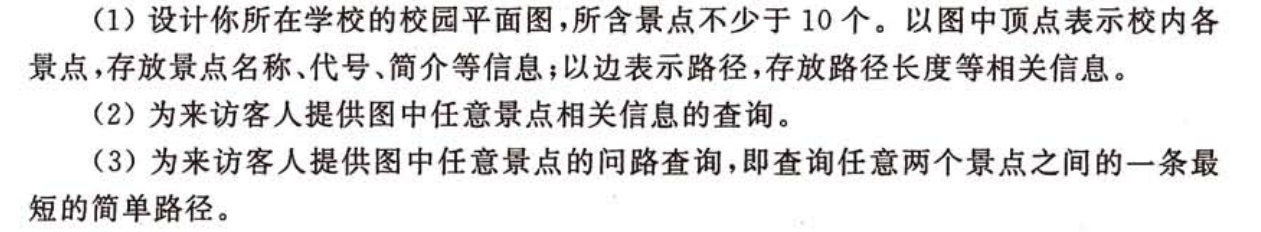
**《数据结构课程设计》大作业**

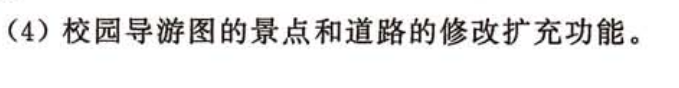
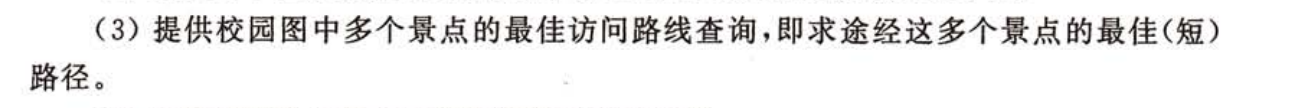
#### 班级：22330112 学号：22330229 姓名：王宇恒 完成时间：2023-12-11

### 需求分析

* 1. 程序实现的功能

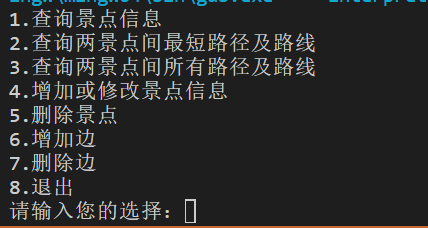


ZLO9OABW_YW{F9ERD$S~(OE

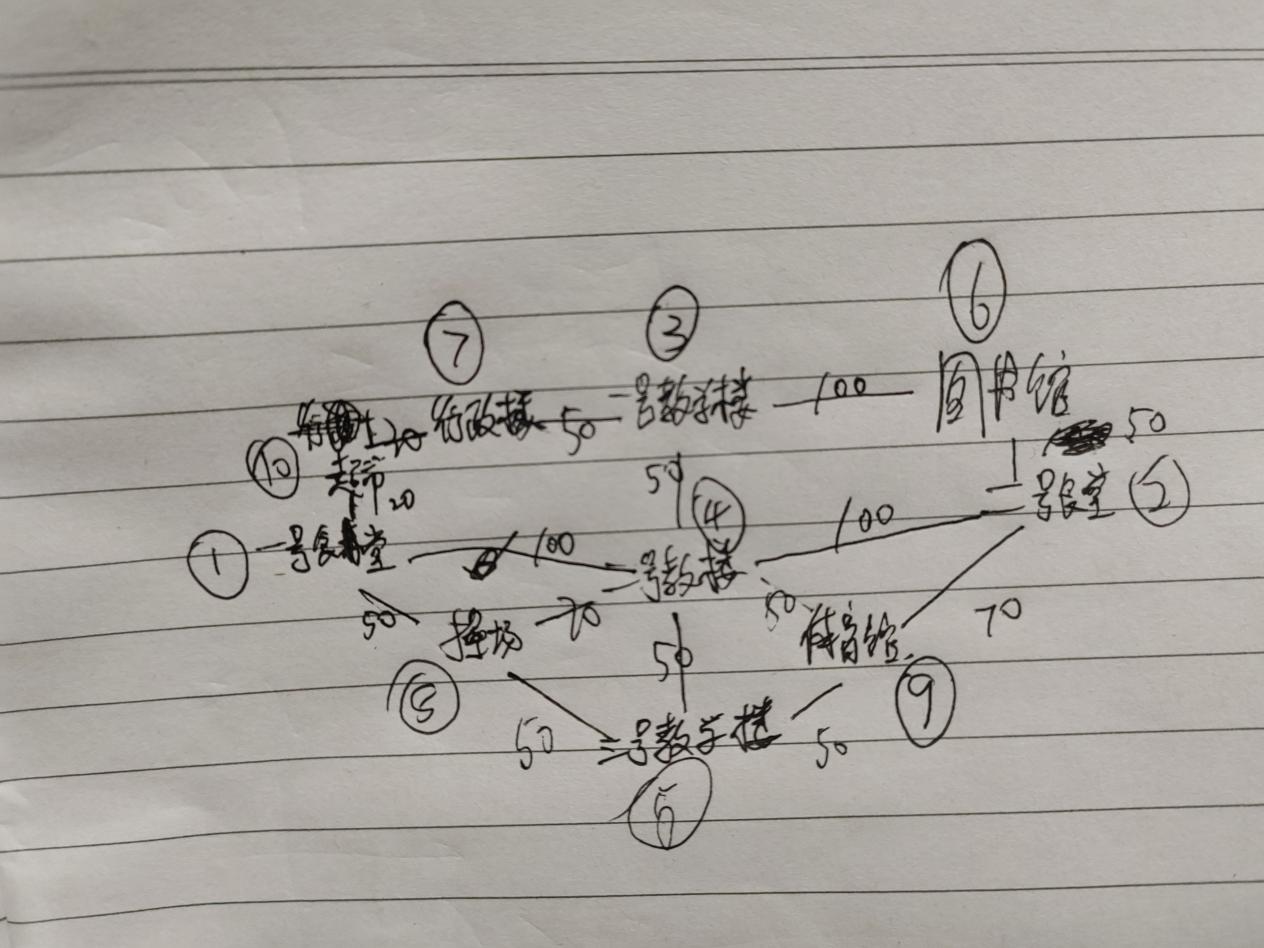


1.2 演示程序以用户和计算机的对话方式执行,即在计算机终端上显示“提示信息”之后,由用户在键盘上输入演示程序中规定的运算命令;相应的输入数据(滤去输入中的非法字符)和运算结果显示在其后。

