计算机与信息工程学院实验报告

••••••••••••••••••••••••••••••••• 密 ••••••••••••••••••••••••••••••••• 封 ••••••••••••••••••••••••••••••••• 线 •••••••••••••••••••••••••••••••••

姓名：杨佳森 学号：2112080106\_专业：\_数据科学与大数据技术\_ 年级：\_2021

课程： 数据结构 主讲教师：\_刘成\_\_ 辅导教师：\_\_\_无\_\_\_\_

实验时间：\_2023\_\_年 \_5\_月 \_30\_\_日 \_下\_午17\_\_时至\_19\_时，

实验地点\_计算机大楼213\_\_\_\_\_\_\_

实验题目： 图综合应用实验——校院导航

实验目的： 通过该实验，使学生数量掌握图的存储、基本操作，以及求两点之间所有可能路径和和最短路径。

实验环境（硬件和软件） PC和vs code

实验内容：

面向学校，构建一个校院导游软件。用无向图表示所在学校的校院景点平面图，图中顶点表示主要景点，存放景点的编号、名称、简介等信息，图中边表示景点之间的道路，存放路径距离等信息。该软件具有以下基本功能，学生可自行扩展功能。

实验步骤：

1、查询景点信息

// 查询景点信息

void searchSpot(const Graph& graph) {

    int spotId;

    cout << "请输入要查询景点的编号：";

    cin >> spotId;

    cout << "景点编号：" << graph.spots[spotId].id << endl

        << "景点名称：" << graph.spots[spotId].name << endl

        << "景点简介：" << graph.spots[spotId].intro << endl;

}

2、查询任意两个景点之间所有可能路径

// 查询任意两个景点之间的所有可能路径

void findAllPaths(const Graph& graph, int start, int end, vector<int>& path, vector<vector<int>>& paths) {

    path.push\_back(start);

    if (start == end) {

        paths.push\_back(path);

    } else {

        const auto& edges = graph.edges[start];

        for (const auto& edge : edges) {

            int nextVex = edge.vex2;

            if (find(path.begin(), path.end(), nextVex) == path.end()) {

                vector<int> newPath = path;

                findAllPaths(graph, nextVex, end, newPath, paths);

            }

        }

    }

    path.pop\_back();

}

void searchPath(const Graph& graph) {

    int start, end;

    cout << "请输入起点和终点编号：";

    cin >> start >> end;

    vector<int> path;

    vector<vector<int>> paths;

    findAllPaths(graph, start, end, path, paths);

    // 输出找到的所有路径

    for (const auto& p : paths) {

        for (int i = 0; i < p.size(); i++) {

            cout << graph.spots[p[i]].name;

            if (i < p.size() - 1) {

                cout << " -> ";

            }

        }

        cout << endl;

    }

}

3、查询两个景点之间最短路径

// 查询任意两个景点之间的最短路径

void searchShortestPath(const Graph& graph) {

    int start, end;

    cout << "请输入起点和终点编号：";

    cin >> start >> end;

    if (start < 0 || start > graph.vexnum || end < 0 || end > graph.vexnum) {

        cout << "无效的起点或终点编号！" << endl;

        return;

    }

    priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq; // 最小优先队列

    vector<int> dist(graph.vexnum + 1, INT\_MAX); // 距离

    vector<bool> visited(graph.vexnum + 1, false); // 是否已访问

    dist[start] = 0; // 起始节点距离为0

    pq.push(make\_pair(dist[start], start)); // 起始节点加入队列

    while (!pq.empty()) {

        // 找到当前距离最近的节点

        int u = pq.top().second;

        pq.pop();

        if (visited[u]) continue; // 如果u已经被访问过了，就跳过本次循环

        visited[u] = true;

        // 更新当前节点可以到达的节点距离

        for (const auto& edge : graph.edges[u]) {

            int v = edge.vex2;

            if (visited[v]) continue; // 如果v已经访问过了，就跳过本次循环

            if (dist[v] > dist[u] + edge.weight) {

                dist[v] = dist[u] + edge.weight;

                pq.push(make\_pair(dist[v], v));

