#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#define MAX\_STATIONS 50

#define MAX\_ROUTES 10

#define MAX\_NAME 50

#define MAX\_EDGES 100

#define BUS\_SPEED 0.25 // 公交车速度，km/min

#define WALK\_SPEED 0.1 // 步行速度，km/min

#define MAX\_PASSWORD 20 // 密码最大长度

// 站点结构体

typedef struct {

int id;

char name[MAX\_NAME];

int is\_bus\_station; // 1为公交站点，0为非公交站点

} Station;

// 线路结构体

typedef struct {

int id;

int station\_count;

int stations[MAX\_STATIONS];

float distances[MAX\_STATIONS]; // 相邻站点间距离

} Route;

// 步行边结构体

typedef struct {

int from;

int to;

float distance;

} WalkEdge;

// 路径结构体

typedef struct {

int path[MAX\_STATIONS];

int path\_length;

float time;

float distance;

int transfers;

char modes[MAX\_STATIONS][10];

} Path;

// 全局变量

Station stations[MAX\_STATIONS];

Route bus\_routes[MAX\_ROUTES];

WalkEdge walk\_edges[MAX\_EDGES];

int station\_count = 0;

int route\_count = 0;

int walk\_edge\_count = 0;

char admin\_password[MAX\_PASSWORD] = "admin123";

int password\_set = 1;

// 保存数据到文件

void save\_data() {

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

if (!fp) {

printf("文件打开失败。\n");

return;

}

fprintf(fp, "%d\n", station\_count);

for (int i = 0; i < station\_count; i++) {

fprintf(fp, "%d %s %d\n", stations[i].id, stations[i].name, stations[i].is\_bus\_station);

}

fprintf(fp, "%d\n", route\_count);

for (int i = 0; i < route\_count; i++) {

fprintf(fp, "%d %d", bus\_routes[i].id, bus\_routes[i].station\_count);

for (int j = 0; j < bus\_routes[i].station\_count; j++) {

fprintf(fp, " %d", bus\_routes[i].stations[j]);

}

for (int j = 0; j < bus\_routes[i].station\_count - 1; j++) {

fprintf(fp, " %.2f", bus\_routes[i].distances[j]);

}

fprintf(fp, "\n");

}

fprintf(fp, "%d\n", walk\_edge\_count);

for (int i = 0; i < walk\_edge\_count; i++) {

fprintf(fp, "%d %d %.2f\n", walk\_edges[i].from, walk\_edges[i].to, walk\_edges[i].distance);

}

fprintf(fp, "%d %s\n", password\_set, admin\_password);

fclose(fp);

printf("数据已保存到文件。\n");

}

// 从文件加载数据

void load\_data() {

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

if (!fp) {

printf("文件打开失败，将使用默认数据初始化。\n");

return;

}

if (fscanf(fp, "%d", &station\_count) != 1) {

fclose(fp);

return;

}

for (int i = 0; i < station\_count; i++) {

if (fscanf(fp, "%d %s %d", &stations[i].id, stations[i].name, &stations[i].is\_bus\_station) != 3) {

fclose(fp);

return;

}

}

if (fscanf(fp, "%d", &route\_count) != 1) {

fclose(fp);

return;

}

for (int i = 0; i < route\_count; i++) {

if (fscanf(fp, "%d %d", &bus\_routes[i].id, &bus\_routes[i].station\_count) != 2) {

fclose(fp);

return;

}

for (int j = 0; j < bus\_routes[i].station\_count; j++) {

if (fscanf(fp, "%d", &bus\_routes[i].stations[j]) != 1) {

fclose(fp);

return;

}

}

for (int j = 0; j < bus\_routes[i].station\_count - 1; j++) {

if (fscanf(fp, "%f", &bus\_routes[i].distances[j]) != 1) {

fclose(fp);

return;

}

}

}

if (fscanf(fp, "%d", &walk\_edge\_count) != 1) {

fclose(fp);

return;

}

walk\_edge\_count = (walk\_edge\_count > MAX\_EDGES) ? MAX\_EDGES : walk\_edge\_count;

for (int i = 0; i < walk\_edge\_count; i++) {

int from, to;

float distance;

if (fscanf(fp, "%d %d %f", &from, &to, &distance) != 3) {

printf("读取步行边 %d 失败，文件格式可能错误。\n", i);

fclose(fp);

return;

}

if (from == 0 && to == 0 && distance == 0.00) {

break;

}

walk\_edges[i].from = from;

walk\_edges[i].to = to;

walk\_edges[i].distance = distance;

}

if (fscanf(fp, "%d %s", &password\_set, admin\_password) != 2) {

fclose(fp);

return;

}

fclose(fp);

printf("数据已从文件加载。\n");

}

// 更新步行边

void update\_walk\_edges() {

walk\_edge\_count = 0;