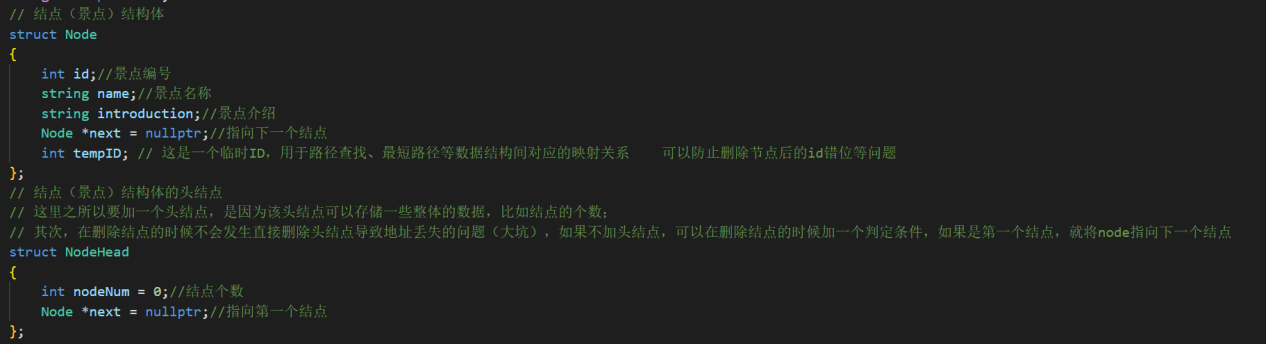
1.3 程序可交互的命令

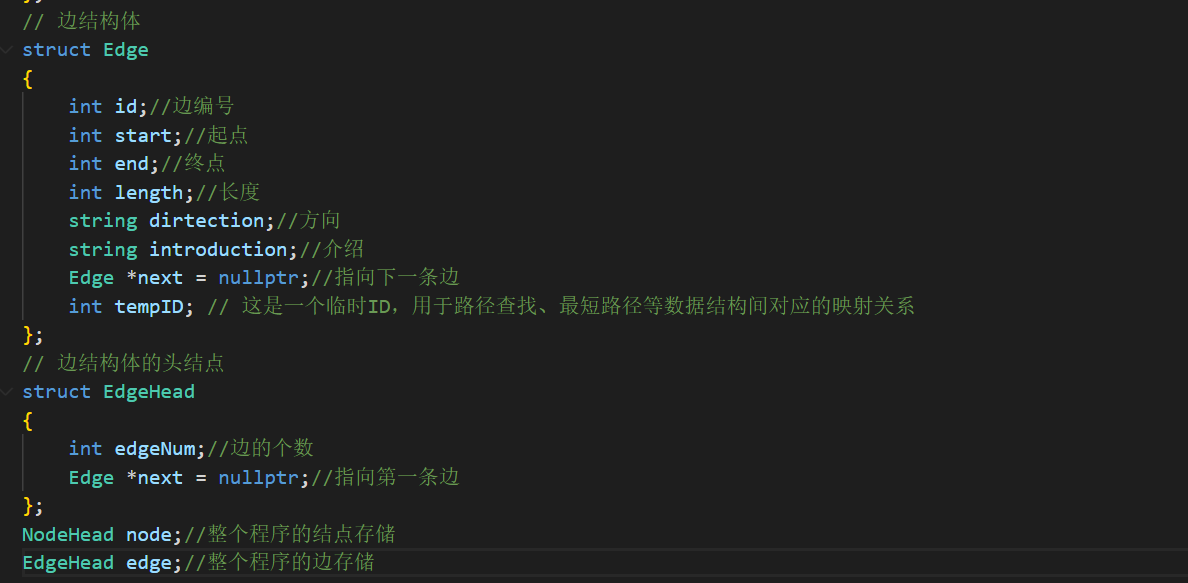


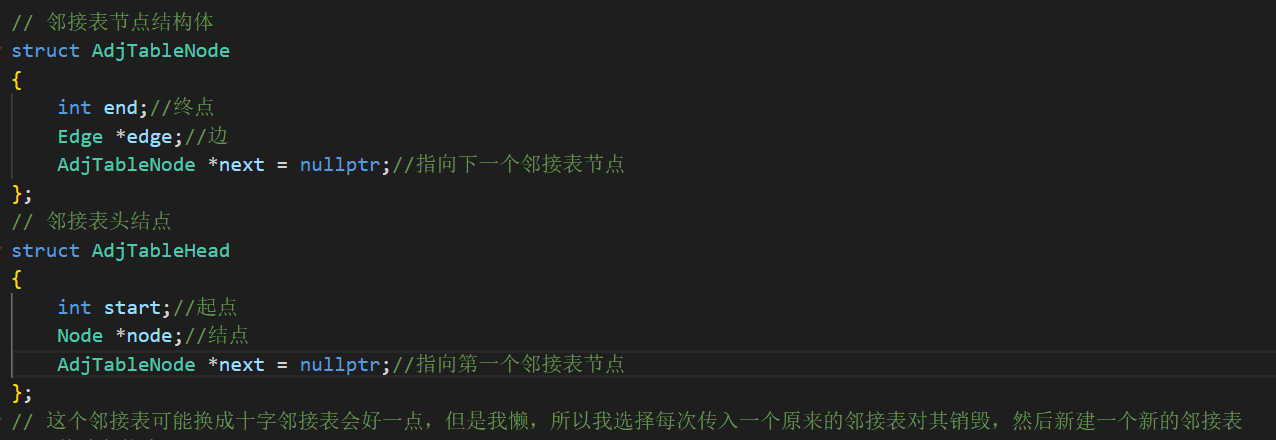
1.4 测试用例

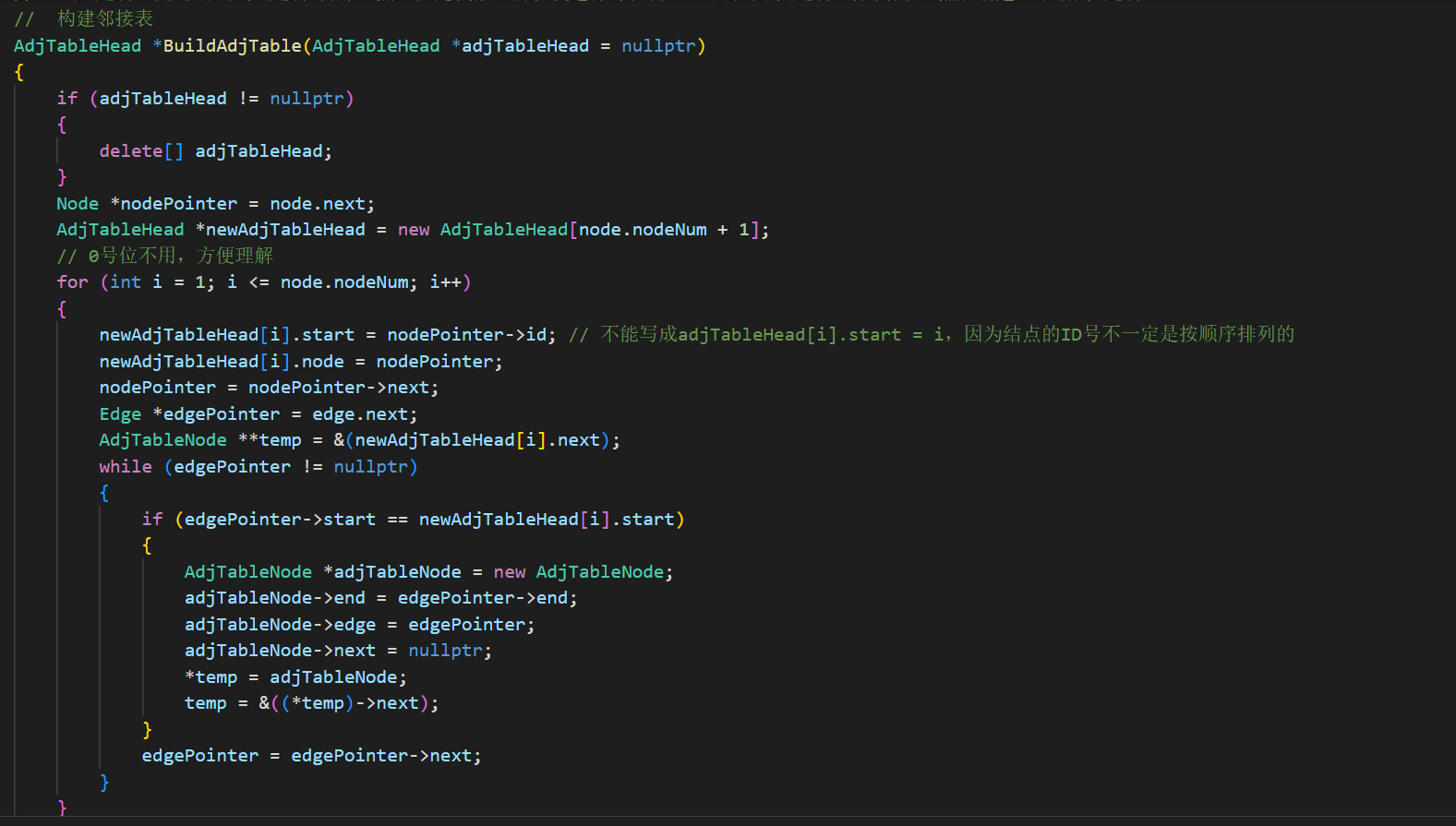


### 概要设计

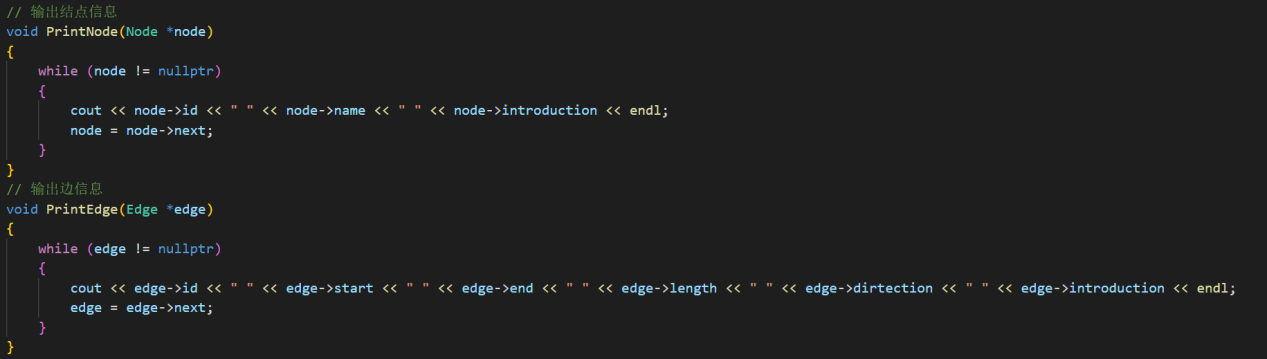


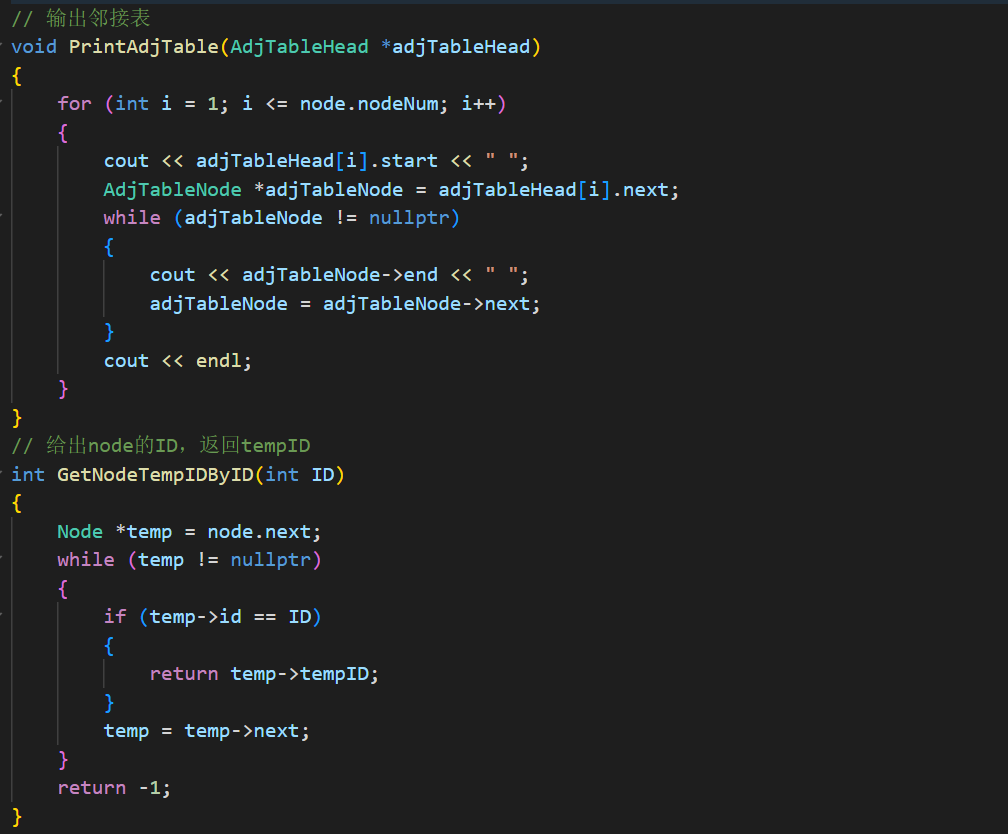


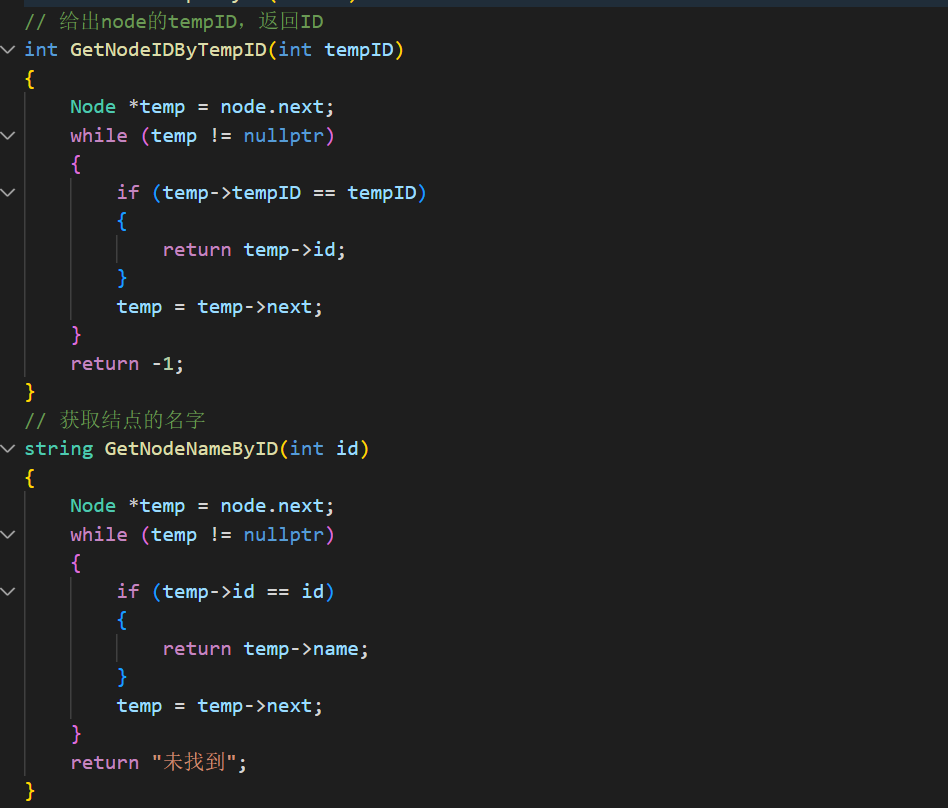


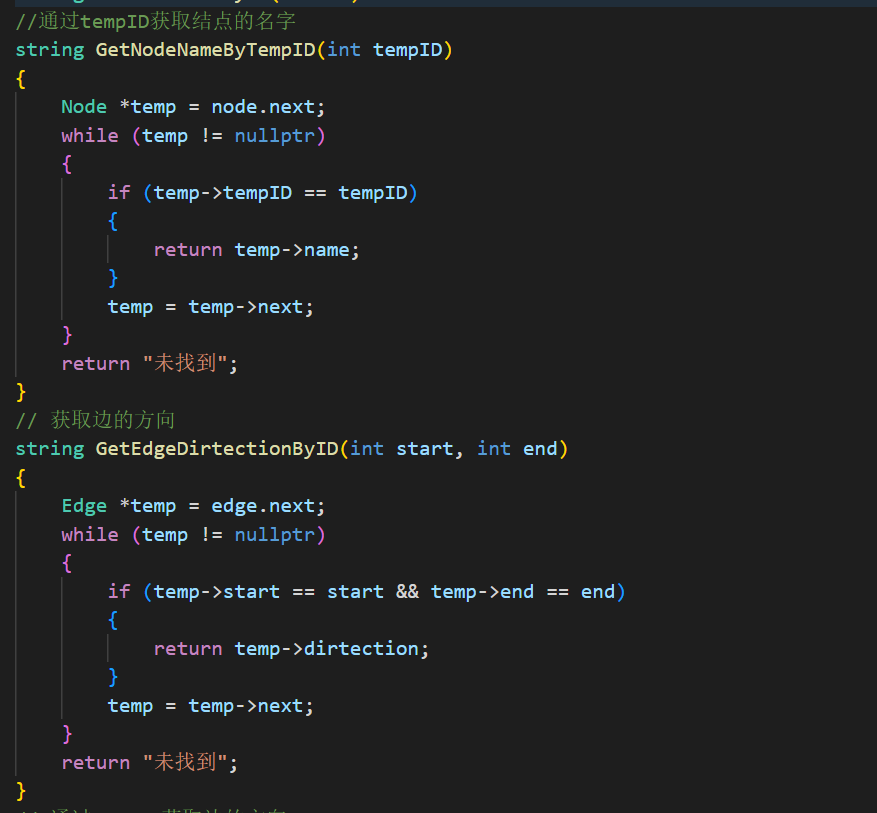


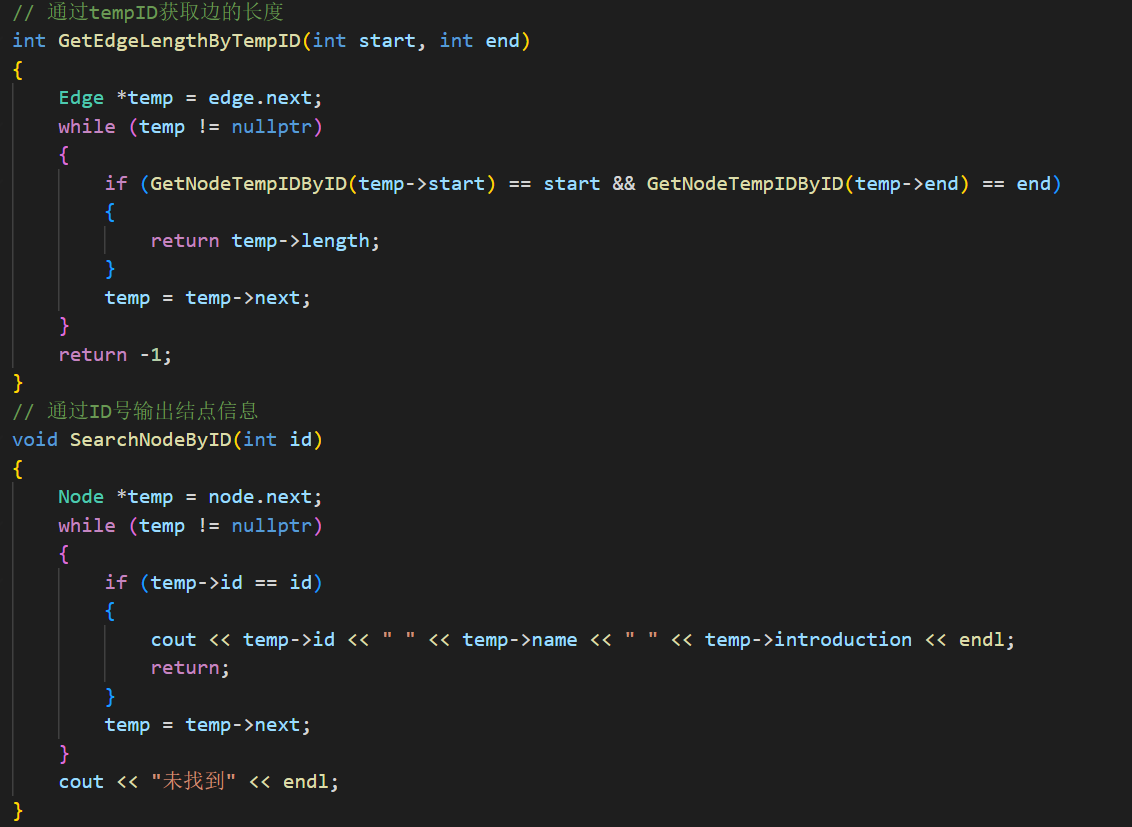
### 详细设计

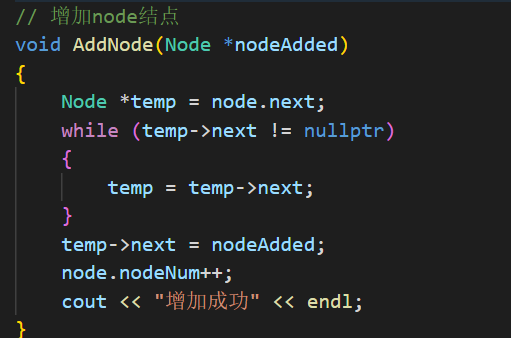


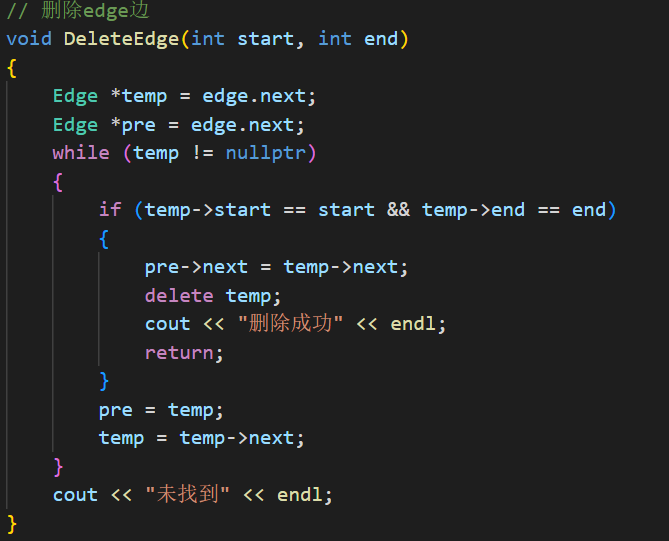


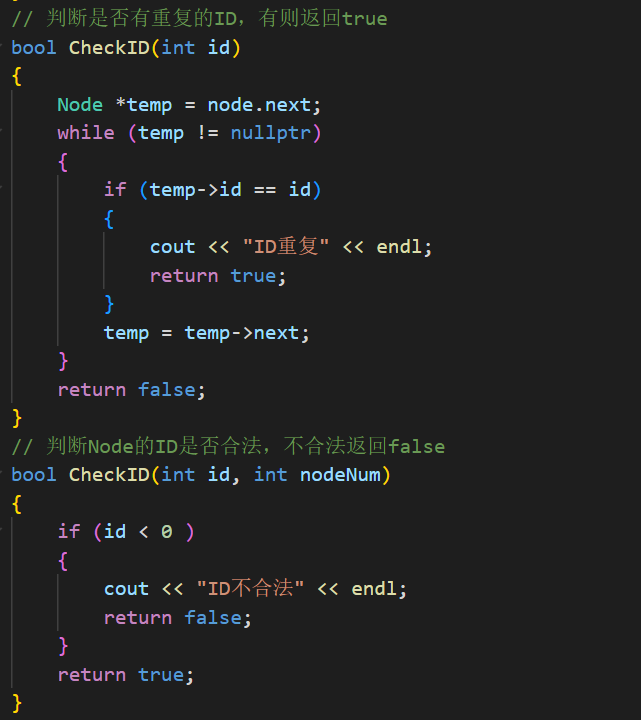


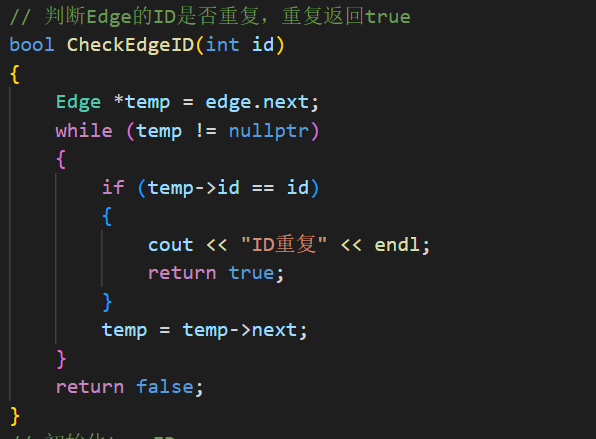


