沈阳航空航天大学

**课 程 设 计 报 告**

课程设计名称：数据结构课程设计

课程设计题目：校园导航系统

**学 院：计算机学院**

**专 业：软件工程**

**班 级：软件1801**

**学 号：183401050120**

**姓 名：何雨泊**

**指导教师：刘香芹**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **沈阳航空航天大学** | | | | | | | | | | | | | | | | | | | | |
| **课程设计任务书** | | | | | | | | | | | | | | | | | | | | |
| **课程设计名称** | **数据结构课程设计** | | | | | | | | | | | **专业** | | | **软件工程** | | | | | |
| **学生姓名** | **何雨泊** | | | **班级** | | | **软件1801** | | | | | **学号** | | | **1834010501** | | | | | |
| **题目名称** | **校园导航系统** | | | | | | | | | | | | | | | | | | | |
| **起止日期** | **2019** | | **年** | **12** | **月** | | **9** | **日起至** | | | **2019** | | | **年** | | | **12** | **月** | **20** | **日止** |
| **课设内容和要求：** | | | | | | | | | | | | | | | | | | | | |
| 沈航的校区面积很大，对于新生来说入校对校园环境不熟悉，设计一校园导游系统，根据功能要求选择合适的存储结构，提供沈航校园办事处大致方位和路径查询等。  （1）设计校园平面图，包含沈航所有景点及办事处，以图中解点表示，结点属性为名称、简介等信息，图中的边表示结点间的道路，存放道路长度等信息。  （2）为来访客人提供图中任意景点相关信息的查询。  为来访客人提供图中任意景点的问路查询，即查询任意两个景点之间的路径（列出所有），并向访客推荐一条最短路径  （3）提供沈航校园电子地图的查看功能，该功能可在界面上显示沈航电子地图  **设计要求：**  1. 选择合适的数据结构； 2. 系统测试数据量应该达到一定规模；  3. 实现对程序性能的验证 | | | | | | | | | | | | | | | | | | | | |
| 参考资料：  [1] 严蔚敏,吴伟民.数据结构[M].北京:清华大学出版社,2012  [2] 吕国英.算法设计与分析[M].北京:清华大学出版社,2006  [3] 徐宝文 李志. C程序设计语言[M]. 北京: 机械工业出版社,2004  [4] Erich Gamma,Richard Helm．设计模式（英文版）[M].北京:机械工业出版社, 2004 | | | | | | | | | | | | | | | | | | | | |
| **课程负责人审核意见：□同意 □不同意 课程负责人签字（盖章）：** | | | | | | | | | | | | | | | | | | | | |
| **指导教师（签名）** | |  | | | |  | | |  | 年 | | |  | | | 月 | |  | 日 | |
| **学生（签名）** | |  | | | |  | | |  | 年 | | |  | | | 月 | |  | 日 | |

**目 录**

[1 题目介绍 1](#_Toc29018)

[1.1 题目介绍 1](#_Toc6352)

[1.2 功能要求 2](#_Toc6138)

[2 系统功能分析 2](#_Toc31754)

[2.1 总体功能分析 2](#_Toc9371)

[2.1.1实现功能的方式 3](#_Toc85)

[2.1.2实现功能的函数和类 3](#_Toc12756)

[3 数据结构设计 4](#_Toc5964)

[3.1 变量及数据类型 4](#_Toc1297)

[3.2 数据存储的数据结构 5](#_Toc13687)

[4 功能模块设计 6](#_Toc1875)

[4.1 主界面模块设计 6](#_Toc14344)

[4.2 导航模块算法设计 7](#_Toc1214)

[4.2.1 主菜单设计 7](#_Toc14344)

[4.2.1 导航模块算法设计 9](#_Toc1214)

[4.2.3 导航模块算法设计 1](#_Toc1214)4

[5 系统测试与运行结果 1](#_Toc19723)6

[5.1 调试及调试分析 1](#_Toc18442)6

[5.2 测试电子地图 2](#_Toc32201)1

[6 总结及建议 2](#_Toc16642)2

[参考文献 2](#_Toc32746)3

[附 录 （关键部分程序清单） 2](#_Toc18348)4

# 1 题目介绍

## 1.1 题目介绍

沈航校区面积很大，对于新生来说初入校对校园环境不熟悉，设计一校园导游系统，根据功能要求选择合适的存储结构，提供沈航校园办事处大致方位和路径查询等。

所谓导航系统，通常是一些基础功能的集合,包括:“定位”、“目的地选择”、“路径计算”和“路径指导”。系统在其较高的性能系列中也提供彩色地图显示。所有这些功能要求有一个道路网的数字化地图，它通常存储在CDROM中。而我在本次课设所要做的导航系统当然没有像百度地图、高德地图导航做的那么商业化，我的这次课设主要就是针对校园内各个景点之间最短路径算法的实现和应用，我也了解到了许多科学前沿的知识和我以前从未知道的东西，此次课设的题目让我更加深入理解了数据结构的知识和应用。

## 1.2 功能要求

（1）设计沈航校园平面图，包含沈航所有景点及办事处，以图中解点表示，结点属性为名称、简介等信息，图中的边表示结点间的道路，存放道路长度等信息。

（2）为来访客人提供图中任意景点相关信息的查询。

为来访客人提供图中任意景点的问路查询，即查询任意两个景点之间的路径（列出所有），并向访客推荐一条最短路径。

（3）提供沈航校园电子地图的查看功能，该功能可在界面上显示沈航电子地图。

（4）通过设计一个算法，能为来访的客人提供图中任意景点相关信息的查询和查看；

（5）实现图中任意两个节点之前最短路径的求法，并在界面上显示出来；

（6）Visio工具的使用来画流程图及思维导图。

# 2 系统功能分析

## 2.1 总体功能分析

**2.2.1 实现功能的方式**

1. 为了直观的表示导航系统导航的功能以及方便用户进行交互，应具备图形用户界面，可以选择使用QT框架来开发程序。
2. 为了便于用户更好的使用导航系统的功能，准确无误地到达目的地，应在系统中添加沈航校园电子地图的查看功能。
3. 要使系统界面显示的更加美观，可以在系统的主界面和各子界面上插入一些图片，如沈航的校徽、指南针的图片，增加了系统的观赏性。
4. 因为系统的数据过于庞大，控制台手动输入数据显得不太合适，因此把各景点之间的路径数据用邻接矩阵存在txt文件中，使用文件读取函数在每次系统启动前读取即可。
5. 在选择了导航的起点和终点以后，可以根据选择起点终点函数返回值判断是哪个景点，然后使用Dijkstra算法求出最短路径，最后显示在文本编辑框内。
6. 由于导航系统需要查询各个景点的基本信息，而这些信息都是中文段落，中文在程序中表示不合适很容易产生乱码，因此可以将这些景点的中文表述存在一个String数组中，数组中的每一个成员都存有一个景点的描述，在使用时直接索引就可以了。

