**校园导游系统报告**

**山东大学软件学院 2017级网络空间安全实验班**

**数据结构课程设计实验报告**

**王泉成**

**2019**

**目录**

**Content**

* 问题描述 ······························ 2
* 设计目标 ······························ 2
* 需求分析 ······························ 2
* Qt以及其他GUI实现介绍 ······················· 4
* 算法介绍 ······························ 5
* 流程分析 ······························ 7
* 概要设计 ······························ 7
* 界面设计 ······························ 9
* 详细设计 ······························ 13
* 软件编译 ······························ 19
* 软件测试 ······························ 20
* 使用说明 ······························
* 工作总结 ······························

**正文**

1. **问题描述**

* 随着现代旅游业的快速发展，图文声像导游方式和实地口语导游方式都已经不能满足现阶段旅游者的需求，信息化的飞速发展造就了地理信息系统(GIS)和全球定位系统(GPS)，促使消费者更多的选择自助游和自驾游等方式出行。而近年来高等院校的发展使得高校也成为了一个景点。如何让游客以最短的时间到达旅游目的地就是我的课程设计所寻求解决的问题。我打算通过最短路径算法结合图像化的表示方法，并结合实际情况以高等院校为例采集所需要的数据，在开发环境下模拟设计**导游咨询系统**，理论上使得游客可以轻松的寻找到最适合自己的旅游线路，并以此为依据合理安排自己的行程。
* 用**无向网**表示校园景点平面图，图中顶点表示主要景点，存放景点的编号、名称、简介等信息，图中的边表示景点间的道路，存放路径长度等信息。要求能够回答有关景点介绍、游览路径等问题。

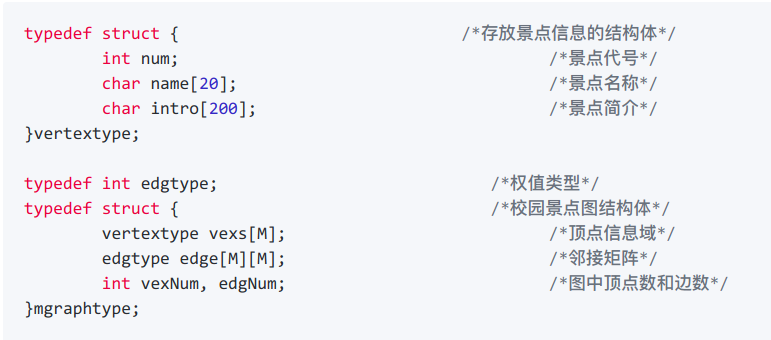
1. **设计目标**

* 查询任意景点的相关信息。
* 查询图中任意两个景点间的最短路径。
* 查询图中任意两个景点间的所有路径。
* 增加、删除、更新有关景点和道路的信息。
* 求多个景点的最佳（最短）游览路径。
* 实现通过图形用户界面进行人机交互。
* 实现C++语言对于文件的读写操作。
* 实现其他额外的附加功能。

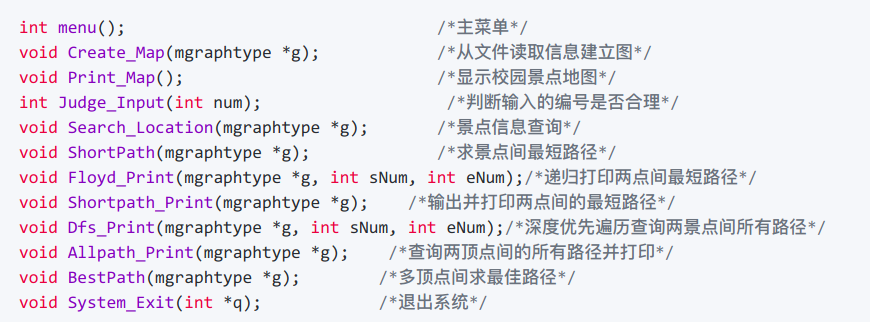
1. **需求分析**

* **所需知识**
* 图的各种遍历算法
* 单源最短路径(Dijkstra算法)
* 所有顶点对的最短路径(Floyd算法)
* 图的基本存储结构(邻接矩阵)
* 文件的读取存储操作
* 界面交互
* **数据结构**

本课题用到的数据结构是图的数据结构，其中用到的是图的邻接矩阵存储结构。



* **基本算法实现**



* **系统开发平台要求**
* 操作系统：Manjaro Linux 18
* 集成开发环境：Visual Studio Code
* 编程语言：C++
* 编译器：gcc与g++
* 其他软件：Qt Designer
* **附加可选功能**
* 登录系统
* 网络服务器