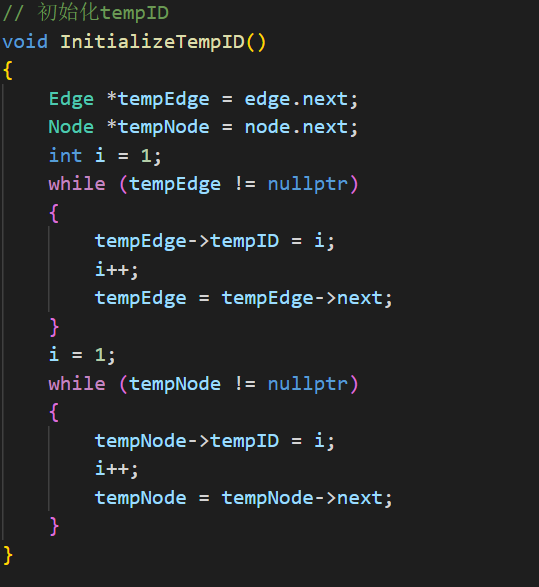












// 读取文件并存储

void ReadFile()

{

    Node \*\*nodePointer = &node.next;

    Edge \*\*edgePointer = &edge.next;

    ifstream file;

    // file.open("Information.json", ios::in);//一般情况下是可以的，但是我的相对路径有问题，原因是我在vscode配置c环境时，修改了默认路径

    file.open("E:/datastructure/Project/Information.json", ios::in);

    if (!file.is\_open())

    {

        cout << "文件打开失败";

        exit(1);

    }

    string str;

    while (getline(file, str))

    {

        if (str == "    \"points\": [") // 这里的优化方式有正则表达式、修改json数据格式，比如id前加一个#，检索数据时就以#为标志

        {

            while (getline(file, str))

            {

                if (str == "    ],")

                {

                    break;

                }

                else if (str == "        {")

                {

                    Node \*tempNode = new Node;

                    getline(file, str);

                    tempNode->id = stoi(str.substr(str.find(":") + 1, MAXLENGTH));//因为stoi，读到""也没事

                    getline(file, str);

                    tempNode->name = str.substr(str.find(":") + 3, str.find(",") - str.find(":") - 4);//这里在数据里添加读取的标识符比较好，比如#

                    getline(file, str);

                    tempNode->introduction = str.substr(str.find(":") + 3, str.find("#") - str.find(":") - 3);

                    getline(file, str);

                    \*nodePointer = tempNode;

                    nodePointer = &(tempNode->next);

                    node.nodeNum++;

                }

            }

        }

        else if (str == "    \"edges\": [")

        {

            while (getline(file, str))

            {

                if (str == "    ],")

                {

                    break;

                }

                else if (str == "        {")