**2.2.2 实现功能的函数和类**

（1）实现导航系统主界面类所需要的函数

MainWindow类的功能是创建主窗口，提供了用于构建应用程序用户界面的框架，该类中的构造函数是创建一个主窗口，即系统运行刚开始的登录界面，析构函数是关闭界面，下列表是该类中其他功能函数。

**表2.1 MainWindow类中函数**

|  |  |  |
| --- | --- | --- |
| **函数名称** | **函数原型** | **实现功能** |
| On\_cx\_currentIndexChanged | Void On\_cx\_currentIndexChanged(int index) | 查询下拉按钮选定信号的槽函数 |
| On\_dh\_begin\_activated | Void On\_dh\_begin\_activated(int index) | 选定导航的起点 |
| On\_dh\_end\_activated | Void On\_dh\_end\_activated(int index) | 选定导航的终点 |
| On\_pu\_1\_clicked() | Void On\_pu\_1\_clicked() | 显示主界面 |
| On\_pu\_2\_clicked() | Void On\_pu\_2\_clicked() | 显示景点查询界面 |
| On\_pu\_3\_clicked() | Void On\_pu\_3\_clicked() | 显示导航界面 |
| On\_pu\_4\_clicked() | Void On\_pu\_4\_clicked() | 打开沈航电子地图 |
| On\_pu\_5\_clicked() | Void On\_pu\_5\_clicked() | 退出导航系统 |

（2） 实现导航功能类所需函数

类Graph\_DG内包含有系统导航的各个函数，实现查询，导航等功能。其中本课设主要的最短路径算法Dijkstra由Dijkstra函数完成，dfs\_all\_line函数的功能是搜索两个景点之间的所有路径，print\_path函数的功能是在文本编辑框内显示两个景点的最短路径。下列表是该类中具体功能函数。

**表2.2 Graph\_DG类中函数**

|  |  |  |
| --- | --- | --- |
| **函数名称** | **函数原型** | **函数功能** |
| print | void print() | 打印邻接矩阵 |
| Dijkstra | void Dijkstra(int begin) | 求出最短路径 |
| dfs\_all\_line | void dfs\_all\_line(int start,int end) | 求出所有路径 |
| print\_path | void print\_path(int end) | 显示最短路径 |

# 3 数据结构设计

## 3.1 变量及数据类型

1. 预定义常量和自定义类型：

#define INT\_MAX=65535。

(2)基本函数的算法用以下形式表示：函数类型 函数名（函数参数表）

//算法说明

{语句序列} //函数名。

(3) 定义QString line[100]; int visited[100],begin; stack<int> sta; struct Dis；class Graph\_DG。

line记录三十个顶点的名称；visited代表顶点是否被访问，sta记录深度遍历过程中所经过的顶点；结构体Dis存储图中各顶点的信息；类Graph\_DG来表示图。

1. Dis中定义的queue<int> path，int value, bool visit；

path存储Dijkstra算法过程后到该节点的最短路径；visit代表该节点是否被访问过；value代表该点的权重。

1. Dijkstra算法过程中定义int count，代表访问的节点个数。
2. 显示最短路径算法过程中定义QString str，int i； str表示最短的路径，用来显示在Textedit文本编辑器上；i代表遍历邻接矩阵过程中节点的数目。
3. 读取文件函数中定义int vexnum,int edge,FILE\*file; vexnum代表图中节点的个数，为30个；edge代表图中各节点两两之间的距离，共30\*30个；file代表读取文件指针，指向包含有邻接矩阵的文件。

**3.2 数据存储的数据结构**

在本次课程设计中因为景点各连接路径数据的庞大以及复杂性，使用节省空间牺牲时间的邻接表存储方法在质量上不如使用牺牲空间节省时间的邻接矩阵存储方法，因此我使用了30\*30的邻接矩阵存储了沈航30个景点两两之间的距离，并通过文件读取。

邻接矩阵的定义：

邻接[矩阵](http://wapbaike.baidu.com/item/%E7%9F%A9%E9%98%B5)（Adjacency Matrix）是表示顶点之间相邻关系的矩阵。设G=(V,E)是一个图，其中V={v1,v2,…,vn}[1] 。G的邻接矩阵是一个具有下列性质的n阶方阵：

①对[无向图](http://wapbaike.baidu.com/item/%E6%97%A0%E5%90%91%E5%9B%BE)而言，邻接矩阵一定是对称的，而且主对角线一定为零（在此仅讨论无向简单图），副对角线不一定为0，[有向图](http://wapbaike.baidu.com/item/%E6%9C%89%E5%90%91%E5%9B%BE)则不一定如此。

②在无向图中，任一顶点i的度为第i列（或第i行）所有非零元素的个数，在有向图中顶点i的出度为第i行所有非零元素的个数，而入度为第i列所有非零元素的个数。

③用邻接矩阵法表示图共需要n^2个空间，由于无向图的邻接矩阵一定具有[对称关系](http://wapbaike.baidu.com/item/%E5%AF%B9%E7%A7%B0%E5%85%B3%E7%B3%BB)，所以扣除对角线为零外，仅需要存储上三角形或下三角形的数据即可，仅需要n（n-1）/2个空间。

