

**数据结构课程实践**

**学号 姓名**

实习报告-实习三

2023

目录

[需求分析 2](#_Toc87012154)

[输入形式与范围 2](#_Toc87012155)

[输出形式 2](#_Toc87012156)

[程序所能达到的功能 2](#_Toc87012157)

[概要设计 2](#_Toc87012159)

[ADT定义 2](#_Toc87012160)

[主程序流程图 2](#_Toc87012161)

[模块层次与调用关系 2](#_Toc87012162)

[详细设计 2](#_Toc87012163)

[主程序伪代码 3](#_Toc87012165)

[调试分析 3](#_Toc87012167)

[遇到的问题与解决 3](#_Toc87012168)

[经验与体会 3](#_Toc87012170)

[用户使用说明 3](#_Toc87012171)

[测试结果 3](#_Toc87012172)

# 需求分析

## 输入形式与范围

分别从文件site.txt和path.txt读取景点信息和路径信息

## 输出形式

输出景点信息；输出两个景点之间的最短路径；输出两个景点之间的所有路径

## 程序所能达到的功能

（1）设计学校的校园平面图，所含景点不少于10个，以图中顶点表示校内各景点，存放景点名称、代号、简介等信息；以边表示路径，存放路径长度等相关信息；

（2）为来访客人提供图中任意景点相关信息的查询；

（3）为来访客人提供景点的问路查询，即已知一个景点，查询到某景点之间的一条最短路径及长度。

（4）提供图中任意景点问路查询，即求任意两个景点之间的所有路径

# 概要设计

## ADT定义

ADT AGragh{

数据对象：V是具有相同特征的数据元素的集合，称为顶点集；

数据关系：

R={VR}

VR={<v,w>|v,w属于V且P(v,w)<v,w>表示从v到w的弧，

谓词P{(c,w)定义了弧<v,w>的意义或信息}

基本操作：

CreateGraph (AGraph&graph, VertexType v, VertexType w, int wei)

初始条件：v,w是图中一条边的两个顶点，wei是边的权重。

操作条件: 按已有的边创建图的邻接表

int FirstNeighbor(AGraph G, int v)

初始条件：v是作为弧尾的顶点。

操作条件: 找到以v为弧尾的第一条边

int NextNeighbor(AGraph G, int v, int w)

初始条件：v是作为弧尾的顶点，w是作为弧头的顶点。

操作条件: 找到以v为弧尾，以w为弧头的边的下一条以v为弧尾的边

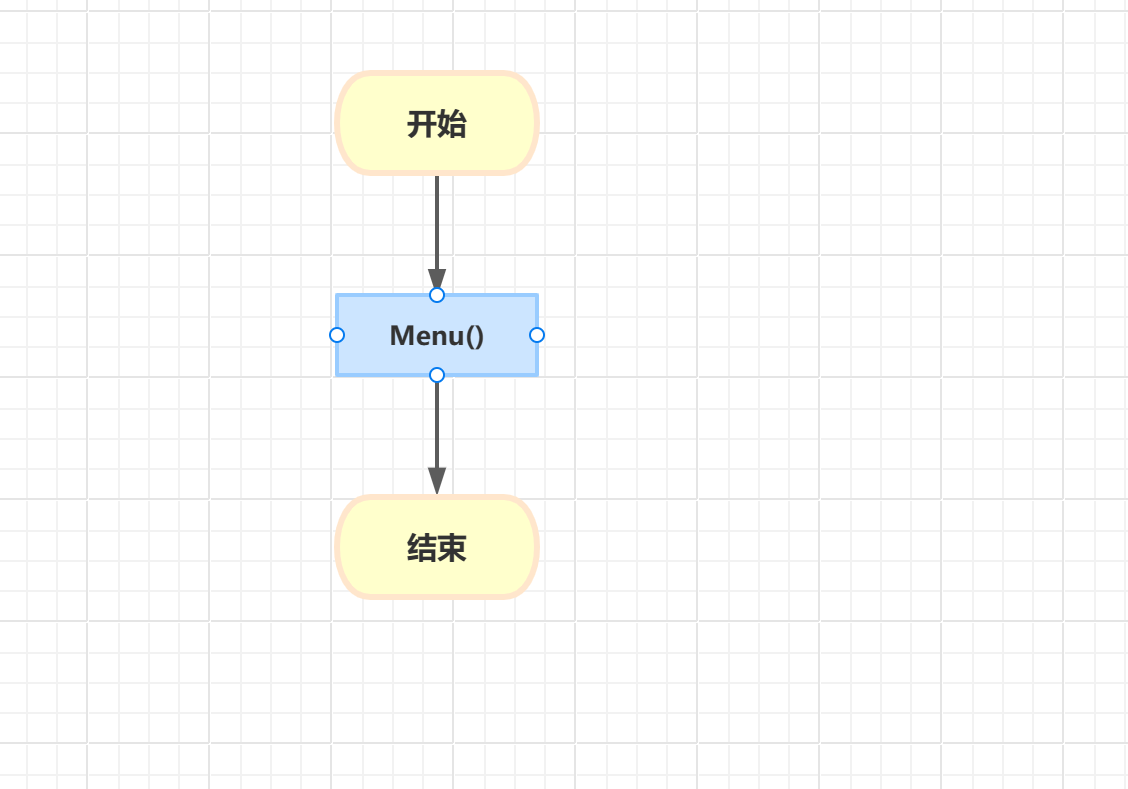
void DFS(int begin, int end, int& i, int& k, int& minindex, int& maxindex, int& minpath, int& maxpath)

初始条件：begin为起点，end为终点，i记录路径序号，k记录路径顶点个数，minindex和maxindex分别记录最短路径和最长路径长度序号下标，minpath和maxpath分别记录最短路径和最长路径长度