                {

                    Edge \*tempEdge = new Edge;

                    getline(file, str);

                    tempEdge->id = stoi(str.substr(str.find(":") + 2,MAXLENGTH));

                    getline(file, str);

                    tempEdge->start = stoi(str.substr(str.find(":") + 2, MAXLENGTH));

                    getline(file, str);

                    tempEdge->end = stoi(str.substr(str.find(":") + 2, MAXLENGTH));

                    getline(file, str);

                    tempEdge->length = stoi(str.substr(str.find(":") + 2, MAXLENGTH));

                    getline(file, str);

                    tempEdge->dirtection = str.substr(str.find(":") + 3, 6);//这里是按字节数来算的，utf-8中一个汉字占3个字节，所以这里是6

                    getline(file, str);

                    tempEdge->introduction = str.substr(str.find(":") + 2, str.find("#") - str.find(":") - 3);

                    getline(file, str);

                    \*edgePointer = tempEdge;

                    edgePointer = &tempEdge->next;

                    edge.edgeNum++;

                }

            }

        }

    }

    file.close();

}

void Menu(AdjTableHead \*adjTableHead)

{

    cout << "1.查询景点信息" << endl;

    cout << "2.查询两景点间最短路径及路线" << endl;

    cout << "3.查询两景点间所有路径及路线" << endl;

    cout << "4.增加或修改景点信息" << endl;

    cout << "5.删除景点" << endl;

    cout << "6.增加边" << endl;

    cout << "7.删除边" << endl;

    cout << "8.退出" << endl;

    cout << "请输入您的选择：";

    int choice;

    cin >> choice;

    switch (choice)

    {

    case 1://查询景点信息

    {

        PrintView();

        cout << "请输入景点编号：";

        int ID;

        cin >> ID;

        SearchNodeByID(ID);

        break;

    }

    case 2://查询两景点间最短路径及路线

    {

        cout << "1.查询两景点间最短路径及路线（无必须经过的景点）" << endl;

        cout << "2.查询两景点间最短路径及路线（有必须经过的景点）" << endl;

        cout << "请输入您的选择：";

        cin >> choice;

        switch (choice)

        {

        case 1:

        {

            PrintView();

            cout << "请输入起点编号：";

            int start;

            cin >> start;

            cout << "请输入终点编号：";

            int end;

            cin >> end;

            ShortestPathByDijkstra(adjTableHead, start, end);

            break;

        }

        case 2:

        {

            PrintView();

            cout << "请输入起点编号：";

            int start;

            cin >> start;

            cout << "请输入终点编号：";

            int end;

            cin >> end;

            cout << "请输入必须经过的景点编号(请以0作为结束输入)：";

            vector<int> mustPass;

            int temp;

            cin >> temp;

            while (temp != 0)

            {

                mustPass.push\_back(temp);

                cin >> temp;

            }

            GetShortestPathThoughMustPoint(adjTableHead, start, end, mustPass);

            break;

        }

        default:

        {

            cout << "输入错误，请重新输入" << endl;

            Menu(adjTableHead);

            break;

        }

        }

        break;

    }

    case 3://查询两景点间所有路径及路线

    {

        PrintView();

        cout << "请输入起点编号：";

        int start;

        cin >> start;

        cout << "请输入终点编号：";

        int end;

        cin >> end;

        AllPathByID(adjTableHead, start, end);

        break;

    }

    case 4://增加或修改景点信息

    {

        cout << "请输入景点ID: ";

        int ID;

        cin >> ID;

        //检查ID是否合法

        if (CheckID(ID,node.nodeNum) == false)

            break;

        cout << "请输入景点名称：";

        string name;

        cin >> name;

        cout << "请输入景点介绍：";

        string introduction;

        cin >> introduction;

        if(CheckID(ID))

        {

            //修改结点

            Node \*temp = node.next;

            while (temp != nullptr)

            {

                if (temp->id == ID)

                {

                    temp->name = name;

                    temp->introduction = introduction;

                    break;

                }

                temp = temp->next;

            }

            cout << "修改成功" << endl;

        }

        else

        {

            //增加结点

            Node \*newNode = new Node;

            newNode->id = ID;

            newNode->name = name;

            newNode->introduction = introduction;

            AddNode(newNode);

        }

        InitializeTempID();

        break;

    }

    case 5://删除景点

    {

        PrintView();

        cout << "请输入要删除的景点编号：";

        int id;

        cin >> id;

        if (CheckID(id,node.nodeNum) == false)

            break;

        DeleteNode(id);

        InitializeTempID();

        break;

    }

    case 6://增加边

    {

        cout << "请输入ID号：";

        int id;

        cin >> id;

        if (CheckEdgeID(id) == true)

            break;

        PrintView();

        cout << "请输入起点编号：";

        int start;

        cin >> start;

        if (CheckID(start,node.nodeNum) == false)

            break;

        cout << "请输入终点编号：";

        int end;

        cin >> end;

        if (CheckID(end,node.nodeNum) == false)

            break;

        cout << "请输入边的长度：";

        int length;

        cin >> length;

        if (length <= 0)

        {

            cout << "长度必须大于0" << endl;

            break;

        }

        cout << "请输入边的方向：";

        string dirtection;

        cin >> dirtection;

        cout << "请输入边的介绍：";

        string introduction;

        cin >> introduction;

        Edge \*newEdge = new Edge;

        newEdge->id = id;

        newEdge->start = start;

        newEdge->end = end;

        newEdge->length = length;

        newEdge->dirtection = dirtection;

        newEdge->introduction = introduction;

        AddEdge(newEdge);

        InitializeTempID();

        break;

    }

    case 7://删除边

    {

        PrintView();

        cout << "请输入起点编号：";

        int start;

        cin >> start;

        if (CheckID(start,node.nodeNum) == false)

            break;

        cout << "请输入终点编号：";

        int end;

        cin >> end;

        if (CheckID(end,node.nodeNum) == false)

            break;

        DeleteEdge(start, end);

        InitializeTempID();

        break;

    }

    case 8://退出

    {

        exit(0);

        break;

    }

    default://输入错误

    {

        cout << "输入错误，请重新输入" << endl;

        Menu(adjTableHead);

        break;

    }

    }

}

迪杰斯特拉求最短路径

// 求两点间最短路径，迪杰斯特拉算法，使用邻接表，输出以start为起点，到各个点的最短路径

void ShortestPathByDijkstra(AdjTableHead \*adjTableHead, int start, int end)

{

    bool isSure[node.nodeNum + 1];  // 是否已经确定最短路径

    int distance[node.nodeNum + 1]; // 起点到各点的距离

    int path[node.nodeNum + 1];     // 各点的前驱节点

    // 初始化

    for (int i = 1; i <= node.nodeNum; i++)

    {

        isSure[i] = false;

        distance[i] = MAXINT;

        path[i] = -1;

    }

    // 遍历邻接表，寻找以start为起点的数据结构，存储到temp

    int startTempID = GetNodeTempIDByID(start);

    AdjTableNode \*temp=adjTableHead[startTempID].next;

    while (temp != nullptr)

    {

        distance[GetNodeTempIDByID(temp->edge->end)] = temp->edge->length;

        path[GetNodeTempIDByID(temp->edge->end)] = startTempID;

        temp = temp->next;

