数据结构

课程设计报告

广州大学 计算机科学与网络工程学院

计算机系 17级软件工程专业1班

林欣煜

（学号：1706300032）

（班内序号：32）

指导教师： 杜娇

2019年6月17日

1. 课程题目

**社交网络图实现（难度系数 1.4）**

[问题描述]：设计并实现一种简单的社交网络模型图。

[基本要求]：

（1） 每个人的信息是一个结点，人与人的联系构成边。个人信息里要有地理坐标信息，以便后续应用中能方便找靠近的人。

（2） 根据输入的任意两个人信息，给出他们之间的联系路径，最少经过多少人构成联系。

（3） 根据位置信息的动态变化，找寻附近能够联络的人，能够通过1次中间人能联络的人等。

（4） 可根据自己的创意添加更多的功能。

二．实验要求

1. 所有的课程设计报告，均要有封面，包括：课题名称、班级、学号、学生姓名和指导教师；
2. 给出自己采用的数据结构；
3. 给出算法设计思想；
4. 给出实现的源程序，并在必要的代码处给出注释；
5. 给出测试数据和结果；
6. 给出算法的时间复杂度、另外可以提出算法的改进方法；
7. 给出结束语：说明完成课程设计的情况，心得体会；

8. 必须在指定的时间内上交课程设计报告;

9. 学习委员提交一张光盘，用黑色笔写清楚班别、课程名；里面的内容包括全班同学的文件。光盘的内容分两个子目录：源程序（所有同学的源程序）、课程设计报告（所有同学的报告的电子版）

1. 开发环境

操作系统：Win10

IDE：VScode

1. 功能模块设计

1. 计算距离最近的人。

使用迪杰斯特拉算法计算彼此之间的最短距离直接解决。

2. 搜索两个人之间的所有路径。

先使用迪杰斯特拉算法计算彼此之间的最短距离，再从起点开始寻找每一个中间点，到达终点后回溯递归搜索其余的可行路径（深度递归搜索）。

3. 给出搜索范围寻找可以联络到的人。

使用迪杰斯特拉算法计算彼此之间的最短距离后和输入的范围距离比较即可。

4. 寻找只需联络一次就能建立联系的人。

使用迪杰斯特拉算法计算彼此之间的最短距离的同时，使用一个数组记录从起点到达其他点所经过的中间点即可。

1. 实验算法

**使用到的数据结构：**

const int INF = 100000000;

const int MAX = 500;

int n, m; // 总人数，有关系的人的对数

int dis[MAX]; // 人与人的距离

int bothDis[MAX][MAX]; // 两个有联系的人之间的距离

int through[MAX]; // 到达目标经过的联络人数

bool vis[MAX] = {false}; // 标记是否已经访问过

vector<int> path; // 存放一次联系中经过的联络人

// 存放每个人的地理坐标

struct People {

int x;

int y;

}

**迪杰斯特拉算法：**

以一个人作为起点start，先将每一个人初始化为都不连通，即把彼此之间的距离设为一个极大值INF。从头开始遍历每一个人，如果当前遍历到的人还没有被访问过（使用一个数组来标记）并且它和起点之间的距离小于INF，即两者连通。则以这个人作为中间人，再遍历每一个人判断从start开始通过中间人到达其他人的距离，和从start开始直接到达其他人的距离哪个更短。如果通过中间人达到的路径更短，则更新路径长度。N趟循环后距离数组dis表示的就是从start开始到其他相应每一个人的最短距离。另外，迪杰斯特拉算法的时间复杂度为O(n^2)。

// 迪杰斯特拉算法

void Dijkstra(int start) {

// 初始化值

fill(dis, dis + MAX, INF); // 起点和其他所有点初始化为无穷远，表示不连通

fill(vis, vis + MAX, false);

fill(through, through + MAX, -1); // 两个人之间经过的联络人数初始化为-1，表示互不联系。当两个人相互直接联系时，through为0。

dis[start] = 0; // 起点只和自己相连通

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

int u = -1, MIN = INF;

// 先找到距离起点最近的一个点

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

if(vis[j] == false && dis[j] < MIN) {

u = j;

MIN = dis[j];

}

}

// 标记为已读

vis[u] = true;

// 没有找到说明剩下的点和起点都不连通

if(u == -1) return;