// 添加默认步行边

int idx = 0;

walk\_edges[idx++] = (WalkEdge){1, 2, 0.25}; walk\_edges[idx++] = (WalkEdge){2, 1, 0.25};

walk\_edges[idx++] = (WalkEdge){1, 11, 1.5}; walk\_edges[idx++] = (WalkEdge){11, 1, 1.5};

walk\_edges[idx++] = (WalkEdge){1, 7, 1.0}; walk\_edges[idx++] = (WalkEdge){7, 1, 1.0};

walk\_edges[idx++] = (WalkEdge){1, 6, 1.0}; walk\_edges[idx++] = (WalkEdge){6, 1, 1.0};

walk\_edges[idx++] = (WalkEdge){2, 3, 0.25}; walk\_edges[idx++] = (WalkEdge){3, 2, 0.25};

walk\_edges[idx++] = (WalkEdge){4, 5, 0.5}; walk\_edges[idx++] = (WalkEdge){5, 4, 0.5};

walk\_edges[idx++] = (WalkEdge){7, 12, 0.5}; walk\_edges[idx++] = (WalkEdge){12, 7, 0.5};

walk\_edges[idx++] = (WalkEdge){11, 12, 0.25}; walk\_edges[idx++] = (WalkEdge){12, 11, 0.25};

walk\_edges[idx++] = (WalkEdge){12, 6, 1.0}; walk\_edges[idx++] = (WalkEdge){6, 12, 1.0};

walk\_edges[idx++] = (WalkEdge){12, 13, 0.25}; walk\_edges[idx++] = (WalkEdge){13, 12, 0.25};

walk\_edges[idx++] = (WalkEdge){6, 5, 1.0}; walk\_edges[idx++] = (WalkEdge){5, 6, 1.0};

walk\_edges[idx++] = (WalkEdge){5, 14, 1.0}; walk\_edges[idx++] = (WalkEdge){14, 5, 1.0};

walk\_edges[idx++] = (WalkEdge){6, 14, 1.0}; walk\_edges[idx++] = (WalkEdge){14, 6, 1.0};

walk\_edges[idx++] = (WalkEdge){13, 14, 1.0}; walk\_edges[idx++] = (WalkEdge){14, 13, 1.0};

walk\_edges[idx++] = (WalkEdge){14, 16, 1.0}; walk\_edges[idx++] = (WalkEdge){16, 14, 1.0};

walk\_edges[idx++] = (WalkEdge){13, 16, 0.5}; walk\_edges[idx++] = (WalkEdge){16, 13, 0.5};

walk\_edges[idx++] = (WalkEdge){16, 15, 0.5}; walk\_edges[idx++] = (WalkEdge){15, 16, 0.5};

walk\_edges[idx++] = (WalkEdge){16, 10, 0.5}; walk\_edges[idx++] = (WalkEdge){10, 16, 0.5};

walk\_edges[idx++] = (WalkEdge){10, 9, 1.0}; walk\_edges[idx++] = (WalkEdge){9, 10, 1.0};

walk\_edges[idx++] = (WalkEdge){9, 15, 0.5}; walk\_edges[idx++] = (WalkEdge){15, 9, 0.5};

walk\_edges[idx++] = (WalkEdge){9, 8, 0.5}; walk\_edges[idx++] = (WalkEdge){8, 9, 0.5};

walk\_edges[idx++] = (WalkEdge){8, 3, 0.5}; walk\_edges[idx++] = (WalkEdge){3, 8, 0.5};

walk\_edges[idx++] = (WalkEdge){9, 1, 1.5}; walk\_edges[idx++] = (WalkEdge){1, 9, 1.5};

walk\_edges[idx++] = (WalkEdge){9, 3, 1.0}; walk\_edges[idx++] = (WalkEdge){3, 9, 1.0};

walk\_edge\_count = idx;

// 为公交线路上的相邻站点添加步行边

for (int r = 0; r < route\_count; r++) {

for (int j = 0; j < bus\_routes[r].station\_count - 1; j++) {

int from = bus\_routes[r].stations[j];

int to = bus\_routes[r].stations[j+1];

float distance = bus\_routes[r].distances[j];

int exists = 0;

for (int k = 0; k < walk\_edge\_count; k++) {

if ((walk\_edges[k].from == from && walk\_edges[k].to == to) ||

(walk\_edges[k].from == to && walk\_edges[k].to == from)) {

exists = 1;

break;

}

}

if (!exists && walk\_edge\_count < MAX\_EDGES - 2) {

walk\_edges[walk\_edge\_count++] = (WalkEdge){from, to, distance};

walk\_edges[walk\_edge\_count++] = (WalkEdge){to, from, distance};

}

}

}

save\_data();

}

// 初始化默认数据

void init\_default\_data() {

station\_count = 16;

stations[0] = (Station){1, "五棵松", 1};

stations[1] = (Station){2, "教务处", 0};

stations[2] = (Station){3, "图书馆", 1};

stations[3] = (Station){4, "教职工活动中心", 1};

stations[4] = (Station){5, "职工医院", 1};

stations[5] = (Station){6, "大礼堂", 1};

stations[6] = (Station){7, "三岔口", 1};

stations[7] = (Station){8, "研究生院", 0};

stations[8] = (Station){9, "耒耜大楼", 1};

stations[9] = (Station){10, "宿舍G区", 1};

stations[10] = (Station){11, "宿舍C区", 0};

stations[11] = (Station){12, "宿舍B区", 1};

stations[12] = (Station){13, "宿舍D区", 1};

stations[13] = (Station){14, "行政二号楼", 1};

stations[14] = (Station){15, "京江楼", 1};

stations[15] = (Station){16, "六食堂", 1};

route\_count = 6;

bus\_routes[0] = (Route){1, 5, {1, 7, 12, 13, 16}, {1.0, 0.5, 0.25, 0.5}};

bus\_routes[1] = (Route){2, 4, {16, 10, 9, 3}, {0.5, 1.0, 1.0}};

bus\_routes[2] = (Route){3, 4, {1, 9, 15, 16}, {1.5, 0.5, 0.5}};

bus\_routes[3] = (Route){4, 4, {12, 6, 5, 4}, {1.0, 1.0, 0.5}};

bus\_routes[4] = (Route){5, 3, {5, 14, 16}, {1.0, 1.0}};

bus\_routes[5] = (Route){6, 4, {13, 14, 6, 1}, {1.0, 1.0, 1.0}};

update\_walk\_edges();