            }

        }

    }

    // 输出最短路径

    if (dist[end] < INT\_MAX) { // 说明存在最短路径

        cout << "最短路径长度为：" << dist[end] << endl;

        cout << "最短路径为：";

        vector<int> path;

        int cur = end;

        while (cur != start) {

            path.push\_back(cur);

            bool foundPrev = false;

            for (const auto& edge : graph.edges[cur]) {

                int prev = edge.vex1;

                int weight = edge.weight;

                if (dist[prev] + weight == dist[cur]) {

                    cur = prev;

                    foundPrev = true;

                    break;

                }

            }

            if (!foundPrev) {

                // 未找到合适的前驱节点，可能存在问题

                break;

            }

        }

        path.push\_back(start);

        for (auto it = path.rbegin(); it != path.rend(); ++it) {

            cout << graph.spots[\*it].name;

            if (it + 1 != path.rend()) {

                cout << " -> ";

            }

        }

        cout << endl;

    }

    else { // 不存在最短路径

        cout << "无法到达终点！" << endl;

    }

}

4、增加、删除、更新景点的路径信息

// 增加、删除、更新景点和道路信息

void modifyGraph(Graph& graph) {

    int choice;

    show1();

    while(true) {

        cout << "请输入您的选择: ";

        cin >> choice;

        switch (choice) {

        case 1: // 增加景点

            if (graph.vexnum >= MAXVEX) {

                cout << "无法增加景点，已达到最大限制！" << endl;

            }

            else {

                cout << "请输入新景点编号、名称、简介：";

                graph.vexnum++;

                cin >> graph.spots[graph.vexnum].id >> graph.spots[graph.vexnum].name >> graph.spots[graph.vexnum].intro;

                cout << "增加景点成功！" << endl;

            }

            break;

        case 2: // 删除景点

            int spotId;

            cout << "请输入要删除的景点编号：";

            cin >> spotId;

            if (spotId < 1 || spotId > graph.vexnum) {

                cout << "不存在该景点！" << endl;

            }

            else {

                // 删除所有和该点相关的边

                for (int i = 1; i <= graph.vexnum; i++) {

                    for (int j = 0; j < graph.edges[i].size(); j++) {

                        Edge edge = graph.edges[i][j];

                        if (edge.vex1 == spotId || edge.vex2 == spotId) {

                            graph.edges[i].erase(graph.edges[i].begin() + j);

                        }

                    }

                }

                // 删除该节点及其信息

                for (int i = spotId; i < graph.vexnum; i++) {

                    graph.spots[i] = graph.spots[i + 1];

                }

                graph.vexnum--;

                cout << "删除成功！" << endl;

            }

            break;

        case 3: // 更新景点信息

            int spotId2;

            cout << "请输入要更新的景点编号：";

            cin >> spotId2;

            if (spotId2 < 1 || spotId2 > graph.vexnum) {

                cout << "不存在该景点！" << endl;

            }

            else {

                cout << "请输入新的景点名称和简介：";

                cin >> graph.spots[spotId2].name >> graph.spots[spotId2].intro;

                cout << "更新景点信息成功！" << endl;

            }

            break;

        case 4: // 增加道路

            int v1, v2, w;

            cout << "请输入要连接的两个景点编号和长度：";

            cin >> v1 >> v2 >> w;

            if (v1 < 1 || v1 > graph.vexnum || v2 < 1 || v2 > graph.vexnum) {

                cout << "不存在该景点！" << endl;

            }

            else {

                Edge edge1 = { v1, v2, w };

                Edge edge2 = { v2, v1, w };

                graph.edges[v1].push\_back(edge1);

                graph.edges[v2].push\_back(edge2);

                graph.edgenum++;

                cout << "增加道路成功！" << endl;

