#include "functions.h"

int main()

{

system("cls"); // очистка командной строки

// переменная для хранения кол-ва считанных структур из текстового

// файла f.txt

int n;

// переменная для хранения кол-ва станций пересадок

int num;

int graph[V][V] = {…};

// объявление массива структур для хранения информации о всех

// городах

struct s1\* info;

// объявление и выделение памяти массива структур для хранения

// информации о поездке пользователя

struct s2\* infoWay = (struct s2\*)calloc(1, sizeof(struct s2));

ReadInfoFromFile1(&info, &n);

// пользовательское меню

infoWay = InitialMenu(graph, info, infoWay, n, &num);

free(info);

free(infoWay);

return 0;

}

#pragma once

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

#include <limits.h>

#include <string.h>

#define N 20 // размер строковых переменных

#define t 10 // кол-во времён в массивах времён

#define V 32 // кол-во городов

// объявление структуры, хранящей информацию о городе (населённого

// пункта)

struct s1

{

char City[N]; // название города (населённого пункта)

char Time1[t][N]; // массив времён отбытия на t элементов

char Time2[t][N]; // массив времён прибытия на t элементов

// номер железно-дорожной линии, на которой расположен город

// (населённый пункт)

int Line;

// расстояние от данного города (населённого пункта) до Минска

int Dist;

};

// объявление структуры, хранящей информацию о поездке пользователя

struct s2

{

// название города (населённого пункта), из которого пользователь

// отправляется

char CityFrom[N];

// название города (населённого пункта), в который пользователь

// отправляется

char CityTo[N];

// массив станций пересадок, максимальное кол-во 3

char Peresadki[3][N];

// массив времён отправления поездов со станций пересадки

char TimePeres1[3][t][N];

// массив времён отправления поездов со станций пересадки

char TimePeres2[3][t][N];

// массив времён отправления из данного города (населённого пункта)

char TimeFrom[t][N];

// массив времён прибытия в данный город (населённый пункт)

char TimeTo[t][N];

// расстояние между городом отправления и городои прибытия

int Distance;

};

// структура для хранения информации о ребре между двумя городами

struct Edge

{

int u; // номер первой станции

int v; // номер второй станции

// вес ребра (расстояние между данными городами)

int weight;

};

// запись структуры, хранящей информацию о поездке пользователя, в

// текстовый файл result.txt

void InputInformationInFileS2(struct s2\* infoWay, int num);

// ввод названия города (населённого пункта), в который пользователь

// отправляется

char\* InputCityTo(struct s1\* info);

// ввод названия города (населённого пункта), из которого пользователь

// отпраляется

char\* InputCityFrom(struct s1\* info);

// функция для проверки открытия файла

void CheckFile(FILE\* fp);

// удаление enter при вводе информации в строковую переменную (char\*

// str/char str[])

void DelEnter(char str[]);

// чтение и массива структур, хранящего информацию о всех городах, из

// текстового файла f.txt

void ReadInfoFromFile1(struct s1\*\* info, int\* n);

// вывод массива структур, хранящего информацию о всех городах

void PrintStruct\_s1(struct s1\* info, int n);

// вывод структуры, хранящей информацию о поездке пользователя

void PrintStruct\_s2(struct s2\* infoWay, int num);

// нахождение кол-во структур в текстовом файле

int NumberOfStructs(FILE\* fp);

// поиск минимального расстояниямежду станциями

int min\_distance(int dist[], int visited[]);

// алгоритм Дейкстры для обхода графа и нахожения кратчайшего пути и

// станций пересадок

struct s2\* dijkstra(int graph[V][V], int start, int end, struct s1\* info, struct s2\* infoWay, int\* num);

// поиск индекса города в массиве структур, хранящем информацию о всех

// городах

int find\_city\_index(struct s1\* info, char\* name);

struct s2\* InitialMenu(int graph[V][V], struct s1\* info, struct s2\* infoWay, int n, int\* num); // пользовательское меню

int returnFunc(); // возвращение к пользовательскому меню

// поиск напрваления поездки (от Минска или к Минску)

int napravl(struct s1\* info, int index1, int index2, int end, int num);

struct s2\* GetTime(struct s1\* info, struct s2\* infoWay, int indexFrom, int indexTo, int napravl, int number); // чтение времён отбытия и прибытия

struct s2\* GetTimeOfPeresadki(struct s1\* info, struct s2\* infoWay, int napravl, int number);

#include "functions.h"

