

 数据结构课程设计

报告书

**题目：五山校区校园导游系统**

**学  院  数学学院**

**专  业  统计学**

**学生姓名 周冠宏**

**学生学号 202330401911**

**指导教师 刘勇平**

**课程编号**040101521

**课程学分  2**

**起始日期 2024年12月26日**

|  |  |
| --- | --- |
| 教  师  评  语 | 教师签名：  日期： |
| 成  绩  评  定 |  |
| 备  注 |  |

**五山校区校园导游系统**

一**、**任务与需求

### 1.1 **主要问题**

本课题旨在设计并实现一个五山校区校园导游系统，为用户提供校园景点的详细介绍、浏览路线规划以及地图信息更新功能。系统需要解决以下问题：

1.景点介绍：用户可以通过系统查看校园内各个景点的详细信息。

2.浏览路线：

A.查看游览路线：输入起点，获取所有可能的浏览路线。

B.查询景点间最短路径：输入起点和终点，获取最短路径。

C.查询景点间所有路径：输入起点和终点，获取所有可能的路径。

3.更新信息：管理员可以通过密码验证后，对校园地图进行动态更新，包括删除景点、删除道路、增加景点、增加道路等功能。

### **1.2 技术要求**

1.系统应具备良好的用户交互界面，方便用户操作。

2.系统应能够高效地计算最短路径和所有路径。

3.系统应支持动态更新地图信息，确保数据的实时性和准确性。

4.系统应具备一定的安全性，修改地图功能需要通过密码验证才能访问。

**1.3指导思想：**

本系统采用模块化设计思想，将功能划分为景点介绍、路线规划和地图更新三大模块。通过邻接矩阵存储校园地图信息，利用 Floyd 算法和 Dijkstra 算法实现路径规划功能。