            }

            break;

        case 5: // 删除道路

            // int v1, v2;

            cout << "请输入要删除的两个景点编号：";

            cin >> v1 >> v2;

            if (v1 < 1 || v1 > graph.vexnum || v2 < 1 || v2 > graph.vexnum) {

                cout << "不存在该景点！" << endl;

            }

            else {

                // 删除该边

                for (int i = 0; i < graph.edges[v1].size(); i++) {

                    Edge edge = graph.edges[v1][i];

                    if (edge.vex2 == v2) {

                        graph.edges[v1].erase(graph.edges[v1].begin() + i);

                        break;

                    }

                }

                for (int i = 0; i < graph.edges[v2].size(); i++) {

                    Edge edge = graph.edges[v2][i];

                    if (edge.vex2 == v1) {

                        graph.edges[v2].erase(graph.edges[v2].begin() + i);

                        break;

                    }

                }

                graph.edgenum--;

                cout << "删除道路成功！" << endl;

            }

            break;

        case 6:

            return;

            break;

        default:

            cout << "输入不合法！" << endl;

            break;

        }

    }

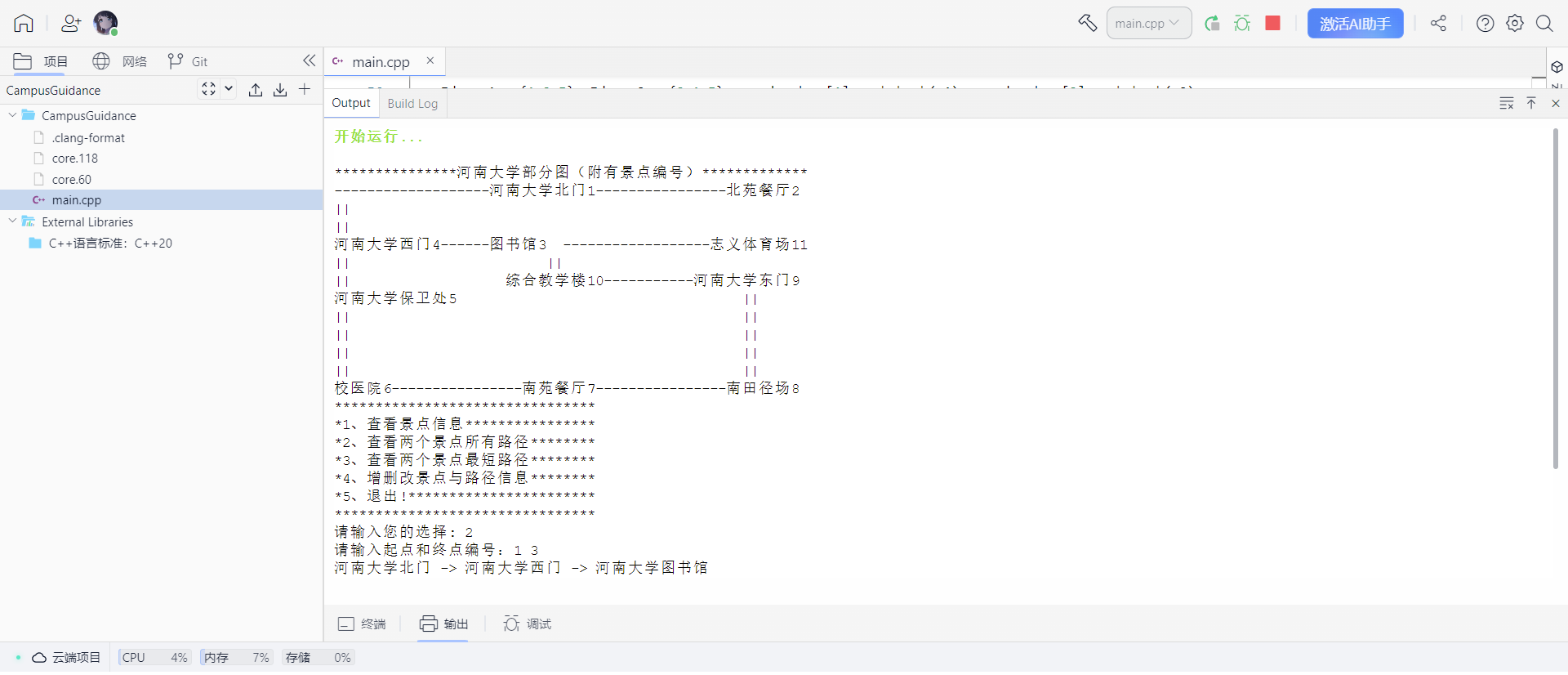
}

实验数据记录：

1）查询各景点的想信息；



2）查询图中任意两个景点之间的所有可能路径；



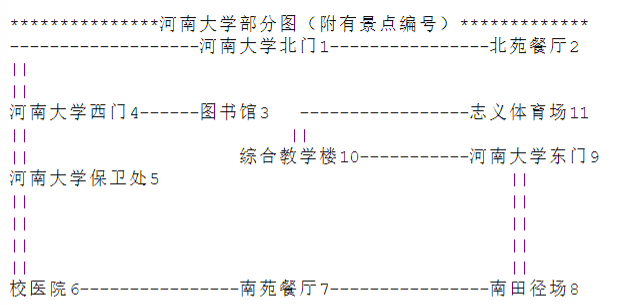
3）查新图中任意两个景点之间的最短路径；



4）增加、删除、更新景点与道路信息



景点参考图：



问题讨论：

查询两个景点之间最短路径：

这个算法使用了 Dijkstra 算法来解决单源最短路径问题。它采用了贪心策略，每次选择当前距离最近的节点进行扩展。通过不断更新节点的最短距离，最终得到起点到所有节点的最短路径。

查询图中任意两个景点之间的最短路径；

这个算法使用了深度优先搜索（DFS）的思想，在搜索过程中，通过递归不断探索所有可能的路径。当找到一条路径时，将其加入到路径集合中，并在递归回溯时，移除当前起点，继续探索其他可能的路径。

源代码：