// запись массива структур в текстовый файл result.txt

void InputInformationInFileS2(struct s2\* infoWay, int num)

{

FILE\* fp = fopen("result.txt", "w");

CheckFile(fp);

// если на маршруте нет пересадок, то запись в файл будет иметь

// следующий вид

if (num == 0)

{

fprintf(fp, "%s %s %s %s %s %s %s %s %s %s %s ->\n-> %s %s %s %s %s %s %s %s %s %s %s -> %d", infoWay[0].CityFrom, infoWay[0].TimeFrom[0], infoWay[0].TimeFrom[1], infoWay[0].TimeFrom[2], infoWay[0].TimeFrom[3], infoWay[0].TimeFrom[4], infoWay[0].TimeFrom[5], infoWay[0].TimeFrom[6], infoWay[0].TimeFrom[7], infoWay[0].TimeFrom[8], infoWay[0].TimeFrom[9],

infoWay[0].CityTo, infoWay[0].TimeTo[0], infoWay[0].TimeTo[1], infoWay[0].TimeTo[2], infoWay[0].TimeTo[3], infoWay[0].TimeTo[4], infoWay[0].TimeTo[5], infoWay[0].TimeTo[6], infoWay[0].TimeTo[7], infoWay[0].TimeTo[8], infoWay[0].TimeTo[9], infoWay[0].Distance);

}

// если на маршруте есть 1 (одна) пересадка, то запись в файл будет иметь // следующий вид

else if (num == 1)

{

fprintf(fp, "%s %s %s %s %s %s %s %s %s %s %s ->\n-> %s %s %s %s %s %s %s %s %s %s %s ->\n-> %s %s %s %s %s %s %s %s %s %s %s ->\n-> %s %s %s %s %s %s %s %s %s %s %s -> %d", infoWay[0].CityFrom, infoWay[0].TimeFrom[0], infoWay[0].TimeFrom[1], infoWay[0].TimeFrom[2], infoWay[0].TimeFrom[3], infoWay[0].TimeFrom[4], infoWay[0].TimeFrom[5], infoWay[0].TimeFrom[6], infoWay[0].TimeFrom[7], infoWay[0].TimeFrom[8], infoWay[0].TimeFrom[9], infoWay[0].Peresadki[0], infoWay[0].TimePeres1[0][0], infoWay[0].TimePeres1[0][1], infoWay[0].TimePeres1[0][2], infoWay[0].TimePeres1[0][3], infoWay[0].TimePeres1[0][4], infoWay[0].TimePeres1[0][5], infoWay[0].TimePeres1[0][6],

infoWay[0].TimePeres1[0][7], infoWay[0].TimePeres1[0][8], infoWay[0].TimePeres1[0][9], infoWay[0].Peresadki[0], infoWay[0].TimePeres2[0][0], infoWay[0].TimePeres2[0][1], infoWay[0].TimePeres2[0][2], infoWay[0].TimePeres2[0][3], infoWay[0].TimePeres2[0][4], infoWay[0].TimePeres2[0][5], infoWay[0].TimePeres2[0][6], infoWay[0].TimePeres2[0][7], infoWay[0].TimePeres2[0][8], infoWay[0].TimePeres2[0][9],

infoWay[0].CityTo, infoWay[0].TimeTo[0], infoWay[0].TimeTo[1], infoWay[0].TimeTo[2], infoWay[0].TimeTo[3], infoWay[0].TimeTo[4], infoWay[0].TimeTo[5], infoWay[0].TimeTo[6], infoWay[0].TimeTo[7],

infoWay[0].TimeTo[8], infoWay[0].TimeTo[9], infoWay[0].Distance);

