《数据结构》课程实践报告

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 院、系 | 计算机学院 | | 年级专业 | 计算机类 | 姓名 |  | 学号 |  |
| 实验布置日期 | | 2019.12.12 | | 提交  日期 | 2019.12.27 | | 成绩 |  |

课程实践实验8：社交网络模型

## 问题描述及要求

设计并实现一个社交网络模型图。

要求：

（1）每个人的信息是一个顶点，两个人相互认识则构成边。

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

（3）可根据自己的创意添加更多的功能。

## 二、概要设计

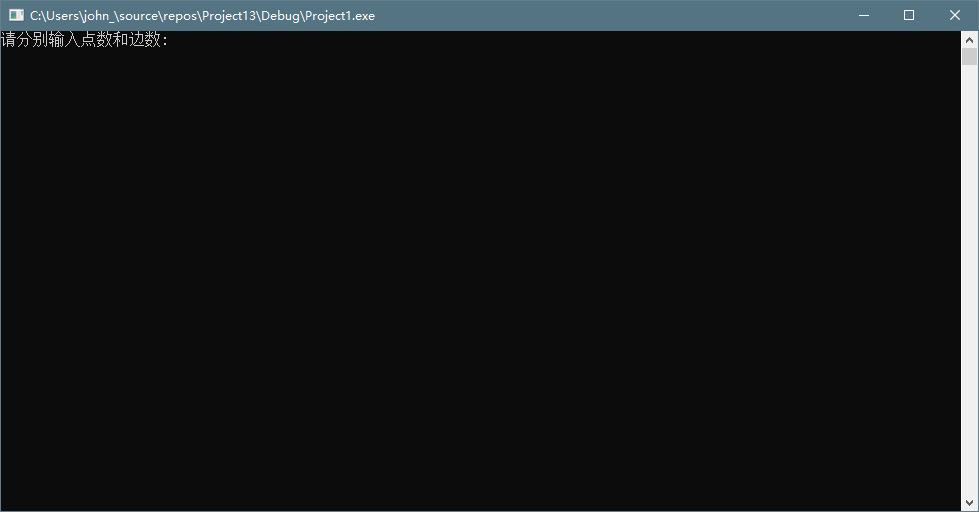
**（1）对实验内容的理解和二次概括。**

**该实验需要使用模板类，设计一个无向图的类，并且实现在图中添加点，边以及计算最短路径等算法的功能。**

**（2）给出系统的功能列表**

* **输入点和边的数量**
* **输入连接起来的点**

**（3）程序运行的界面设计**



**（4）程序结构设计，包括：对已有程序的使用，自己将设计哪些程序文件，各部分关系描述。**

将设计一个无向图类，利用链表和线性表的结构存放点和边。

## 三、详细设计

开始

输入需要相连的边

生成无向图

输入点和边的个数

通过边与点的关系，生成领接矩阵

终止

## 四、实验结果

紧扣题目要求设计提供相应的测试方法和结果。可以给出具体的测试用例，每个测试用例一般可列出：

* 测试输入：
* 
* 测试目的：检查程序是否正常运作
* 正确输出：
* 实际输出：
* 
* 测试结论：通过

## 五、实验分析与探讨

测试结果的分析：结果较为正常。

**实验设计、实现过程中遇到的问题：输入数据不合法时，系统会自动省略，而并非提示错误。**

## 六、小结

本次实验完成了无向图网络的构建，内容较为简易，通过本实验可以看出领接矩阵，从而得到各个节点之间的关系。

## 附录：源代码

#include <iostream>

#include <stack>

#include <list>

#include <queue>

#include <string>

using namespace std;

#define MAX\_VERTEX\_NUM 20

enum VisitIf { unvisited, visited };

VisitIf visit[MAX\_VERTEX\_NUM];

class EBox {

public:

VisitIf mark; //访问标记

int ivex, jvex; //该边依附的两个顶点位置

EBox \*ilink, \*jlink; //分别指向依附这两个顶点的下一条边

};

class VexBox {

public:

string data;

EBox \*firstedge; //指向第一条依附该顶点的边

};

class AMLGraph {

public:

VexBox adjmulist[MAX\_VERTEX\_NUM];

int vexnum, edgenum; //无向图的当前顶点数和边数

};

struct Node {

Node \*next;

Node \*front;

int data;

};

int LocateVex(VexBox adj[], int num, string v) {

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

if (adj[i].data == v) {

return i;

}

}

}

void CreateGraph(AMLGraph \*G) {

cout << "请输入顶点数和弧数: " << endl;

cin >> G->vexnum >> G->edgenum;

cout << "请输入顶点: " << endl;

for (int i = 0; i < G->vexnum; i++) {

cin >> G->adjmulist[i].data;

G->adjmulist[i].firstedge = NULL;