1. **Qt以及其他GUI实现介绍**



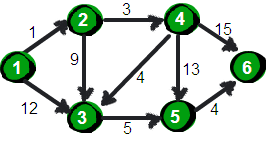
* **Qt说明**
* Qt 是一个著名的 C++ 应用程序框架。你并不能说它只是一个 GUI 库，因为 Qt 十分庞大，并不仅仅是 GUI 组件。使用 Qt，在一定程度上你获得的是一个**“一站式”**的解决方案：不再需要研究 STL，不再需要 C++ 的<string>，不再需要到处去找解析 XML、连接数据库、访问网络的各种第三方库，因为 Qt 自己内置了这些技术。
* Qt 和 wxWidgets 一样，也是一个标准的 C++ 库。但是它的语法类似于 Java 的 Swing，十分清晰，而且使用信号槽（signal/slot）机制，让程序看起来很明白——这也是很多人优先选择 Qt 的一个很重要的原因。不过，所谓“成也萧何，败也萧何”。这种机制虽然很清楚，但是它所带来的后果是你需要使用 Qt 的 moc 对程序进行预处理，才能够再使用标准的 make 或者 nmake 进行正常的编译，并且信号槽的调用要比普通的函数调用慢大约一个数量级（Qt 4 文档中说明该数据，但 Qt 5 尚未有官方说明）。Qt 的界面也不是原生风格的，尽管 Qt 使用 style 机制十分巧妙地模拟了原生界面。另外值得一提的是，Qt 不仅仅能够运行在桌面环境中，还可以运行在**嵌入式平台以及手机平台**。
* **Glade说明**

Glade是一个RAD工具，可以快速轻松地开发GTK +工具包和GNOME桌面环境的用户界面。Glade中设计的用户界面保存为XML，通过使用GtkBuilder GTK +对象，可以根据需要动态加载应用程序。通过使用GtkBuilder，Glade XML文件可用于多种编程语言，包括C，C ++，C＃，Vala，Java，Perl，Python等。Glade是根据GNU GPL许可证发布的自由软件。

* **GTK的说明**
* 本教程介绍了如何在Python中编写GTK + 3应用程序。在学习本教程之前，建议您合理掌握Python编程语言。与标准输出（控制台/终端）交互相比，GUI编程引入了新的问题。 您必须知道如何创建和运行Python文件，理解基本的解释器错误，以及使用字符串，整数，浮点数和布尔值。对于本教程中更高级的小部件，需要熟悉列表和元组。虽然本教程描述了GTK + 3中最重要的类和方法，但它不应该作为API参考。有关API的详细说明，请参阅GTK + 3参考手册。还有一个特定于Python的参考资料。

1. **算法介绍**
2. **迪杰斯特拉算法**

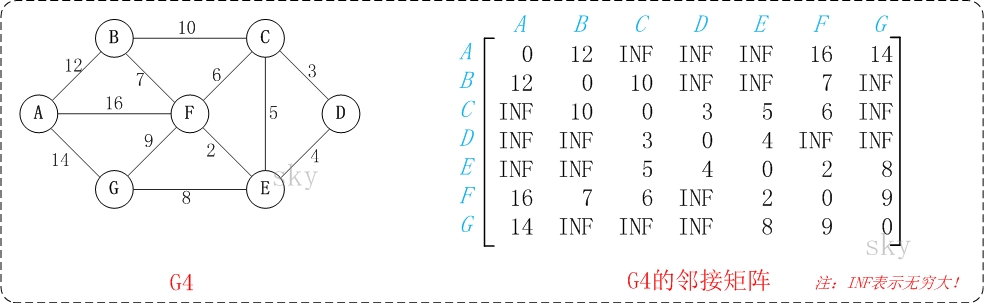
* 迪杰斯特拉算法是由荷兰计算机科学家狄克斯特拉于1959年提出的，因此又叫狄克斯特拉算法。是**从一个顶点到其余各顶点的最短路径算法，解决的是有向图中最短路径问题**。迪杰斯特拉算法主要特点是以起始点为中心向外层层扩展，直到扩展到终点为止。