#include <climits>

#include <iostream>

#include <cstring>

#include <vector>

#include <queue>

#include <map>

using namespace std;

#define MAXVEX 100 // 最大顶点数

// 景点结构体

struct Spot {

    int id=0; //景点编号

    string name; // 景点名称

    string intro;// 景点简介

};

// 边结构体

struct Edge {

    int vex1; // 边的起点

    int vex2; // 边的终点

    int weight; // 边权重，即路径长度

};

// 图结构体

struct Graph {

    int vexnum=0; // 顶点数

    int edgenum=0; // 边数

    Spot spots[MAXVEX]; // 顶点信息

    vector<Edge> edges[MAXVEX]; // 边信息

};

// 创建图

Graph createGraph() {

    Graph graph;

    graph.vexnum = 11;

    graph.edgenum = 11;

    graph.spots[1].id = 1; graph.spots[1].name = "河南大学北门"; graph.spots[1].intro = "河南大学的北门";

    graph.spots[2].id = 2; graph.spots[2].name = "北苑餐厅"; graph.spots[2].intro = "北苑的校内餐厅";

    graph.spots[3].id = 3; graph.spots[3].name = "河南大学图书馆"; graph.spots[3].intro = "河南大学的图书馆";

    graph.spots[4].id = 4; graph.spots[4].name = "河南大学西门"; graph.spots[4].intro = "河南大学的西门";

    graph.spots[5].id = 5; graph.spots[5].name = "河南大学保卫处"; graph.spots[5].intro = "河南大学的保卫处";

    graph.spots[6].id = 6; graph.spots[6].name = "河南大学校医院"; graph.spots[6].intro = "河南大学的校医院";

    graph.spots[7].id = 7; graph.spots[7].name = "南苑餐厅"; graph.spots[7].intro = "南苑的餐厅";

    graph.spots[8].id = 8; graph.spots[8].name = "南田径场"; graph.spots[8].intro = "东操";

    graph.spots[9].id = 9; graph.spots[9].name = "河南大学东门"; graph.spots[9].intro = "河南大学的东门";

    graph.spots[10].id = 10; graph.spots[10].name = "综合教学楼"; graph.spots[10].intro = "综合教学楼，一二三四五六七";

    graph.spots[11].id = 11; graph.spots[11].name = "志义体育场"; graph.spots[11].intro = "志义体育场足球场";