    }

    isSure[startTempID] = true;

    distance[startTempID] = 0;

    // 主循环

    for (int i = 1; i <= node.nodeNum; i++)

    {

        int min = MAXINT;

        int minIndex = -1;

        // 找到未确定最短路径的最小距离点

        for (int j = 1; j <= node.nodeNum; j++)

        {

            if (!isSure[j] && distance[j] < min)

            {

                min = distance[j];

                minIndex = j;

            }

        }

        if (minIndex == -1)

        {

            break;

        }

        isSure[minIndex] = true;

        // 更新通过这个点可到的点的距离，此时还不一定是最短距离

        temp = adjTableHead[minIndex].next;

        while (temp != nullptr)

        {

            if (!isSure[GetNodeTempIDByID(temp->edge->end)] && distance[GetNodeTempIDByID(temp->edge->end)] > distance[minIndex] + temp->edge->length)

            {

                distance[GetNodeTempIDByID(temp->edge->end)] = distance[minIndex] + temp->edge->length;

                path[GetNodeTempIDByID(temp->edge->end)] = minIndex;

            }

            temp = temp->next;

        }

    }

    int endTempID = GetNodeTempIDByID(end);

    // 输出

    int tempEnd = endTempID;

    if (path[tempEnd] == startTempID)

    {

        cout << GetNodeNameByID(start) << "向" << GetEdgeDirtectionByID(start, end) << "方向"

             << "->" << GetNodeNameByID(end);

        cout << endl

             << "距离为" << distance[endTempID] << endl;

        return;

    }

    while (path[tempEnd] != startTempID)

    {

        vector<int> reversePath;

        while (path[tempEnd] != startTempID)

        {

            reversePath.push\_back(tempEnd);

            tempEnd = path[tempEnd];

        }

        reversePath.push\_back(tempEnd);

        reversePath.push\_back(startTempID);

        for (int i = reversePath.size() - 1; i >= 0; i--)

        {

            cout << GetNodeNameByTempID(reversePath[i]);

            if (i != 0)

            {

                cout << "向" << GetEdgeDirtectionByTempID(reversePath[i], reversePath[i - 1]) << "方向" << GetEdgeLengthByTempID(reversePath[i], reversePath[i - 1]) << "->";

            }

        }

    }

    cout << endl << "距离为" << distance[endTempID] << endl;

}

Dfs求全部路径

void dfs(AdjTableHead \*AdjTableHead, int start, int end, bool isVisited[], vector<int> path)

{

    isVisited[start] = true;

    if (start == end)

    {

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

        {

            cout << GetNodeNameByTempID(path[i]);

            if (i != path.size() - 1)

            {

                cout << "->";

            }

        }

        cout << endl;

    }

    else

    {

        AdjTableNode \*temp = AdjTableHead[start].next;

        while (temp != nullptr)

        {

            if (!isVisited[GetNodeTempIDByID(temp->edge->end)])

            {

                path.push\_back(GetNodeTempIDByID(temp->edge->end));

                dfs(AdjTableHead, GetNodeTempIDByID(temp->edge->end), end, isVisited, path);

                path.pop\_back();

            }

            temp = temp->next;

        }

    }

    isVisited[start] = false;

}

void AllPathByID(AdjTableHead \*AdjTableHead, int start, int end)

{

    start = GetNodeTempIDByID(start);

    end = GetNodeTempIDByID(end);

    bool isVisited[node.nodeNum + 1] = {0};

    vector<int> path;

    path.push\_back(start);

    dfs(AdjTableHead, start, end, isVisited, path);

}

求两点间经过某点的最佳路径

// 求经过的路径长度

int GetPathLength(vector<int> path)

{

    int length = 0;

    for (int i = 0; i < path.size() - 1; i++)

    {

        length += GetEdgeLengthByTempID(path[i], path[i + 1]);

    }

    return length;

}

// 输出经过mustPath的最短路径

void PrintShortestPathThoughMustPoint(vector<vector<int>> &allPath, vector<int> mustPath)

{

    int min = MAXINT;

    int minIndex = -1;

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

    {

        int temp = 0;

        for (int j = 0; j < allPath[i].size(); j++)

        {

            for (int k = 0; k < mustPath.size(); k++)

            {

                if (allPath[i][j] == mustPath[k])

                {

                    temp++;

                }

            }

        }

        if (temp == mustPath.size() && GetPathLength(allPath[i]) < min)

        {

            min = GetPathLength(allPath[i]) ;

            minIndex = i;

        }

    }

    if (minIndex == -1)

    {

        cout << "不存在经过必须经过的点的路径" << endl;

    }

    else

    {

        cout << "经过必须经过的点的最短路径为：" << endl;

        for (int i = 0; i < allPath[minIndex].size(); i++)

        {

            cout << GetNodeNameByTempID(allPath[minIndex][i]);

            if (i != allPath[minIndex].size() - 1)

            {

                cout << "->";

            }

        }

        cout << endl;

        cout << "最小路径长度为： " << min << endl;

    }

}

//该重载是用于将所有路径存在allPath中

void dfs(AdjTableHead \*AdjTableHead, int start, int end, bool isVisited[], vector<int> path, vector<vector<int>> &allPath)

{

    isVisited[start] = true;

    if (start == end)

    {

        vector<int> temp;

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

        {

            temp.push\_back(path[i]);

        }

        allPath.push\_back(temp);

    }

    else

    {

        AdjTableNode \*temp = AdjTableHead[start].next;

        while (temp != nullptr)

        {

            if (!isVisited[GetNodeTempIDByID(temp->edge->end)] && GetNodeTempIDByID(temp->edge->end) != -1)

            {

                path.push\_back(GetNodeTempIDByID(temp->edge->end));

                dfs(AdjTableHead, GetNodeTempIDByID(temp->edge->end), end, isVisited, path, allPath);

                path.pop\_back();

            }

            temp = temp->next;

        }

    }

    isVisited[start] = false;

}

// 求两点间全部路径，将所有路径存储到一个二维动态数组中，遍历该二维动态数组，比较是否存在必须经过的地点，如果有，记录路径长度。

void GetShortestPathThoughMustPoint(AdjTableHead \*AdjTableHead, int start, int end, vector<int> mustPath)

{

    start = GetNodeTempIDByID(start);

    end = GetNodeTempIDByID(end);

    bool isVisited[node.nodeNum + 1] = {0};

    vector<int> path;

    path.push\_back(start);

    vector<vector<int>> allPath;

    dfs(AdjTableHead, start, end, isVisited, path, allPath);

    PrintShortestPathThoughMustPoint(allPath, mustPath);

}

### 四、调试分析

4.1 读取json数据文件时使用了相对路径，但无法成功读取 。原因为在我配置vscode的c环境时，更改了其相对位置的默认路径，导致其默认路径指向了mingw的文件夹下。