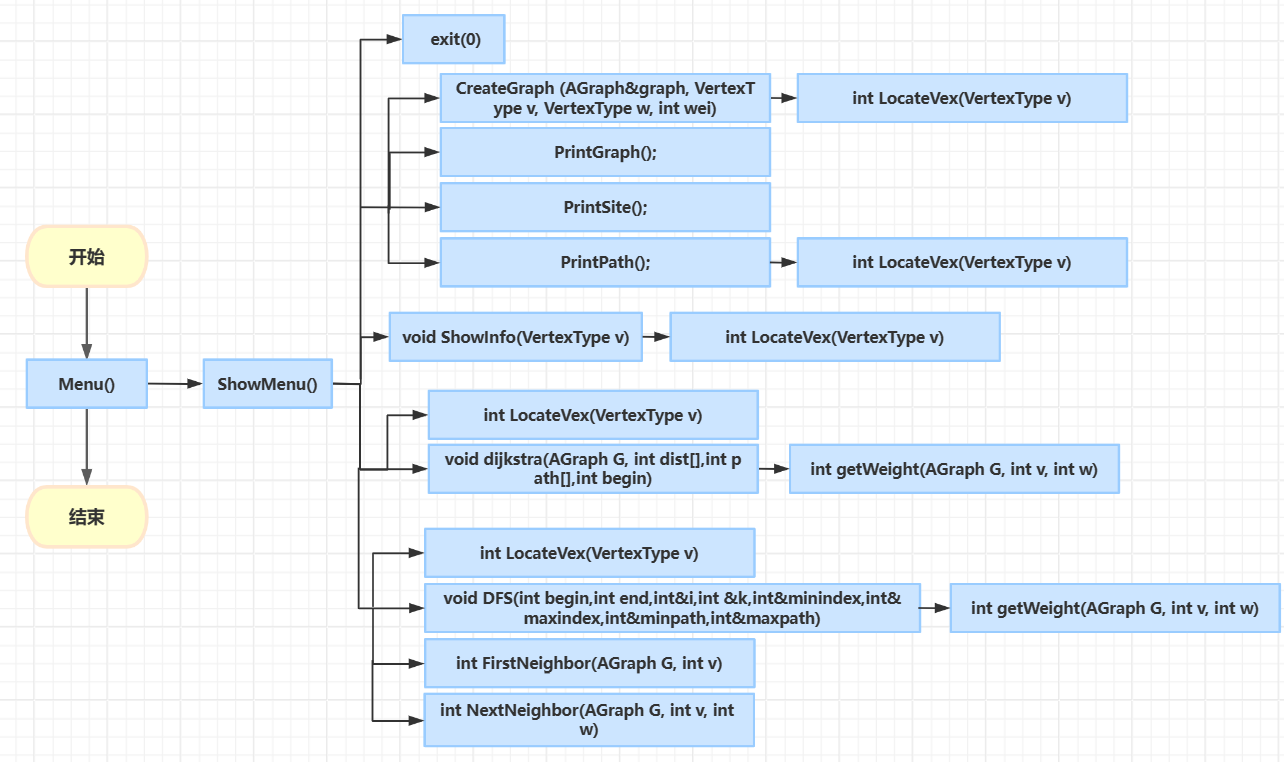
操作结果：对图进行深度优先遍历，输出两个顶点之间的所有路径，最短路径和最长路径。

}

## 主程序流程图



## 模块层次与调用关系



# 详细设计

#include<iostream>

#include<fstream>

#include<sstream>

#include<opencv.hpp>

#define INFINITY 99999

using namespace std;

using namespace cv;

#define MaxVertexNum 100

#define MAXSIZE 1000

typedef char VertexType;

typedef int EdgeType;

//边

typedef struct EdgeNode {

int adjvex;

EdgeType weight;

struct EdgeNode\* next;

}EdgeNode;

//点

typedef struct VertexNode {

VertexType data;//景点代号

string name;//景点名称

string brief;//景点简介

EdgeNode\* firstedge;

}VertexNode, AdjList[MaxVertexNum];

//邻接表

typedef struct {

AdjList adjList;

int numVertexes, numEdges;

}AGraph;

AGraph graph;

bool visited[MaxVertexNum];

int\* dist = new int[graph.numVertexes];//（迪杰斯特拉）记录点到各点的最短路径长度

int\* path = new int[graph.numVertexes];//记录路径下标

//找点

int LocateVex(VertexType v) {

for (int i = 0; i < graph.numVertexes; i++)

if (graph.adjList[i].data == v)

return i;

cout << "输入错误！";

exit(0);

}

//获取边的权重

int getWeight(AGraph G, int v, int w) {

EdgeNode\* edge = G.adjList[v].firstedge;

while (edge != NULL) {

if (edge->adjvex == w)

return edge->weight;

edge = edge->next;

}

return INFINITY;

}

//建图

void CreateGraph(AGraph&graph, VertexType v, VertexType w, int wei) {

EdgeNode\* tail;

int i = 0, j = 0;

int indexv = LocateVex(v);

int indexw = LocateVex(w);

EdgeNode\* e1 = new EdgeNode;

tail = graph.adjList[indexv].firstedge;

graph.adjList[indexv].firstedge = e1;

e1->next = tail;

e1->adjvex = indexw;

e1->weight = wei;

EdgeNode\* e2 = new EdgeNode;

tail = graph.adjList[indexw].firstedge;

graph.adjList[indexw].firstedge = e2;

e2->next = tail;

e2->adjvex = indexv;

e2->weight = wei;

}

//第一条边

int FirstNeighbor(AGraph G, int v) {

if(G.adjList[v].firstedge!=NULL)

return G.adjList[v].firstedge->adjvex;

return -1;