// 以u为中间点，判断从start直接到t，和从start经过u再到t哪个的路径更短

for(int t=0; t<n; t++) {

// t未访问 && 中间点u和t连通 && 以u为中介可以时dis[t]更短

if(vis[t] == false && bothDis[u][t] != INF && dis[u] + bothDis[u][t] < dis[t]) {

dis[t] = dis[u] + bothDis[u][t];

through[t] = through[u] + 1;

}

}

}

}

**深度递归搜索：**

使用vector这个容器来装一条路径中出现的所有人。从起点开始，先将起点start加入到vector中，遍历所有的人，先判断当前循环到的人是否已经存在vector中，不存在的话再判断start和当前这个人是否有联系，有的话就以这个人为起点，递归调用函数进行搜索。每次遍历完后就把vector顶部的人弹出去（vector是先进后出），从而可以遍历到所有可能的路径。在函数头去判断起点start和终点end是否相等，相等的话说明已经找到了目标对象，这时候则遍历整个vector，打印出两个人建立联系经过的所有人。打印完成后还需要把vector顶部元素弹出去，继续回溯寻找另外的路径。

// 深度递归搜索和start相关联的其他所有人

// 使用vector来装路径。后进先出

void DFS(int start, int end) {

path.push\_back(start);

// 找到了目标，输出关联两者时经过的所有联络人

if(start == end) {

for(vector<int>::iterator it=path.begin(); it!=path.end(); it++) {

cout<<\*it<<" ";

}

cout<<endl;

path.pop\_back();

return;

}

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

// 对于已经在path中的结点不重复搜索

int mark = find(path.begin(), path.end(), i) == path.end();

if(mark) {

// start 得和 i 有联系才递归搜索

if(bothDis[start][i] != INF) {

DFS(i, end);

}

}

}

path.pop\_back();

}

1. 源代码

#include <bits/stdc++.h>

using namespace std;

const int INF = 100000000;

const int MAX = 500;

int n, m; // 总人数，有关系的人的对数

int dis[MAX]; // 人与人的距离

int bothDis[MAX][MAX]; // 两个有联系的人之间的距离

int through[MAX]; // 到达目标经过的联络人数

bool vis[MAX] = {false}; // 标记是否已经访问过

vector<int> path; // 存放一次联系中经过的联络人

// 存放每个人的地理坐标

struct People {

int x;

int y;

} people[MAX];

// 输入相关的数据并初始化

void input() {

cout<<"请输入总人数"<<endl;

cin>>n;

cout<<"请分别输入编号为0-"<<n-1<<"的人所在的横纵坐标"<<endl;

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

cin>>people[i].x>>people[i].y;

}

fill(bothDis[0], bothDis[0] + MAX \* MAX, INF);

cout<<"请输入有关系的人的对数"<<endl;

cin>>m;

cout<<"请输入两个有联系的人的编号"<<endl;

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

int a, b;

cin>>a>>b;

bothDis[a][b] = bothDis[b][a] = sqrt(pow(people[a].x - people[b].x, 2) + pow(people[a].y - people[b].y, 2));

}

}

// 迪杰斯特拉算法

void Dijkstra(int start) {

// 初始化值

fill(dis, dis + MAX, INF); // 起点和其他所有点初始化为无穷远，表示不连通

fill(vis, vis + MAX, false);

fill(through, through + MAX, -1); // 两个人之间经过的联络人数初始化为-1，表示互不联系。当两个人相互直接联系时，through为0。

dis[start] = 0; // 起点只和自己相连通

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

int u = -1, MIN = INF;

// 先找到距离起点最近的一个点

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

if(vis[j] == false && dis[j] < MIN) {

u = j;

MIN = dis[j];

}

}

// 标记为已读

vis[u] = true;

// 没有找到说明剩下的点和起点都不连通

if(u == -1) return;

// 以u为中间点，判断从start直接到t，和从start经过u再到t哪个的路径更短

for(int t=0; t<n; t++) {

// t未访问 && 中间点u和t连通 && 以u为中介可以时dis[t]更短

if(vis[t] == false && bothDis[u][t] != INF && dis[u] + bothDis[u][t] < dis[t]) {

dis[t] = dis[u] + bothDis[u][t];

through[t] = through[u] + 1;

