

Министерство цифрового развития, связи и массовых коммуникаций Российской
Федерации
СибГУТИ

Кафедра ПМиК

КУРСОВОЙ ПРОЕКТ
“Структуры и алгоритмы обработки данных”
ВАРИАНТ 140

Выполнил:
студент группы ИП-216
Русецкий А.С.

Проверил:
Старший преподаватель
Кафедры ПМиК
Солодов П.С

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	4
2.4. Метод кодирования	5
3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ АЛГОРИТМОВ.....	6
4. ОПИСАНИЕ ПРОГРАММЫ	8
4.1. Основные переменные	8
4.2. Описание подпрограмм.....	9
5. ТЕКСТ ПРОГРАММЫ	10
6. РЕЗУЛЬТАТЫ	19
7. ВЫВОДЫ.....	23

1. ПОСТАНОВКА ЗАДАЧИ

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

Упорядочить данные по **номеру отдела и ФИО**, используя **Цифровую сортировку (Digital Sort)**. Упорядоченные данные вывести на экран.

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

Из записей очереди построить **дерево оптимального поиска методом A1 по номеру отдела**, 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 её с энтропией исходного файла.

База данных "Предприятие"

Структура записи:

ФИО сотрудника: текстовое поле 30 символа

формат <Фамилия>_<Имя>_<Отчество>

Номер отдела: целое число

Должность: текстовое поле 22 символа

Дата рождения: текстовое поле 10 символов

формат дд-мм-гг

Пример записи из БД:

Петров_Иван_Иванович_____

130

начальник_отдела_____

15-03-46

Вариант условий упорядочения и ключи поиска (К):

$S = 1$ - по номеру отдела и ФИО, K = номер отдела;

Ключ в дереве – первые 3 буквы фамилии.

2. ОСНОВНЫЕ ИДЕИ И ХАРАКТЕРИСТИКИ ПРИМЕНЯЕМЫХ МЕТОДОВ

2.1. Метод сортировки

Цифровая сортировка (Digital Sort)

Цифровая сортировка является одним из методов сортировки последовательностей.

Пусть дана последовательность из S чисел, представленных в m – ичной системе счисления. Каждое число состоит из L цифр $d_1d_2\dots d_L$, $0 \leq d_i \leq m - 1$, $i=1..L$. 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. Приведем более детальный алгоритм цифровой сортировки.

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.

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:

1. Выбранный элемент равен X . Поиск завершён.
2. Выбранный элемент меньше X . Продолжаем поиск в правой половине массива.
3. Выбранный элемент больше X . Продолжаем поиск в левой половине массива.

Дадим верхнюю оценку трудоёмкости алгоритма двоичного поиска. 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

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

Припишем каждой вершине дерева V_i вес w_i , пропорциональный частоте поиска этой вершины (например, если из каждых 100 операций поиска 15 операций приходятся на вершину V_1 , то $w_1=15$). Сумма весов всех вершин дает вес дерева W . Каждая вершина V_i расположена на высоте h_i , корень расположен на высоте 1. Высота вершины равна количеству операций сравнения, необходимых для поиска этой вершины. Определим средневзвешенную высоту дерева с n вершинами следующим образом: $h_{cp}=(w_1h_1+w_2h_2+\dots+w_nh_n)/W$. Дерево поиска, имеющее минимальную средневзвешенную высоту, называется деревом оптимального поиска (ДОП).

Алгоритм (А1) предлагает в качестве корня использовать вершину с наибольшим весом. Затем среди оставшихся вершин снова выбирается вершина с наибольшим весом и помещается в левое или правое поддереву в зависимости от ее значения, и т.д.

2.4. Метод кодирования

Алфавитный код Гилберта – Мура

Е. Н. Гилбертом и Э. Ф. Муром был предложен метод построения алфавитного кода, для которого $L_{cp} < H(p_1, \dots, p_n) + 2$. Пусть символы алфавита некоторым образом упорядочены, например, $a_1 \leq a_2 \leq \dots \leq a_n$. Код σ называется алфавитным, если кодовые слова лексикографически упорядочены, т.е. $\sigma(a_1) \leq \sigma(a_2) \leq \dots \leq \sigma(a_n)$.

Процесс построения кода происходит следующим образом.

1. Вычислим величины Q_i , $i=1, n$:

$$Q_1=p_1/2,$$

$$Q_2=p_1+p_2/2,$$

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

...

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

2. Представим суммы Q_i в двоичном виде.

3. В качестве кодовых слов возьмем $\lceil -\log p_i \rceil + 1$ младших бит в двоичном представлении Q_i , $i = 1, \dots, n$.

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

a_i	P_i	Q_i	L_i	кодовое 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

Рисунок 1 – Таблица «Код Гилберта-Мура»

Средняя длина кодового слова не превышает значения энтропии плюс 2. Действительно, $L_{cp} = 4 \cdot 0.18 + 4 \cdot 0.18 + 3 \cdot 0.36 + 5 \cdot 0.07 + 5 \cdot 0.09 + 5 \cdot 0.12 = 3.92 < 2.37 + 2$

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ АЛГОРИТМОВ

1. Интерфейс программы

Данный код реализует загрузку и вывод базы данных, представленной в бинарном файле "testBase2.dat". Сначала создается динамический массив указателей на структуру base, где каждый элемент Интерфейс программы реализован в бесконечном цикле с возможностью выхода из него, при нажатии клавиши «Esc». Выбор пункта меню реализован с помощью функции switch().

