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 А1 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 "Пpедпpиятие"

Стpуктуpа записи:

ФИО сотpудника: текстовое поле 30 символа

фоpмат <Фамилия>\_<Имя>\_<Отчество>

Hомеp отдела: целое число

Должность: текстовое поле 22 символа

Дата pождения: текстовое поле 10 символов

фоpмат дд-мм-гг

Пpимеp записи из БД:

Петpов\_Иван\_Иванович\_\_\_\_\_\_\_\_\_\_\_\_

130

начальник\_отдела\_\_\_\_\_\_

15-03-46

Ваpиант условий упоpядочения и ключи поиска (К):

C = 1 - по номеpу отдела и ФИО, К = номеp отдела;

Ключ в дереве – первые 3 буквы фамилии.

1. ОСНОВНЫЕ ИДЕИ И ХАРАКТЕРИСТИКИ ПРИМЕНЯЕМЫХ МЕТОДОВ

# Метод сортировки

Цифровая сортировка (Digital Sort)

Цифровая сортировка является одним из методов сортировки последовательностей.

Пусть дана последовательность из S чисел, представленных в m – ичной системе счисления. Каждое число состоит из L цифр d1d2…dL, 0 ≤ di ≤ m – 1, i=1..L. Сначала числа из списка S распределяются по m очередям, причём номер очереди определяется последней цифрой каждого числа. Затем полученные очереди соединяются в список, для которого все действия повторяются, но распределение по очередям производится в соответствии со следующей цифрой и т.д.

Цифровой метод может успешно использоваться не только для сортировки чисел, но и для сортировки любой информации, представленной в памяти компьютера. Необходимо лишь рассматривать каждый байт ключа сортировки как цифру, принимающую значения от 0 до 255. Тогда для сортировки потребуется m=256 очередей. Для выделения каждого байта ключа сортировки можно использовать массив Digit, наложенный в памяти компьютера на поле элемента последовательности, по которому происходит сортировка. Приведем более детальный алгоритм цифровой сортировки.

Для цифровой сортировки М<const L(*m+n*). При фиксированных *m* и L М=O(*n*) при *n → ∞*, что значительно быстрее остальных рассмотренных методов. Однако если длина чисел L велика, то метод может проигрывать обычным методам сортировки. Кроме того, Метод применим только, если задача сортировки сводится к задаче упорядочивания чисел, что не всегда возможно.

Метод обеспечивает устойчивую сортировку.

# 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 сравнение для первой версии, одно сравнение для второй версии. Количество итераций не больше, чем ⎡log2 n⎤ . Таким образом, трудоёмкость двоичного поиска в обоих случаях

С=O(log n), n → ∞.

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

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 поддерево 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 Lcp < H( p1 ,...,pn ) + 2. Пусть символы алфавита некоторым образом упорядочены, например, a1≤a2≤…≤an. Код σ называется алфавитным, если кодовые слова лексикографически упорядочены, т.е. σ (a1) ≤σ (a2) ≤ ... ≤σ (an).

Процесс построения кода происходит следующим образом.

1. Вычислим величины Qi , i=1,n:

Q1=p1/2,

Q2=p1+p2/2,

Q3=p1+p2+p3/2,

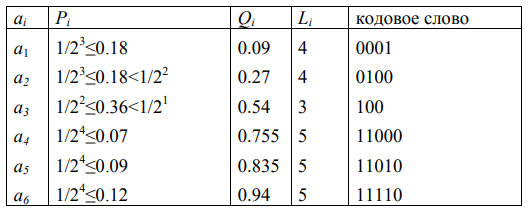
…

Qn=p1+p2+…+pn-1+pn/2.

2. Представим суммы Qi в двоичном виде.

3. В качестве кодовых слов возьмем ⎡− log pi⎤ +1 младших бит в двоичном представлении Qi , i =1,...,n.