因此使用邻接矩阵存储和读取数据非常方便！



**图3.1 邻接矩阵存储图**

# 4 功能模块设计

## 4.1 主界面模块设计

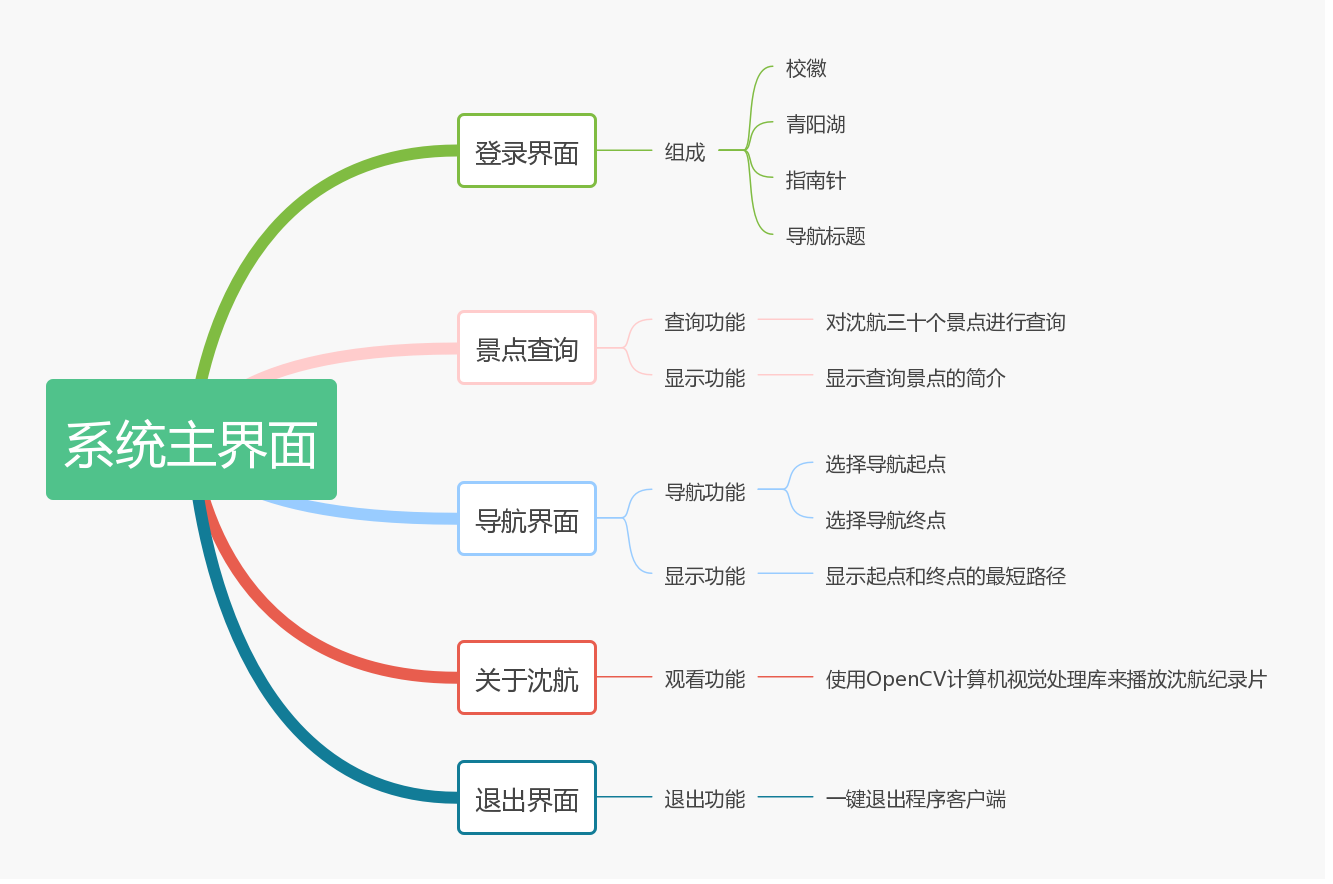
为了实现导航相应功能，首先设计一个包含多个功能按钮的主菜单窗口，然后再为这些按钮配上相应的功能。

程序运行后，菜单的内容如下：



**图4.1 登录界面****示意图**

1. 主登录界面由校徽、青阳湖、指南针、导航标题等图案组成。
2. 景点查询功能界面由查询窗口和显示窗口构成，查询窗口包含一个下拉框，可以选择沈航的三十个景点进行查询；显示窗口由一个初始化为空白的文本编辑框构成，可以显示查询的景点的信息。
3. 导航界面由导航选择窗口和路径显示窗口构成，导航选择窗口由选择导航起点和选择导航终点两个下拉框组成；路径显示窗口也由一个初始化为空白的文本编辑框构成，显示两个景点之间的最短路径。
4. 地图显示界面可以显示沈航的电子地图。
5. 退出键可以一键退出该导航系统。



**图4.2主界面思维导图**

## 4.2 导航模块算法设计

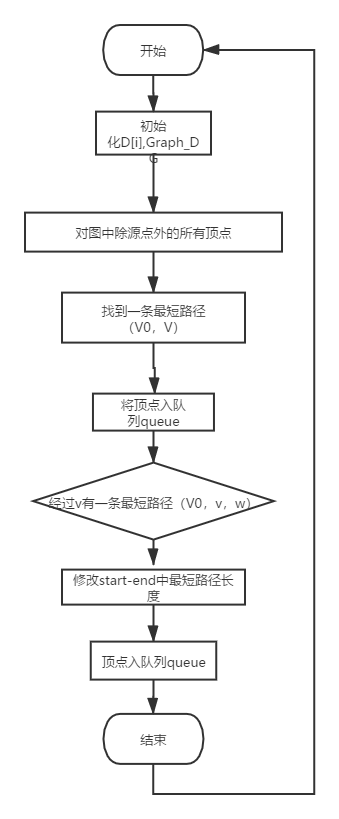
**4.2.1 最短路径（Dijkstra）**

Dijkstra算法采用的是一种贪心的策略，声明一个数组dis来保存源点到各个顶点的最短距离和一个保存已经找到了最短路径的顶点的集合：T，初始时，原点 s 的路径权重被赋为 0 （dis[s] = 0）。若对于顶点 s 存在能直接到达的边（s,m），则把dis[m]设为w（s, m）,同时把所有其他（s不能直接到达的）顶点的路径长度设为无穷大。初始时，集合T只有顶点s。

然后，从dis数组选择最小值，则该值就是源点s到该值对应的顶点的最短路径，并且把该点加入到T中，OK，此时完成一个顶点，

然后，我们需要看看新加入的顶点是否可以到达其他顶点并且看看通过该顶点到达其他点的路径长度是否比源点直接到达短，如果是，那么就替换这些顶点在dis中的值。

然后，又从dis中找出最小值，重复上述动作，直到T中包含了图的所有顶点。



**图4.3 Dijkstra算法流程图**

具体算法：

void Graph\_DG::Dijkstra(int begin) {

//首先初始化我们的dis数组

int i;

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

//设置当前的路径

//dis[i].path = "v" + QString(begin) + "-->v" + QString(i + 1);

dis[i].path.push(i+1);

dis[i].value = arc[begin - 1][i];

}

//设置起点的到起点的路径为0

dis[begin - 1].value = 0;

dis[begin - 1].visit = true;

int count = 1;

//计算剩余的顶点的最短路径（剩余this->vexnum-1个顶点）

while (count != this->vexnum) {

//temp用于保存当前dis数组中最小的那个下标

//min记录的当前的最小值

int temp = 0;

int min = INT\_MAX;

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

if (!dis[i].visit && dis[i].value < min) {

min = dis[i].value;

temp = i;

}

}

//cout << temp + 1 << " "<<min << endl;

//把temp对应的顶点加入到已经找到的最短路径的集合中

dis[temp].visit = true;

++count;

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

//注意这里的条件arc[temp][i]!=INT\_MAX必须加，不然会出现溢出，从而造成程序异常

if (!dis[i].visit && arc[temp][i] != INT\_MAX && (dis[temp].value + arc[temp][i]) < dis[i].value) {

//如果新得到的边可以影响其他为访问的顶点，那就就更新它的最短路径和长度

dis[i].value = dis[temp].value + arc[temp][i];

queue<int> q=dis[temp].path;

q.push(i+1);

dis[i].path=q;