2. Особенности реализации бинарного поиска и построения очереди

База данных открывается в функции menu(). Считывается база данных в список структур в функции Read_base(FILE *fp, list *rbase), где struct base – список, в котором хранится база данных. Считывание производится независимо от желания пользователя, в то время как большинство остальных функций он может выбрать посредством меню. После считывания в список структур, файл закрывается.

За вывод элементов базы данных отвечает процедура void Print(list *base) которая представляет возможность просмотра базы данных постранично.

3. Вспомогательные функции и процедуры для сортировки данных

База данных сортируется после запуска программы. Для сортировки базы данных используется процедура void digital_sort(list *&head, int sort). Она сортирует список сначала по полю должности, а потом по ФИО. Для быстрого доступа к отсортированной и

неотсортированной базе данных, перед вызовом процедуры сортировки, делается копия списка, хранящего элементы базы данных, с помощью процедуры `void copy_base(list *a, list *b)`.

Особенности реализации бинарного поиска

Бинарный поиск по отсортированной базе данных осуществляется в процедуре `void BSearch(list **A, int Number)`. Результатом работы процедуры `void BSearch(list **A, int Number)` является индекс найденного элемента в индексном массиве, удовлетворяющий условию поиска. При отсутствии элементов с заданным ключом, программа выводит сообщение о том, что таких элементов нет.

Вспомогательные функции и процедуры для построения дерева оптимального поиска (приближенный алгоритм A1).

Построение дерева осуществляется в процедуре `void A1(int L, int R, list **mas)`. Записи заносятся в дерево в процедуре `void add_vertex(vertex *&p, list* mas, int w)`. Обход дерева слева направо осуществляется в процедуре `void LR_print(vertex *p, int& count)`. Поиск по дереву выполняется в процедуре `void TreeSearch(vertex* p, char* name)`.

4. Кодирование данных

Кодирование данных начинается с процедуры `void GillbertMoorCode()`, которая открывает файл базы данных для чтения, заполняет массив структур для алфавита кодовых слов всеми возможными символами, считывает из файла символы и считает их вероятности, закрывает файл, удаляет пустые символы, т.е. те, которые не встретились в файле и сортирует полученный алфавит по вероятностям. В процедуре считается длина кодового слова и само кодовое слово. В процедуре `void CodePrint()` осуществляется подсчет и вывод средней длины кодового слова, энтропии, а также выводит символы, их вероятности, длины кодовых слов и сами кодовые слова на монитор.

4. ОПИСАНИЕ ПРОГРАММЫ

4.1. Основные переменные

```
struct base
```

```
{  
    char FIO[30];  
    short int Department;  
    char Position[22];  
    char Date[10];  
};
```

Структура для хранения элемента базы данных. Всего таких элементов:

const int N = 4000 - размер базы данных.

```
struct list {  
    base *data;  
    list *next;  
    list *prev;  
};
```

Структура для создания списка, в котором хранятся элементы базы данных.

```
struct vertex {  
    base *data;  
    int w;  
    int h;  
    vertex *Next;  
    vertex *Left;  
    vertex *Right;  
};
```

Структура, представляющая дерево оптимального поиска (A2).

int *W – вес элементов, из которых строится дерево.

```
struct GM_code {  
    float p;  
    float q;
```



```

    int l;

    char a;

}

```

Структура, представляющая собой информацию о символе для формирования кодового слова.

GM_code A[M]; - массив для всех символов

GM_code B[alphabet_num]; - массив для всех появляющихся символов

const int M = 256 – число символов в алфавите

const int alphabet_num = 81 – количество элементов в итоговом алфавите, исключая пустые символы

float entropy = 0 - энтропия

float lgm = 0 – средняя длина кодового слова

int sum = 0 – счётчик всех символов в файле

4.2. Описание подпрограмм

Процедуры начальной обработки базы данных:

1. void Read_base(FILE *fp, list *rbase) – считывает базу данных и создает список.
2. Void Print(list *Base); - визуальный вывод базы данных.
3. Void Print_sort_mas(list **mas) – вывод отсортированной базы данных

Процедуры сортировки:

4. void copy_base(list *a, list *b) – делает копию списка, в котором хранится база данных.
5. void digital_sort(list *&head, int sort) – сортирует базу данных по номеру отдела и ФИО.

Процедуры и функции для поиска в отсортированной базе данных:

6. void BSearch(list **A, int Number) – бинарный поиск по ключу года.

Процедуры построения дерева оптимального поиска (A2):

7. void add_vertex(vertex *&p, list *mas, int w) – добавление элемента в дерево.
8. void A1(int L, int R, list **mas) – построение дерева оптимального поиска, приближенный алгоритм A1.

9. void LR_print(vertex *p, int& count) – вывод дерева.
10. void TreeSearch(vertex *p, char *data) – поиск в дереве.
11. void delete_tree(vertex* p, char* name) – удаление дерева.

Процедуры и функции кодирования базы данных:

12. void GilbertMoorCode() – считывание символов базы данных, подсчет их вероятностей и преобразование алфавита, создание кодовых слов.
13. void CodePrint() – вывод статистики и алфавита с вероятностями.
14. void Menu() – запуск интерфейса базы данных

Основная программа:

15. main() - основная программа, в которой происходит запуск меню.