**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.

  
 Рисунок 1 – Таблица «Код Гилберта-Мура»

Средняя длина кодового слова не превышает значения энтропии плюс 2. Действительно, Lср=4. 0.18+4. 0.18+3. 0.36+5. 0.07+5. 0.09+5. 0.12=3.92<2.37+2

1. ОСОБЕННОСТИ РЕАЛИЗАЦИИ АЛГОРИТМОВ
2. 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( ).

1. 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 посредствам 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.

1. 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 А1).

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).

1. 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.

1. 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 (А1).

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, исключающем 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:

1. void copy\_base(list \*a, list \*b) – делает копию списка, в котором хранится база данных.
2. void digital\_sort(list \*&head, int sort) – сортирует базу данных по номеру отдела и ФИО.

Процедуры и функции для поиска в отсортированной базе данных:

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

Процедуры построения дерева оптимального поиска (А1):

1. void add\_vertex(vertex \*&p, list \*mas, int w) – добавление элемента в дерево.
2. void A1(int L, int R, list \*\*mas) – построение дерева оптимального поиска, приближенный алгоритм А1.
3. void LR\_print(vertex \*p, int& count) – вывод дерева.
4. void TreeSearch(vertex \*p, char \*data) – поиск в дереве.
5. void delete\_tree(vertex\* p, char\* name) – удаление дерева.

Процедуры и функции кодирования базы данных:

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

Основная программа:

1. main( ) - основная программа, в которой происходит запуск меню.
2. 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;

}

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;

}

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("| ¹ | 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 (strncmp(p->data->FIO,mas->data->FIO,3) == 0) {

add\_vertex(p->Next, mas, w);

}

else if (strncmp(p->data->FIO,mas->data->FIO,3) > 0) {

add\_vertex(p->Left, mas, w);

}

else if (strncmp(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(strncmp(p->data->FIO,name,3)>0){

TreeSearch(p->Left,name);

}else{

if(strncmp(p->data->FIO,name,3)<0){

TreeSearch(p->Right,name);

}else{

if(strncmp(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 <<" pozitions ("<< 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);

SetConsoleCP(866);

cout<<endl;

// printf("+------+----------+----------+----------------+\n");

// printf("|%6d| UniqSize | Height | Mid. height |\n", N);

// printf("+------+----------+----------+----------------+\n");

// printf("| A1 |%10d|%10d|%16.2f|\n", size(root), height(root), (double)sdp(root, 1) / size(root));

// printf("+------+----------+----------+----------------+\n");

cout<<endl<< "Element find: "<< endl;

SetConsoleCP(866);

cin >> street;

SetConsoleCP(1251);

char \*spc = " ";

strcat(street, spc);

if(strcmp(street,"0")!=0){

TreeSearch(root,street);

}

break;

}

case 5:{

CodePrint();

break;

}

case 6:

// Опция 4: Exit

system("PAUSE");

return 0;

default:

break;

}

\_getch(); // Ждем нажатия любой клавиши перед повторным вызовом Menu()

Menu();

return 0;

}

int main()

{

Menu();

return 0;