}

//其他边

int NextNeighbor(AGraph G, int v, int w) {

if (v != -1) {

EdgeNode\* edge = G.adjList[v].firstedge;

while (edge != NULL && edge->adjvex != w) edge = edge->next;

if (edge != NULL && edge->next != NULL) return edge->next->adjvex;

}

return -1;

}

//递归打印迪杰斯特拉路径

void print\_the\_path(int path[],int begin, int i) {

if (i == begin) return;

print\_the\_path(path, begin,path[i]);

cout << "->" << graph.adjList[i].name << '(' << graph.adjList[i].data << ')';

}

//查找所有路径

void DFS(int begin,int end,int&i,int &k,int&minindex,int&maxindex,int&minpath,int&maxpath) {

visited[begin] = true;

int sum;//统计路径长度

path[k++] = begin;

if (begin == end) {

sum = 0;

cout << "路线"<<i<<"："<<graph.adjList[path[0]].name<<'('<<graph.adjList[path[0]].data<<')';

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

sum += getWeight(graph,path[j - 1], path[j]);

cout <<"——>"<< graph.adjList[path[j]].name << '(' << graph.adjList[path[j]].data << ')';

}

cout <<endl<<"路径长度为："<<sum<< endl;

if (sum < minpath) {

minpath = sum;

minindex = i;

}

if (sum > maxpath) {

maxpath = sum;

maxindex = i;

}

i++;

}

for (int w = FirstNeighbor(graph, begin); w != -1; w = NextNeighbor(graph, begin, w)) {

if (!visited[w])

DFS(w,end,i,k,minindex,maxindex,minpath,maxpath);

}

visited[begin] = false; //一条简单路径处理完，退回一个顶点继续遍历

k--;

}

//迪杰斯特拉算法

void dijkstra(AGraph G, int dist[],int path[],int begin) {

int i, j, u, min;

int weight;

for (i = 0; i < G.numVertexes; i++) {

path[i] = -1;

visited[i] = false;

dist[i] = INFINITY;

}

EdgeNode\* edge = G.adjList[begin].firstedge;

while (edge != NULL) {

dist[edge->adjvex] = edge->weight;

path[edge->adjvex] = begin;

edge = edge->next;

}

visited[begin] = true;

dist[begin] = 0;

//找最小路径的弧头

for (i = 1; i < G.numVertexes; i++) {

min = INFINITY;

for (j = 0; j < G.numVertexes; j++) {

if (visited[j] == false && dist[j] < min) {

min = dist[j];

u = j;

}

}

visited[u] = true;

//更新下一次最短路径

for (j = 0; j < G.numVertexes; j++) {

weight = getWeight(G, u, j);

if (visited[j] == false && dist[u] + weight < dist[j]) {

dist[j] = dist[u] + weight;

path[j] = u;

}

}

}

}

//打印景点信息

void PrintSite() {

cout << "杭州电子科技大学景点信息：" << endl;

cout << "代号" << " " << "景点" << " " << "简介" << endl;

for (int i = 0; i < graph.numVertexes; i++) {

int num = 0;

VertexNode v=graph.adjList[i];

cout <<' '<< v.data << " " << v.name << " ";

while (num<(int)v.brief.size()) {

cout << v.brief[num++];

if (num % 30 == 0)cout <<endl << " " ;

}

cout << endl<<endl;

}

}

//打印邻接表

void PrintGraph() {

for (int i = 0; i < graph.numVertexes; i++) {

VertexNode v = graph.adjList[i];

EdgeNode\* first = v.firstedge;

cout << i << ' ' << v.data;

while (first != NULL) {

cout <<' '<<graph.adjList[first->adjvex].data;

first = first->next;

}

cout << endl;

}

cout << endl;

}

//打印所有边

void PrintPath() {

FILE\* fp;

if ((fp = fopen("path.txt", "r")) == NULL) {

cout << "文件打开失败！" << endl;

exit(0);

}

VertexType v, w;

int weight;

int i = 0;

while (fscanf(fp, "%c %c %d\n", &v, &w, &weight) != EOF) {

int indexv=LocateVex(v);

int indexw = LocateVex(w);

cout << i+1 << '.';

cout << graph.adjList[indexv].name << '(' << graph.adjList[indexv].data << ')'

<< "------" << graph.adjList[indexw].name << '(' << graph.adjList[indexw].data << ')'

<< " 路径长度：" << weight << endl;

i++;

}

fclose(fp);

}

//打印指定景点信息

void ShowInfo(VertexType v) {

int index=LocateVex(v);

cout << "景点" << graph.adjList[index].data << graph.adjList[index].name << "的简介：" << endl;

cout << graph.adjList[index].brief << endl;

}

void ShowMenu() {

cout << " =============================================================================\n";

cout << "|| ★★★★★★★杭州电子科技大学校园十景导览★★★★★★★ ||\n";

cout << "||============================================================================||\n";

cout << "||============================================================================||\n";

cout << "|| 【0】--- 退出 ||\n";

cout << "|| 【1】--- 创建杭电十景图 ||\n";

cout << "|| 【2】--- 查询杭电十景信息 ||\n";

cout << "|| 【3】--- 查询两个景点之间的最短路径 ||\n";

cout << "|| 【4】--- 查询两个景点之间的所有路径 ||\n";

cout << "|| 【5】--- 打印杭电十景分布图 ||\n";

cout << " ==============================================================================\n";

cout << "请输入数字来选择对应的功能：";

}

void Menu() {

while (1)

{

system("cls");

ShowMenu();

int num;

if (!(cin >> num))

{

cout << "输入格式错误！请重新输入：" << endl;