* 首先，引入一个辅助向量D，它的每个分量D[i]表示当前所找到的 Dijkstra算法运行动画过程 Dijkstra算法运行动画过程 从起始点 （即源点 ）到其它每个顶点 的长度。例如，D[3] = 2表示从起始点到顶点3的路径相对最小长度为2。这里强调相对就是说在算法执行过程中D的值是在不断逼近最终结果但在过程中不一定就等于长度。
* D的初始状态为：若从v 到v[i]有弧（即从v到v[i]存在连接边），则D[i]为弧上的权值（即为从v到v[i]的边的权值）；否则置D[i]为∞。显然，长度为 D[j]= Min{ D |v[i]∈V } 的路径就是从v出发到顶点v[j]的长度最短的一条路径，此路径为(v,v[j])。
* 那么，下一条长度次短的是哪一条呢？也就是找到从源点v到下一个顶点的最短路径长度所对应的顶点，且这条最短路径长度仅次于从源点v到顶点v[j]的最短路径长度。 假设该次短路径的终点是v[k]，则可想而知，这条路径要么是(v,v[k])，或者是(v,v[j],v[k])。它的长度或者是从v到v[k]的弧上的权值，或者是D[j]加上从v[j]到v[k]的弧上的权值。
* 一般情况下，假设S为已求得的从源点v出发的最短路径长度的顶点的集合，则可证明：下一条次最短路径（设其终点为x）要么是弧(v,x)，或者是从源点v出发的中间只经过S中的顶点而最后到达顶点 的路径。 因此，下一条长度次短的的最短路径长度必是D[j]= Min{ D[i] |v[i]∈V-S }，其中D 要么是弧( v,v[i])上的权值，或者是D[i]( v[k]∈S)和弧(v[k] ,v[i] )上的权值之和。

1. **弗洛伊德算法**

* **在计算机科学中，Floyd-Warshall算法是一种在具有正或负边缘权重（但没有负周期）的加权图中找到最短路径的算法。**算法的单个执行将找到所有顶点对之间的最短路径的长度（加权）。虽然它不返回路径本身的细节，但是可以通过对算法的简单修改来重建路径。该算法的版本也可用于查找关系R的传递闭包，或（与Schulze投票系统相关）在加权图中所有顶点对之间的最宽路径。
* Floyd-Warshall算法是动态规划的一个例子，并在1962年由Robert Floyd以其当前公认的形式出版。然而，它基本上与Bernard Roy在1959年先前发表的算法和1962年的Stephen Warshall中找到图形的传递闭包基本相同，并且与Kleene的算法密切相关 在1956年）用于将确定性有限自动机转换为正则表达式。算法作为三个嵌套for循环的现代公式首先由Peter Ingerman在1962年描述。



* 从任意节点i到任意节点j的最短路径不外乎2种可能，1是直接从i到j，2是从i经过若干个节点k到j。所以，我们假设Dis(i,j)为节点u到节点v的最短路径的距离，对于每一个节点k，我们检查Dis(i,k) + Dis(k,j) < Dis(i,j)是否成立，如果成立，证明从i到k再到j的路径比i直接到j的路径短，我们便设置Dis(i,j) = Dis(i,k) + Dis(k,j)，这样一来，当我们遍历完所有节点k，Dis(i,j)中记录的便是i到j的最短路径的距离。

1. **流程分析**

首先，启动程序，可以有左侧提供了信息录入，信息查询，路径查询以及多景点之间路径查询的功能入口。其中信息录入用来录入景点数据，信息查询用来查询景点的信息，路径查询可以帮助查询两个景点之间最佳的游览路径以及所有路径，多经典查询将这种最佳路径扩展到了多个景点之间。

然后，根据不同的功能选项可有不同的进度，本程序提供相对完善的功能来实现这些操作。

1. **概要设计**
2. **存储景点信息**

使用结构体来存储景点信息，每个景点作为一个单独的景点类对象，含有景点编号（也就是唯一标识符），景点名称和景点简介的信息。同时应该使用文件读写操作确保景点信息能够保存在本地并且可供查询。