5. ТЕКСТ ПРОГРАММЫ

```
#include <fstream>
#include <iostream>
#include <conio.h>
#include <Windows.h>
#include <iomanip>
#include <cstdio>
#include <cstring>
#include <cmath>
using namespace std;

const int N=4000;
const int M=256;
int sum=0;
int code[M][M];
float entropy = 0, lgm = 0;
int fcompression = 0, cfcompression = 0;
int *W;
const int alphabet_num=81;

struct base
{
    char FIO[30];
    short int Department;
    char Position[22];
    char Date[10];
};

struct list {
    base *data;
    list* next;
    list* prev;
};

};

struct Vertex {
    base *data;
    int w;
    int h;
    Vertex *Next;
    Vertex* Left;
    Vertex* Right;
};
Vertex *root = NULL;

struct GM_code {
    float p;
    float q;
    int l;
    char a;
};
GM_code A[M];
GM_code B[alphabet_num];

void Read_base(FILE *fp, list *rbase){
    base *struk = new base();
    fread((base *)struk, sizeof(base), 1, fp);
    rbase->data=struk;
    rbase->prev=NULL;
    rbase->next=NULL;

    for (int i = 1; i < N; i++) {
        struk = new base();
        list *rbase_prev;
```

```

fread((base*)struk, sizeof(base), 1, fp);
rbase_prev = rbase;
rbase = rbase->next = new list();
rbase->data = struk;
rbase->prev = rbase_prev;
rbase->next = NULL;
}
}

int BSearch (list **A, int Number) {
    int l = 0, r = N, m = 0, fam_pos=0;
    //char buffer[10];
    while (l < r) {
        m=(l+r)/2;
        //strcpy(buffer, A[m]->data->Date);
        //cout << &buffer[6] << endl;
        if (A[m]->data->Department ==
Number) {
            return m;
        }
        if (A[m]->data->Department <
Number)
            l = m+1;
        else r = m-1;
    }
    return -1;
}

int compare_name(char* a, char* b) {
    char bufferA[30];
    char bufferB[30];
    strcpy(bufferA, a);
    strcpy(bufferB, b);

    int space_pos_a = strchr(a, ' ') - a;
    int space_pos_b = strchr(b, ' ') - b;

    int count =4;

    if(space_pos_a < space_pos_b){
        count = space_pos_a;
    }
    else{
        count = space_pos_b;
    }
    count -= 1;

    if(strncmp(bufferA, bufferB, count) > 0){
        return 1;
    }
    if(strncmp(bufferA, bufferB, count) < 0){
        return -1;
    }

    if(space_pos_a > space_pos_b){
        return 1;
    }
}

}

if(space_pos_a < space_pos_b){
    return -1;
}

strcpy(bufferA, a);
strcpy(bufferB, b);
bufferA[space_pos_a]='a';
bufferB[space_pos_b]='a';
space_pos_a = strchr(bufferA, ' ') - bufferA;
space_pos_b = strchr(bufferB, ' ') - bufferB;

count = 4;

if(space_pos_a < space_pos_b){
    count = space_pos_a;
}
else{
    count = space_pos_b;
}
count -= 1;

if(strncmp(bufferA, bufferB, count) > 0){
    return 1;
}
if(strncmp(bufferA, bufferB, count) < 0){
    return -1;
}

if(space_pos_a > space_pos_b){
    return 1;
}
}

```

```

        if(space_pos_a >= space_pos_b){
            return 1;
        }
        return -1;
    }

void digital(list* h1, list* t1, list* h2, list* t2, list*& hr,
list*& tr, int sort) {
    if(sort == 0){
        if (compare_name(h1->data->FIO,
h2->data->FIO)>0) {
            hr = h1;
            h1 = h1->next;
        }
        else {
            hr = h2;
            h2 = h2->next;
        }
    }
    else{
        if (h1->data->Department < h2-
>data->Department) {
            hr = h1;
            h1 = h1->next;
        }
        else {
            hr = h2;
            h2 = h2->next;
        }
    }
    tr = hr;
    while (t1->next != h1 && t2->next != h2) {
        if(sort == 0){
            if (compare_name(h1->data-
>FIO, h2->data->FIO)>0) {
                tr->next = h1;
                h1 = h1->next;
                tr = tr->next;
            }
            else {
                tr->next = h2;
                h2 = h2->next;
                tr = tr->next;
            }
        }
        else{
            if (h1->data->Department <
h2->data->Department) {
                tr->next = h1;
                h1 = h1->next;
                tr = tr->next;
            }
            else {
                tr->next = h2;
                h2 = h2->next;
                tr = tr->next;
            }
        }
    }
}

```

```

    }
    if (t1->next != h1) {
        tr->next = h1;
        tr = t1;
    }
    else if (t2->next != h2) {
        tr->next = h2;
        tr = t2;
    }
    tr->next = NULL;
}

void digital_sort(list*& head, int sort) {
    list* t = new list;
    t->data = NULL;
    t->next = head;
    int k = 1;
    int e = 0;
    list* hp, * tp, * h1, * t1, * h2, * t2, * hr, * tr;
    while (k < 4000) {
        hp = t;
        while (hp != NULL) {
            if (hp->next == NULL) {
                break;
            }
            h1 = hp->next;
            t1 = h1;
            for (int i = 1; i < k; i++) {
                if (t1->next ==
NULL) {
                    break;
                }
                t1 = t1->next;
            }
            if (t1->next == NULL) {
                break;
            }
            h2 = t1->next;
            t2 = h2;
            t1->next = NULL;
            for (int i = 1; i < k; i++) {
                if (t2->next ==
NULL) {
                    break;
                }
                t2 = t2->next;
            }
            tp = t2->next;
            t1->next = NULL;
            t2->next = NULL;
            digital(h1, t1, h2, t2, hr,
tr, sort);
            hp->next = hr;
            head = hr;
            tr->next = tp;
            hp = tr;
        }
        k *= 2;
    }
}