}

else {

switch (num) {

case 0:

exit(0);

case 1://创建校园十景图

{

system("cls");

char buf[MAXSIZE];

int num = 0,line\_num=0;

fstream out;

out.open("site.txt", ios::in);

while (!out.eof()) {

out.getline(buf, MAXSIZE, '\n');

line\_num++;

if (line\_num % 3 == 1) {

graph.adjList[(line\_num - 1) / 3].data = buf[0];

graph.adjList[(line\_num - 1) / 3].firstedge = NULL;

}

if (line\_num % 3 == 2)

graph.adjList[(line\_num - 1) / 3].name = buf;

if (line\_num % 3 == 0)

graph.adjList[(line\_num - 1) / 3].brief = buf;

}

out.close();

graph.numVertexes = line\_num / 3;

FILE\* fp;

if ((fp = fopen("path.txt", "r")) == NULL) {

cout << "文件打开失败！" << endl;

exit(0);

}

VertexType v, w;

int weight;

while (fscanf(fp, "%c %c %d\n", &v, &w, &weight) != EOF) {

CreateGraph(graph, v, w, weight);

num++;

}

fclose(fp);

graph.numEdges = num;

cout << "创建成功" << endl;

cout << "校园十景邻接图：" << endl;

PrintGraph();

cout << "校园十景信息：" << endl;

PrintSite();

cout << "校园十景路径：" << endl;

PrintPath();

break;

}

case 2://查询校园十景信息

{

system("cls");

for (int i = 0; i < graph.numVertexes; i++) {

cout << graph.adjList[i].data << '.' << graph.adjList[i].name << endl;

}

cout << "请输入需要查询的景点代号：";

VertexType ch;

cin >> ch;

system("cls");

ShowInfo(ch);

break;

}

case 3://查询两个景点之间的最短路径

{

system("cls");

VertexType v, w;

for (int i = 0; i < graph.numVertexes; i++) {

cout << graph.adjList[i].data << '.' << graph.adjList[i].name << endl;

}

cout << "请输入需要查询的起点景点代号：";

cin >>v;

cout << "请输入需要查询的终点景点代号：";

cin >>w;

int begin = LocateVex(v);

int end = LocateVex(w);

dijkstra(graph, dist, path, begin);

cout << "从" << graph.adjList[begin].name << '(' << graph.adjList[begin].data << ')'

<< "到" << graph.adjList[end].name << '(' << graph.adjList[end].data << ')' << "的最短路线为：";

cout << graph.adjList[begin].name << '(' << graph.adjList[begin].data << ')';

print\_the\_path(path, begin,end);

cout << " 路径长度为" << dist[end]<<endl;

break;

}

case 4://查询景点之间的所有路径

{

system("cls");

VertexType v, w;

for (int i = 0; i < graph.numVertexes; i++) {

cout << graph.adjList[i].data << '.' << graph.adjList[i].name << endl;

}

cout << "请输入需要查询的起点景点代号：";

cin >> v;

cout << "请输入需要查询的终点景点代号：";

cin >> w;

int begin = LocateVex(v);

int end = LocateVex(w);

for (int i = 0; i < graph.numVertexes; i++) {

visited[i] = false;

path[i] = -1;

}

int k = 0,i=1;

int minindex, maxindex;

int minpath=MAXSIZE, maxpath=0;

cout << "从" << graph.adjList[begin].name << '(' << graph.adjList[begin].data << ')'

<< "到" << graph.adjList[end].name << '(' << graph.adjList[end].data << ')'

<< "的所有路线为："<<endl;

DFS(begin, end,i,k,minindex,maxindex,minpath,maxpath);

cout << endl;

cout << "最长路线为第" << maxindex << "条路线，路径长度为" << maxpath<<endl;

cout << "最短路线为第" << minindex << "条路线，路径长度为" << minpath<<endl;

break;

}

case 5:

{

String str = "杭电十景.jpg";

Mat image = imread(str);

namedWindow("HDU", WINDOW\_NORMAL);

imshow("HDU", image);

waitKey(0);

break;

}

}

}

system("pause");

}

}

int main() {

Menu();

return 0;

}

# 调试分析

## 遇到的问题与解决

1. 如何逐行读取文件信息？

用C++读取文件的函数out实现逐行读取，用buffer字符数组接收读取的内容，用line\_num计算读取的行数

1. 如何输出两个景点之间的所有路径？

使用DFS深度遍历算法，用visited[]记录顶点是否访问过，用path[]记录经过的顶点，用k记录顶点的个数。在DFS找到路径输出并出栈的时候，把顶点设置为未访问，k减一，表示路径退回，如此递归即可输出所有路径