}

// 根据站点名称查找ID

int find\_station\_id(char \*name) {

for (int i = 0; i < station\_count; i++) {

if (strcmp(stations[i].name, name) == 0) {

return stations[i].id;

}

}

return -1;

}

// 验证输入是否为有效浮点数

int is\_valid\_float(char \*str) {

int has\_dot = 0, has\_digit = 0;

for (int i = 0; str[i]; i++) {

if (isdigit(str[i])) has\_digit = 1;

else if (str[i] == '.' && !has\_dot) has\_dot = 1;

else return 0;

}

return has\_digit;

}

// 更新站点是否为公交站点

void update\_bus\_station\_status() {

for (int i = 0; i < station\_count; i++) {

stations[i].is\_bus\_station = 0;

for (int r = 0; r < route\_count; r++) {

for (int j = 0; j < bus\_routes[r].station\_count; j++) {

if (bus\_routes[r].stations[j] == stations[i].id) {

stations[i].is\_bus\_station = 1;

break;

}

}

if (stations[i].is\_bus\_station) break;

}

}

}

// 验证密码

int verify\_password() {

if (!password\_set) return 1;

char input\_password[MAX\_PASSWORD];

printf("请输入管理员密码：");

scanf("%s", input\_password);

return strcmp(input\_password, admin\_password) == 0;

}

// 管理密码

void manage\_password() {

int sub\_choice;

while (1) {

printf("\n=== 密码管理 ===\n");

printf("1. 修改密码\n");

printf("2. 取消密码\n");

printf("3. 设置密码\n");

printf("4. 返回管理员菜单\n");

printf("5. 退出系统\n");

printf("请输入选项：");

scanf("%d", &sub\_choice);

switch (sub\_choice) {

case 1:

if (verify\_password()) {

printf("请输入新密码：");

scanf("%s", admin\_password);

password\_set = 1;

save\_data();

printf("密码修改成功。\n");

} else {

printf("密码错误。\n");

}

break;

case 2:

if (verify\_password()) {

password\_set = 0;

strcpy(admin\_password, "");

save\_data();

printf("密码已取消。\n");

} else {

printf("密码错误。\n");

}

break;

case 3:

printf("请输入新密码：");

scanf("%s", admin\_password);

password\_set = 1;

save\_data();

printf("密码设置成功。\n");

break;

case 4:

return;

case 5:

printf("感谢使用本系统。\n");

exit(0);

default:

printf("无效选项。\n");

}

}

}

// 添加新站点

void add\_new\_station() {

if (station\_count >= MAX\_STATIONS) {

printf("站点数量已达上限。\n");

return;

}

Station new\_station;

new\_station.id = station\_count + 1;

printf("请输入站点名称：");

scanf("%s", new\_station.name);

new\_station.is\_bus\_station = 1;

stations[station\_count++] = new\_station;

save\_data();

printf("新站点添加成功：ID=%d, 名称=%s, 是否公交站点=%d\n",

new\_station.id, new\_station.name, new\_station.is\_bus\_station);

}

// 添加公交线路

void add\_bus\_route() {

if (route\_count >= MAX\_ROUTES) {

printf("公交线路已达上限。\n");

return;

}

Route new\_route;

new\_route.id = route\_count + 1;

printf("请输入站点数量：");

int station\_count\_input;

if (scanf("%d", &station\_count\_input) != 1 || station\_count\_input <= 0 || station\_count\_input > MAX\_STATIONS) {

printf("站点数量无效。\n");

while (getchar() != '\n');

return;

}

new\_route.station\_count = station\_count\_input;

printf("请按行驶顺序输入站点名称（每行一个）：\n");

for (int i = 0; i < new\_route.station\_count; i++) {

char name[MAX\_NAME];

scanf("%s", name);

int id = find\_station\_id(name);

if (id == -1) {

printf("站点 %s 不存在。\n", name);

return;

}

new\_route.stations[i] = id;

}

printf("请输入相邻站点间距离（km，用空格分隔）：");

for (int i = 0; i < new\_route.station\_count - 1; i++) {

char dist\_str[20];

scanf("%s", dist\_str);

if (!is\_valid\_float(dist\_str)) {

printf("无效距离输入。\n");

return;

}

new\_route.distances[i] = atof(dist\_str);

if (new\_route.distances[i] <= 0) {

printf("距离必须为正数。\n");

return;

}

}

bus\_routes[route\_count++] = new\_route;

update\_bus\_station\_status();

update\_walk\_edges();

printf("新公交线路添加成功：线路ID=%d，站点数=%d\n站点：", new\_route.id, new\_route.station\_count);

for (int i = 0; i < new\_route.station\_count; i++) {

printf("%s", stations[new\_route.stations[i]-1].name);

if (i < new\_route.station\_count - 1) {

printf("-(%.2fkm)->", new\_route.distances[i]);

}

}

printf("\n");