}

// если на маршруте более 1 (одной) пересадки, то запись в файл будет

// иметь следующий вид

else

{

fprintf(fp, "%s %s %s %s %s %s %s %s %s %s %s ->\n-> %s %s %s %s %s %s %s %s %s %s %s ->\n-> %s %s %s %s %s %s %s %s %s %s %s ->\n-> %s %s %s %s %s %s %s %s %s %s %s ->\n-> %s %s %s %s %s %s %s %s %s %s %s ->\n-> %s %s %s %s %s %s %s %s %s %s %s -> %d", infoWay[0].CityFrom, infoWay[0].TimeFrom[0], infoWay[0].TimeFrom[1], infoWay[0].TimeFrom[2], infoWay[0].TimeFrom[3], infoWay[0].TimeFrom[4], infoWay[0].TimeFrom[5], infoWay[0].TimeFrom[6], infoWay[0].TimeFrom[7], infoWay[0].TimeFrom[8], infoWay[0].TimeFrom[9], infoWay[0].Peresadki[0], infoWay[0].TimePeres1[0][0], infoWay[0].TimePeres1[0][1], infoWay[0].TimePeres1[0][2], infoWay[0].TimePeres1[0][3], infoWay[0].TimePeres1[0][4], infoWay[0].TimePeres1[0][5], infoWay[0].TimePeres1[0][6], infoWay[0].TimePeres1[0][7], infoWay[0].TimePeres1[0][8], infoWay[0].TimePeres1[0][9], infoWay[0].Peresadki[0], infoWay[0].TimePeres2[0][0], infoWay[0].TimePeres2[0][1], infoWay[0].TimePeres2[0][2], infoWay[0].TimePeres2[0][3], infoWay[0].TimePeres2[0][4], infoWay[0].TimePeres2[0][5], infoWay[0].TimePeres2[0][6], infoWay[0].TimePeres2[0][7], infoWay[0].TimePeres2[0][8], infoWay[0].TimePeres2[0][9], infoWay[0].Peresadki[1], infoWay[0].TimePeres1[1][0], infoWay[0].TimePeres1[1][1], infoWay[0].TimePeres1[1][2], infoWay[0].TimePeres1[1][3], infoWay[0].TimePeres1[1][4], infoWay[0].TimePeres1[1][5], infoWay[0].TimePeres1[1][6], infoWay[0].TimePeres1[1][7], infoWay[0].TimePeres1[1][8], infoWay[0].TimePeres1[1][9], infoWay[0].Peresadki[1], infoWay[0].TimePeres2[1][0], infoWay[0].TimePeres2[1][1], infoWay[0].TimePeres2[1][2], infoWay[0].TimePeres2[1][3], infoWay[0].TimePeres2[1][4], infoWay[0].TimePeres2[1][5], infoWay[0].TimePeres2[1][6], infoWay[0].TimePeres2[1][7], infoWay[0].TimePeres2[1][8], infoWay[0].TimePeres2[1][9],

infoWay[0].CityTo, infoWay[0].TimeTo[0], infoWay[0].TimeTo[1], infoWay[0].TimeTo[2], infoWay[0].TimeTo[3], infoWay[0].TimeTo[4], infoWay[0].TimeTo[5], infoWay[0].TimeTo[6], infoWay[0].TimeTo[7],

infoWay[0].TimeTo[8], infoWay[0].TimeTo[9], infoWay[0].Distance);

}

fprintf(fp, "\n");

fclose(fp);

}

// проверка, есть ли введённый город в массиве структур

int CheckCity(struct s1\* info,char\* str)

{

// пусть изначально города нет в массиве структур

int flag = 0;

// цикл для прохода по всем элементам (городам) массива структур

for (int i = 0; i < V; i++)

// если введённый город совпал с городом из массива структур, то

// flag = 1

if (strcmp(info[i].City, str) == 0)

flag = 1;

if (flag == 0)

printf("\nThere is no this city. Try again please!");

return flag;

}

// ввод названия города (населённого пункта), в который пользователь едет

char\* InputCityTo(struct s1\* info)

{

// есть ли введённый город в массиве структур

int flag;

char\* StrCityTo;

do

{

rewind(stdin);

flag = 0;

printf("\nEnter the city, where you are going: ");

// выделение динамической памяти для ввода названия города

StrCityTo = (char\*)calloc(N, sizeof(char));

fgets(StrCityTo, N, stdin);

DelEnter(StrCityTo);

// проверка, есть ли введённый город в массиве структур

flag = CheckCity(info, StrCityTo);

// если данного города нет, то пользователь вводит город снова

} while (!flag);

return StrCityTo;

}

// ввод названия города (населённого пункта), из которого едет пользователь

char\* InputCityFrom(struct s1\* info)

{

// есть ли введённый город в массиве структур

int flag;

char\* StrCityFrom;

do

{

rewind(stdin);

flag = 0;

printf("\nEnter the city, where you are coming from: ");

// выделение динамической памяти для ввода названия города

StrCityFrom = (char\*)calloc(N, sizeof(char));

fgets(StrCityFrom, N, stdin);

DelEnter(StrCityFrom);

// проверка, есть ли введённый город в массиве структур

flag = CheckCity(info, StrCityFrom);