## 经验与体会

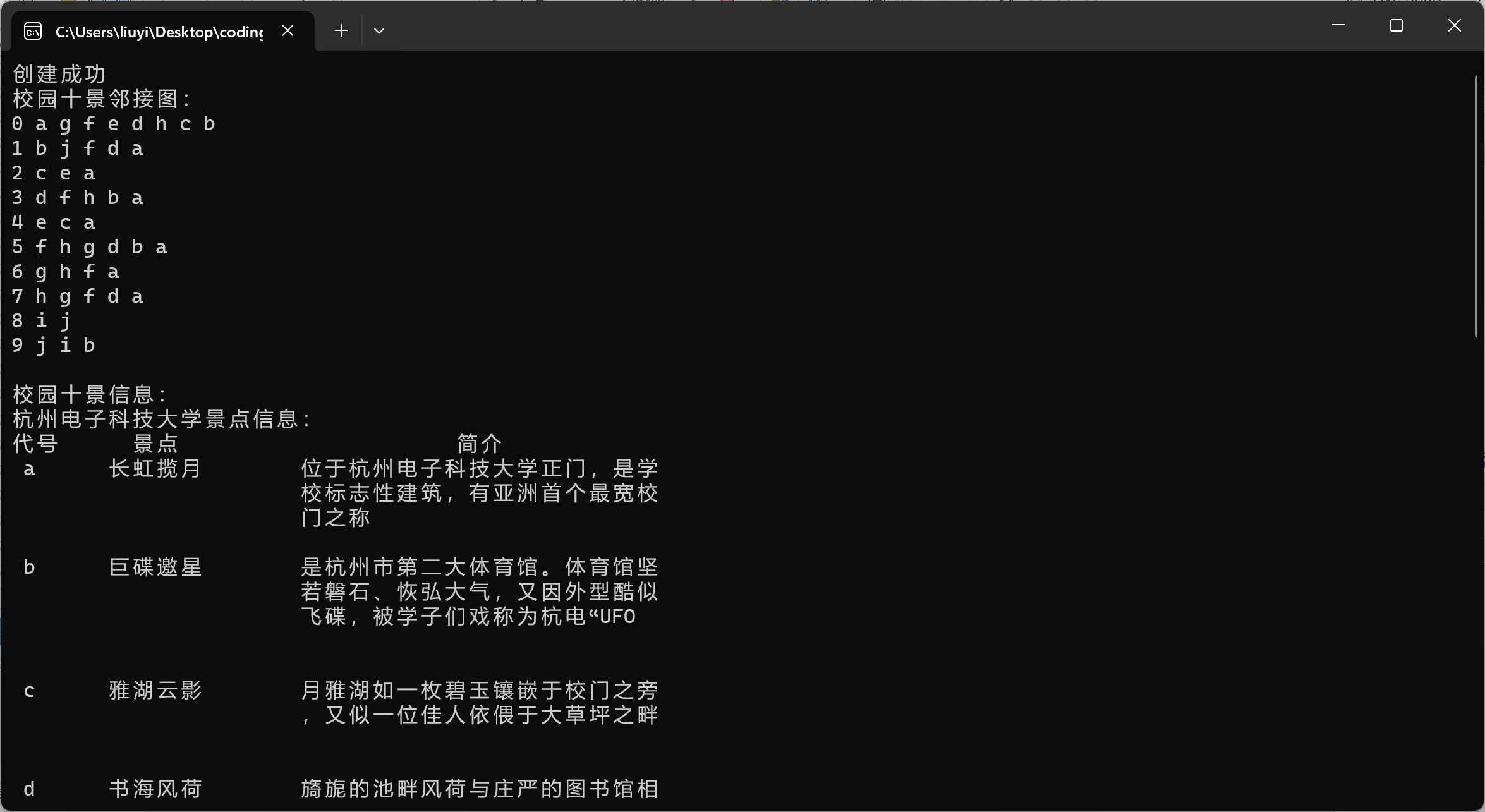
1. **项目规划和设计：** 在着手编写代码之前，花一些时间仔细规划和设计项目。明确项目的目标，确定所需的功能和特性，以及项目的整体架构。这有助于避免后期的混乱和修改。
2. **选择适当的数据结构：** 根据项目的需求，选择适当的数据结构是至关重要的。本次实践需要使用图来表示校园地图。在实习过程中更加能理解每种数据结构的优势和劣势，以及它们如何在项目中相互作用。