    Edge e1 = {1,2,5}; Edge e2 = {2,1,5}; graph.edges[1].push\_back(e1); graph.edges[2].push\_back(e2);

    // Edge e3 = {1,3,5}; Edge e4 = {3,1,5}; graph.edges[1].push\_back(e3); graph.edges[3].push\_back(e4);

    Edge e5 = {1,4,10}; Edge e6 = {4,1,10}; graph.edges[1].push\_back(e5); graph.edges[4].push\_back(e6);

    Edge e7 = {3,4,5}; Edge e8 = {4,3,5}; graph.edges[3].push\_back(e7); graph.edges[4].push\_back(e8);

    Edge e9 = {4,5,1}; Edge e10 = {5,4,1}; graph.edges[4].push\_back(e9); graph.edges[5].push\_back(e10);

    Edge e11 = {5,6,10}; Edge e12 = {6,5,10}; graph.edges[5].push\_back(e11); graph.edges[6].push\_back(e12);

    Edge e13 = {6,7,5}; Edge e14 = {7,6,5}; graph.edges[6].push\_back(e13); graph.edges[7].push\_back(e14);

    Edge e15 = {7,8,10}; Edge e16 = {8,7,10}; graph.edges[7].push\_back(e15); graph.edges[8].push\_back(e16);

    Edge e17 = {8,9,10}; Edge e18 = {9,8,10}; graph.edges[8].push\_back(e17); graph.edges[9].push\_back(e18);

    Edge e19 = {9,10,10}; Edge e20 = {10,9,10}; graph.edges[9].push\_back(e19); graph.edges[10].push\_back(e20);

    Edge e21 = {10,11,8}; Edge e22 = {11,10,8}; graph.edges[10].push\_back(e21); graph.edges[11].push\_back(e22);

    return graph;

}

// 查询景点信息

void searchSpot(const Graph& graph) {

    int spotId;

    cout << "请输入要查询景点的编号：";

    cin >> spotId;

    cout << "景点编号：" << graph.spots[spotId].id << endl

        << "景点名称：" << graph.spots[spotId].name << endl

        << "景点简介：" << graph.spots[spotId].intro << endl;

}

// 查询任意两个景点之间的所有可能路径

void findAllPaths(const Graph& graph, int start, int end, vector<int>& path, vector<vector<int>>& paths) {

    path.push\_back(start);

    if (start == end) {

        paths.push\_back(path);

    } else {

        const auto& edges = graph.edges[start];

        for (const auto& edge : edges) {

            int nextVex = edge.vex2;

            if (find(path.begin(), path.end(), nextVex) == path.end()) {

                vector<int> newPath = path;

                findAllPaths(graph, nextVex, end, newPath, paths);

            }

        }

    }

    path.pop\_back();

}

void searchPath(const Graph& graph) {

    int start, end;

    cout << "请输入起点和终点编号：";

    cin >> start >> end;

    vector<int> path;

    vector<vector<int>> paths;

    findAllPaths(graph, start, end, path, paths);

    // 输出找到的所有路径

    for (const auto& p : paths) {

        for (int i = 0; i < p.size(); i++) {

            cout << graph.spots[p[i]].name;

            if (i < p.size() - 1) {

                cout << " -> ";

            }

        }

        cout << endl;

    }

}

// 查询任意两个景点之间的最短路径

void searchShortestPath(const Graph& graph) {

    int start, end;

    cout << "请输入起点和终点编号：";

    cin >> start >> end;

    if (start < 0 || start > graph.vexnum || end < 0 || end > graph.vexnum) {

        cout << "无效的起点或终点编号！" << endl;

        return;