```

```

void Print(list *Base) {
    int i, g, N = 4000;
    char n;
    SetConsoleCP(1251);

    i = 0;
    g = 0;
    bool exitLoop = false;
    printf("\n");
    printf("\n");
    printf("\n");
    printf("\n");
    while (!exitLoop) {
        for (i = g; i < N && i < g + 20; i++) {
            if (Base != nullptr) {
                std::cout << i + 1 << "\t" << Base->data-
>FIO << "\t" << Base->data->Department << "\t" <<
Base->data->Position << "\t" << Base->data->Date <<
std::endl;
                Base = Base->next;
            }
        }

        if (i >= N) {
            std::cout << "Reached the end of data." <<
std::endl;
            std::cout << "Press 'f' for forward, 'b' for
backward, 'e' to exit, 'n' to go to a specific record: " <<
endl;
            n = _getch();
        } else {
            std::cout << "Press 'f' for forward, 'b' for
backward, 'e' to exit, 'n' to go to a specific record: " <<
endl;
            n = _getch();
        }

        if (n == 'f') {
            g += 20;
        } else if (n == 'b') {
            g -= 20;
            if (g < 0) {
                g = 0;
            }
        } else if (n == 'e') {
            exitLoop = true;
        } else if (n == 'n') {
            std::cout << "\nEnter record number: ";
            int recordNumber;
            std::cin >> recordNumber;

            if (recordNumber > 0 && recordNumber <= N)
            {
                g = ((recordNumber - 1) / 20) * 20;
            } else {
                std::cout << "Invalid record number. Press
Enter to continue." << std::endl;
                while (_getch() != 'r') {}
            }
        }
    }
}

```

```

    } else {
        std::cout << "Invalid input. Exiting." <<
std::endl;
        return;
    }
}

void Print_sort_mas(list **mas) {
    int i, g, N = 4000;
    char n;
    SetConsoleCP(1251);

    i = 0;
    g = 0;
    bool exitLoop = false;
    printf("\n");
    printf("\n");
    printf("\n");
    printf("\n");
    while (!exitLoop) {
        for (i = g; i < N && i < g + 20; i++) {
            std::cout << i + 1 << "\t" <<
mas[i]->data->FIO << "\t" << mas[i]->data-
>Department << "\t" << mas[i]->data->Position << "\t"
<< mas[i]->data->Date << std::endl;
        }

        if (i >= N) {
            std::cout << "Reached the
end of data." << std::endl;
            std::cout << "Press 'f' for
forward, 'b' for backward, 'e' to exit, 'n' to go to a
specific record: " << endl;
            n = _getch();
        } else {
            std::cout << "Press 'f' for
forward, 'b' for backward, 'e' to exit, 'n' to go to a
specific record: " << endl;
            n = _getch();
        }

        if (n == 'f') {
            g += 20;
        } else if (n == 'b') {
            g -= 20;
            if (g < 0) {
                g = 0;
            }
        } else if (n == 'e') {
            exitLoop = true;
        } else if (n == 'n') {
            std::cout << "\nEnter record
number: ";

            int recordNumber;
            std::cin >> recordNumber;

            if (recordNumber > 0 &&
recordNumber <= N) {
                g = ((recordNumber - 1) / 20) * 20;
            }
        }
    }
}

```

```

    } else {
        std::cout << "Invalid record number. Press
Enter to continue." << std::endl;
        while (_getch() != 'r') {} // Ожидаем
нажатия Enter перед продолжением
    }
    } else if (n != 'y') {
        std::cout << "Invalid input.
Exiting." << std::endl;
        return;
    }
}

void copy_base(list *a, list *b) {
    b->prev = NULL;
    b->data = a->data;
    for (int i = 1; i < N; i++) {
        a = a->next;
        b->next = new list;
        b->next->prev = b;
        b = b->next;
        b->data = a->data;
    }
    b->next = NULL;
}

void CodePrint(){
    lgm=0;
    SetConsoleCP(866);
    printf("\n\nCode Gilbert-Moore: \n\n");
    printf("| ^ | Symbol | Propability |   Code
word | Length \n");
    SetConsoleCP(1251);
    for (int i = 0; i < alphabet_num; i++) {
        // Используем условие проверки, что строка не
пуста
        if (B[i].l > 0) {
            // Заменяем символ B[i].a на "-" при i = 56
или 57
            char currentChar = (i == 56 || i == 57) ? '-':
B[i].a;
            printf("| %2d |   %c   | %2.6f   |", i,
currentChar, B[i].q);
            for (int j = 1; j <= B[i].l; j++)
                printf("%d", code[i][j]);
            for (int j = B[i].l + 1; j < 14; j++)
                printf(" ");
            printf("   | %4d \n", B[i].l);
            lgm += B[i].p * B[i].l;
        } else {
            // Выводим "-" вместо пустого символа во
второй колонке
            printf("| %2d |   -   | %2.6f   |", i, B[i].q);
            for (int j = 0; j < 12; j++) // Уменьшаем на 1,
чтобы учесть "-"
                printf(" ");
            printf("   | %4d \n", B[i].l);
            lgm += B[i].p * B[i].l; // уточнено место для
lgm в случае пустой строки

