  22-07-93   Глебов Н Е
 8 Евграфов Остап Феофанович       30000  14-01-96   Батырова О Й
 9 Ахиллесов Филимон Демьянович    35000  21-10-94   Батырова О Й
10 Янов Ахиллес Герасимович        25000  17-09-94   Патриков Ж К
11 Патриков Клим Янович            15000  08-10-97   Глебов У Л
12 Власов Влас Архипович           25000  23-02-96   Глебов Н Е
13 Феофанова Изольда Евграфовна    35000  23-02-93   Янов И Ж
14 Тихонов Филимон Демьянович      50000  26-02-94   Архипов П В
15 Герасимов Остап Власович        40000  26-12-94   Патриков Ж К
16 Глебов Жак Остапович            40000  24-11-95   Архипов П В
17 Никодимо Поликарп Никодимович   45000  17-09-93   Глебов У Л
18 Герасимова Степанида Яновна     25000  07-04-94   Янов О С
19 Пантелемоно Евграф Гедеонович   25000  25-03-95   Патриков Ж В
20 Евграфо Пантелемон Патрикович   45000  22-07-96   Янов И Ж
D or d - next page, A or a - previous page, E or e - exit
```

Рисунок 2. Неотсортированная база данных

```
1 Власов Ахмед Ромуальдович       5000  01-01-93   Архипов П В
2 Гедеонов Архип Пантелемонович   5000  01-01-93   Глебов У Л
3 Муамаров Батыр Тихонович        5000  08-01-93   Янов О С
4 Жаков Патрик Герасимович        5000  08-01-93   Патриков Ж К
5 Ахиллесов Патрик Герасимович    5000  09-01-93   Глебов У Л
6 Гедеонова Нинель Герасимовна    5000  13-01-93   Феофанова К В
7 Глебова Фекла Сабировна         5000  14-01-93   Батырова О Й
8 Поликарпов Ян Пантелемонович    5000  18-01-93   Янов И Ж
9 Жаков Ян Демьянович             5000  26-01-93   Янов И Ж
10 Ахиллесова Алсу Мстиславовна    5000  18-02-93   Янов И Ж
11 Герасимов Василиса Евграфовна   5000  26-02-93   Янов И Ж
12 Александрова Ада Глебовна       5000  26-02-93   Янов И Ж
13 Батыров Евграф Жакович          5000  27-02-93   Феофанова К В
14 Янов Сабир Глебович             5000  27-02-93   Феофанова К В
15 Демьянов Ромуальд Глебович      5000  08-03-93   Янов О С
16 Остапов Хасан Муамарович        5000  09-03-93   Патриков Ж В
17 Герасимо Виолетта Никодимовна   5000  09-03-93   Патриков Ж К
18 Ахиллесова Фекла Глебовна       5000  17-03-93   Архипов П В
19 Глебов Ромуальд Янович          5000  25-03-93   Батырова О Й
20 Янова Изабелла Тихоновна        5000  03-04-93   Ахиллесов Н О
D or d - next page, A or a - previous page, E or e - exit
```

Рисунок 3. Отсортированная база по сумме вклада и дате.