// если данного города нет, то пользователь вводит город снова

} while (!flag);

return StrCityFrom;

}

// функция для проверки открытия файла

void CheckFile(FILE\* fp)

{

if (fp == NULL) // проверка открытия файла

{

printf("ERROR");

return;

}

}

void DelEnter(char str[]) // удаление \n при вводе информации

{

int l = strlen(str);

str[l - 1] = '\0';

}

// чтение информации о всех городах в массив структур info

void ReadInfoFromFile1(struct s1\*\* info, int\* n)

{

FILE\* fp = fopen("f.txt", "r");

CheckFile(fp);

// узнаём кол-во структур в файле

int num = NumberOfStructs(fp);

// выделение динамической памяти для массива структур

\*info = (struct s1\*)calloc(num , sizeof(struct s1));

// устанавливаем курсор на начало файал f.txt

fseek(fp, 0, SEEK\_SET);

// цикл для чтения информации о num городах в массив структур из файла // f.txt

for (int i = 0; i < num; i++)

// чтение полей структуры

fscanf(fp, "%s %d %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %d", &((\*info + i)->City), &((\*info + i)->Line), &((\*info + i)->Time1[0]), &((\*info + i)->Time1[1]), &((\*info + i)->Time1[2]), &((\*info + i)->Time1[3]), &((\*info + i)->Time1[4]), &((\*info + i)->Time1[5]), &((\*info + i)->Time1[6]), &((\*info + i)->Time1[7]), &((\*info + i)->Time1[8]), &((\*info + i)->Time1[9]), &((\*info + i)->Time2[0]), &((\*info + i)->Time2[1]), &((\*info + i)->Time2[2]), &((\*info + i)->Time2[3]), &((\*info + i)->Time2[4]), &((\*info + i)->Time2[5]), &((\*info + i)->Time2[6]), &((\*info + i)->Time2[7]), &((\*info + i)->Time2[8]), &((\*info + i)->Time2[9]), &((\*info + i)->Dist));

fclose(fp);

\*n = num;

}

// вывод массива структур, хранящего информацию о всех городах

void PrintStruct\_s1(struct s1\* info, int n)

{

printf("\nThe information about all cities:\n");

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

printf("| %-14s | %-5s | %-5s | %-5s | \n", "City", "Time1", "Time2","Distance");

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

// цикл для вывода информации о всех городах

for (int i = 0; i < n; i++)

{

printf("| %-14s | %-5s | %-5s | %-8d | \n", info[i].City, info[i].Time1[0], info[i].Time2[0], info[i].Dist);

// цикл для вывода массива времён

for (int j = 1; j < t; j++)

printf("| \t\t | %-5s | %-5s | \t | \n",info[i].Time1[j], info[i].Time2[j]);

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

}

}

// вывод структуры, хранящей информации о поездке пользователя

void PrintStruct\_s2(struct s2\* infoWay, int num)

{

// если пересадок нет, то форма вывода будет следующей

if (num == 0)

{

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

printf("| %-14s | %-8s | %-15s | %-5s | %-9s | \n", "CityFrom", "TimeFrom", "CityTo", "TimeTo", "Distance");

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

printf("| %-14s | %-8s | %-15s | %-7s | %-9d | \n", infoWay[0].CityFrom, infoWay[0].TimeFrom[0], infoWay[0].CityTo, infoWay[0].TimeTo[0], infoWay[0].Distance);

// цикл для вывода массива времён

for (int j = 1; j < t; j++)

printf("| \t\t | %-8s | \t\t | %-7s | \t | \n", infoWay[0].TimeFrom[j], infoWay[0].TimeTo[j]);

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

}

// если пересадка 1 (одна), то форма вывода будет следующей

else if (num == 1)

{

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

printf("| %-14s | %-8s | %-14s | %-8s || %-8s | %-15s | %-6s | %-9s | \n", "CityFrom", "TimeFrom", "Peresadka", "Time1", "Time2", "CityTo", "TimeTo", "Distance");

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

printf("| %-14s | %-8s | %-14s | %-8s || %-8s | %-15s | %-7s | %-9d | \n", infoWay[0].CityFrom, infoWay[0].TimeFrom[0], infoWay[0].Peresadki[0], infoWay[0].TimePeres1[0][0], infoWay[0].TimePeres2[0][0], infoWay[0].CityTo, infoWay[0].TimeTo[0], infoWay[0].Distance);