}

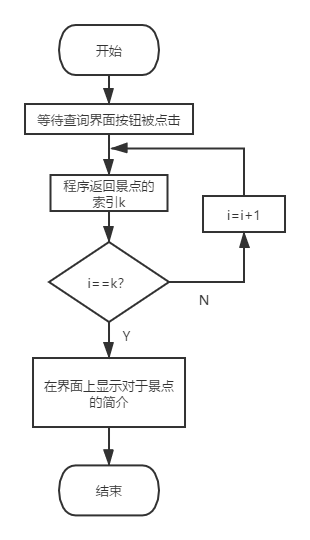
}

}

}

**4.2.2 地点查找（Find）**

这次查找其实函数其实很简单，界面在下拉框选中某个景点时，根据信号与槽机制，程序会返回一个对于景点的编号，接着我们的查找函数对应于编号输出显示。



**图4.4 地点查找函数流程图**

详细算法：

void MainWindow::on\_cx\_currentIndexChanged(int index)

{

//将当前索引赋值给变量index

index = ui->cx->currentIndex();

qDebug()<<"Index"<< index;

if(index==0)

{

ui->cx\_edit->setText("位于学校正门，是学校部分机关部门，直属附属单位办公场所，涉及到学生事务的部门有教务处，计财处等");

}

else if(index==1)

{

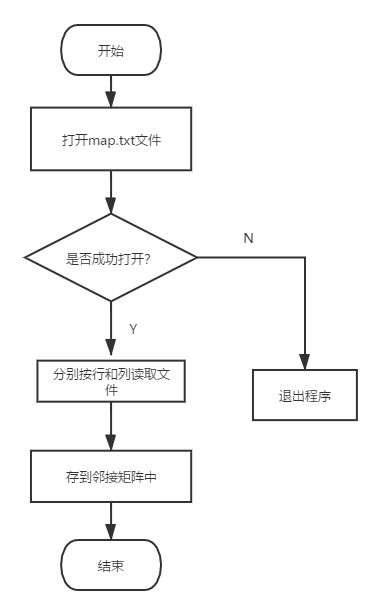
ui->cx\_edit->setText("位于行政楼北侧，经济与管理学院，安全工程学院，马克思主义学院在此办公");

}

}

**4.2.3 文件读取（Read）**

文件读取使用fopen函数，对map.txt文件进行按行和列的读取



**图4.5文件读取函数流程图**

具体代码：

Graph\_DG read\_file()

{

int vexnum=30; //顶点数

int edge=900; //边数

FILE \*file;

if ((file = fopen("C://Users/hyb/Desktop/Work/map.txt", "r+")) == NULL)

{

cout<<"123\n";

exit(0);

}

Graph\_DG Graph(vexnum, edge);

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

for(int j=0;j<vexnum;j++)

{

if(j==29)

fscanf(file,"%d\n",&Graph.arc[i][j]);

else

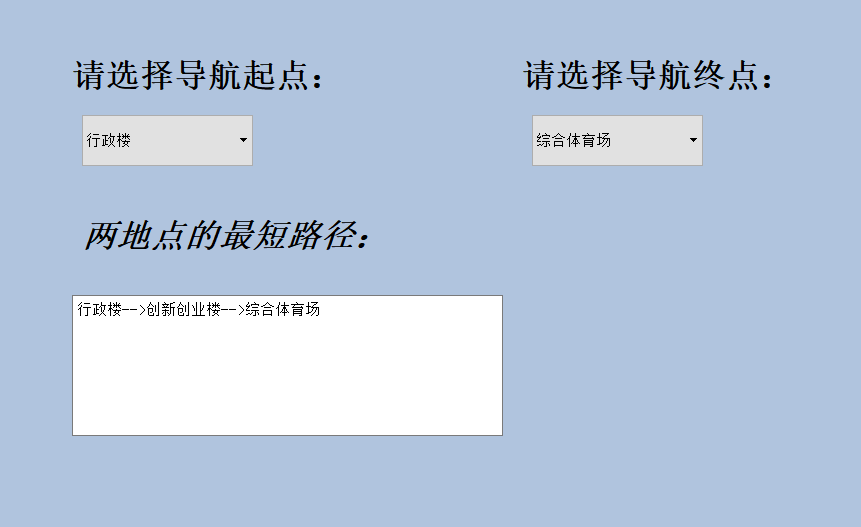
fscanf(file,"%d ",&Graph.arc[i][j]);

}

return Graph;

}

导航函数使用两个下拉框作为导航搜索起点和终点的来源，当点击起点下拉框时，得到一个起点的int类型begin变量，当点击终点下拉框时，得到一个终点的int类型的end变量，然后执行Dijkstra算法进行导航，得到的路径显示在界面上。



**图4.6 导航功能示意图**

# 5 系统测试与运行结果

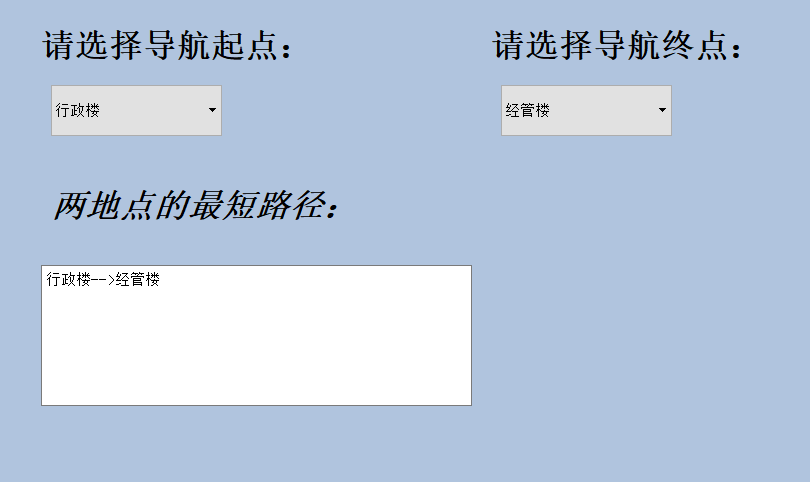
## 5.1 调试及测试分析

程序调试是将编制的程序投入实际运行前，用手工或编译程序等方法进行测试，修正语法错误和逻辑错误的过程。这是保证计算机信息系统正确性的必不可少的步骤。编完计算机程序，必须送入计算机中测试。根据测试时所发现的错误，进一步诊断，找出原因和具体的位置进行修正。