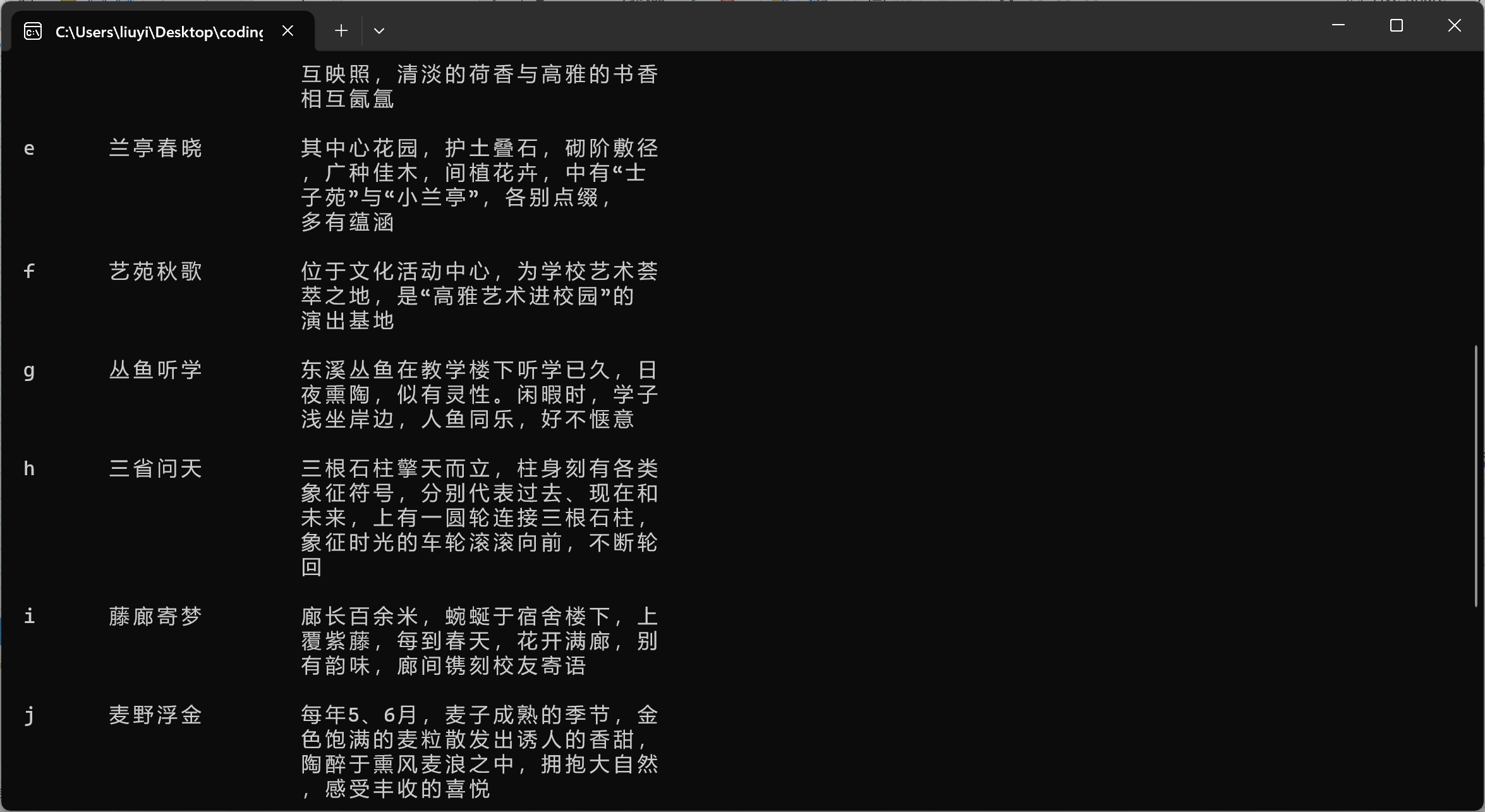
# 用户使用说明

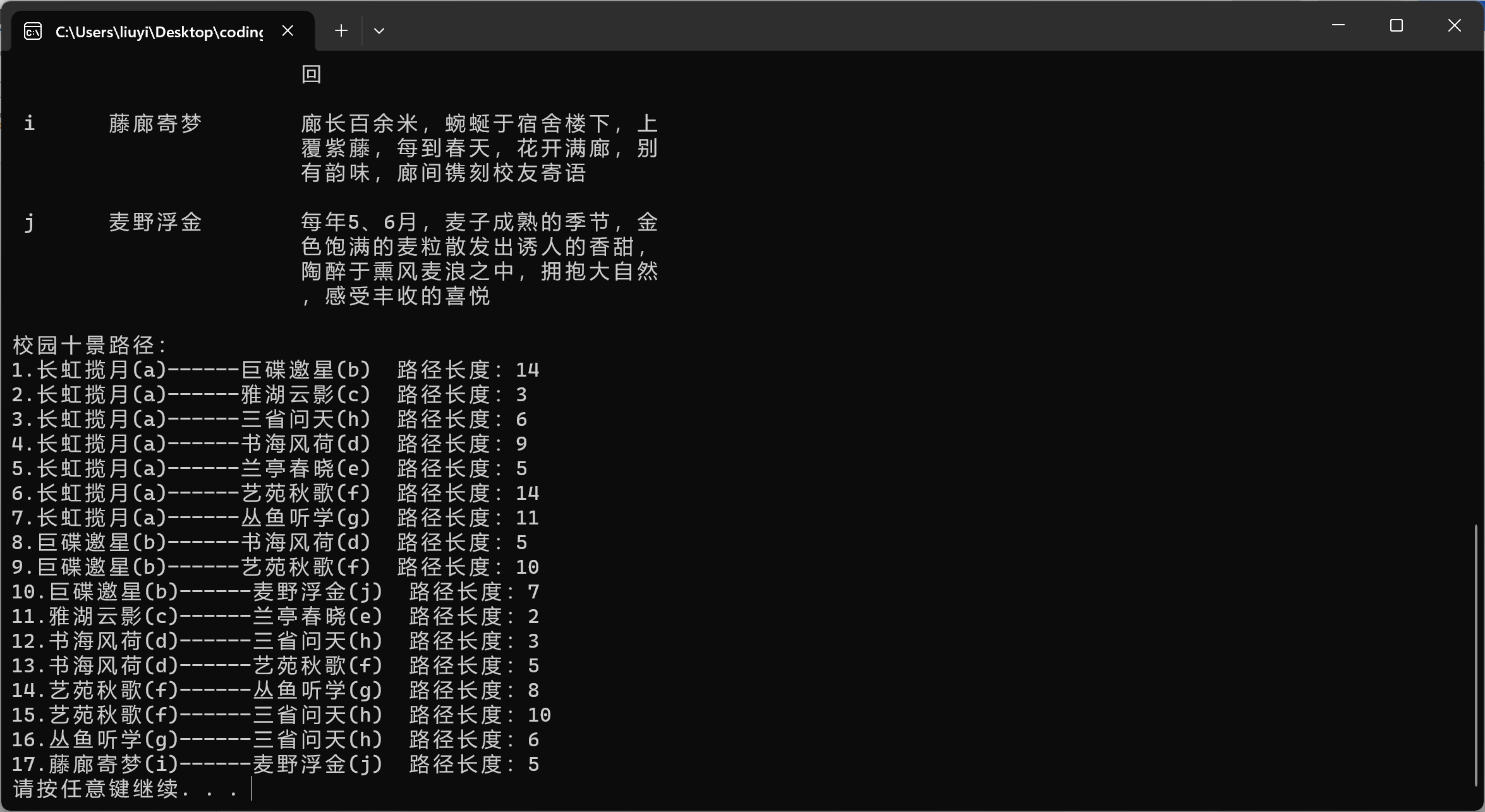
用户进入菜单，根据自身需求输入相应的序号，根据各个功能的提示可以得到所需的信息，如景点信息、最短路径信息和所有路径信息