```

```

    }
}

SetConsoleCP(866);

printf("\nN = %d\n", alphabet_num);
printf("\n  Entropy \t Average lenght\n");
printf(" %10f    %10.5f    \n", entropy, lgm);

    cout << endl << endl << entropy+2 << " > " <<
lgm << endl << endl;
}

void GilbertMoorCode(){
    int i,j;
    FILE *fp;
    fp = fopen("testBase2.dat", "rb");
    for (i = 0; i < M; i++) {
        A[i].p = 0;
        A[i].q = 0;
        A[i].l = 0;
        A[i].a = (char)(i-128);
    }
    while (!feof(fp)) {
        char c;
        fscanf(fp, "%c", &c);
        if (feof(fp))
            break;
        //printf("%c",c);
        //cout << c<< " - " << (int)c << endl;
        A[c+128].p +=1;
        A[c+128].a = c;
        sum++;
    }
    fclose(fp);
    for (i = 0, j = 0; i < M; i++){
        if(A[i].p!=0){
            A[i].p /=sum;
            B[j]=A[i];
            entropy += A[i].p *
abs(log(A[i].p) / log(2));
            j++;
        }
    }

    for (i = 0; i < alphabet_num; i++){
        B[i].q = B[i-1].q + B[i].p/2;
        B[i].l = ceil(-log(B[i].p) / log(2)) + 1;
    }

    for (i = 0; i < alphabet_num; i++)
    {
        for (j = 0; j <= B[i].l; j++)
        {
            B[i].q *= float(2);
            code[i][j] = floor(B[i].q);
            while (B[i].q >= 1)
                B[i].q -= 1;
        }
    }
}

```

```

    }
}

int size(Vertex *x)
{
    if (x == NULL) {
        return 0;
    }
    else {
        return 1 + size(x->Left) + size(x->Right);
    }
}

int maxi(int x, int y)
{
    if (x > y) return x;
    return y;
}

int height(Vertex *x)
{
    if (x == NULL) {
        return 0;
    }
    else {
        return 1 + max(height(x->Left),
height(x->Right));
    }
}

int sdp(Vertex *x, int l)
{
    if (x == NULL) {
        return 0;
    }
    else {
        return l + sdp(x->Left, l + 1) + sdp(x->Right, l + 1);
    }
}

void LR_print(Vertex* p, int& count) {
    if (p != NULL) {
        LR_print(p->Left, count);
        cout << count+1 << " " << p->data->FIO << "\t" << p->data->Department << "\t" << p->data->Position << "\t" << p->data->Date << endl;
        count++;
        LR_print(p->Next, count);
        LR_print(p->Right, count);
    }
}

void add_vertex(Vertex *&p, list* mas, int w) {
    if (p == NULL) {
        p = new Vertex;
        p->data = mas->data;
        p->w = w;
        p->Next = NULL;

```

```

        p->Left = NULL;
        p->Right = NULL;
    }
    else if (strcmp(p->data->FIO,mas->data->FIO,3) == 0) {
        add_vertex(p->Next, mas, w);
    }
    else if (strcmp(p->data->FIO,mas->data->FIO,3) > 0) {
        add_vertex(p->Left, mas, w);
    }
    else if (strcmp(p->data->FIO,mas->data->FIO,3) < 0) {
        add_vertex(p->Right, mas, w);
    }
}

void A1(int L, int R, list **mas) {
    int wes = 0, sum = 0;
    int i;
    if (L <= R) {
        for (i = L; i <= R; i++) {
            wes = wes + W[i];
        }
        for (i = L; i < R; i++) {
            if ((sum < (wes / 2)) &&
(sum + W[i]) > (wes / 2)) {
                break;
            }
            sum = sum + W[i];
        }
        //cout << L << " - " << R << " - " <<
wes << " - " << i << " - " << W[i] << endl;
        add_vertex(root, mas[i], W[i]);
        A1(L, i - 1, mas);
        A1(i + 1, R, mas);
    }
}

void TreeSearch (Vertex* p, char* name){
    if(p!=NULL)
    {
        if(strcmp(p->data->FIO,name,3)>0){
            TreeSearch(p->Left,name);
        }else{
            if(strcmp(p->data->FIO,name,3)<0){
                TreeSearch(p->Right,name);
            }else{
                if(strcmp(p->data->FIO,name,3)==0){
                    cout << p->data->FIO << "\t" << p->data->Department << "\t"
<< p->data->Position << "\t" << p->data->Date <<
endl;
                    TreeSearch(p->Next,name);
                }
            }
        }
    }
}