此外，还可以录入景点之间的路径的数据，方便计算最短路径的问题。

1. **维护景点信息**

通过一个特定的函数读取存储景点信息的文件，并且可以写入新的信息以及修改信息。

1. **单源最短路径**

使用迪杰斯特拉算法可以求单源最短路径问题（不过边的权重需要为正数）。

1. **两点之间最短路径**

使用弗洛伊德算法或者迪杰斯特拉算法都可以解决这个问题，具体实现时会有具体的分析。

1. **任意两点之间最短路径**

弗洛伊德算法是一种在具有正或负边缘权重（但没有负周期）的加权图中找到最短路径的算法。

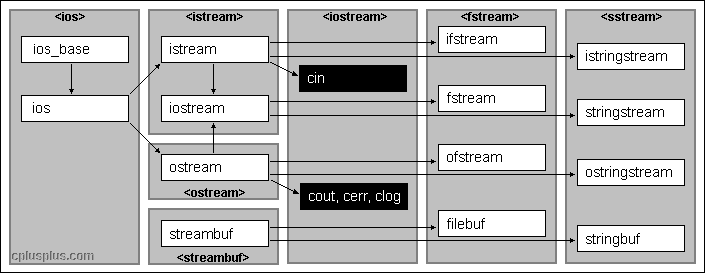
1. **多景点之间最佳路径**

对于多个景点求最佳路径的问题，可以转化为先到第一个景点，再到第二个景点，最后到最后一个景点的问题，如果途中经过需要经过的某个节点，就将其标记为已访问，最后可以得到一条路径，本质上是对弗洛伊德算法进行改进。

1. **图形界面**

使用Qt设计师进行图形界面的开发，同时加入信号与槽可以方便使用C++函数进行调用与对接。

1. **文件读写操作**



* **相关的头文件**
* #include <fstream>
* **需要相关的类**
* fstream提供三种类，实现C++对文件的操作
* ofstream：写操作，由ostream引申而来
* ifstream：读操作，由istream引申而来
* fstream ：同时读写操作，由iostream引申而来
* **文件的类型**
* 文本文件和二进制文件
* **文件读写的步骤**
* 包含的头文件：#include <fstream>
* 创建流
* 打开文件(文件和流关联)
* 读写 (写操作：<<,put( ), write( ) 读操作： >> , get( ),getline( ), read( ))
* 关闭文件：把缓冲区数据完整地写入文件， 添加文件结束标志， 切断流对象和外部文件的连接

1. **界面设计**
2. **界面布局**

界面整体采用左边栏为功能入口，中间部分为主体功能实现位置的布局。其中顶栏处为程序大标题。通常在开发一个实际的应用程序时会尽量做到界面友好，最常使用的方法就是使用图形 用户界面GUI。建立这样一个界面友好、占用资源少、高性能、便于移植、可配置的GUI界面设计，能够使用户的学习和使用更为方便容易。用户不需知道后台的应用程序究竟是怎样执行各种命令的，而只需了解可见界面组件的使用方法；用户也不需知道命令是怎样执行的，只要通过与界面交流就可以使指定的行为得到正确执行，对输入的通信信号进行一系列的处理。

1. **左侧快捷入口**

左侧提供了信息录入，信息查询，路径查询以及多景点之间路径查询的功能入口。其中信息录入用来录入景点数据，信息查询用来查询景点的信息，路径查询可以帮助查询两个景点之间最佳的游览路径以及所有路径，多经典查询将这种最佳路径扩展到了多个景点之间。



1. **主体部分**

* **信息录入**

可以让用户很方便的录入景点编号，景点名称，景点简介的景点基本信息。也提供了录入两个景点之间存在的直接路径的信息功能。



* **信息查询**

输入景点名称，点击查询景点信息就可以很方便的查询景点所有的信息，包括景点编号，名称，简介以及其中包含的路径信息。



* **路径查询**

输入道路的起点和终点景点就可以查询出最短路径以及所有存在的路径信息。



* **多景点查询**

查询多个景点之间游览路径的信息。

1. **关于窗体**

提供了一个说明本程序的窗体。

1. **托盘图标**

在 Qt5当中，本身就可以实现系统托盘的功能，而且不仅仅局限于Win32应用程序，在Linux系统中同样可以实现。

即使是Win32应用程序例如QQ、WeChat，使用Wine兼容层可以在Linux下运行，当然这种情况下运行起来质量极其糟糕。也可以通过一定的手段，让它的托盘图标不是悬浮窗显 示，而是在Linux的托盘中。