}

}

}

}

// 深度递归搜索和start相关联的其他所有人

// 使用vector来装路径。后进先出

void DFS(int start, int end) {

path.push\_back(start);

// 找到了目标，输出关联两者时经过的所有联络人

if(start == end) {

for(vector<int>::iterator it=path.begin(); it!=path.end(); it++) {

cout<<\*it<<" ";

}

cout<<endl;

path.pop\_back();

return;

}

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

// 对于已经在path中的结点不重复搜索

int mark = find(path.begin(), path.end(), i) == path.end();

if(mark) {

// start 得和 i 有联系才递归搜索

if(bothDis[start][i] != INF) {

DFS(i, end);

}

}

}

path.pop\_back();

}

// 功能实现

// 1. 计算和路人甲距离最近的人

// 2. 搜索两个人之间的所有路径

// 3. 给出搜索范围寻找可以联络到的人

// 4. 寻找只需联络一次就能建立联系的人

int main() {

// 初始化并输入测试数据

input();

while(1) {

// 1. 计算和路人甲距离最近的人

{

cout<<"请输入任意一个人的编号用来计算和他距离最近的人"<<endl;

int one;

cin>>one;

Dijkstra(one);

int minDis = INF;

// 先找出和有联系的人之间的最短距离

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

if(i != one && dis[i] < minDis) {

minDis = dis[i];

}

}

if(minDis == INF) {

cout<<"编号为"<<one<<"的人和其他人都没有联系"<<endl;

}

else {

cout<<"和编号为"<<one<<"的人距离最近的人的编号是: ";

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

if(dis[i] == minDis) {

cout<<i<<" ";

}

}

cout<<endl;

cout<<"最短的距离是"<<minDis<<endl;

}

cout<<endl;

}

// 2. 搜索两个有联系的人之间的所有路径

{

cout<<"请分别输入要搜索的两个人的编号用来搜索它们之间的路径"<<endl;

int a, b;

Dijkstra(a);

cin>>a>>b;

cout<<a<<"和"<<b<<"之间的路径有"<<endl;

DFS(a, b);

cout<<"中间经过最少的联络人数为"<<through[b]<<endl;

cout<<endl;

}

// 3. 给出搜索范围寻找可以联络到的人

{

cout<<"请分别输入任意一个人的编号和要搜索的范围"<<endl;

int one;

cin>>one;

int range;

cin>>range;

Dijkstra(one);

cout<<"和"<<one<<"距离不超过"<<range<<"的人有"<<endl;

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

if(i != one && dis[i] <= range) {

cout<<i<<" ";

}

}

cout<<endl<<endl;

}

// 4. 寻找只需联络一次就能建立联系的人

{

cout<<"请输入任意一个人的编号用来寻找和他只需要一个联络人就能联系到的人"<<endl;

int one;

cin>>one;

Dijkstra(one);

cout<<"和"<<one<<"建立联系只需要经过一个联络人的有"<<endl;

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

if(through[i] == 1) {

cout<<i<<" ";

}

}

cout<<endl<<endl;