}

// 修改站点信息

void modify\_station\_info(int route\_idx) {

printf("当前线路ID=%d，站点：", bus\_routes[route\_idx].id);

for (int j = 0; j < bus\_routes[route\_idx].station\_count; j++) {

printf("%s", stations[bus\_routes[route\_idx].stations[j]-1].name);

if (j < bus\_routes[route\_idx].station\_count - 1) {

printf("-(%.2fkm)->", bus\_routes[route\_idx].distances[j]);

}

}

printf("\n");

int sub\_choice;

printf("1. 修改站点名称\n");

printf("2. 修改相邻站点距离\n");

printf("3. 返回\n");

printf("请输入选项：");

scanf("%d", &sub\_choice);

switch (sub\_choice) {

case 1: {

char old\_name[MAX\_NAME], new\_name[MAX\_NAME];

printf("请输入要修改的站点名称：");

scanf("%s", old\_name);

int station\_idx = -1;

for (int j = 0; j < bus\_routes[route\_idx].station\_count; j++) {

if (strcmp(stations[bus\_routes[route\_idx].stations[j]-1].name, old\_name) == 0) {

station\_idx = bus\_routes[route\_idx].stations[j] - 1;

break;

}

}

if (station\_idx == -1) {

printf("线路中未找到站点 %s。\n", old\_name);

return;

}

printf("请输入新站点名称：");

scanf("%s", new\_name);

if (find\_station\_id(new\_name) != -1) {

printf("站点名称 %s 已存在。\n", new\_name);

return;

}

strcpy(stations[station\_idx].name, new\_name);

save\_data();

printf("站点名称修改成功：%s -> %s\n", old\_name, new\_name);

break;

}

case 2: {

printf("请输入要修改的相邻站点对（格式：站点1 站点2）：");

char name1[MAX\_NAME], name2[MAX\_NAME];

scanf("%s %s", name1, name2);

int idx1 = -1, idx2 = -1;

for (int j = 0; j < bus\_routes[route\_idx].station\_count; j++) {

if (strcmp(stations[bus\_routes[route\_idx].stations[j]-1].name, name1) == 0) idx1 = j;

if (strcmp(stations[bus\_routes[route\_idx].stations[j]-1].name, name2) == 0) idx2 = j;

}

if (idx1 == -1 || idx2 == -1 || abs(idx1 - idx2) != 1) {

printf("站点对无效或非相邻站点。\n");

return;

}

int dist\_idx = idx1 < idx2 ? idx1 : idx2;

printf("请输入 %s 到 %s 的新距离（km）：", name1, name2);

char dist\_str[20];

scanf("%s", dist\_str);

if (!is\_valid\_float(dist\_str)) {

printf("无效距离输入。\n");

return;

}

float new\_dist = atof(dist\_str);

if (new\_dist <= 0) {

printf("距离必须为正数。\n");

return;

}

bus\_routes[route\_idx].distances[dist\_idx] = new\_dist;

save\_data();

printf("距离修改成功：%s 到 %s 的距离更新为 %.2fkm\n", name1, name2, new\_dist);

break;

}

case 3:

return;

default:

printf("无效选项。\n");

}

}

// 修改公交线路

void modify\_bus\_route() {

int route\_id;

printf("请输入要修改的线路ID：");

if (scanf("%d", &route\_id) != 1 || route\_id <= 0) {

printf("无效线路ID。\n");

while (getchar() != '\n');

return;

}

int route\_idx = -1;

for (int i = 0; i < route\_count; i++) {

if (bus\_routes[i].id == route\_id) {

route\_idx = i;

break;

}

}

if (route\_idx == -1) {

printf("未找到该线路。\n");

return;

}

int sub\_choice;

printf("\n=== 修改公交线路 ===\n");

printf("1. 仅修改站点信息（名称或距离）\n");

printf("2. 仅修改公交线路（站点顺序和数量）\n");

printf("3. 返回\n");

printf("请输入选项：");

scanf("%d", &sub\_choice);

switch (sub\_choice) {

case 1:

modify\_station\_info(route\_idx);

break;

case 2: {

printf("请输入新的站点数量：");

int new\_count;

if (scanf("%d", &new\_count) != 1 || new\_count <= 0 || new\_count > MAX\_STATIONS) {

printf("站点数量无效。\n");

while (getchar() != '\n');

return;

}

bus\_routes[route\_idx].station\_count = new\_count;

printf("请按新行驶顺序输入站点名称（每行一个）：\n");

for (int j = 0; j < bus\_routes[route\_idx].station\_count; j++) {

char name[MAX\_NAME];

scanf("%s", name);

int id = find\_station\_id(name);

if (id == -1) {

printf("站点 %s 不存在。\n", name);

return;

}

bus\_routes[route\_idx].stations[j] = id;

}

printf("请输入新的相邻站点间距离（km，用空格分隔）：");

for (int j = 0; j < bus\_routes[route\_idx].station\_count - 1; j++) {

char dist\_str[20];

scanf("%s", dist\_str);

if (!is\_valid\_float(dist\_str)) {

printf("无效距离输入。\n");

return;

}

bus\_routes[route\_idx].distances[j] = atof(dist\_str);

if (bus\_routes[route\_idx].distances[j] <= 0) {

printf("距离必须为正数。\n");

return;

}

}

update\_bus\_station\_status();

update\_walk\_edges();

printf("公交线路修改成功：线路ID=%d，站点数=%d\n站点：", bus\_routes[route\_idx].id, bus\_routes[route\_idx].station\_count);

for (int j = 0; j < bus\_routes[route\_idx].station\_count; j++) {

printf("%s", stations[bus\_routes[route\_idx].stations[j]-1].name);

if (j < bus\_routes[route\_idx].station\_count - 1) {

printf("-(%.2fkm)->", bus\_routes[route\_idx].distances[j]);