    }

    priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq; // 最小优先队列

    vector<int> dist(graph.vexnum + 1, INT\_MAX); // 距离

    vector<bool> visited(graph.vexnum + 1, false); // 是否已访问

    dist[start] = 0; // 起始节点距离为0

    pq.push(make\_pair(dist[start], start)); // 起始节点加入队列

    while (!pq.empty()) {

        // 找到当前距离最近的节点

        int u = pq.top().second;

        pq.pop();

        if (visited[u]) continue; // 如果u已经被访问过了，就跳过本次循环

        visited[u] = true;

        // 更新当前节点可以到达的节点距离

        for (const auto& edge : graph.edges[u]) {

            int v = edge.vex2;

            if (visited[v]) continue; // 如果v已经访问过了，就跳过本次循环

            if (dist[v] > dist[u] + edge.weight) {

                dist[v] = dist[u] + edge.weight;

                pq.push(make\_pair(dist[v], v));

            }

        }

    }

    // 输出最短路径

    if (dist[end] < INT\_MAX) { // 说明存在最短路径

        cout << "最短路径长度为：" << dist[end] << endl;

        cout << "最短路径为：";

        vector<int> path;

        int cur = end;

        while (cur != start) {

            path.push\_back(cur);

            bool foundPrev = false;

            for (const auto& edge : graph.edges[cur]) {

                int prev = edge.vex1;

                int weight = edge.weight;

                if (dist[prev] + weight == dist[cur]) {

                    cur = prev;

                    foundPrev = true;

                    break;

                }

            }

            if (!foundPrev) {

                // 未找到合适的前驱节点，可能存在问题

                break;

            }

        }

        path.push\_back(start);

        for (auto it = path.rbegin(); it != path.rend(); ++it) {

            cout << graph.spots[\*it].name;

            if (it + 1 != path.rend()) {

                cout << " -> ";

            }

        }

        cout << endl;

    }

    else { // 不存在最短路径

        cout << "无法到达终点！" << endl;

    }

}

//

void show1() {

    cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

    cout << "\*1、增加景点\*\*\*\*\*\*" << endl;

    cout << "\*2、删除景点\*\*\*\*\*\*" << endl;

    cout << "\*3、更新景点信息\*\*" << endl;

    cout << "\*4、增加路径\*\*\*\*\*\*" << endl;

    cout << "\*5、删除路径\*\*\*\*\*\*" << endl;

    cout << "\*6、退出!\*\*\*\*\*\*\*\*\*" << endl;

    cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

}

// 增加、删除、更新景点和道路信息

void modifyGraph(Graph& graph) {

    int choice;

    show1();

    while(true) {

        cout << "请输入您的选择: ";

        cin >> choice;

        switch (choice) {

        case 1: // 增加景点

            if (graph.vexnum >= MAXVEX) {

                cout << "无法增加景点，已达到最大限制！" << endl;

            }

            else {

                cout << "请输入新景点编号、名称、简介：";

                graph.vexnum++;

                cin >> graph.spots[graph.vexnum].id >> graph.spots[graph.vexnum].name >> graph.spots[graph.vexnum].intro;

                cout << "增加景点成功！" << endl;

            }

            break;

        case 2: // 删除景点

            int spotId;

            cout << "请输入要删除的景点编号：";

            cin >> spotId;

            if (spotId < 1 || spotId > graph.vexnum) {

                cout << "不存在该景点！" << endl;

            }

            else {

                // 删除所有和该点相关的边

                for (int i = 1; i <= graph.vexnum; i++) {

                    for (int j = 0; j < graph.edges[i].size(); j++) {

                        Edge edge = graph.edges[i][j];

                        if (edge.vex1 == spotId || edge.vex2 == spotId) {

                            graph.edges[i].erase(graph.edges[i].begin() + j);

                        }

                    }

                }

                // 删除该节点及其信息

                for (int i = spotId; i < graph.vexnum; i++) {

                    graph.spots[i] = graph.spots[i + 1];

                }

                graph.vexnum--;

                cout << "删除成功！" << endl;