}

}

}

1. 实验结果及分析

**测试样例：**

总人数：

8

编号为0-7的人所在的横纵坐标：

2 7

4 0

6 4

8 0

10 2

10 5

14 4

16 8

有关系的人的对数：

7

两个有联系的人的编号：

0 2

2 3

1 3

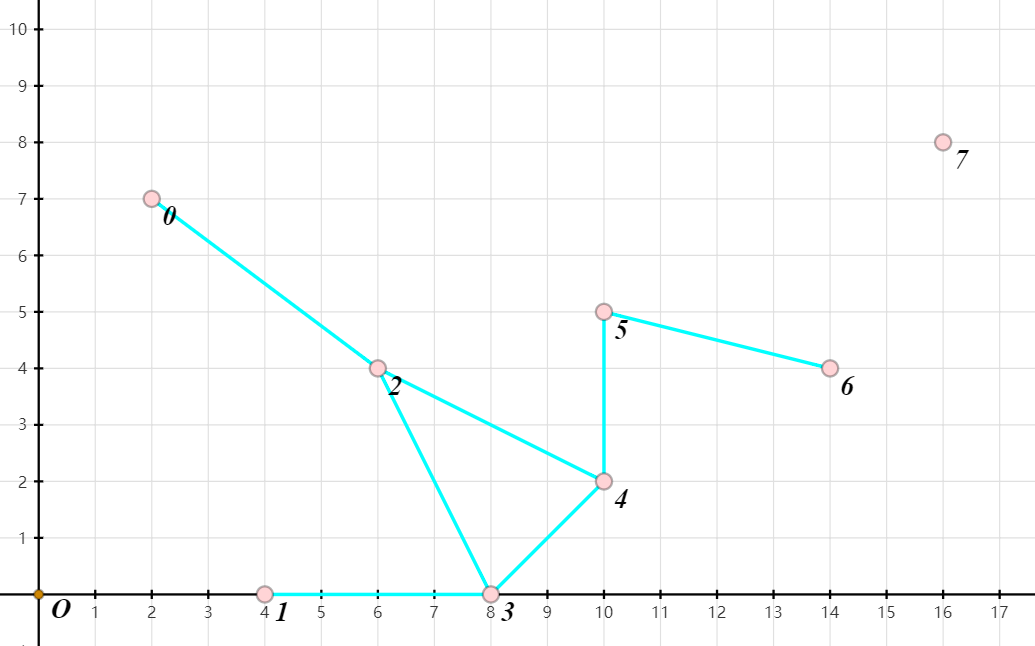
3 4

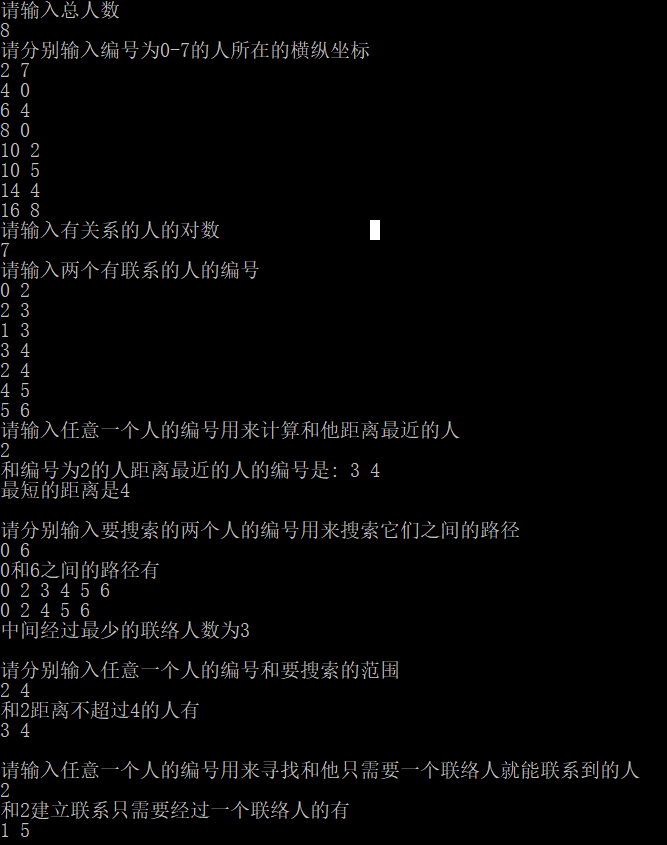
2 4

4 5

5 6

**可视化样例：**





1. 心得体会

完成了本次课程设计的要求。本题是对最短路径的考察，涉及到的算法有迪杰斯特拉算法和深度递归搜索，还有一些对STL容器的使用比如vector。通过本次实验，重新学习了一遍迪杰斯特拉算法的思想，并重温了DFS。

1. 实验的改进意见和建议

本题的解题代码中，是通过每个人的坐标点来计算彼此之间的距离，因为统一是使用int，所以对于距离的计算不够精确，只精确到了整数。另外声明了多个全局变量，在大规模的团队合作中很容易造成全局变量污染以及代码不小心修改。但鉴于本次只是单个算法题比较简单，所以就直接使用全局变量方便在各个函数中使用。

本次实验的指导书，有一些细节性的东西没有说清楚。比如两个人有联系的人是直接使用和中间人的距离之和，还是使用两个人之间的位移等。此外，在指导书中可以提供一个测试样例，给出输入输出的形式。