}

}

printf("\n");

break;

}

case 3:

return;

default:

printf("无效选项。\n");

}

}

// 从线路中删除站点

void delete\_station\_from\_route() {

int route\_id;

char station\_name[MAX\_NAME];

printf("请输入线路ID：");

if (scanf("%d", &route\_id) != 1 || route\_id <= 0) {

printf("无效线路ID。\n");

while (getchar() != '\n');

return;

}

printf("请输入要删除的站点名称：");

scanf("%s", station\_name);

int station\_id = find\_station\_id(station\_name);

if (station\_id == -1) {

printf("站点 %s 不存在。\n", station\_name);

return;

}

int route\_idx = -1;

for (int i = 0; i < route\_count; i++) {

if (bus\_routes[i].id == route\_id) {

route\_idx = i;

break;

}

}

if (route\_idx == -1) {

printf("未找到该线路。\n");

return;

}

int delete\_idx = -1;

for (int j = 0; j < bus\_routes[route\_idx].station\_count; j++) {

if (bus\_routes[route\_idx].stations[j] == station\_id) {

delete\_idx = j;

break;

}

}

if (delete\_idx == -1) {

printf("线路中未找到该站点。\n");

return;

}

int new\_count = bus\_routes[route\_idx].station\_count - 1;

int new\_stations[MAX\_STATIONS];

float new\_distances[MAX\_STATIONS];

for (int j = 0, k = 0; j < bus\_routes[route\_idx].station\_count; j++) {

if (j != delete\_idx) {

new\_stations[k] = bus\_routes[route\_idx].stations[j];

if (j < bus\_routes[route\_idx].station\_count - 1 && k < new\_count - 1) {

new\_distances[k] = bus\_routes[route\_idx].distances[j];

}

k++;

}

}

if (delete\_idx > 0 && delete\_idx < bus\_routes[route\_idx].station\_count - 1) {

printf("请输入站点 %s 到 %s 的新距离（km）：",

stations[new\_stations[delete\_idx-1]-1].name, stations[new\_stations[delete\_idx]-1].name);

char dist\_str[20];

scanf("%s", dist\_str);

if (!is\_valid\_float(dist\_str)) {

printf("无效距离输入。\n");

return;

}

new\_distances[delete\_idx-1] = atof(dist\_str);

if (new\_distances[delete\_idx-1] <= 0) {

printf("距离必须为正数。\n");

return;

}

}

bus\_routes[route\_idx].station\_count = new\_count;

for (int j = 0; j < new\_count; j++) {

bus\_routes[route\_idx].stations[j] = new\_stations[j];

if (j < new\_count - 1) {

bus\_routes[route\_idx].distances[j] = new\_distances[j];

}

}

update\_bus\_station\_status();

update\_walk\_edges();

printf("站点删除成功：线路ID=%d，剩余站点数=%d\n站点：", bus\_routes[route\_idx].id, bus\_routes[route\_idx].station\_count);

for (int j = 0; j < bus\_routes[route\_idx].station\_count; j++) {

printf("%s", stations[bus\_routes[route\_idx].stations[j]-1].name);

if (j < bus\_routes[route\_idx].station\_count - 1) {

printf("-(%.2fkm)->", bus\_routes[route\_idx].distances[j]);

}

}

printf("\n");

}

// 添加站点到线路

void add\_station\_to\_route() {

int route\_id;

char station\_name[MAX\_NAME];

int position;

printf("请输入线路ID：");

if (scanf("%d", &route\_id) != 1 || route\_id <= 0) {

printf("无效线路ID。\n");

while (getchar() != '\n');

return;

}

printf("请输入要添加的站点名称：");

scanf("%s", station\_name);

printf("请输入添加位置（0为起始位置）：");

if (scanf("%d", &position) != 1) {

printf("无效位置输入。\n");

while (getchar() != '\n');

return;

}

int station\_id = find\_station\_id(station\_name);

if (station\_id == -1) {

printf("站点 %s 不存在。\n", station\_name);

return;

}

int route\_idx = -1;

for (int i = 0; i < route\_count; i++) {

if (bus\_routes[i].id == route\_id) {

route\_idx = i;

break;

}

}

if (route\_idx == -1) {

printf("未找到该线路。\n");

return;

}

if (position < 0 || position > bus\_routes[route\_idx].station\_count) {

printf("位置无效。\n");

return;

}

for (int j = bus\_routes[route\_idx].station\_count; j > position; j--) {

bus\_routes[route\_idx].stations[j] = bus\_routes[route\_idx].stations[j-1];

if (j < bus\_routes[route\_idx].station\_count) {

bus\_routes[route\_idx].distances[j] = bus\_routes[route\_idx].distances[j-1];

}

}

bus\_routes[route\_idx].stations[position] = station\_id;

bus\_routes[route\_idx].station\_count++;

stations[station\_id-1].is\_bus\_station = 1;

if (position < bus\_routes[route\_idx].station\_count - 1) {

printf("请输入该站点到下一站的距离（km）：");

char dist\_str[20];

scanf("%s", dist\_str);

if (!is\_valid\_float(dist\_str) || atof(dist\_str) <= 0) {

printf("无效距离输入。\n");

return;

}

bus\_routes[route\_idx].distances[position] = atof(dist\_str);