}

for (int i = 0; i < G->edgenum; i++) {

cout << "请输入边的两个顶点: " << endl;

string v1, v2;

cin >> v1 >> v2;

int i1 = LocateVex(G->adjmulist, G->vexnum, v1);

int i2 = LocateVex(G->adjmulist, G->vexnum, v2);

//cout << i1 << "...." << i2 << endl;

EBox \*p = new EBox;

p->mark = unvisited;

p->ivex = i1;

p->jvex = i2;

p->ilink = G->adjmulist[i1].firstedge;

p->jlink = G->adjmulist[i2].firstedge;

G->adjmulist[i1].firstedge = p;

G->adjmulist[i2].firstedge = p;

}

}

int FirstAdjVex(AMLGraph G, int v) {

int i = v;

if (G.adjmulist[i].firstedge) {

if (G.adjmulist[i].firstedge->ivex == i) {

return G.adjmulist[i].firstedge->jvex;

}

else {

return G.adjmulist[i].firstedge->ivex;

}

}

else

return -1;

}

//v是G中的某个顶点，w是v的邻接顶点

//返回v的(相对于w的)下一个邻接顶点。若w是v的最后一个邻接点，则返回空

int NextAdjVex(AMLGraph G, int v, int w) {

int i1 = v;

int i2 = w;

EBox \*p = G.adjmulist[i1].firstedge;

while (p) {

if (p->ivex == i1 && p->jvex != i2) {

p = p->ilink;

}

else {

if (p->ivex != i2 && p->jvex == i1) {

p = p->jlink;

}

else {

break;

}

}

}

if (p && p->ivex == i1 && p->jvex == i2) {

p = p->ilink;

if (p&&p->ivex == i1) {

return p->jvex;

}

else if (p&&p->jvex == i1)

return p->ivex;

}

if (p && p->ivex == i2 && p->jvex == i1) {

p = p->jlink;

if (p&&p->ivex == i1) {

return p->jvex;

}

else if (p&&p->jvex == i1)

return p->ivex;

}

return -1;

}

void BFSReverse(AMLGraph G) {

queue<int> a;

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

visit[i] = unvisited;

}

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

if (visit[i] == unvisited) {

cout << "连通图：" << endl;

a.push(i);

visit[i] = visited;

cout << G.adjmulist[i].data << endl;

while (!a.empty()) {

int v = a.front();

a.pop();

for (int w = FirstAdjVex(G, v); w >= 0; w = NextAdjVex(G, v, w)) {

if (visit[w] == unvisited) {

a.push(w);

cout << G.adjmulist[w].data << endl;

visit[w] = visited;

}

}

}

}

}

}

int BFSPath(AMLGraph G, string v1, string v2) {

if (v1 == v2) {

cout << "the same point!" << endl;

return 0;

}

int i1 = LocateVex(G.adjmulist, G.vexnum, v1);

int i2 = LocateVex(G.adjmulist, G.vexnum, v2);

//int num = 0;

Node \*head;

Node \*rear;

head = NULL;

rear = NULL;

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

visit[i] = unvisited;

}

visit[i1] = visited;

Node \*temp = new Node;

temp->data = i1;

temp->front = NULL;

temp->next = NULL;

head = rear = temp;

stack<string> a;

while (head) {

int v = head->data;

for (int w = FirstAdjVex(G, v); w >= 0; w = NextAdjVex(G, v, w)) {

if (visit[w] == unvisited) {

Node \*temp0 = new Node;

temp0->data = w;

temp0->front = head;

temp0->next = NULL;

rear->next = temp0;

rear = temp0;

visit[w] = visited;

if (rear->data == i2) {

Node \*p = rear;

while (p) {

a.push(G.adjmulist[p->data].data);

p = p->front;

}

cout << a.top();

a.pop();

while (!a.empty()) {

cout << "->" << a.top();

a.pop();

}

cout << endl;

return 1;

}

}

}

head = head->next;

}

cout << "Not Find!" << endl;

return 0;

}

void PrintGraph(AMLGraph G)

{

EBox \*p;

for (int i = 0; i < G.vexnum; ++i)

{

p = G.adjmulist[i].firstedge;

while (p != NULL)

{

if (p->ivex == i) //判断相等才能知道连接上的是ivex还是jvex;

{

cout << G.adjmulist[p->ivex].data << "------" << G.adjmulist[p->jvex].data << endl;

//printf("%d--%d\n", G.adjmulist[p->ivex].data, G.adjmulist[p->jvex].data);

p = p->ilink;

}

else

{

cout << G.adjmulist[p->jvex].data << "------" << G.adjmulist[p->ivex].data << endl;

p = p->jlink;

}

}

}

}

int main() {

AMLGraph g;

CreateGraph(&g);

//PrintGraph(g);

BFSReverse(g);

//DFSTraverse(g);

while (1) {

string v1, v2;

cout << "请输入你想查询的两点: " << endl;

cin >> v1 >> v2;

BFSPath(g, v1, v2);

}

return 0;

}