```

```

}

void seth(Vertex *p)
{
    if (p) {
        if (p->Next) {
            p->Next->h = p->h + 1;
        }
        if (p->Left) {
            p->Left->h = p->h + 1;
        }
        if (p->Right) {
            p->Right->h = p->h + 1;
        }
        seth(p->Left);
        seth(p->Right);
        seth(p->Next);
    }
}

int med(int L, int R)
{
    float sl = 0, sr;
    for (int i = L; i < R; i++)
        sl += A[i].q;
    sr = A[R].q;
    int m = R;
    while (sl >= sr)
    {
        m--;
        sl -= A[m].q;
        sr += A[m].q;
    }
    return m;
}

void fano(int L, int R, int k)
{
    if (L < R)
    {
        k++;
        int m = med(L, R);
        for (int i = L; i <= R; i++)
        {
            if (i <= m)
                code[i][k] = 0;
            else
                code[i][k] = 1;
            A[i].l++;
        }
        fano(L, m, k);
        fano(m + 1, R, k);
    }
}

int Menu()
{
    // Остальной код без изменений...
    system("cls");
    setlocale(LC_ALL, "Russian");

```

```

FILE* fp;
int search_start=0;
int search=N-1;

struct GM_code {
    float p;
    float q;
    int l;
    char a;
};
GM_code A[M];
GM_code B[alphabet_num];

fp = fopen("testBase2.dat", "rb");
list* OriginBase = new list;
list* SortBase = new list;
Read_base(fp, OriginBase);
fclose(fp);
copy_base(OriginBase, SortBase);
digital_sort(SortBase, 0);
digital_sort(SortBase, 1);
list *mas[N];
for(int i=0; i<N; i++){
    mas[i]=SortBase;
    SortBase=SortBase->next;
}
W = new int[N];
int SoD;
char input; // Переменная для хранения
введённого символа
GilbertMoorCode();

int enter = 0;
char *spc = " ";

std::cout << "\t-----Menu-----"
<< std::endl;
std::cout << "\t1. View database" << std::endl;
std::cout << "\t2. View sorted database (Digital
Sort)" << std::endl;
std::cout << "\t3. Bsearch" << std::endl;
std::cout << "\t4. Tree (A1)" << std::endl;
std::cout << "\t5. Code" << std::endl;
std::cout << "\t6. Exit" << std::endl;
std::cout << "\t-----"
<< std::endl;

input = _getch(); // Считываем символ с
клавиатуры

// Проверяем введенный символ
switch (input)
{
    case '1':
        SoD = 1;
        break;
    case '2':
        SoD = 2;
        break;

```



```

case '3':
    SoD = 3;
    break;
case '4':
    SoD = 4;
    break;
case '5':
    SoD = 5;
    break;
case '6':
    SoD = 6;
    break;
default:
    // Если введен символ, отличный от 1, 2, 3
или 4, возвращаемся в меню
    Menu();
    return 0;
}

switch (SoD)
{
    case 1:
        // Опция 1: View database
        Print(OriginBase);
        break;
    case 2:
        // Опция 2: View sorted database (Digital Sort)
        Print_sort_mas(mas);
        break;
    case 3:
    {
        cout << endl;
        cout << "Enter
department" << endl;

        int numb = 0;
        cin >> numb;
        if(numb >= 0){
            search =
BSearch(mas, numb);

            int fam_pos = 0;
            if(search == -1){
                cout <<
"This department doesn't exists" << endl;
            }
            else{
                do{

                    if(search == 0){

                        break;

                    }

                    else{

                        search--;

                    }

                    if(mas[search]->data->Department != numb){

                        search++;

                        break;

                    }

                    while(true);

                    search_start = search;

                    do{

                        search++;

                        if(search == N){

                            search--;

                            break;

                        }

                        if(mas[search]->data->Department != numb){

                            break;

                        }

                    }

                    while(true);

                    SetConsoleCP(1251);

                    cout
                    << endl << endl << "Founded " << search - search_start << "
                    positions (" << search_start << " - " << search -
                    1 << ")" << endl;

                    for(int
                    i = search_start; i < search; i++){

                        cout << i << " " << mas[i]->data->FIO << "\t"
                        << mas[i]->data->Department << "\t" << mas[i]->data-
                        >Position << "\t" << mas[i]->data->Date << endl;

                        W[i] = rand()
                        % 99 + 1;

                    }

                    SetConsoleCP(866);

                    A1(search_start, search, mas);
                    root->h = 1;
                    seth(root);

                }

                break;

            }

            case 4: {

                int count = 0;
                char street[18];
                SetConsoleCP(1251);
                printf("\n");
                printf("\n");
                LR_print(root, count);
            }
        }
    }
}

```


6. РЕЗУЛЬТАТЫ

```
-----Menu-----
1. View database
2. View sorted database (Digital Sort)
3. Bsearch
4. Tree (A1)
5. Code
6. Exit
-----
```

Рисунок 2 — Внешний вид меню

1	Остапова Алсу Власовна	130	штукатур-маляр	18-02-57
2	Глебо Виолетта Пантелемоновна	120	начальник лаборатории	04-08-38
3	Янов Александр Ромуальдович	120	научный сотрудник	01-11-12
4	Поликарпов Жак Янович	200	научный сотрудник	26-08-61
5	Климов Пантелемон Батырович	150	слесарь-сантехник	01-01-50
6	Филимонов Зосим Архипович	240	ученый секретарь	01-03-73
7	Зосимова Изабелла Архиповна	160	ученый секретарь	25-09-76
8	Демьянов Глеб Сабирович	90	начальник лаборатории	08-03-70
9	Ахиллесов Остап Никодимович	200	начальник отдела	26-02-19
10	Патриков Зосим Евграфович	50	инженер	26-02-36
11	Ромуальдова Нинель Жаковна	110	начальник лаборатории	01-03-45
12	Гедеонов Феофан Ахмедович	230	научный сотрудник	08-11-26
13	Архипов Демьян Хасанович	20	инженер	10-02-11
14	Демьянов Ахиллес Жакович	110	штукатур-маляр	28-03-33
15	Патриков Жак Хасанович	50	начальник отдела	19-08-30
16	Филимонов Влас Зосимович	170	слесарь-сантехник	09-08-64
17	Мстиславов Филимон Сабирович	50	начальник сектора	25-03-63
18	Александ Поликарп Ахиллесович	190	штукатур-маляр	25-03-17
19	Хасанов Ян Сабирович	10	ученый секретарь	08-09-63
20	Ромуальдов Зосим Сабирович	130	начальник лаборатории	24-11-28