}

1. РЕЗУЛЬТАТЫ

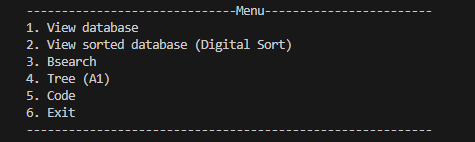


Рисунок 2 — Внешний вид меню

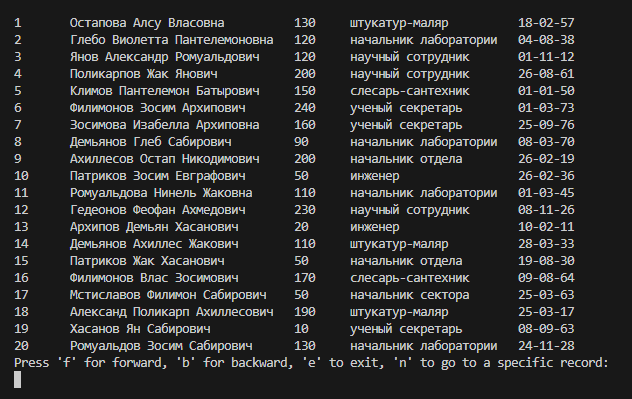


Рисунок 3 — Вывод базы данных

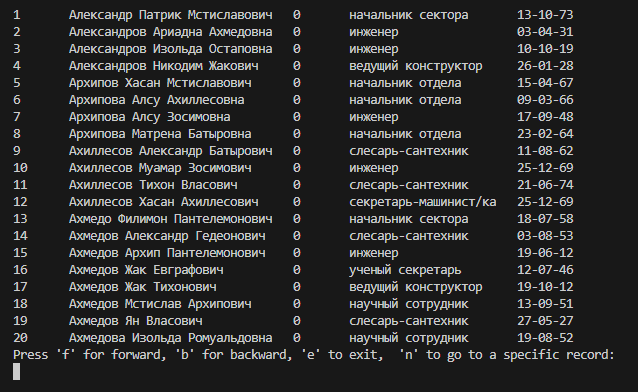


Рисунок 4 — Вывод отсортированноый базы данных



Рисунок 5 —Запрос номера отдела для поиска

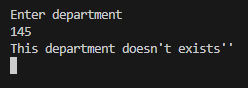


Рисунок 6 – уведомление при наборе несуществующего отдела

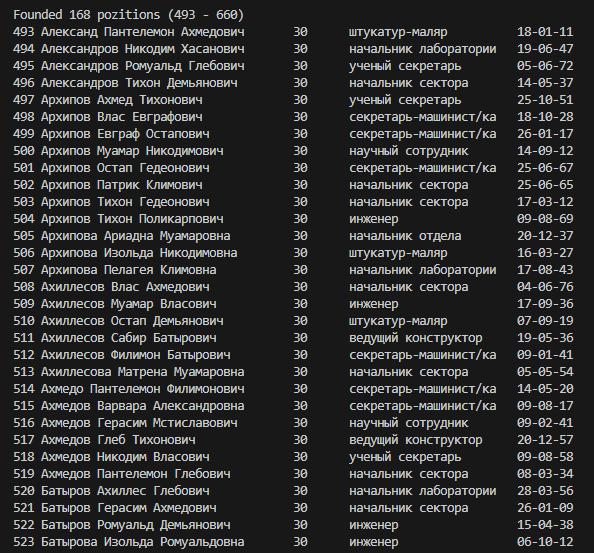


Рисунок 7 — Результат поиска



Рисунок 8 — Вывод дерева

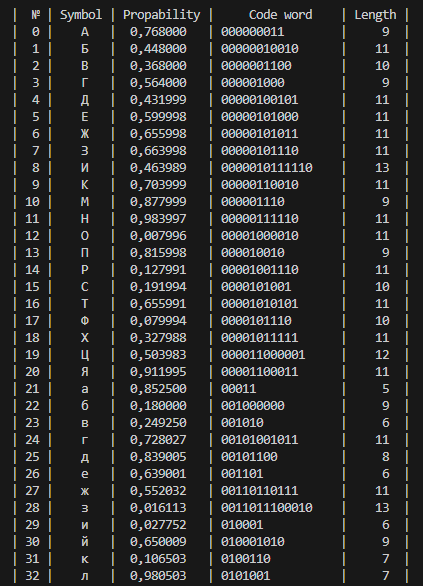


Рисунок 9 — Вывод вероятностей, кодовых слов и длин символов

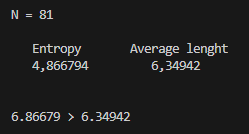


Рисунок 10 — Вывод энтропии и средней длины

1. 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 А1 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