下面是我随机找的十组测试数据对本次课设程序的测试。

1. 设置导航起点为行政楼，导航终点为经管楼。

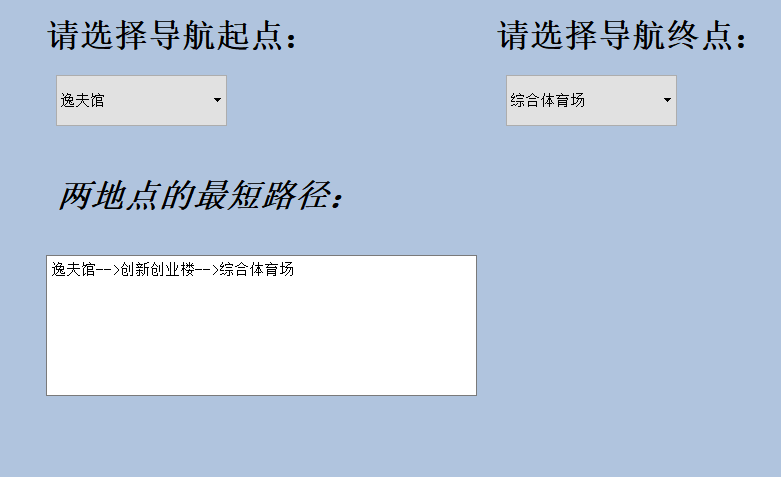
运行的结果如下图所示。



**图5.1 第一组测试数据示意图**

1. 导航起点为逸夫馆，导航终点为综合体育场。

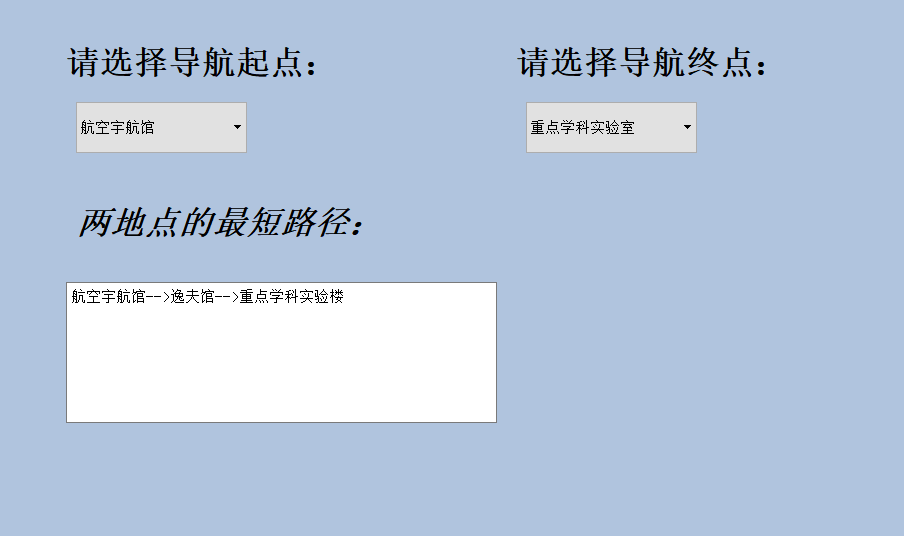
运行的结果如下图所示。



**图5.2 第二组测试数据示意图**

（3）导航起点为航空宇航馆，导航终点为重点学科实验室

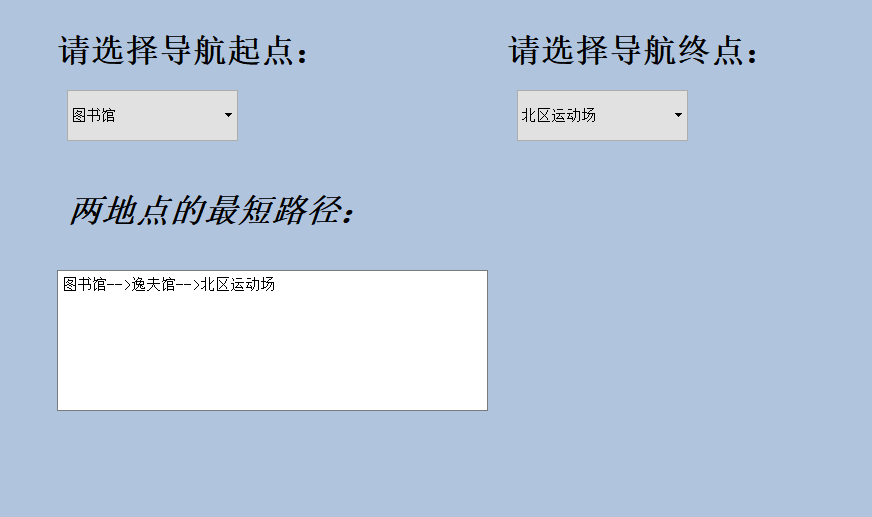
运行的结果如下图所示。



**图5.3第三组测试数据示意图**

（4）导航起点为图书馆，导航终点为北区运动场

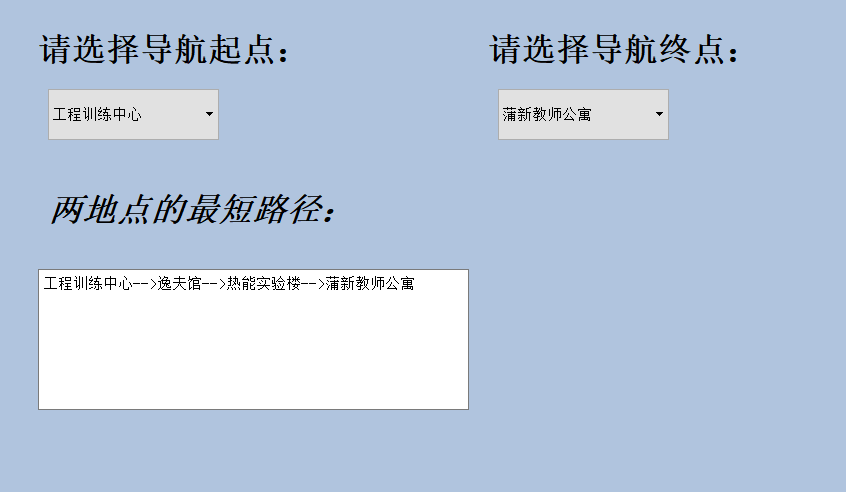
运行的结果如下图所示。



**图5.4 第四组测试数据示意图**

（5）导航起点为工程训练中心，导航终点为蒲新教师公寓

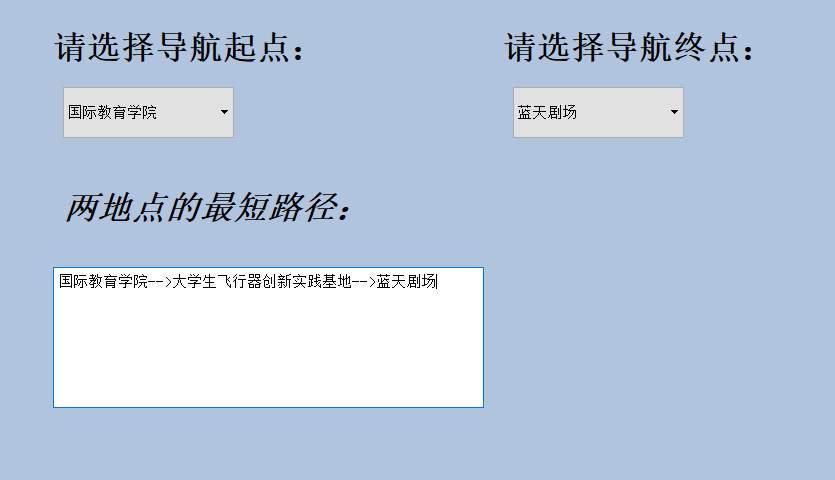
运行的结果如下图所示。



**图5.5第五组测试数据示意图**

（6）导航起点为国际教育学院，导航终点为蓝天剧场

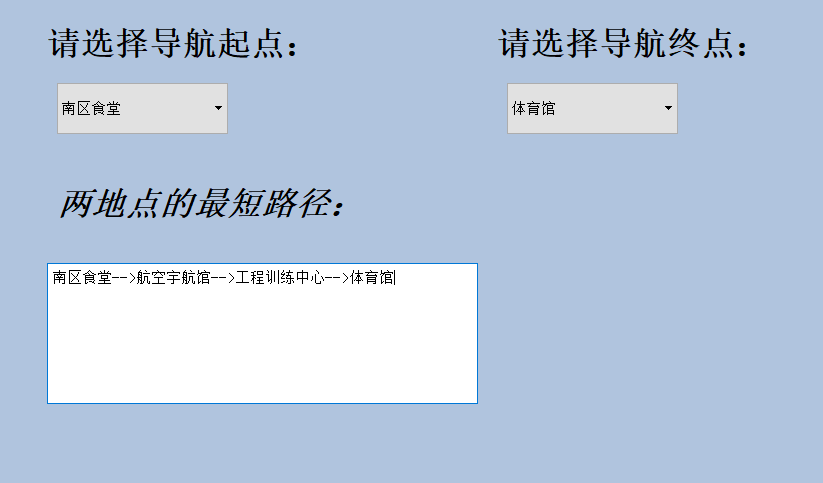