```
391 Гедеонов Гедеон Сабирович       5000  07-09-97   Феофанова К В
392 Мстислав Саломея Мстиславовна   5000  13-09-97   Патриков Ж В
393 Пантелемонов Клим Сабирович     5000  13-09-97   Глебов Н Е
394 Жаков Тихон Зосимович           5000  14-09-97   Патриков Ж В
395 Ахиллесов Изабелла Демьяновна   5000  18-09-97   Архипов П В
396 Патрикова Арабелла Гедеоновна   5000  25-09-97   Янов И Ж
397 Хасанов Жак Патрикович          5000  06-10-97   Ахиллесов Н О
398 Никодимова Варвара Глебовна     5000  06-10-97   Ахиллесов Н О
399 Мстиславов Феофан Остапович     5000  10-10-97   Феофанова К В
400 Архипов Остап Батырович         5000  13-10-97   Батырова О Й
401 Поликарпо Евграф Ромуальдович   5000  13-10-97   Глебов Н Е
402 Мстиславов Архип Глебович       5000  19-10-97   Батырова О Й
403 Евграфова Варвара Филимоновна   5000  19-10-97   Янов О С
404 Патрикова Виолетта Жаковна      5000  19-10-97   Феофанова К В
405 Глебов Александр Демьянович     5000  24-10-97   Янов И Ж
406 Остапов Поликарп Жакович        5000  02-11-97   Феофанова К В
407 Феофанов Герасим Мстиславович   5000  05-11-97   Патриков Ж В
408 Мстиславо Поликарп Демьянович   5000  05-11-97   Янов О С
409 Поликарпова Ада Тихоновна       5000  05-11-97   Ахиллесов Н О
410 Сабиров Герасим Мстиславович    5000  08-11-97   Янов И Ж
411 Патриков Батыр Ромуальдович     5000  08-11-97   Патриков Ж К
412 Жакова Саломея Сабировна        5000  08-11-97   Глебов Д Й
413 Александр Гедеон Поликарпович   5000  24-11-97   Ахиллесов Н О
414 Муамарова Ариадна Герасимовна   5000  24-11-97   Батырова О Й
415 Климов Феофан Батырович         5000  24-11-97   Глебов Д Й
416 Патриков Александр Патрикович   5000  25-11-97   Янов О С
417 Поликарпов Изабелла Хасановна   5000  01-12-97   Патриков Ж К
418 Муамаров Архип Демьянович       5000  01-12-97   Янов И Ж
419 Янов Евграф Хасанович           5000  06-12-97   Феофанова К В
420 Филимонов Александр Остапович   5000  09-12-97   Янов О С
421 Остапо Пелагея Пантелемоновна   5000  21-12-97   Янов О С
422 Евграфов Пантелемон Остапович   5000  21-12-97   Феофанова К В
423 Евграфов Пантелемон Хасанович   5000  26-12-97   Янов И Ж
```

Рисунок 4. Очередь из записей, полученных в результате поиска (сумма вклада равная 5000)

```
391 Пантелемонов Клим Сабирович     5000  13-09-97   Глебов Н Е
392 Мстислав Саломея Мстиславовна   5000  13-09-97   Патриков Ж В
393 Жаков Тихон Зосимович           5000  14-09-97   Патриков Ж В
394 Ахиллесов Изабелла Демьяновна   5000  18-09-97   Архипов П В
395 Патрикова Арабелла Гедеоновна   5000  25-09-97   Янов И Ж
396 Никодимова Варвара Глебовна     5000  06-10-97   Ахиллесов Н О
397 Хасанов Жак Патрикович          5000  06-10-97   Ахиллесов Н О
398 Мстиславов Феофан Остапович     5000  10-10-97   Феофанова К В
399 Поликарпо Евграф Ромуальдович   5000  13-10-97   Глебов Н Е
400 Архипов Остап Батырович         5000  13-10-97   Батырова О Й
401 Патрикова Виолетта Жаковна      5000  19-10-97   Феофанова К В
402 Евграфова Варвара Филимоновна   5000  19-10-97   Янов О С
403 Мстиславов Архип Глебович       5000  19-10-97   Батырова О Й
404 Глебов Александр Демьянович     5000  24-10-97   Янов И Ж
405 Остапов Поликарп Жакович        5000  02-11-97   Феофанова К В
406 Поликарпова Ада Тихоновна       5000  05-11-97   Ахиллесов Н О
407 Мстиславо Поликарп Демьянович   5000  05-11-97   Янов О С
408 Феофанов Герасим Мстиславович   5000  05-11-97   Патриков Ж В
409 Жакова Саломея Сабировна        5000  08-11-97   Глебов Д Й
410 Патриков Батыр Ромуальдович     5000  08-11-97   Патриков Ж К
411 Сабиров Герасим Мстиславович    5000  08-11-97   Янов И Ж
412 Климов Феофан Батырович         5000  24-11-97   Глебов Д Й
413 Муамарова Ариадна Герасимовна   5000  24-11-97   Батырова О Й
414 Александр Гедеон Поликарпович   5000  24-11-97   Ахиллесов Н О
415 Патриков Александр Патрикович   5000  25-11-97   Янов О С
416 Муамаров Архип Демьянович       5000  01-12-97   Янов И Ж
417 Поликарпов Изабелла Хасановна   5000  01-12-97   Патриков Ж К
418 Янов Евграф Хасанович           5000  06-12-97   Феофанова К В
419 Филимонов Александр Остапович   5000  09-12-97   Янов О С
420 Евграфов Пантелемон Остапович   5000  21-12-97   Феофанова К В
421 Остапо Пелагея Пантелемоновна   5000  21-12-97   Янов О С
422 Евграфов Пантелемон Хасанович   5000  26-12-97   Янов И Ж
```

Рисунок 5. Двоичное Б-дерево, ключ в дереве - дата вклада