Press 'f' for forward, 'b' for backward, 'e' to exit, 'n' to go to a specific record:
█

Рисунок 3 — Вывод базы данных

1	Александр Патрик Мстиславович	0	начальник сектора	13-10-73
2	Александров Ариадна Ахмедовна	0	инженер	03-04-31
3	Александров Изольда Остаповна	0	инженер	10-10-19
4	Александров Никодим Жакович	0	ведущий конструктор	26-01-28
5	Архипов Хасан Мстиславович	0	начальник отдела	15-04-67
6	Архипова Алсу Ахиллесовна	0	начальник отдела	09-03-66
7	Архипова Алсу Зосимовна	0	инженер	17-09-48
8	Архипова Матрена Батыровна	0	начальник отдела	23-02-64
9	Ахиллесов Александр Батырович	0	слесарь-сантехник	11-08-62
10	Ахиллесов Муамар Зосимович	0	инженер	25-12-69
11	Ахиллесов Тихон Власович	0	слесарь-сантехник	21-06-74
12	Ахиллесов Хасан Ахиллесович	0	секретарь-машинист/ка	25-12-69
13	Ахмедо Филимон Пантелемонович	0	начальник сектора	18-07-58
14	Ахмедов Александр Гедеонович	0	слесарь-сантехник	03-08-53
15	Ахмедов Архип Пантелемонович	0	инженер	19-06-12
16	Ахмедов Жак Евграфович	0	ученый секретарь	12-07-46
17	Ахмедов Жак Тихонович	0	ведущий конструктор	19-10-12
18	Ахмедов Мстислав Архипович	0	научный сотрудник	13-09-51
19	Ахмедов Ян Власович	0	слесарь-сантехник	27-05-27
20	Ахмедова Изольда Ромуальдовна	0	научный сотрудник	19-08-52

Press 'f' for forward, 'b' for backward, 'e' to exit, 'n' to go to a specific record:

Рисунок 4 — Вывод отсортированной базы данных

```
Enter department
30
```

Рисунок 5 —Запрос номера отдела для поиска

```
Enter department
145
This department doesn't exists''

```

Рисунок 6 – уведомление при наборе несуществующего отдела

Founded 168 positions (493 - 660)			
493 Александр Пантелемон Ахмедович	30	штукатур-маляр	18-01-11
494 Александр Никодим Хасанович	30	начальник лаборатории	19-06-47
495 Александр Ромуальд Глебович	30	ученый секретарь	05-06-72
496 Александр Тихон Демьянович	30	начальник сектора	14-05-37
497 Архипов Ахмед Тихонович	30	ученый секретарь	25-10-51
498 Архипов Влас Евграфович	30	секретарь-машинист/ка	18-10-28
499 Архипов Евграф Остапович	30	секретарь-машинист/ка	26-01-17
500 Архипов Муамар Никодимович	30	научный сотрудник	14-09-12
501 Архипов Остап Гедеонович	30	секретарь-машинист/ка	25-06-67
502 Архипов Патрик Климович	30	начальник сектора	25-06-65
503 Архипов Тихон Гедеонович	30	начальник сектора	17-03-12
504 Архипов Тихон Поликарпович	30	инженер	09-08-69
505 Архипова Ариадна Муамаровна	30	начальник отдела	20-12-37
506 Архипова Изольда Никодимовна	30	штукатур-маляр	16-03-27
507 Архипова Пелагея Климовна	30	начальник лаборатории	17-08-43
508 Ахиллесов Влас Ахмедович	30	начальник сектора	04-06-76
509 Ахиллесов Муамар Власович	30	инженер	17-09-36
510 Ахиллесов Остап Демьянович	30	штукатур-маляр	07-09-19
511 Ахиллесов Сабир Батырович	30	ведущий конструктор	19-05-36
512 Ахиллесов Филимон Батырович	30	секретарь-машинист/ка	09-01-41
513 Ахиллессова Матрена Муамаровна	30	начальник сектора	05-05-54
514 Ахмедо Пантелемон Филимонович	30	секретарь-машинист/ка	14-05-20
515 Ахмедов Варвара Александровна	30	секретарь-машинист/ка	09-08-17
516 Ахмедов Герасим Мстиславович	30	научный сотрудник	09-02-41
517 Ахмедов Глеб Тихонович	30	ведущий конструктор	20-12-57
518 Ахмедов Никодим Власович	30	ученый секретарь	09-08-58
519 Ахмедов Пантелемон Глебович	30	начальник сектора	08-03-34
520 Батыров Ахиллес Глебович	30	начальник лаборатории	28-03-56
521 Батыров Герасим Ахмедович	30	начальник сектора	26-01-09