运行的结果如下图所示。



**图5.6第六组测试数据示意图**

（7）导航起点为南区食堂，导航终点为体育馆

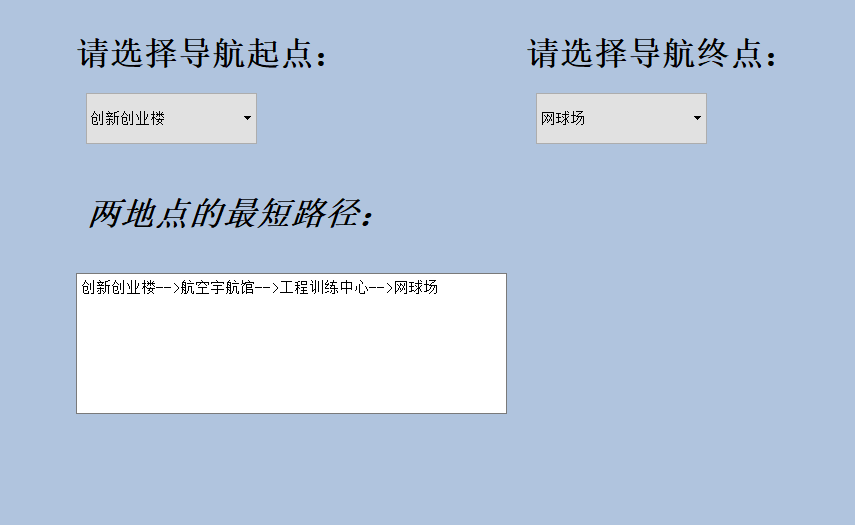
运行的结果如下图所示。



**图5.7第七组测试数据示意图**

（8）导航起点为创新创业楼，导航终点为网球场

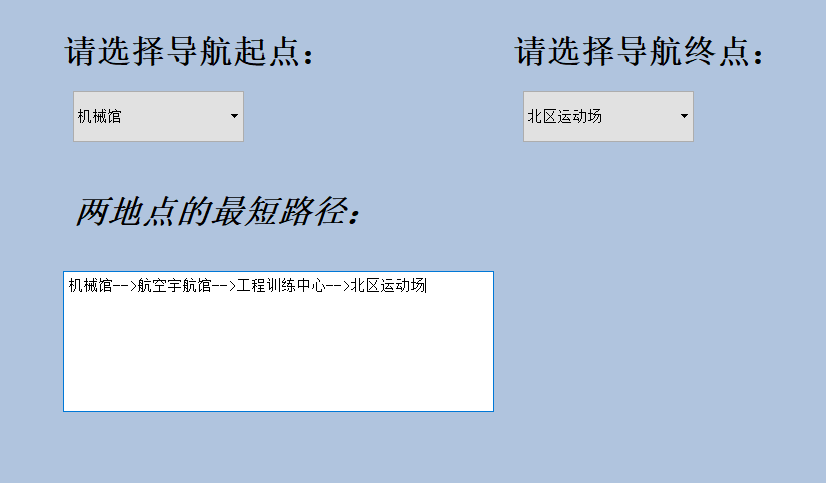
运行的结果如下图所示。



**图5.8第八组测试数据示意图**

（9）导航起点为机械馆，导航终点为北区运动场

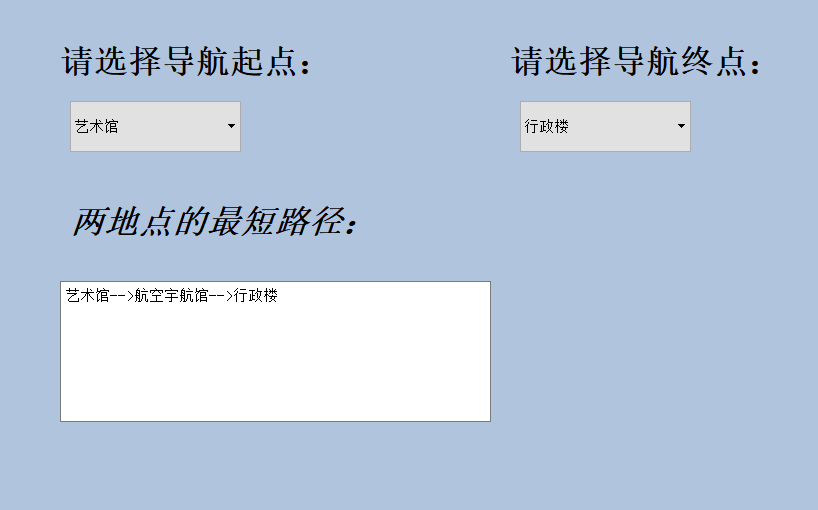
运行的结果如下图所示。



**图5.9第九组测试数据示意图**

1. 导航起点为艺术馆，导航终点为行政楼

运行的结果如下图所示。



**图5.10第十组测试数据示意图**

**5.2测试电子地图**

既然设计了导航系统，自然也少不了地图的使用，系统中也有沈航的电子地图，出处是沈阳航空航天官网。

 **5.11 沈航校园电子地图**6 总结及建议

首先在本次课设中非常感谢刘香芹老师细心指导和严格要求！由衷感谢老师提供给我们这样一个锻炼自己的机会，让我们第一次感受到学来的知识不只是用来完成试卷的。课程设计使我们发现考试真的并不是最重要，而是学会一种快速有效的学习方法。