// цикл для вывода массива времён

for (int j = 1; j < t; j++)

printf("| \t\t | %-8s | \t\t | %-8s || %-8s | \t\t | %-7s | \t | \n", infoWay[0].TimeFrom[j], infoWay[0].TimePeres1[0][j], infoWay[0].TimePeres2[0][j], infoWay[0].TimeTo[j]);

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

}

// если пересадок 2 (две), то форма вывода будет следующей

else if (num == 2)

{

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

printf("| %-14s | %-8s | %-14s | %-8s || %-8s | %-15s | %-6s || %-7s | %-15s | %-5s | %-9s | \n", "CityFrom", "TimeFrom", "Peresadka1", "Time1", "Time2", "Peresadka2", "Time1", "Time2", "CityTo", "TimeTo", "Distance");

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

printf("| %-14s | %-8s | %-14s | %-8s || %-8s | %-15s | %-7s || %-7s | %-15s | %-7s | %-9d | \n", infoWay[0].CityFrom, infoWay[0].TimeFrom[0], infoWay[0].Peresadki[0], infoWay[0].TimePeres1[0][0], infoWay[0].TimePeres2[0][0], infoWay[0].Peresadki[1], infoWay[0].TimePeres1[1][0], infoWay[0].TimePeres2[1][0], infoWay[0].CityTo, infoWay[0].TimeTo[0], infoWay[0].Distance);

// цикл для вывода массива времён

for (int j = 1; j < t; j++)

printf("| \t\t | %-8s | \t\t | %-8s || %-8s | \t\t | %-7s || %-7s | \t\t | %-7s | \t | \n", infoWay[0].TimeFrom[j], infoWay[0].TimePeres1[0][j], infoWay[0].TimePeres2[0][j], infoWay[0].TimePeres1[1][j], infoWay[0].TimePeres2[1][j], infoWay[0].TimeTo[j]);

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

}

else return;

}

int NumberOfStructs(FILE\* fp) // поиск кол-ва структур в файле

{

int i = 0;

char str[256];

// считываем строки до тез пор, пока не дойдём до конца

while (fgets(str, 256, fp) != NULL)

i++;

return i;

}

// функция для возвращения пользовательского меню

int returnFunc()

{

int choice;

printf("\nEnter the any button to return the menu: ");

scanf\_s("%d", &choice);

rewind(stdin);

return 1;

}

struct s2\* InitialMenu(int graph[V][V], struct s1\* info, struct s2\* infoWay, int n, int\* num) // пользовательское меню

{

system("cls");

printf("1.Enter information about the trip.\n");

printf("2.Display information about all cities of the railway.\n");

printf("3.Display information about your last trip.\n");

printf("4.Save your trip way information in TXT file.\n");

printf("5.Exit.\n");

// объявление и выделение памяти для хранения названия города, из

// которого едет пользователь

char\* StrCityFrom = (char\*)calloc(N, sizeof(char));

// объявление и выделение памяти для хранения названия города, в

// который пользователь едет

char\* StrCityTo = (char\*)calloc(N, sizeof(char));

// индекс города, из которого едет пользователь, в массиве структур

int IndexFrom;

// индекс города, в который едет пользователь, в массиве структур

int IndexTo;

int choice = getchar(); // выбор опции

rewind(stdin);

switch (choice)

{

case '1': // ввод информации о поездке

system("cls"); // очитска консоли

// ввод названия города, из которого едет пользователь

StrCityFrom = InputCityFrom(info);

// ввод названия города, в который едет пользователь

StrCityTo = InputCityTo(info);

// сохранение названий городов в структуру, хранящую информацию о

// поездке

strcpy(infoWay[0].CityFrom, StrCityFrom);

strcpy(infoWay[0].CityTo, StrCityTo);

// получение индекса города, из которого едет пользователь, в массиве

// структур

IndexFrom = find\_city\_index(info, StrCityFrom);

// получение индекса города, в который едет пользователь, в массиве

// структур

IndexTo = find\_city\_index(info, StrCityTo);

// алгорит Дейкстры для получения информации о станциях пересадок,

// если такие есть, нахождение кратчайшего пути между станциями и

// сохранение данной информации в структуру

infoWay = dijkstra(graph, IndexFrom, IndexTo, info, infoWay, num);

returnFunc(); // возвращение в пользовательское меню

// вывод пользовательского меню и сохранение информации в структуру

infoWay = InitialMenu(graph, info, infoWay, n, num);

break;

case '2': // вывод информации о всех городах

system("cls");

printf("The information about all cities:\n");