Рисунок 7 — Результат поиска

1) Александр Тихон Демьянович	30	начальник сектора	14-05-37
2) Александр Ромуальд Глебович	30	ученый секретарь	05-06-72
3) Александр Никодим Хасанович	30	начальник лаборатории	19-06-47
4) Александр Пантелемон Ахмедович	30	штукатур-маляр	18-01-11
5) Александр Влас Климович	30	штукатур-маляр	21-12-14
6) Архипов Муамар Никодимович	30	научный сотрудник	14-09-12
7) Архипов Влас Евграфович	30	секретарь-машинист/ка	18-10-28
8) Архипов Ахмед Тихонович	30	ученый секретарь	25-10-51
9) Архипов Евграф Остапович	30	секретарь-машинист/ка	26-01-17
10) Архипова Изольда Никодимовна	30	штукатур-маляр	16-03-27
11) Архипов Тихон Гедеонович	30	начальник сектора	17-03-12
12) Архипов Остап Гедеонович	30	секретарь-машинист/ка	25-06-67
13) Архипов Патрик Климович	30	начальник сектора	25-06-65
14) Архипов Тихон Поликарпович	30	инженер	09-08-69
15) Архипова Ариадна Муамаровна	30	начальник отдела	20-12-37
16) Архипова Пелагея Климовна	30	начальник лаборатории	17-08-43
17) Ахиллесов Сабир Батырович	30	ведущий конструктор	19-05-36
18) Ахиллесов Влас Ахмедович	30	начальник сектора	04-06-76
19) Ахиллесов Остап Демьянович	30	штукатур-маляр	07-09-19
20) Ахиллесов Муамар Власович	30	инженер	17-09-36
21) Ахиллессова Матрена Муамаровна	30	начальник сектора	05-05-54
22) Ахиллесов Филимон Батырович	30	секретарь-машинист/ка	09-01-41
23) Ахмедов Герасим Мстиславович	30	научный сотрудник	09-02-41
24) Ахмедо Пантелемон Филимонович	30	секретарь-машинист/ка	14-05-20
25) Ахмедов Варвара Александровна	30	секретарь-машинист/ка	09-08-17
26) Ахмедов Пантелемон Глебович	30	начальник сектора	08-03-34
27) Ахмедов Глеб Тихонович	30	ведущий конструктор	20-12-57
28) Ахмедов Никодим Власович	30	ученый секретарь	09-08-58
29) Батыров Ромуальд Демьянович	30	инженер	15-04-38
30) Батыров Герасим Ахмедович	30	начальник сектора	26-01-09
31) Батыров Ахиллес Глебович	30	начальник лаборатории	28-03-56

Рисунок 8 — Вывод дерева

№	Symbol	Propability	Code word	Length
0	А	0,768000	000000011	9
1	Б	0,448000	00000010010	11
2	В	0,368000	0000001100	10
3	Г	0,564000	000001000	9
4	Д	0,431999	00000100101	11
5	Е	0,599998	00000101000	11
6	Ж	0,655998	00000101011	11
7	З	0,663998	00000101110	11
8	И	0,463989	0000010111110	13
9	К	0,703999	00000110010	11
10	М	0,877999	000001110	9
11	Н	0,983997	00000111110	11
12	О	0,007996	00001000010	11
13	П	0,815998	000010010	9
14	Р	0,127991	00001001110	11
15	С	0,191994	0000101001	10
16	Т	0,655991	00001010101	11
17	Ф	0,079994	0000101110	10
18	Х	0,327988	00001011111	11
19	Ц	0,503983	000011000001	12
20	Я	0,911995	00001100011	11
21	а	0,852500	00011	5
22	б	0,180000	001000000	9
23	в	0,249250	001010	6
24	г	0,728027	00101001011	11
25	д	0,839005	00101100	8
26	е	0,639001	001101	6
27	ж	0,552032	00110110111	11
28	з	0,016113	0011011100010	13
29	и	0,027752	010001	6
30	й	0,650009	010001010	9
31	к	0,106503	0100110	7
32	л	0,980503	0101001	7

Рисунок 9 — Вывод вероятностей, кодовых слов и длин символов

```

N = 81

Entropy      Average lenght
4,866794     6,34942

6.86679 > 6.34942

```

Рисунок 10 — Вывод энтропии и средней длины

7. ВЫВОДЫ

В ходе выполнения курсового проекта были успешно реализованы все поставленные задачи, подразумевавшие использование различных алгоритмов для обработки и анализа данных. Алгоритмы сортировки и метод Цифровой сортировки использовались для упорядочивания базы данных по номеру отдела и ФИО, обеспечивая систематизированный доступ к информации.

Были внедрены алгоритмы двоичного поиска и создания очереди для оптимизации процесса поиска по ключу в упорядоченной базе. Это позволяет эффективно и быстро находить необходимые записи и предоставляет пользователю интуитивно понятный интерфейс для взаимодействия с данными.

Построение дерева оптимального поиска методом A1 и реализация поиска по дереву дополнительно расширили функциональность программы, обеспечивая более сложные запросы и операции поиска.

Завершая проект, было реализовано кодирование данных кодом Гилберта-Мура. Построенный код был не только успешно выведен на экран, но и подвергнут анализу для определения средней длины кодового слова. Эта метрика была сравнена с энтропией исходного файла, что позволяет оценить эффективность сжатия и оптимальность выбранного кодирования.

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