}

if (position > 0) {

printf("请输入上一站到该站点的距离（km）：");

char dist\_str[20];

scanf("%s", dist\_str);

if (!is\_valid\_float(dist\_str) || atof(dist\_str) <= 0) {

printf("无效距离输入。\n");

return;

}

bus\_routes[route\_idx].distances[position-1] = atof(dist\_str);

}

update\_bus\_station\_status();

update\_walk\_edges();

printf("站点添加成功：线路ID=%d，站点数=%d\n站点：", bus\_routes[route\_idx].id, bus\_routes[route\_idx].station\_count);

for (int j = 0; j < bus\_routes[route\_idx].station\_count; j++) {

printf("%s", stations[bus\_routes[route\_idx].stations[j]-1].name);

if (j < bus\_routes[route\_idx].station\_count - 1) {

printf("-(%.2fkm)->", bus\_routes[route\_idx].distances[j]);

}

}

printf("\n");

}

// 查询站点经过的公交线路

void query\_station\_routes() {

char station\_name[MAX\_NAME];

printf("请输入站点名称：");

scanf("%s", station\_name);

int station\_id = find\_station\_id(station\_name);

if (station\_id == -1) {

printf("未找到该站点。\n");

return;

}

printf("经过%s的公交线路：\n", station\_name);

int found = 0;

for (int i = 0; i < route\_count; i++) {

for (int j = 0; j < bus\_routes[i].station\_count; j++) {

if (bus\_routes[i].stations[j] == station\_id) {

found = 1;

printf("线路%d：", bus\_routes[i].id);

float total\_distance = 0.0;

for (int k = 0; k < bus\_routes[i].station\_count; k++) {

printf("%s", stations[bus\_routes[i].stations[k]-1].name);

if (k < bus\_routes[i].station\_count - 1) {

printf("-(%.2fkm)->", bus\_routes[i].distances[k]);

total\_distance += bus\_routes[i].distances[k];

}

}

printf("\n总距离：%.2fkm，总时间：%.2f分钟\n", total\_distance, total\_distance / BUS\_SPEED);

}

}

}

if (!found) {

printf("该站点没有公交线路经过。\n");

}

}

// 显示所有公交线路

void display\_all\_routes() {

printf("所有校内公交线路：\n");

for (int i = 0; i < route\_count; i++) {

printf("线路%d：", bus\_routes[i].id);

float total\_distance = 0.0;

for (int j = 0; j < bus\_routes[i].station\_count; j++) {

printf("%s", stations[bus\_routes[i].stations[j]-1].name);

if (j < bus\_routes[i].station\_count - 1) {

printf("-(%.2fkm)->", bus\_routes[i].distances[j]);

total\_distance += bus\_routes[i].distances[j];

}

}

printf("\n总距离：%.2fkm，总时间：%.2f分钟\n", total\_distance, total\_distance / BUS\_SPEED);

}

}

// Dijkstra算法

Path dijkstra(int start\_id, int end\_id, int mode) {

float dist[MAX\_STATIONS]; // 主要度量（时间/换乘次数/距离）

float total\_time[MAX\_STATIONS]; // 总用时（用于最少换乘模式比较）

float total\_dist[MAX\_STATIONS]; // 总距离

int prev[MAX\_STATIONS];

int visited[MAX\_STATIONS];

char modes[MAX\_STATIONS][10];

int transfers[MAX\_STATIONS];

int prev\_route[MAX\_STATIONS]; // 记录上一个使用的线路ID

Path result = {{0}, 0, 99999.0, 99999.0, 999, {""}};

for (int i = 0; i < station\_count; i++) {

dist[i] = 99999.0;

total\_time[i] = 99999.0;

total\_dist[i] = 99999.0;

prev[i] = -1;

visited[i] = 0;

transfers[i] = 999;

prev\_route[i] = -1;

strcpy(modes[i], "");

}

dist[start\_id-1] = 0;

total\_time[start\_id-1] = 0;

total\_dist[start\_id-1] = 0;

transfers[start\_id-1] = 0;

// 处理非公交站点终点（最少换乘）

int target\_id = end\_id;

if (stations[end\_id-1].is\_bus\_station == 0 && mode == 1) {

float min\_dist = 99999.0;

for (int i = 0; i < walk\_edge\_count; i++) {

if (walk\_edges[i].from == end\_id && stations[walk\_edges[i].to-1].is\_bus\_station) {

if (walk\_edges[i].distance < min\_dist) {

min\_dist = walk\_edges[i].distance;

target\_id = walk\_edges[i].to;

}

}

if (walk\_edges[i].to == end\_id && stations[walk\_edges[i].from-1].is\_bus\_station) {

if (walk\_edges[i].distance < min\_dist) {

min\_dist = walk\_edges[i].distance;

target\_id = walk\_edges[i].from;

}

}

}

}

for (int i = 0; i < station\_count; i++) {

int min\_idx = -1;

float min\_dist = 99999.0;

for (int j = 0; j < station\_count; j++) {

if (!visited[j]) {

if (mode == 1) {

if (min\_idx == -1 || transfers[j] < transfers[min\_idx] ||

(transfers[j] == transfers[min\_idx] && total\_time[j] < total\_time[min\_idx])) {

min\_dist = dist[j];

min\_idx = j;

}

} else if (dist[j] < min\_dist) {

min\_dist = dist[j];

min\_idx = j;

}

}

}

if (min\_idx == -1 || min\_idx == target\_id-1) break;

visited[min\_idx] = 1;