// вывод массива структур, хранящего информацию о всех городах

PrintStruct\_s1(info, n);

returnFunc();

infoWay = InitialMenu(graph, info, infoWay, n, num);

break;

case '3': // вывод информации о поездке

system("cls");

printf("The information about your trip:\n");

// если в стрктуре отсутствует инофрмация о городе, из которого едет

// пользователь, то информции о поездке нет

if (infoWay[0].CityFrom == NULL)

{

printf("There is no information about your trip!\n");

returnFunc();

infoWay = InitialMenu(graph, info, infoWay, n, num);

break;

}

else

// иначе вывод структуры, хранящей информацию о поездке

// пользователя

PrintStruct\_s2(infoWay, \*num);

returnFunc();

infoWay = InitialMenu(graph, info, infoWay, n, num);

break;

// сохранение информации о поездке пользователя в текстовый файл

case '4':

system("cls");

// запись структуры в текстовый файл

InputInformationInFileS2(infoWay, \*num);

printf("TXT file is successfuly created. Name of this file is result.txt.\n");

returnFunc();

infoWay = InitialMenu(graph, info, infoWay, n, num);

break;

case '5': // закончить выполнение программы

system("cls");

printf("Thank you fo using our service. Goodbye!");

break;

// при вводе другого варианта, выводится сообщение

default:

printf("There is no this option. Try again please!");

}

return infoWay;

}

// функция для поиска индекса вершины с наименьшим расстоянием от

// стартовой вершины

int min\_distance(int dist[], int visited[])

{

// изначальное расстояние принимается за бесконечно большое число - min,

int min = INT\_MAX, min\_index;

// цикл для прохождения по всем городам

for (int i = 0; i < V; i++)

{

if (visited[i] == 0 && dist[i] <= min)

{

min = dist[i];

// запоминаем индекс элемента с минимальным расстоянием

min\_index = i;

}

}

return min\_index;

}

// функция реализующая алгоритм Дейкстры с модификацией для учёта

// станций пересадок

struct s2\* dijkstra(int graph[V][V], int start, int end, struct s1\* info, struct s2\* infoWay, int\* num)

{

int dist[V]; // массив расстояний от стартовой вершины

int visited[V]; // массив посещенных вершин

int prev[V]; // массив предыдущих вершин на пути

int number = -1;

int numb = 0;

// цикл для инициализации переменных для каждого города

for (int i = 0; i < V; i++)

{

// инициализация всех расстояний как "бесконечность"

dist[i] = INT\_MAX;

// отметка всех вершин как не посещённых

visited[i] = 0;

prev[i] = -1;

}

dist[start] = 0;

// поиск кратчайшего пути для всех вершин графа

for (int i = 0; i < V - 1; i++)

{

// поиск вершины с наименьшей стоимостью среди вершин, ещё не

// включённых в кратчайший путь

int u = min\_distance(dist, visited);

// отметка выбранной вершины как посещённой

visited[u] = 1;

// обновление расстояний до смежных вершин, учитывая станции

// пересадок

for (int v = 0; v < V; v++)

{

// Если вершина ещё не включена в кратчайший путь,

// есть ребро из выбранной вершины до смежной вершины,

// и расстояние от источника до выбранной вершины

// плюс стоимость ребра меньше текущего расстояния до смежной

// вершины, то обновляем расстояние до смежной вершины

if (visited[v] == 0 && graph[u][v] && dist[u] != INT\_MAX && dist[u] + + graph[u][v] < dist[v])

{

dist[v] = dist[u] + graph[u][v];

prev[v] = u;

}

}

}

// если расстояние до конечного города поездки равно "бесконечности", то // путь не найден

if (dist[end] == INT\_MAX)

{

printf("No path found.\n");

exit(-1);

}

// текущее положение начинается с конечного города поездки

//int index = 0;

int current = end;

// индекс города, где будет осуществленна ближайшая пересадка

int IndexTransfer;

int nap; // направление

// сохранение последовательности предыдущих вершин на пути

int ind = prev[current];

int flag = 0; // город не является пересадкой

// если начальный или конечный город является городом-пересадкой, то

if (info[start].Line == 8 || info[start].Line == 7 || info[end].Line == 8 || info[end].Line == 7)

flag = 1; // обозначаем это

// пока не пройдём всю последовательность предыдущих вершин на

// маршруте

while (prev[current] != -1)