# 测试结果

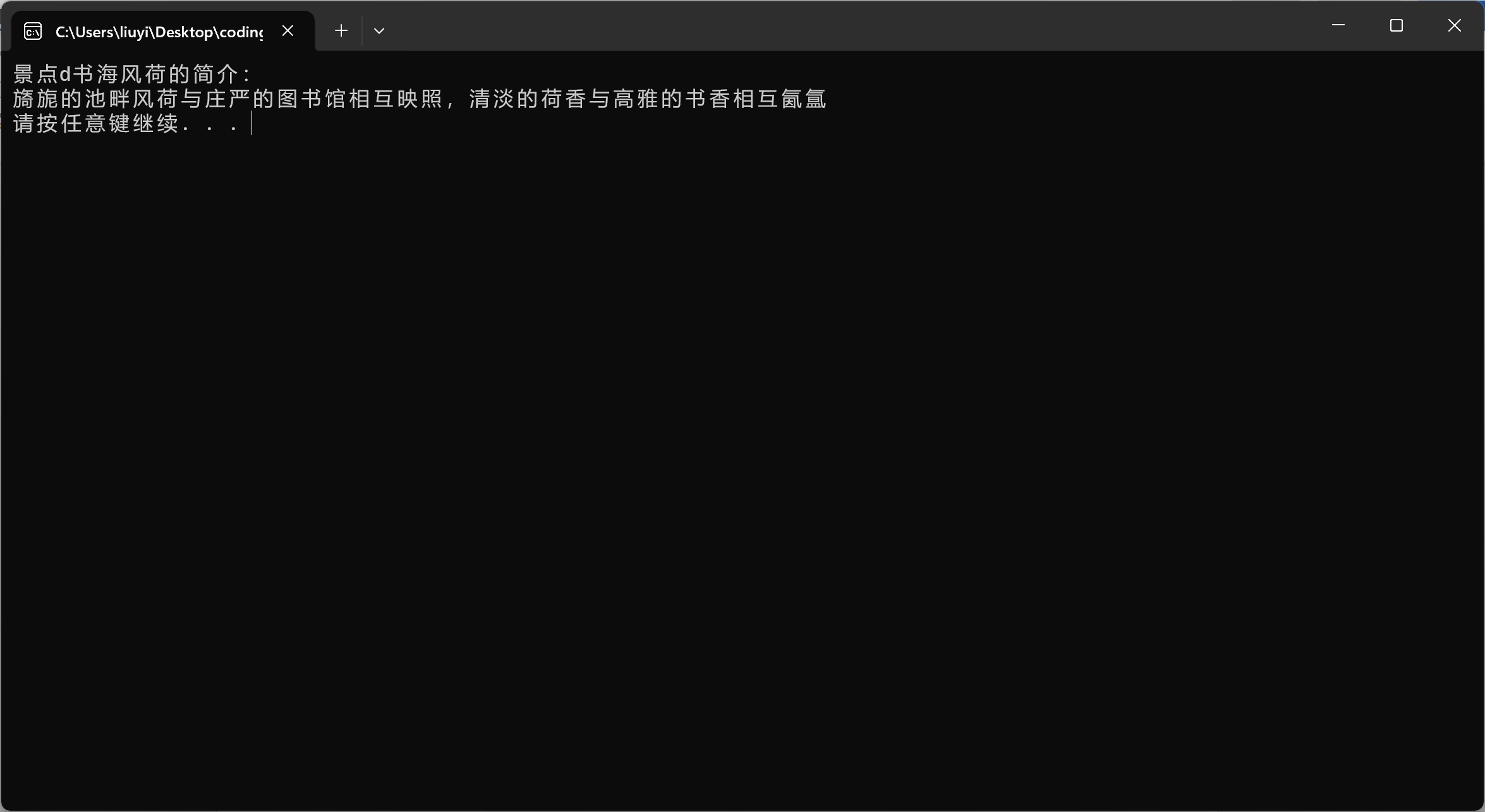
创建景点的图邻接表



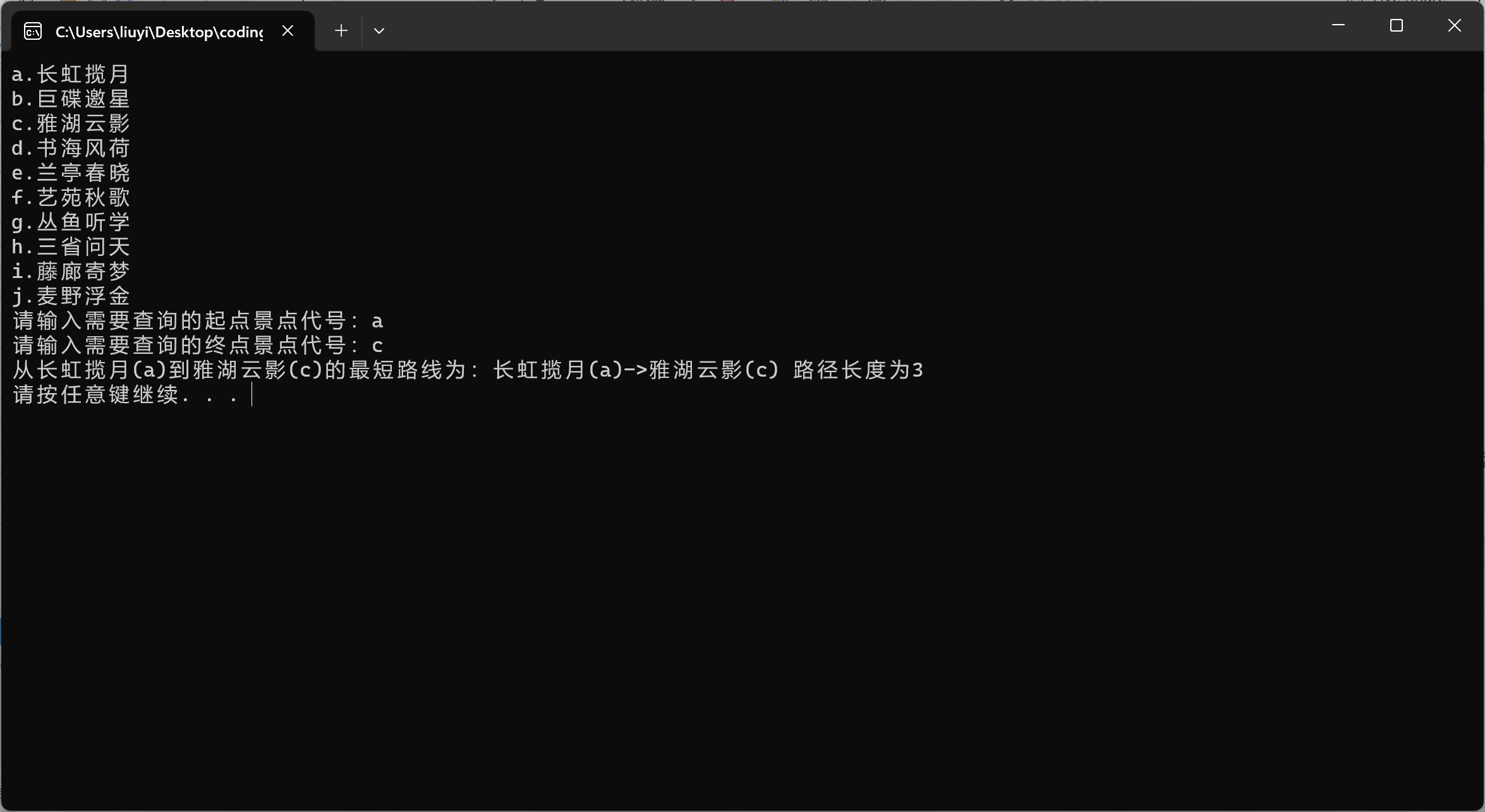




查询景点信息