二**、**总体设计

### ****2.1 设计原理****

本系统基于图论中的邻接矩阵存储校园地图信息，利用 Floyd 算法和 Dijkstra 算法实现路径规划功能。系统分为以下模块：

**1.景点介绍模块**：提供校园内各个景点的详细信息。

**2.路线规划模块**：

A.查询最短路径：使用 Floyd 算法实现。

B.查询多个景点最短路径：使用Dijkstra 算法实现。

**3.地图更新模块**：管理员可以通过密码验证后，动态更新校园地图信息。

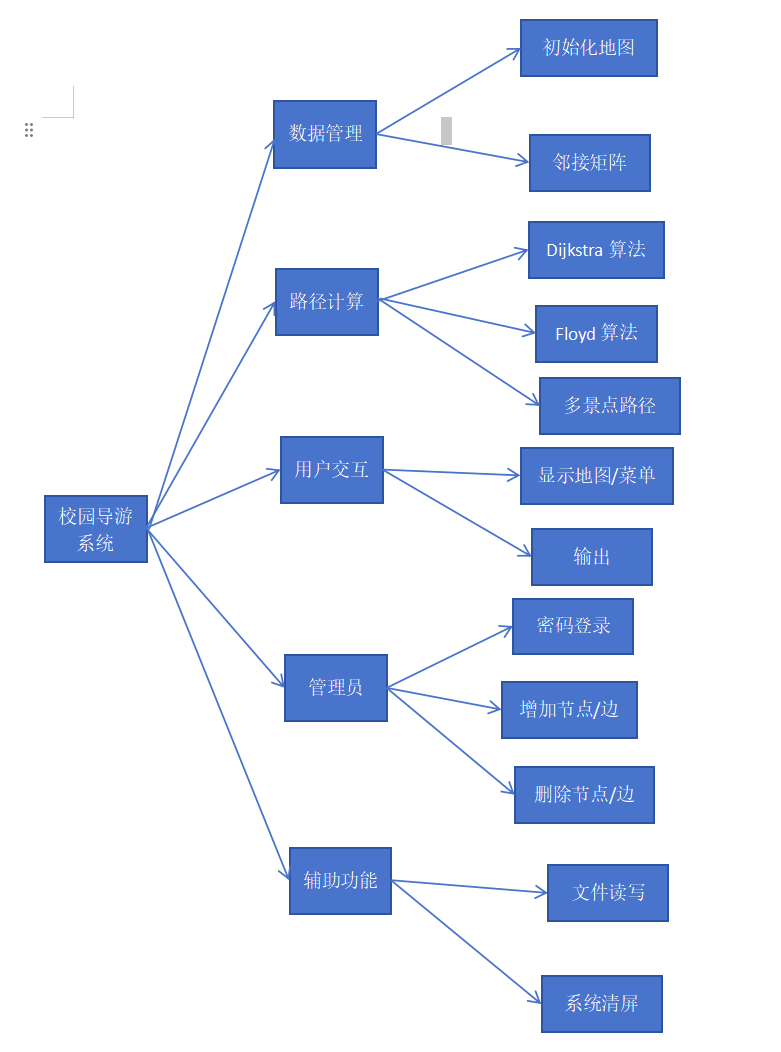


图1 功能模块图

### ****2.2 方案选择****

**1.数据结构**：采用邻接矩阵存储校园地图信息，便于快速访问和更新。

**2.算法选择**：

Dijkstra 算法：适用于单源最短路径问题，时间复杂度为 O(n²)。

Floyd 算法：适用于多源最短路径问题，时间复杂度为 O(n³)。

**3.安全性**：管理员功能通过密码验证，确保系统数据的安全性。

**三**、**详细设计与编码**

任务：

 浏览路线：（1）查看地图。（2）查询景点间最短路径：输入起点及终点，得到最短路径。（3）查询多个景点间最短路径：输入起点及要参观的景点，得到最短路径。

更新信息：需要输入密码来解锁以下功能(1)删除结点：将指定景点删除。 (2)删除边：将景点之间的道路删除。 (3)增加结点：添加一个新的景点。 (4)增加边：为景点间添加一条新的路线。 (5)返回主页。

**3.1 数据结构定义**

#define MAXV 20

#define INF 32767

struct VertexType {

string name;

string introduction;

}; // 顶点类型

struct MatGraph {

int edges[MAXV][MAXV];

int n, e;

VertexType vexs[MAXV];

}; // 邻接矩阵

**3.2 功能模块化实现**

**1.展示地图**

**void DisplayMap() {**

**system("start C:\\Users\\admin\\Desktop\\C++\\camp\_map.png");**

**}**

1. **查询顶点间最短路径**

// Floyd算法

void Floyd(MatGraph g) {

int num1, num2;

cout << "请输入需要查找的两个景点编号：" << endl;

cin >> num1 >> num2;

if (num1 < 0 || num1 >= g.n || num2 < 0 || num2 >= g.n) {

cout << "输入的景点编号不合法！" << endl;

return;

}

int A[MAXV][MAXV], path[MAXV][MAXV];

for (int i = 0; i < g.n; i++) {

for (int j = 0; j < g.n; j++) {

A[i][j] = g.edges[i][j];

if (i != j && g.edges[i][j] != INF) {

path[i][j] = i;

} else {

path[i][j] = -1;

}

}

}

for (int k = 0; k < g.n; k++) {

for (int i = 0; i < g.n; i++) {

for (int j = 0; j < g.n; j++) {

if (A[i][j] > A[i][k] + A[k][j]) {

A[i][j] = A[i][k] + A[k][j];

path[i][j] = path[k][j];

}

}

}

}

Display(g, A, path, num1, num2);

}

1. **查询多个景点的最短路径**

// Dijkstra算法

void dijkstra(MatGraph g, int v, int dist[], int path[]) {

int S[MAXV];

int mindist, i, j, u;

for (i = 0; i < g.n; i++) {

dist[i] = g.edges[v][i]; // 初始化距离

S[i] = 0; // 将S置空，S[i]=1表示已标记

if (g.edges[v][i] < INF)

path[i] = v;

else

path[i] = -1; // 初始点开始没边

}

S[v] = 1; path[v] = 0;

for (i = 0; i < g.n - 1; i++) {

mindist = INF;

for (j = 0; j < g.n; j++)

if (S[j] == 0 && dist[j] < mindist) {

u = j;

mindist = dist[j];

}

S[u] = 1; // 顶点u加入S中

for (j = 0; j < g.n; j++) // 更新未标记的顶点的最短路径

if (S[j] == 0)

if (g.edges[u][j] < INF && dist[u] + g.edges[u][j] < dist[j]) {

dist[j] = dist[u] + g.edges[u][j]; // 更新

path[j] = u; // 更新前一点

}

}

}

// 增加的功能，输出选定的几个顶点之间的最近邻近点策略

void allpath(MatGraph g) {

int dist[MAXV], path[MAXV];

int m;

do {

cout << "请输入你要去的景点个数：" << endl;

cin >> m;

if (m > g.n) cout << "输入值大于景点个数，请重新输入" << endl;