{

// если линии разные, то

if (info[current].Line != info[prev[current]].Line)

// +1 к кол-ву пересадок на маршруте

number = number + 1;

// становимся на предыдущиую вершину на маршруте

current = prev[current];

}

// сохраняем кол-во пересадок на маршруте

numb = number;

// возвращаем последовательнось предыдущих вершин на пути

prev[current] = ind;

// если пересадки есть, то

if (number > 0)

{

// снова проходим по всей последовательности предыдущих вершин на

// маршруте

while (prev[current] != -1)

{

// если линии разные, то

if (info[current].Line != info[prev[current]].Line)

{

// сохраняем название города, как город-пересадку

strcpy(infoWay[0].Peresadki[numb - 1], info[prev[current]].City);

numb = numb - 1; // уменьшаем кол-во пресадок

// если кол-во пересадок == 0, то заканчиваем цикл

if (numb == 0)

break;

}

// становимся на предыдущую вершину на маршруте

current = prev[current];

}

}

printf("\n");

// получаем индекс ближайшей пересадки

IndexTransfer = find\_city\_index(info, infoWay[0].Peresadki[0]);

// определяем напрваление (если nap == 1, то поездке от центра, если == 0, // то к центру)

nap = napravl(info, start, IndexTransfer, end, number);

// считываем нужные времена отбытия и прибытия

infoWay = GetTime(info, infoWay, start, end, nap, number);

// считываем нужные времена прибытия и отбытия со станций пересадок

infoWay = GetTimeOfPeresadki(info, infoWay, nap, number);

// сохраняем расстояние между начальным и конечным городами

infoWay[0].Distance = dist[end];

// сохраняем кол-во пересадок

\*num = number;

return infoWay;

}

// функция для получения индекса города

int find\_city\_index(struct s1\* info, char\* name)

{

// цикл для прохождения по всем городам массива структур

for (int i = 0; i < V; i++)

{

// если есть совпадение, то возвращаем i, то есть индекс данного города в // массиве структур

if (strcmp(info[i].City, name) == 0)

return i;

}

return -1; // иначе возвращаем -1

}

// определение напрваления поездки (от центра или к центру) для чтения

// необходимых масивов времён

int napravl(struct s1\* info, int index1, int index2, int end, int num)

{

// первое расстояние принимаем равным расстоянию города, из которого

// едет пользователь

int dist1 = info[index1].Dist;

int dist2; // второе расстояние

int napravl;

if (num == 0) // если станций пересадок нет, то...

// второе расстояние принимаем равным расстоянию города, в который

// едет пользователь

dist2 = info[end].Dist;

else // если станции пересадок есть, то...

// второе расстояни принимаем равным расстоянию города, в котором

// осуществляется ближайшая пересадка с начала поездки

dist2 = info[index2].Dist;

// если разность данных расстояний < 0, то

if ((dist1 - dist2) < 0)

napravl = 1; // поездка от центра

else

napravl = 0; // поездка к центру

return napravl;

}

// функция сохранения нужных времён прибытия и отбытия

struct s2\* GetTime(struct s1\* info, struct s2\* infoWay, int indexFrom, int indexTo, int napravl, int num)

{

if (napravl == 0) // ЕСЛИ поездке к центру, то

{

if (num == 0) // если кол-во пересадок == 0, то

{

for (int i = 0; i < t; i++) // цикл для чтения массива времён

{

// считываем время отбытия из начального города

strcpy(infoWay[0].TimeFrom[i], info[indexFrom].Time2[i]);

// считываем время прибытия в конечный город

strcpy(infoWay[0].TimeTo[i], info[indexTo].Time2[i]);

}

}

else // иначе

{

for (int i = 0; i < t; i++) // цикл для чтения массива времён

{

// считываем время отбытия из начального города

strcpy(infoWay[0].TimeFrom[i], info[indexFrom].Time2[i]);

// считываем время прибытия в конечный город учитывая пересдку

strcpy(infoWay[0].TimeTo[i], info[indexTo].Time1[i]);

}

}

}

else // ИНАЧЕ (поездка от центра), то

{

if (num == 0) // если кол-во пересадок == 0, то

{

for (int i = 0; i < t; i++) // цикл для чтения массива времён

{

// считываем время отбытия из начального города

strcpy(infoWay[0].TimeFrom[i], info[indexFrom].Time1[i]);

// считываем время прибытия в конечный город

strcpy(infoWay[0].TimeTo[i], info[indexTo].Time1[i]);