// 公交线路

for (int r = 0; r < route\_count; r++) {

int found = 0;

int route\_start\_idx = -1;

for (int j = 0; j < bus\_routes[r].station\_count; j++) {

if (bus\_routes[r].stations[j] == min\_idx + 1) {

found = 1;

route\_start\_idx = j;

break;

}

}

if (found) {

for (int j = route\_start\_idx + 1; j < bus\_routes[r].station\_count; j++) {

float segment\_dist = 0.0;

int transfer = 0;

if (prev[min\_idx] != -1 && strcmp(modes[min\_idx], "公交") == 0 && prev\_route[min\_idx] != bus\_routes[r].id) {

transfer = 1;

}

for (int k = route\_start\_idx; k < j; k++) {

segment\_dist += bus\_routes[r].distances[k];

}

float time = segment\_dist / BUS\_SPEED;

int next\_station = bus\_routes[r].stations[j] - 1;

if (!visited[next\_station]) {

float new\_dist = (mode == 0) ? dist[min\_idx] + time :

(mode == 1) ? dist[min\_idx] + (transfer ? 1 : 0) :

dist[min\_idx] + segment\_dist;

float new\_total\_time = total\_time[min\_idx] + time;

float new\_total\_dist = total\_dist[min\_idx] + segment\_dist;

int new\_transfers = transfers[min\_idx] + transfer;

if (mode == 1) {

if (new\_transfers < transfers[next\_station] ||

(new\_transfers == transfers[next\_station] && new\_total\_time < total\_time[next\_station])) {

dist[next\_station] = new\_dist;

total\_time[next\_station] = new\_total\_time;

total\_dist[next\_station] = new\_total\_dist;

prev[next\_station] = min\_idx;

transfers[next\_station] = new\_transfers;

strcpy(modes[next\_station], "公交");

prev\_route[next\_station] = bus\_routes[r].id;

}

} else if (new\_dist < dist[next\_station]) {

dist[next\_station] = new\_dist;

total\_time[next\_station] = new\_total\_time;

total\_dist[next\_station] = new\_total\_dist;

prev[next\_station] = min\_idx;

transfers[next\_station] = new\_transfers;

strcpy(modes[next\_station], "公交");

prev\_route[next\_station] = bus\_routes[r].id;

}

}

}

}

}

// 步行边

for (int j = 0; j < walk\_edge\_count; j++) {

if (walk\_edges[j].from == min\_idx + 1 && !visited[walk\_edges[j].to-1]) {

float time = walk\_edges[j].distance / WALK\_SPEED;

float new\_dist = (mode == 0) ? dist[min\_idx] + time :

(mode == 1) ? dist[min\_idx] :

dist[min\_idx] + walk\_edges[j].distance;

float new\_total\_time = total\_time[min\_idx] + time;

float new\_total\_dist = total\_dist[min\_idx] + walk\_edges[j].distance;

if (mode == 1) {

if (transfers[min\_idx] < transfers[walk\_edges[j].to-1] ||

(transfers[min\_idx] == transfers[walk\_edges[j].to-1] && new\_total\_time < total\_time[walk\_edges[j].to-1])) {

dist[walk\_edges[j].to-1] = new\_dist;

total\_time[walk\_edges[j].to-1] = new\_total\_time;

total\_dist[walk\_edges[j].to-1] = new\_total\_dist;

prev[walk\_edges[j].to-1] = min\_idx;

transfers[walk\_edges[j].to-1] = transfers[min\_idx];

strcpy(modes[walk\_edges[j].to-1], "步行");

prev\_route[walk\_edges[j].to-1] = -1;

}

} else if (new\_dist < dist[walk\_edges[j].to-1]) {

dist[walk\_edges[j].to-1] = new\_dist;

total\_time[walk\_edges[j].to-1] = new\_total\_time;

total\_dist[walk\_edges[j].to-1] = new\_total\_dist;

prev[walk\_edges[j].to-1] = min\_idx;

transfers[walk\_edges[j].to-1] = transfers[min\_idx];

strcpy(modes[walk\_edges[j].to-1], "步行");

prev\_route[walk\_edges[j].to-1] = -1;

}

}

}

}

// 处理非公交站点终点

if (target\_id != end\_id) {

for (int j = 0; j < walk\_edge\_count; j++) {

if ((walk\_edges[j].from == target\_id && walk\_edges[j].to == end\_id) ||

(walk\_edges[j].to == target\_id && walk\_edges[j].from == end\_id)) {

float time = walk\_edges[j].distance / WALK\_SPEED;

dist[end\_id-1] = (mode == 0) ? dist[target\_id-1] + time :

(mode == 1) ? dist[target\_id-1] :

dist[target\_id-1] + walk\_edges[j].distance;

total\_time[end\_id-1] = total\_time[target\_id-1] + time;

total\_dist[end\_id-1] = total\_dist[target\_id-1] + walk\_edges[j].distance;

prev[end\_id-1] = target\_id-1;

transfers[end\_id-1] = transfers[target\_id-1];

strcpy(modes[end\_id-1], "步行");

prev\_route[end\_id-1] = -1;

break;

}

}

}

if (dist[end\_id-1] != 99999.0) {

result.time = total\_time[end\_id-1];

result.distance = total\_dist[end\_id-1];

result.path\_length = 0;

int current = end\_id - 1;

while (current != -1) {

result.path[result.path\_length] = current + 1;

strcpy(result.modes[result.path\_length], modes[current]);

result.path\_length++;

current = prev[current];