} while (m > g.n);

int num[m];

cout << "请输入您的起始点：" << endl;

do {

cin >> num[0];

if (num[0] < 0 || num[0] >= g.n) cout << "输入值有错误，请重新输入" << endl;

} while (num[0] < 0 || num[0] >= g.n);

cout << "请输入你要去的景点编号：" << endl;

for (int i = 1; i < m; i++) {

do {

cin >> num[i];

if (num[i] < 0 || num[i] >= g.n) cout << "输入值有错误，请重新输入" << endl;

} while (num[i] < 0 || num[i] >= g.n);

}

int visit[g.n], mark[m]; // mark[]存放每一步去的景点

memset(visit, 0, sizeof(visit)); // 初始化visit数组

visit[num[0]] = 1;

mark[0] = num[0]; // 第一步为起始点

for (int j = 1; j < m; j++) {

dijkstra(g, mark[j - 1], dist, path);

int min\_dist = INF;

int next\_vertex = -1;

for (int i = 0; i < m; i++) { // 遍历所有待选景点

int vertex = num[i];

if (!visit[vertex] && dist[vertex] < min\_dist) {

min\_dist = dist[vertex];

next\_vertex = vertex;

}

}

if (next\_vertex == -1) break; // 无可用路径

mark[j] = next\_vertex;

visit[next\_vertex] = 1;

}

cout << "最佳路线为：" << endl;

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

cout << mark[i] << " -->";

}

cout << mark[m-1];

cout << endl;

}

1. **学校景点介绍**

void introduce(MatGraph g) {

int num;

cout << "请输入查询的景点编号：" << endl;

cin >> num;

if (num < 0 || num >= g.n) {

cout << "输入的景点编号不合法！" << endl;

return;

}

cout << g.vexs[num].name << endl;

cout << g.vexs[num].introduction << endl;

}

1. **更改图信息**

void ChangeMap(MatGraph &g) {

string key;

int num;

cout << "请输入密钥：" << endl;

cin >> key;

if (key != "2848192063") {

cout << "您无权更改地图" << endl;

return;

}

while (true) {

cout << "请选择您的操作：" << endl;

cout << "0.返回菜单" << endl;

cout << "1.删除景点" << endl;

cout << "2.删除道路" << endl;

cout << "3.增加景点" << endl;

cout << "4.增加道路" << endl;

cin >> num;

switch (num) {

case 1: DeleteNode(g); break;

case 2: DeleteArc(g); break;

case 3: AddNode(g); break;

case 4: AddArc(g); break;

case 0: return;

default: cout << "未找到该功能，请输入有效选项！" << endl; break;

}

}

}

1. **main函数**

int main() {

int key;

InitMap(g);

while (true) {

menu();

cout << "请输入您要选择的功能：" << endl;

cin >> key;

switch (key) {

case 1:system("cls");DisplayMap();break;

case 2:system("cls");Floyd(g);break;

case 3:system("cls");allpath(g);break;

case 4:system("cls");introduce(g);break;

case 5:system("cls");ChangeMap(g);break;

case 0:system("cls");cout<<"谢谢使用"<<endl; return 0;

default: cout << "未找到该功能，请输入有效选项！" << endl; break;