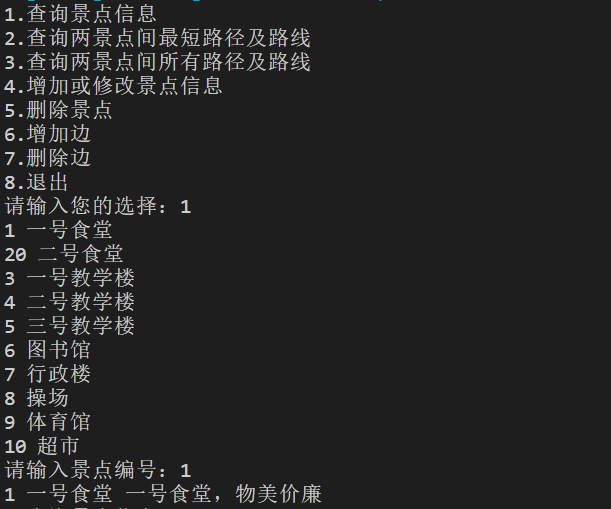
解决方法为修改为绝对路径，或者修改vscode指向的默认相对路径。

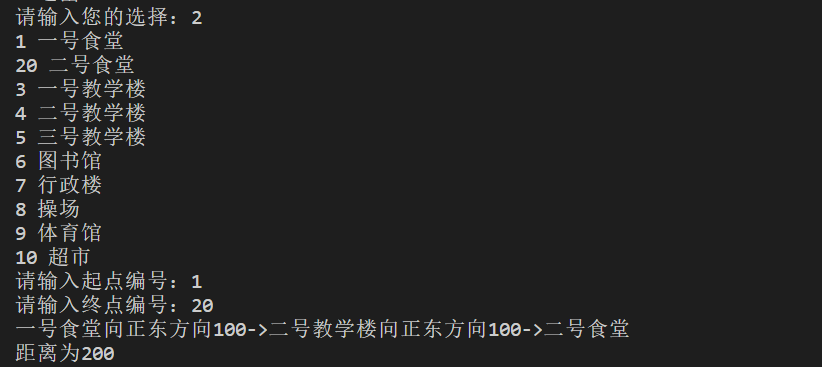
4.2 对读取到的json数据文件进行数据存储时，使用了字符串相等的方式来判定。该方法效率低下，可以换成正则表达式、使用特殊字符标记内容以读取等等。

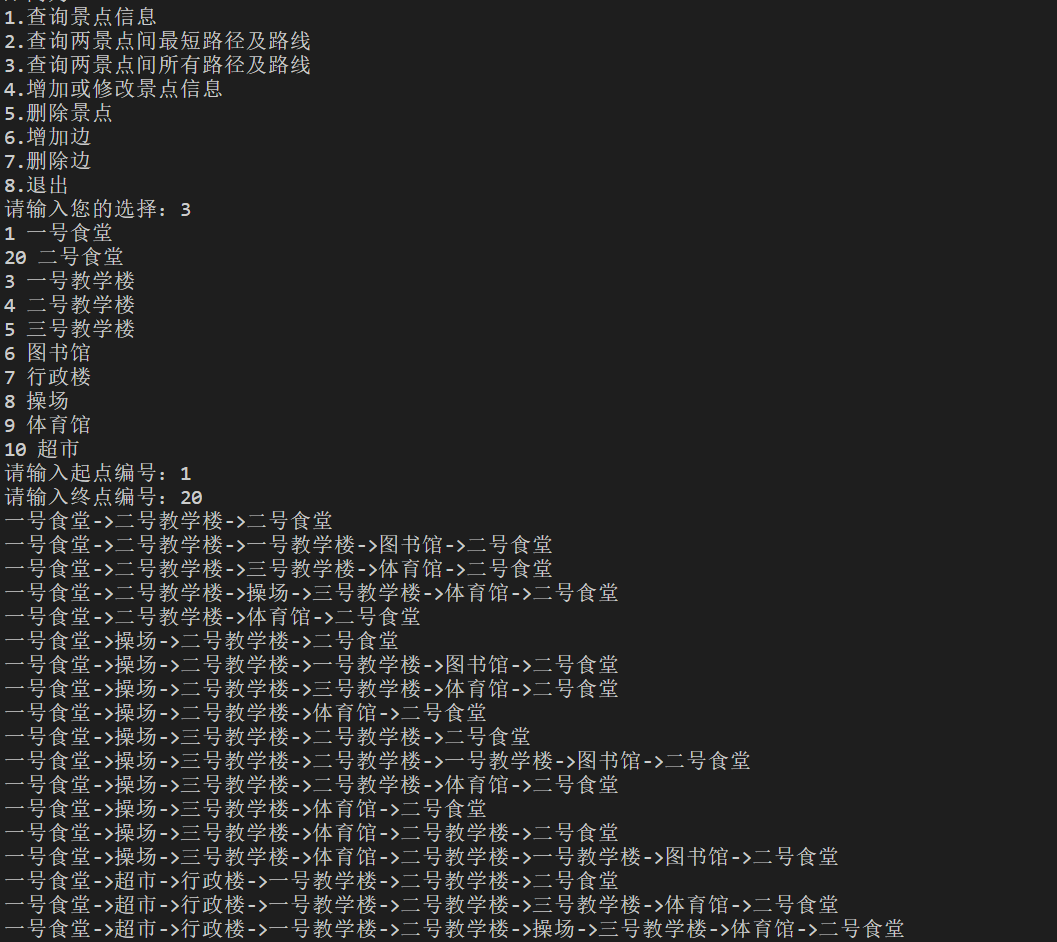
4.3 希望通过输出所有路径为文件的方式，来减少后续计算机运行的量。但是输出的文件名我取了中文名，导致无法成功输出。解决方法为输出文件名为英文即可。

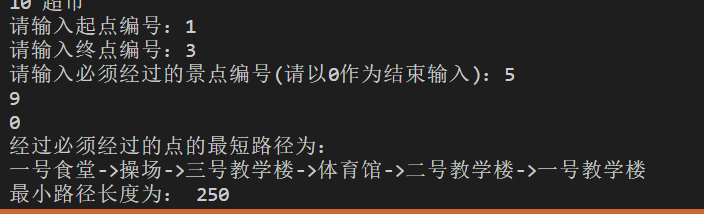
4.4 创建存储景点、路径的存储结构时，一开始使用指针创建，创建失败。解决方法为使用指针的指针创建。

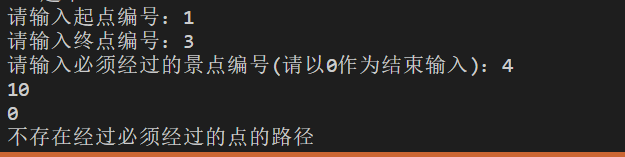
### 五、测试结果











### 六、总结

数据结构这门课程注重理论与实践相结合。通过学习各种数据结构如数组、链表、栈、队列、树、图等，我不仅掌握了它们的基本概念和性质，还学会了如何在实际问题中应用它们。同时，通过实现各种算法，我更加深入地理解了算法的设计与分析方法。这门课程还强调解决问题的能力。通过学习这门课程，我学会了如何将复杂问题分解为若干个子问题，然后运用合适的数据结构和算法来解决这些子问题。这种解决问题的能力不仅在计算机领域有用，在其他领域也同样重要。