```
Enter date in format: 'dd-mm-yy': 08-11-97

1 Сабиров Герасим Мстиславович   5000  08-11-97   Янов И Ж
2 Патриков Батыр Ромуальдович    5000  08-11-97   Патриков Ж К
3 Жакова Саломея Сабировна       5000  08-11-97   Глебов Д Й
```

Рисунок 6. Поиск по дереву

Symbol:	Probability of a symbol:	Code word:	Length of the cod
	0.289316	00	2
о	0.0618672	0100	4
а	0.050625	0101	4
в	0.0488125	0110	4
	0.046875	01110	5
и	0.0413945	01111	5
-	0.03125	10000	5
л	0.0264531	10001	5
е	0.0258008	10010	5
н	0.0242734	100110	6
т	0.021293	100111	6
р	0.0211602	10100	5
9	0.0190234	101010	6
с	0.0157383	101011	6
м	0.0155039	101100	6
1	0.0136055	101101	6
т	0.0128438	101110	6
ч	0.0110625	101111	6
к	0.010543	110000	6
А	0.00979297	1100010	7
х	0.00898828	1100011	7
Г	0.00883203	110010	6
П	0.0088125	1100110	7
д	0.00849219	1100111	7
б	0.00801172	1101000	7
п	0.00777734	1101001	7
2	0.00759375	1101010	7
В	0.00665234	1101011	7
7	0.00620313	1101100	7
З	0.00617578	1101101	7
б	0.00611719	1101110	7

З	0.00617578	1101101	7
б	0.00611719	1101110	7
4	0.00589453	11011110	8
Ж	0.00580078	11011111	8
О	0.00578906	1110000	7
5	0.00559766	1110001	7
И	0.00502734	1110010	7
ф	0.00467578	11100110	8
Ф	0.00465625	11100111	8
Н	0.00461719	1110100	7
К	0.00442187	11101010	8
Я	0.00425781	11101011	8
8	0.00374219	11101100	8
М	0.00372266	11101101	8
С	0.00360938	11101110	8
ь	0.00358203	11101111	8
у	0.00330078	11110000	8
ы	0.003	11110001	8
Б	0.003	11110010	8
Д	0.00298437	111100110	9
Е	0.00291406	111100111	9
Й	0.00290625	11110100	8
я	0.00217187	111101010	9
г	0.00174609	111101011	9
:	0.00169141	11110110	8
Ш	0.00169141	111101110	9
►	0.00167969	111101111	9
'	0.00167969	111110000	9
‼	0.00165234	111110001	9
Т	0.00160547	111110010	9
@	0.00158203	111110011	9
Ь	0.00158203	111110100	9
Р	0.00156641	111110101	9
├	0.00156641	111110110	9
╚	0.00153125	1111101110	10
З	0.00152734	1111101111	10
X	0.00152344	111111000	9
N	0.00151563	111111001	9
a	0.00151563	111111010	9
u	0.00149609	111111011	9
Р	0.00146875	111111100	9
У	0.0014375	111111101	9
Л	0.0014375	111111110	9
╕	0.00139453	1111111110	10
з	0.000546875	1111111111	10

Entrophy:	Avarage length:	Code redundancy
4.63419	4.67381	0.0396237

Рисунки 7-9. Примеры кодовых слов 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.