}

result.transfers = transfers[end\_id-1];

}

return result;

}

// 路线规划

void plan\_route(int mode) {

char start\_name[MAX\_NAME], end\_name[MAX\_NAME];

int start\_id, end\_id;

if (mode != 2) {

printf("请输入起点名称：");

scanf("%s", start\_name);

printf("请输入终点名称：");

scanf("%s", end\_name);

start\_id = find\_station\_id(start\_name);

end\_id = find\_station\_id(end\_name);

if (start\_id == -1 || end\_id == -1) {

printf("起点或终点不存在。\n");

return;

}

} else {

start\_id = 1; // 五棵松

printf("请输入终点名称：");

scanf("%s", end\_name);

end\_id = find\_station\_id(end\_name);

if (end\_id == -1) {

printf("终点不存在。\n");

return;

}

}

Path result = dijkstra(start\_id, end\_id, mode);

if (result.path\_length == 0) {

printf("无法找到合适的路线。\n");

return;

}

printf("最优路线（%s）：\n", mode == 0 ? "最短时间" : mode == 1 ? "最少换乘" : "最短距离（从五棵松）");

float total\_distance = 0.0;

float total\_time = 0.0;

for (int i = result.path\_length - 1; i >= 0; i--) {

printf("%s", stations[result.path[i]-1].name);

if (i > 0) {

printf(" %s ", result.modes[i-1]);

if (strcmp(result.modes[i-1], "步行") == 0) {

for (int j = 0; j < walk\_edge\_count; j++) {

if (walk\_edges[j].from == result.path[i] && walk\_edges[j].to == result.path[i-1]) {

printf("(%.2fkm)", walk\_edges[j].distance);

total\_distance += walk\_edges[j].distance;

total\_time += walk\_edges[j].distance / WALK\_SPEED;

break;

}

}

} else {

int route\_id = -1;

float segment\_distance = 0.0;

for (int r = 0; r < route\_count; r++) {

int found\_from = -1, found\_to = -1;

for (int j = 0; j < bus\_routes[r].station\_count; j++) {

if (bus\_routes[r].stations[j] == result.path[i]) found\_from = j;

if (bus\_routes[r].stations[j] == result.path[i-1]) found\_to = j;

}

if (found\_from != -1 && found\_to != -1 && found\_from < found\_to) {

route\_id = bus\_routes[r].id;

for (int j = found\_from; j < found\_to; j++) {

segment\_distance += bus\_routes[r].distances[j];

}

break;

}

}

printf("(%.2fkm, 线路%d)", segment\_distance, route\_id);

total\_distance += segment\_distance;

total\_time += segment\_distance / BUS\_SPEED;

}

printf(" -> ");

}

}

printf("\n总距离：%.2fkm\n总时间：%.2f分钟\n换乘次数：%d\n",

total\_distance, total\_time, result.transfers);

}

// 管理员菜单

void admin\_menu() {

int sub\_choice;

while (1) {

printf("\n=== 管理员功能 ===\n");

printf("1. 添加新站点\n");

printf("2. 新增公交线路\n");

printf("3. 修改公交线路\n");

printf("4. 从线路中删除站点\n");

printf("5. 添加站点到线路\n");

printf("6. 密码管理\n");

printf("7. 查看所有公交线路\n");

printf("8. 返回主菜单\n");

printf("9. 退出系统\n");

printf("请输入选项：");

scanf("%d", &sub\_choice);

switch (sub\_choice) {

case 1: add\_new\_station(); break;

case 2: add\_bus\_route(); break;

case 3: modify\_bus\_route(); break;

case 4: delete\_station\_from\_route(); break;

case 5: add\_station\_to\_route(); break;

case 6: manage\_password(); break;

case 7: display\_all\_routes(); break;

case 8: return;

case 9:

printf("感谢使用本系统。\n");

exit(0);

default: printf("无效选项。\n");

}

}

}

// 学生菜单

void student\_menu() {

int sub\_choice;

while (1) {

printf("\n=== 学生功能 ===\n");

printf("1. 查询站点经过的公交线路\n");

printf("2. 显示所有公交线路\n");

printf("3. 最短时间路线规划\n");

printf("4. 最少换乘路线规划\n");

printf("5. 从五棵松出发的最短距离路线\n");

printf("6. 返回主菜单\n");

printf("7. 退出系统\n");

printf("请输入选项：");

scanf("%d", &sub\_choice);

switch (sub\_choice) {

case 1: query\_station\_routes(); break;

case 2: display\_all\_routes(); break;

case 3: plan\_route(0); break;

case 4: plan\_route(1); break;

case 5: plan\_route(2); break;

case 6: return;

case 7:

printf("感谢使用本系统。\n");

exit(0);

default: printf("无效选项。\n");

}

}

}

// 主菜单

void menu() {

int choice;

while (1) {

printf("\n=== 校园公交查询系统 ===\n");

printf("1. 管理员功能\n");

printf("2. 学生功能\n");

printf("3. 退出系统\n");

printf("请输入选项：");

scanf("%d", &choice);

switch (choice) {

case 1:

if (verify\_password()) {

admin\_menu();

} else {

printf("密码错误。\n");

}

break;

case 2:

student\_menu();

break;

case 3:

printf("感谢使用本系统。\n");

return;

default:

printf("无效选项。\n");

}

}

}

int main() {

load\_data();

if (station\_count == 0) {

init\_default\_data();

}

menu();

return 0;

}