总结地说，我的这次校园导航系统数据结构课程设计是一门系统化的设计，它将我们在课堂上所学到的数据结构知识灵活地运用到了实际问题的应用中去了。校园导航系统，实际上就是使用一个导航算法，然后使用一些可视化程序将导航数据可视化，将导航的结果直观地展现给我们。当然，不仅观赏性需要做得好，导航算法的要求才是至关重要的，本次导航的主要算法是最短路径，我搜集到最短路径的算法有Dijkstra算法、Bellmanford算法、Floyd算法、Johnson算法。在同时兼顾数据量要求和算法性能下，我选择了Dijkstra算法，可以解决带有非负权值的图中的单源最短路径问题。

经过这次课设的锻炼，使我更加理解数据结构这门课程。在设计综合校园导航算法时，我不仅懂得了许多算法设计思想，而且更加了解了循环的使用。过程很痛苦，好在结果还算顺心，学到了好多可视化界面设计的知识还有QT中信号与槽机制的特性。在设计过程中我还学会了最短路径（Dijkstra）算法，更加深入的了解了广度优先搜索在解决实际应用过程中的过程。而且我和平常写程序输入也不一样，我采用了文件读取的方法，因为我的输入是一个30\*30的邻接矩阵，手动输入基本不可能，做出来其实还是蛮有成就感的！并且在这次课设的过程中，我自行定义了好多数据类型和函数，不断的改进算法，为了就是使程序达到正确性、可读性、健壮性的结果。

最后我想说一下我的建议：

1. 我们应把被动学习转变为主动学习，不在是用学到的知识解题，而是在实际运用时遇到什么学什么，重在把知识应用于实际。

（2）这门课程设计能很好的培养大家的创新意识，是一种很好的课设改革方式，我们希望可选的题目增多一些，以后继续开展下去。

# 参考文献

[1] 严蔚敏,吴伟民.数据结构[M].北京:清华大学出版社,2012

[2] 吕国英.算法设计与分析[M].北京:清华大学出版社,2006

[3] 徐宝文 李志. C程序设计语言[M]. 北京: 机械工业出版社,2004

[4] Erich Gamma,Richard Helm 设计模式（英文版）[M].北京:机械工业出版社,2004

[5] 霍亚飞.Qt Creator快速入门[M]. 北京：北京航空航天大学出版社,2012

[6] (美)Alan Ezust.C++设计模式--基于Qt4开源跨平台开发框架[M]. 北京: 机械工业出版社,2004

[7] 霍亚飞.Qt及Qt Quick开发实战精解[M]. 北京：北京航空航天大学出版社,2009

[8] 程杰.大话数据结构[M]. 北京：清华大学出版社,2011

[9] (美)克林伯格. 算法设计[M]. 北京：清华大学出版社,2007

# 附 录 （关键部分程序清单）

**Mainwindow.cpp程序**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <iostream>

#include <QDebug>

#include <QString>

#include <string>

#include <QTextCodec>

#include <QCoreApplication>

#include <queue>

#include <stack>

using namespace std;

QString line[100]={"行政楼","经管楼","逸夫馆","工程训练中心","航空宇航馆","综合教学楼","图书馆","机械馆","艺术馆","体育馆","综合体育场","大学生活动中心","蓝天剧场","大学生飞行器创新实践基地","创新创业楼","国际教育学院","南区体育场","南区足球场","南区食堂","友谊会馆","南区篮球场","校医院","南区学生公寓","网球场","北区食堂","北区学生公寓","北区运动场","通航实验楼","重点学科实验楼","热能实验楼","蒲新教师公寓"};

int visited[100];

int begin0;

stack<int> sta;

MainWindow::MainWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

QPalette palette(this->palette());

palette.setColor(QPalette::Background, QColor(176, 196, 222));

this->setPalette(palette);

this->setStyleSheet("background-color:blue;");

ui->setupUi(this);

setWindowTitle("校园导航系统");

ui->min\_line->hide();

ui->all\_line->hide();

ui->cx->hide();

ui->cx\_edit->hide();

ui->cx\_label->hide();

ui->dh\_begin->hide();

ui->dh\_end->hide();

ui->cx\_label->hide();

ui->cx\_begin\_na->hide();

ui->cx\_end\_na->hide();

ui->cx\_label->hide();

ui->all\_na->hide();

ui->min\_na->hide();

connect(this->ui->pu\_1,SIGNAL(click),this,SLOT(on\_pu\_1\_clicked));

connect(this->ui->pu\_2,SIGNAL(click),this,SLOT(on\_pu\_2\_clicked));

connect(this->ui->pu\_3,SIGNAL(click),this,SLOT(on\_pu\_3\_clicked));

connect(this->ui->pu\_4,SIGNAL(click),this,SLOT(on\_pu\_4\_clicked));

connect(this->ui->pu\_5,SIGNAL(click),this,SLOT(on\_pu\_5\_clicked));

}

/\*

本程序是使用Dijkstra算法实现求解最短路径的问题

采用的邻接矩阵来存储图

\*/

//记录起点到每个顶点的最短路径的信息

struct Dis {

queue<int> path;

int value;

bool visit;

Dis() {

visit = false;

value = 0;

//path = "";

}

};

class Graph\_DG {

public:

int vexnum; //图的顶点个数

int edge; //图的边数

int \*\*arc; //邻接矩阵

Dis \* dis; //记录各个顶点最短路径的信息

public:

//构造函数

Graph\_DG(int vexnum, int edge);

//析构函数

~Graph\_DG();

// 判断我们每次输入的的边的信息是否合法

//顶点从1开始编号

bool check\_edge\_value(int start, int end, int weight);

//打印邻接矩阵

void print();

//求最短路径

void Dijkstra(int begin);

//求所有路径

void dfs\_all\_line(int start,int end);

//打印最短路径

void print\_path(int end);

};

//构造函数

Graph\_DG::Graph\_DG(int vexnum, int edge) {

//初始化顶点数和边数

this->vexnum = vexnum;

this->edge = edge;

//为邻接矩阵开辟空间和赋初值

arc = new int\*[this->vexnum];

dis = new Dis[this->vexnum];

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

arc[i] = new int[this->vexnum];

for (int k = 0; k < this->vexnum; k++) {

//邻接矩阵初始化为无穷大

arc[i][k] = INT\_MAX;

}

}

}

//析构函数

Graph\_DG::~Graph\_DG() {

delete[] dis;

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

delete this->arc[i];

}

delete arc;

}

Graph\_DG read\_file()

{

int vexnum=30; //顶点数

int edge=900; //边数

FILE \*file;

if ((file = fopen("C://Users/hyb/Desktop/Work/map.txt", "r+")) == NULL)

{

cout<<"123\n";

exit(0);

}

Graph\_DG Graph(vexnum, edge);

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

for(int j=0;j<vexnum;j++)

{

if(j==29)

fscanf(file,"%d\n",&Graph.arc[i][j]);

else

fscanf(file,"%d ",&Graph.arc[i][j]);