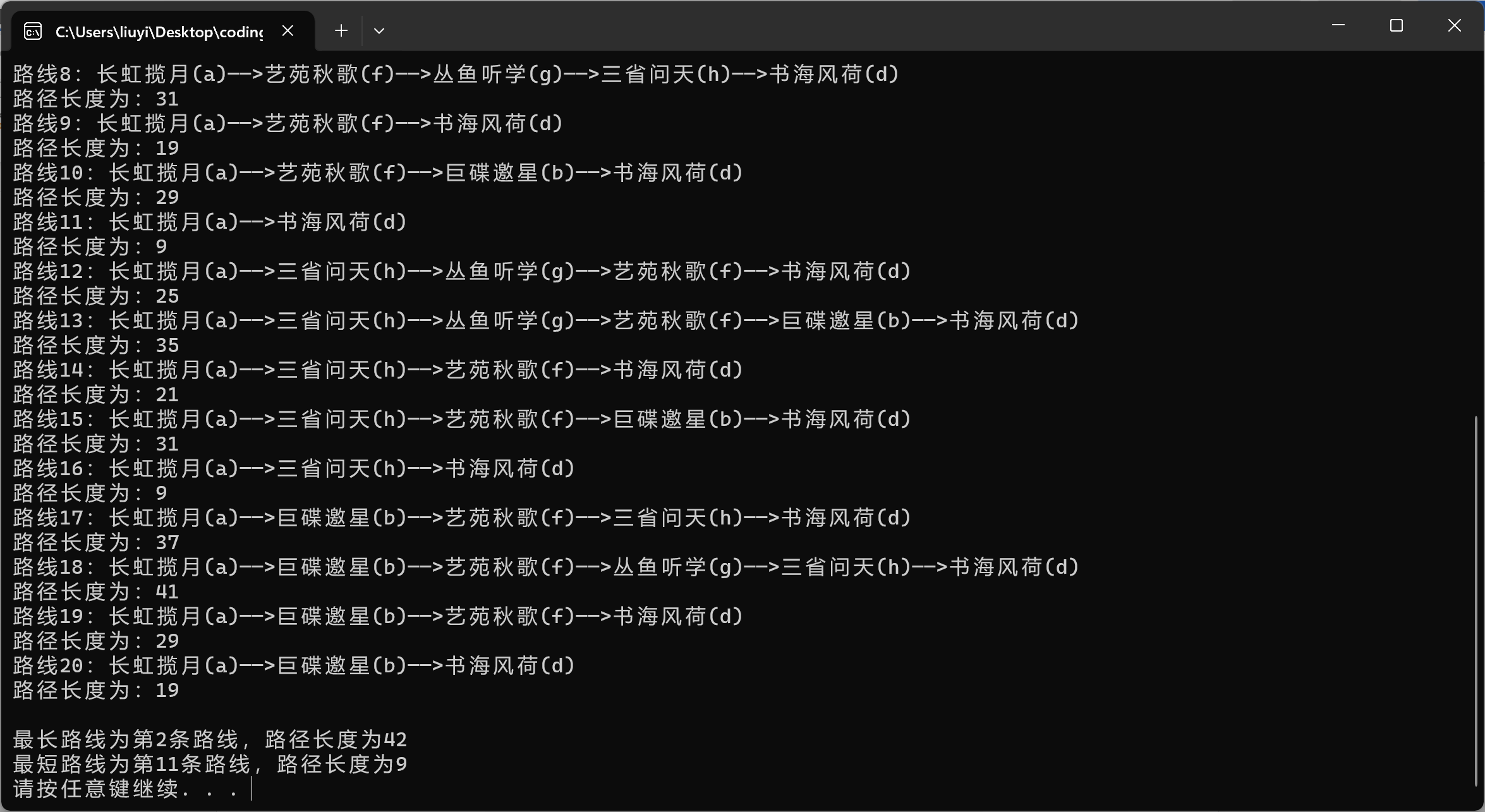


查询两个景点之间的最短路径及其长度



查询两个景点之间的所有路径及其长度，并输出最短路径长度和最长路劲长度





打印杭电十景地图