            }

            break;

        case 3: // 更新景点信息

            int spotId2;

            cout << "请输入要更新的景点编号：";

            cin >> spotId2;

            if (spotId2 < 1 || spotId2 > graph.vexnum) {

                cout << "不存在该景点！" << endl;

            }

            else {

                cout << "请输入新的景点名称和简介：";

                cin >> graph.spots[spotId2].name >> graph.spots[spotId2].intro;

                cout << "更新景点信息成功！" << endl;

            }

            break;

        case 4: // 增加道路

            int v1, v2, w;

            cout << "请输入要连接的两个景点编号和长度：";

            cin >> v1 >> v2 >> w;

            if (v1 < 1 || v1 > graph.vexnum || v2 < 1 || v2 > graph.vexnum) {

                cout << "不存在该景点！" << endl;

            }

            else {

                Edge edge1 = { v1, v2, w };

                Edge edge2 = { v2, v1, w };

                graph.edges[v1].push\_back(edge1);

                graph.edges[v2].push\_back(edge2);

                graph.edgenum++;

                cout << "增加道路成功！" << endl;

            }

            break;

        case 5: // 删除道路

            // int v1, v2;

            cout << "请输入要删除的两个景点编号：";

            cin >> v1 >> v2;

            if (v1 < 1 || v1 > graph.vexnum || v2 < 1 || v2 > graph.vexnum) {

                cout << "不存在该景点！" << endl;

            }

            else {

                // 删除该边

                for (int i = 0; i < graph.edges[v1].size(); i++) {

                    Edge edge = graph.edges[v1][i];

                    if (edge.vex2 == v2) {

                        graph.edges[v1].erase(graph.edges[v1].begin() + i);

                        break;

                    }

                }

                for (int i = 0; i < graph.edges[v2].size(); i++) {

                    Edge edge = graph.edges[v2][i];

                    if (edge.vex2 == v1) {

                        graph.edges[v2].erase(graph.edges[v2].begin() + i);

                        break;

                    }

                }

                graph.edgenum--;

                cout << "删除道路成功！" << endl;

            }

            break;

        case 6:

            return;

            break;

        default:

            cout << "输入不合法！" << endl;

            break;

        }

    }

}

void show2() {

    cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

    cout << "\*1、查看景点信息\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

    cout << "\*2、查看两个景点所有路径\*\*\*\*\*\*\*\*" << endl;

    cout << "\*3、查看两个景点最短路径\*\*\*\*\*\*\*\*" << endl;

    cout << "\*4、增删改景点与路径信息\*\*\*\*\*\*\*\*" << endl;

    cout << "\*5、退出!\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

    cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

}

void show3() {

    cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*河南大学部分图（附有景点编号）\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

    cout << "-------------------河南大学北门1----------------北苑餐厅2" << endl;

    cout << "||                                                      " << endl;

    cout << "||                                                      " << endl;

    cout << "河南大学西门4------图书馆3   -----------------志义体育场11" << endl;

    cout << "||                          ||                            " << endl;

    cout << "||                     综合教学楼10-----------河南大学东门9" << endl;

    cout << "河南大学保卫处5                                   ||     " << endl;

    cout << "||                                                ||     " << endl;

    cout << "||                                                ||     " << endl;

    cout << "||                                                ||     " << endl;

    cout << "||                                                ||     " << endl;

    cout << "校医院6----------------南苑餐厅7----------------南田径场8  " << endl;

}

// 主函数

int main() {

    show3();

    Graph graph = createGraph();

    while (true) {

        int choice;

        show2();

        cout << "请输入您的选择: ";

        cin >> choice;

        switch (choice) {

        case 1:

            searchSpot(graph);

            break;

        case 2:

            searchPath(graph);

            break;

        case 3:

            searchShortestPath(graph);

            break;

        case 4:

            modifyGraph(graph);

            break;

        case 5:

            return 0;

        default:

            cout << "输入不合法！" << endl;

            break;

        }

    }

    return 0;

}