}

return Graph;

}

Graph\_DG G=read\_file();

// 判断我们每次输入的的边的信息是否合法

//顶点从1开始编号

bool Graph\_DG::check\_edge\_value(int start, int end, int weight) {

if (start<1 || end<1 || start>vexnum || end>vexnum || weight < 0) {

return false;

}

return true;

}

void Graph\_DG::print() {

//cout << "图的邻接矩阵为：" << endl;

int count\_row = 0; //打印行的标签

int count\_col = 0; //打印列的标签

//开始打印

while (count\_row != this->vexnum) {

count\_col = 0;

while (count\_col != this->vexnum) {

if (arc[count\_row][count\_col] == INT\_MAX)

cout << "∞" << " ";

else

cout << arc[count\_row][count\_col] << " ";

++count\_col;

}

cout << endl;

++count\_row;

}

}

void Graph\_DG::Dijkstra(int begin) {

//首先初始化我们的dis数组

int i;

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

//设置当前的路径

//dis[i].path = "v" + QString(begin) + "-->v" + QString(i + 1);

dis[i].path.push(i+1);

dis[i].value = arc[begin - 1][i];

}

//设置起点的到起点的路径为0

dis[begin - 1].value = 0;

dis[begin - 1].visit = true;

int count = 1;

//计算剩余的顶点的最短路径（剩余this->vexnum-1个顶点）

while (count != this->vexnum) {

//temp用于保存当前dis数组中最小的那个下标

//min记录的当前的最小值

int temp = 0;

int min = INT\_MAX;

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

if (!dis[i].visit && dis[i].value < min) {

min = dis[i].value;

temp = i;

}

}

//cout << temp + 1 << " "<<min << endl;

//把temp对应的顶点加入到已经找到的最短路径的集合中

dis[temp].visit = true;

++count;

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

//注意这里的条件arc[temp][i]!=INT\_MAX必须加，不然会出现溢出，从而造成程序异常

if (!dis[i].visit && arc[temp][i] != INT\_MAX && (dis[temp].value + arc[temp][i]) < dis[i].value) {

//如果新得到的边可以影响其他为访问的顶点，那就就更新它的最短路径和长度

dis[i].value = dis[temp].value + arc[temp][i];

queue<int> q=dis[temp].path;

q.push(i+1);

dis[i].path=q;

}

}

}

}

void Graph\_DG::print\_path(int end) {

QString str;

str = "v" + QString(end);

int i=0;

while(i!=this->vexnum)

{

if(i==end)

{

QString min\_line;

while(!dis[i].path.empty())

{

qDebug()<<dis[i].path.front();

min\_line+=line[dis[i].path.front()-1];

dis[i].path.pop();

}

qDebug()<<min\_line;

break;

}

i++;

}

}

//检验输入边数和顶点数的值是否有效，可以自己推算为啥：

//顶点数和边数的关系是：((Vexnum\*(Vexnum - 1)) / 2) < edge

bool check(int Vexnum, int edge) {

if (Vexnum <= 0 || edge <= 0 || ((Vexnum\*(Vexnum - 1)) / 2) < edge)

return false;

return true;

}

void init(bool \*visit)

{

for(int i=0;i<100;i++)

visit[i]=false;

}

QString all;

void Graph\_DG::dfs\_all\_line(int start,int end)//深搜入栈查询所有路径

{

visited[start] = 1;//visited数组存储各定点的遍历情况，true为已遍历（标记）

sta.push(start);//入栈

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

{

if (start== end) {//找到终点

stack<int> s=sta;

while(!s.empty())

{

qDebug()<<s.top();

s.pop();

}

sta.pop();//出栈

visited[start] = false;

break;

}

if (!visited[j]) {//该顶点未被访问过

dfs\_all\_line(j,end);

}

if (j == G.vexnum - 1 ) {//如果该顶点无其它出度

sta.pop();

visited[start] = 0;

}

}

}

MainWindow::~MainWindow()

{

delete ui;

}

//主界面

void MainWindow::on\_pu\_1\_clicked()

{

ui->main\_pic->show();

ui->min\_line->hide();

ui->all\_line->hide();

ui->cx->hide();

ui->cx\_edit->hide();

ui->cx\_label->hide();

ui->dh\_begin->hide();

ui->dh\_end->hide();

ui->cx\_label->hide();

ui->cx\_begin\_na->hide();

ui->cx\_end\_na->hide();

ui->cx\_label->hide();

ui->all\_na->hide();

ui->min\_na->hide();

}

//导航界面

void MainWindow::on\_pu\_3\_clicked()

{

ui->main\_pic->hide();

ui->cx\_begin\_na->show();

ui->cx\_end\_na->show();

ui->dh\_begin->show();

ui->dh\_end->show();

//ui->all\_line->show();

//ui->all\_na->show();

ui->min\_line->show();

ui->min\_na->show();

//查询界面关掉

ui->cx->hide();

ui->cx\_label->hide();

ui->cx\_edit->hide();

}

//查询界面

void MainWindow::on\_pu\_2\_clicked()

{

ui->cx->show();

ui->cx\_label->show();

ui->cx\_edit->show();

//导航的界面关掉

ui->main\_pic->hide();

ui->cx\_begin\_na->hide();

ui->cx\_end\_na->hide();

ui->dh\_begin->hide();

ui->dh\_end->hide();

ui->all\_line->hide();

ui->all\_na->hide();

ui->min\_line->hide();

ui->min\_na->hide();

}

//关于

void MainWindow::on\_pu\_4\_clicked()

{

}

//退出键

void MainWindow::on\_pu\_5\_clicked()

{

this->close();

}

//导航起点下拉按钮

void MainWindow::on\_dh\_begin\_activated(int begin)

{

//将当前索引赋值给变量index

begin = ui->dh\_begin->currentIndex();

begin0=begin;

G.Dijkstra(begin+1);

qDebug()<<"begin"<< begin;

//ui->all\_line->setText(a);

}

//导航终点下拉按钮

void MainWindow::on\_dh\_end\_activated(int end)

{

//将当前索引赋值给变量index

end = ui->dh\_end->currentIndex();

//最短路径

int i=0;

QString min\_line=line[begin0]+"-->";

while(i!=G.vexnum)

{

if(i==end)

{

while(!G.dis[i].path.empty())

{

qDebug()<<G.dis[i].path.front();

min\_line+=line[G.dis[i].path.front()-1];

G.dis[i].path.pop();

if(!G.dis[i].path.empty())

min\_line+="-->";

}

ui->min\_line->setText(min\_line);

break;

}

i++;

}

//所有路径

int num=0;

//G.dfs\_all\_line(begin0,end);

//ui->all\_line->setText(all);

qDebug()<<"end"<< end;

}

**main.cpp程序**

#include "mainwindow.h"

#include <QApplication>

#include <stdio.h>

#include <stdlib.h>

#include <iostream>

#include <algorithm>

#include <QDebug>

using namespace std;

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}