}

}

}

**四**、调试与**结果分析**

1. 查看地图

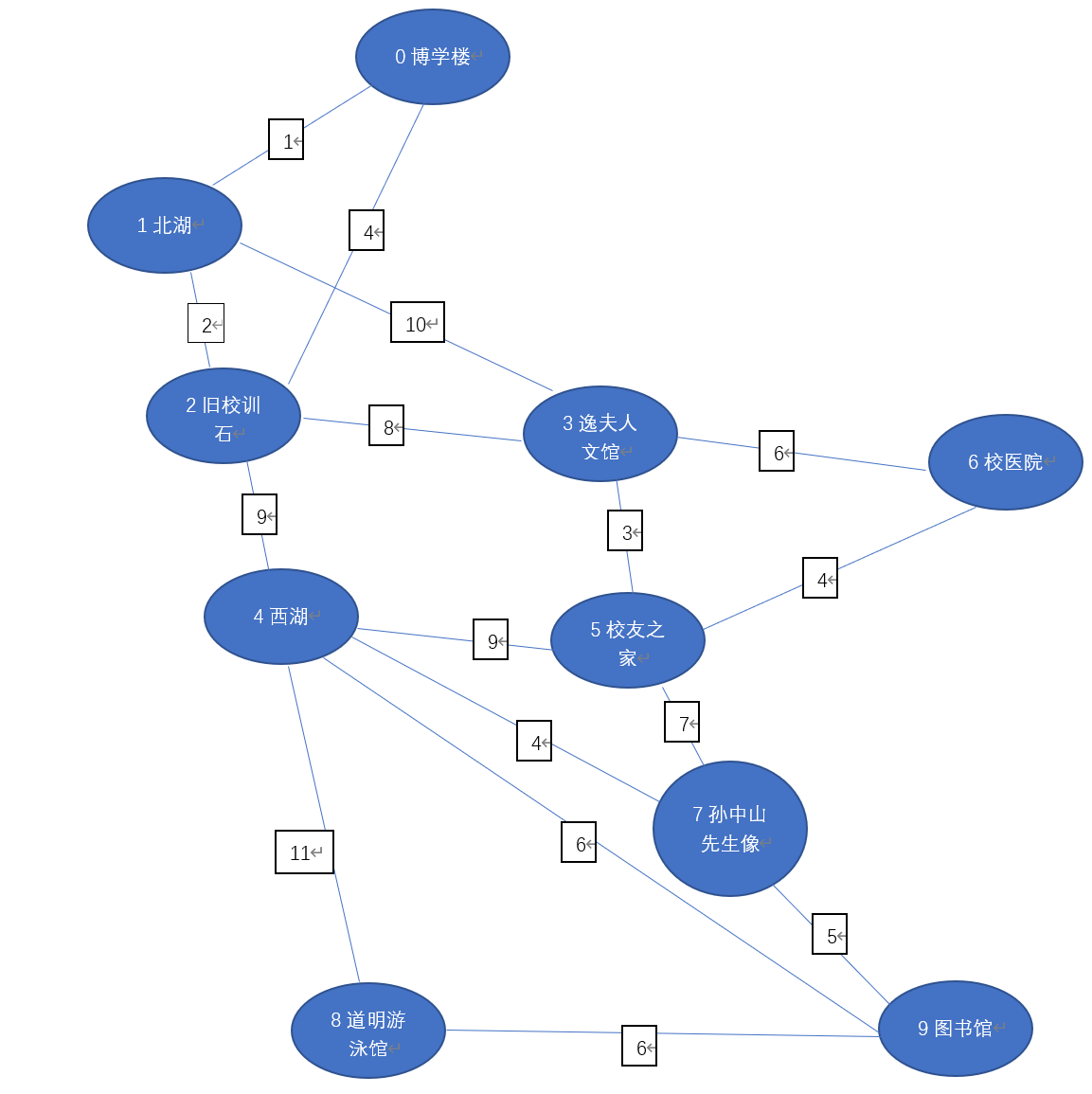


图2 五山校区平面图

1. Main函数

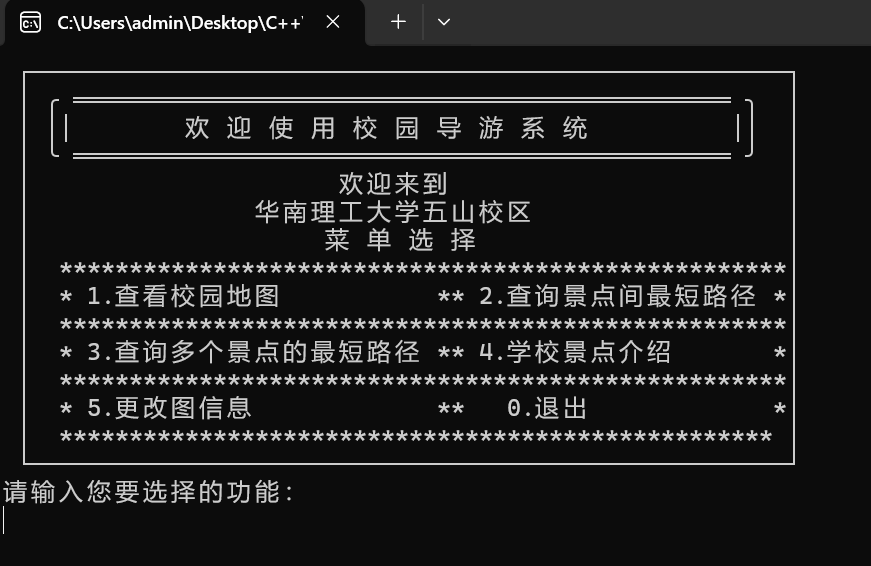


图3 初始菜单

1. 查询景点间最短路径

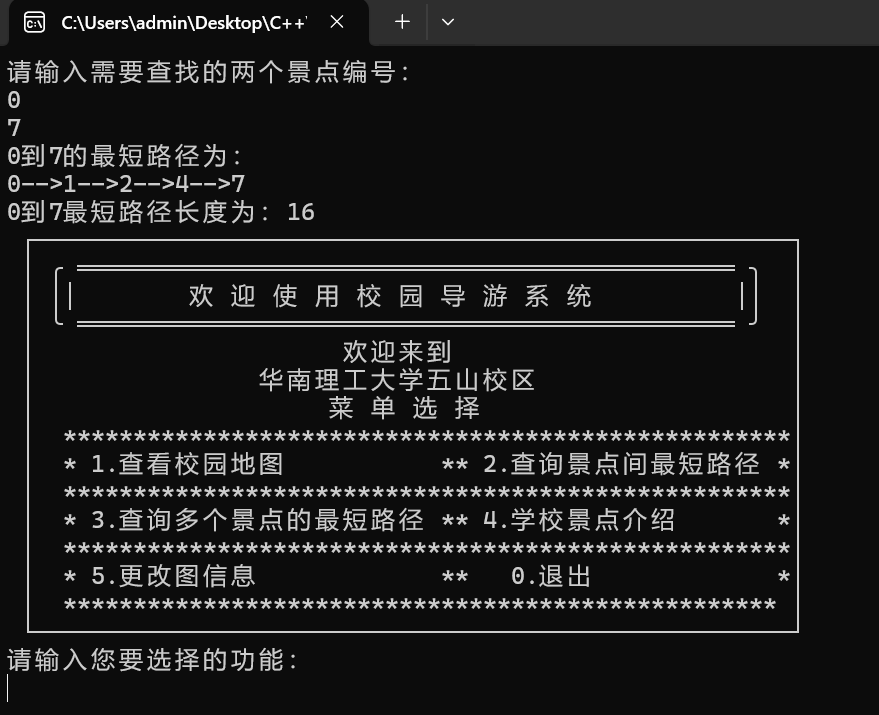


图4 查询景点间最短路径

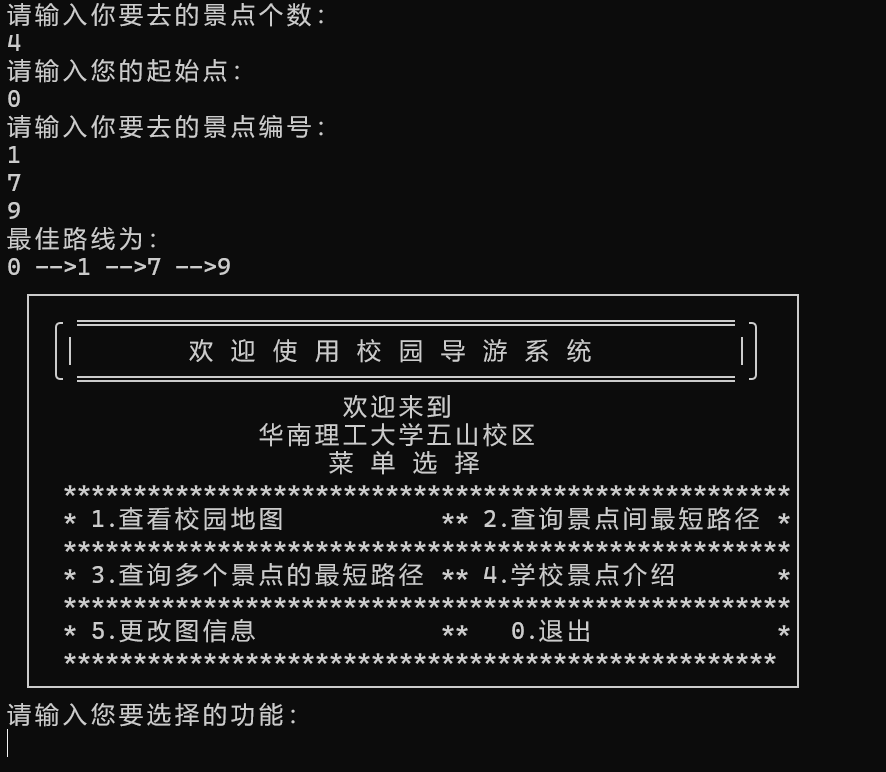