}

}

else // иначе

{

for (int i = 0; i < t; i++)

{

// считываем время отбытия из начального города учитывая

// пересадку

strcpy(infoWay[0].TimeFrom[i], info[indexFrom].Time1[i]);

// считываем время прибытия в конечный город

strcpy(infoWay[0].TimeTo[i], info[indexTo].Time2[i]);

}

}

}

return infoWay;

}

// функция сохранения нужных времён прибытия и отбытия станций

// пересадок

struct s2\* GetTimeOfPeresadki(struct s1\* info, struct s2\* infoWay, int napravl, int number)

{

if (napravl == 0) // ЕСЛИ поездке к центру, то

{

// цикл для прохождения по всем городам-пересадкам

for (int j = 0; j < number; j++)

for (int i = 0; i < t; i++) // цикл для чтения массива времён

{

// если второй пересадкой является Минск, то

if ((j == 1) && strcmp(info[0].City, infoWay[0].Peresadki[j]) == 0)

{

// считываем время прибытия в город-пересадку

strcpy(infoWay[0].TimePeres1[j][i], info[find\_city\_index(info, infoWay[0].Peresadki[j])].Time2[i]);

// считываем время отбытия из города-пересадки

strcpy(infoWay[0].TimePeres2[j][i], info[find\_city\_index(info, infoWay[0].Peresadki[j])].Time1[i]);

}

else if (j == 1)

{

// считываем нужное время прибытия в город-пересадку

strcpy(infoWay[0].TimePeres1[j][i], info[find\_city\_index(info, infoWay[0].Peresadki[j])].Time1[i]);

// считываем нужное время отбытия из города-пересадки

strcpy(infoWay[0].TimePeres2[j][i], info[find\_city\_index(info, infoWay[0].Peresadki[j])].Time1[i]);

}

// если первой пересадкой является город Орша, то

else if ((j == 0) && strcmp(info[8].City, infoWay[0].Peresadki[j]) == 0)

{

// считываем время прибытия в город-пересадку

strcpy(infoWay[0].TimePeres1[j][i], info[find\_city\_index(info, infoWay[0].Peresadki[j])].Time2[i]);

// считываем время отбытия из города-пересадки

strcpy(infoWay[0].TimePeres2[j][i], info[find\_city\_index(info, infoWay[0].Peresadki[j])].Time2[i]);

}

else

{

// считываем время прибытия в город-пересадку

strcpy(infoWay[0].TimePeres1[j][i], info[find\_city\_index(info, infoWay[0].Peresadki[j])].Time2[i]);

// считываем время отбытия из города-пересадки

strcpy(infoWay[0].TimePeres2[j][i], info[find\_city\_index(info, infoWay[0].Peresadki[j])].Time1[i]);

}

}

}

else // ИНАЧЕ (поездка от центра), то

{

for (int j = 0; j < number; j++)

for (int i = 0; i < t; i++)

{

if ((j == 1) && strcmp(info[0].City, infoWay[0].Peresadki[j]) == 0)

{

strcpy(infoWay[0].TimePeres1[j][i], info[find\_city\_index(info, infoWay[0].Peresadki[j])].Time1[i]);

strcpy(infoWay[0].TimePeres2[j][i], info[find\_city\_index(info, infoWay[0].Peresadki[j])].Time1[i]);

}

else if (j == 1)

{

strcpy(infoWay[0].TimePeres1[j][i], info[find\_city\_index(info, infoWay[0].Peresadki[j])].Time2[i]);

strcpy(infoWay[0].TimePeres2[j][i], info[find\_city\_index(info, infoWay[0].Peresadki[j])].Time2[i]);

}

else if ((j == 0) && strcmp(info[8].City, infoWay[0].Peresadki[j]) == 0)

{

strcpy(infoWay[0].TimePeres1[j][i], info[find\_city\_index(info, infoWay[0].Peresadki[j])].Time2[i]);

strcpy(infoWay[0].TimePeres2[j][i], info[find\_city\_index(info, infoWay[0].Peresadki[j])].Time2[i]);

}

else

{

strcpy(infoWay[0].TimePeres1[j][i], info[find\_city\_index(info, infoWay[0].Peresadki[j])].Time1[i]);

strcpy(infoWay[0].TimePeres2[j][i], info[find\_city\_index(info, infoWay[0].Peresadki[j])].Time2[i]);

}

}

}

return infoWay;

}