1. 查询多个景点间的最短路径 

图5 查询多个景点间的最短路径

1. 学校景点介绍

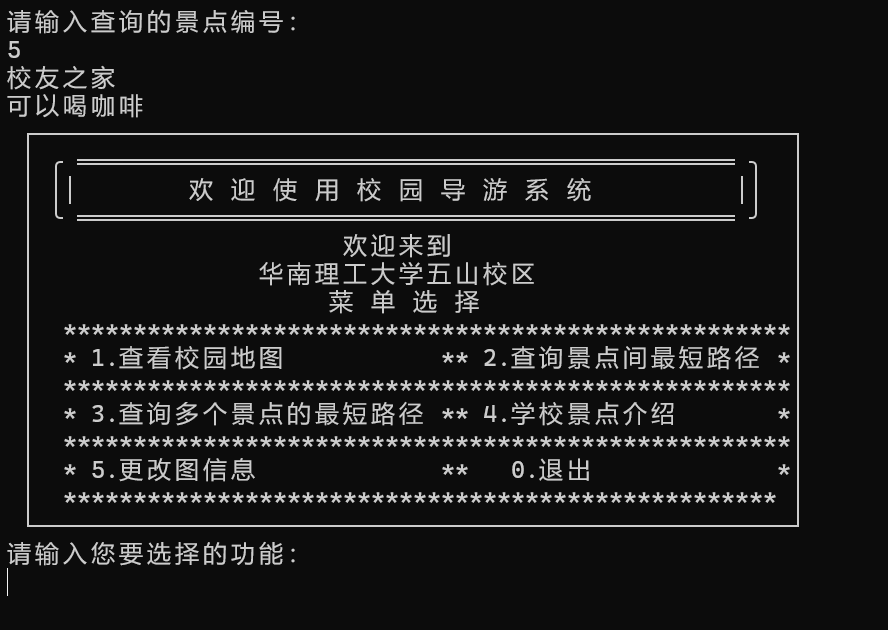


图6 介绍页面

1. 更改图信息

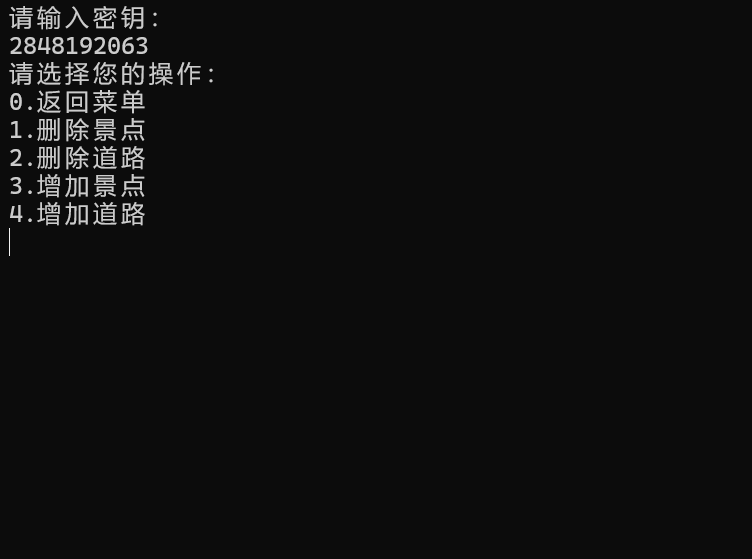


图7 更改信息

**五**、**课程设计总结**

### ****5.1 收获与体会****

通过本次课程设计，我深入理解了图论算法（如 Dijkstra 算法和 Floyd 算法）的实际应用，掌握了邻接矩阵存储图结构的方法。同时，我提高了代码编写和调试能力，明白真正编写一个程序的各个步骤，增强了解决实际问题的能力。

### ****5.2 遇到的问题与解决****

**问题**：地图更新功能实现复杂。  
**解决**：采用模块化设计，将功能分解为多个子模块，降低了实现难度。

### ****5.3 改进方向****

1.使用QT增加UI图形化界面，提升用户体验。

2.支持更多路径规划算法，如 A\* 算法。

3.使用数据库等知识将地图数据能直接导入，不用自己输入。

参考文献

[1]李春葆. 数据结构教程（第5版）.  北京：清华大学出版社. 2017