**大一下Python项目托盘图标演示界面**

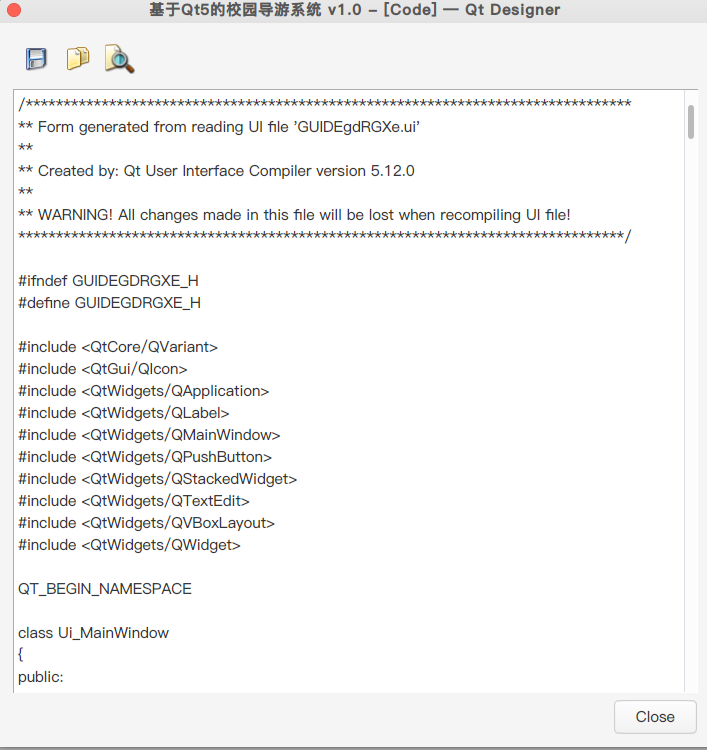


1. **设计概要**

图形界面的设计使用Qt5来实现，使用Qt可以很方便的绘制出图形界面并且生成对应的C++代码。其中重点介绍一下QStackedWidget，提供了多页面切换的布局，一次只能看到一个界面。

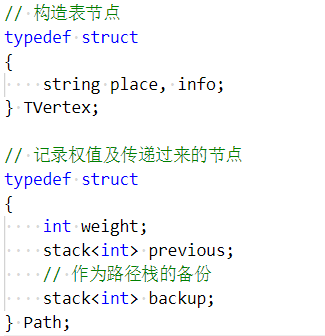
* QStackedWidget采用堆栈的思想，将不同的widget通过addwidget（）函数放进该类的对象中，并默认编号，也可显式编号，默认从0开始。
* indexOf()函数获取显示widget的页号或编号。
* setCurrentIndex(int) 来显示该页号的widget。

用QStackedWidget结合QListWidget来实现页面切换比较简单，主要包括堆栈页面设置，窗口显示列表显示和信号与槽的连接等。Qt Creator给出的示例更加完善，还使用了图标按键，使得[界面](https://www.baidu.com/s?wd=%E7%95%8C%E9%9D%A2&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "/home/y_wang/文档\\x/_blank)更加好看。



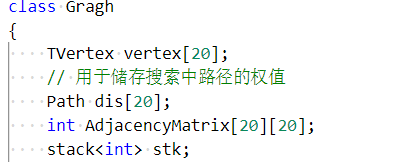
1. **详细设计**
2. **图中顶点**

* 顶点用来记录景点的名称以及信息，可以使用一个顶点数组来代表所有的景点，数组中每个元素的下标可以看作景点标号。
* 此外还应该记录权值以及传递过来的节点，中间会使用堆栈。



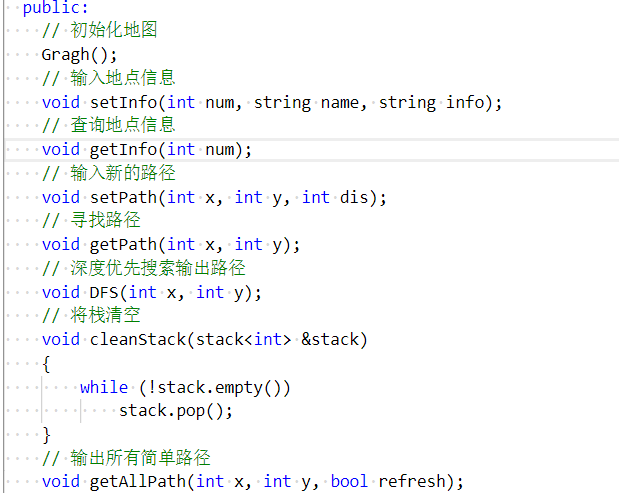
1. **Graph类定义**

* 首先，图的类中应该有存储顶点信息的数组
* 此外，还有存储顶点对之间路径权重的邻接矩阵
* 还应该有遍历时需要的堆栈（我使用了标准库中的stack）以及记录权值以及传递数据的path类型数组



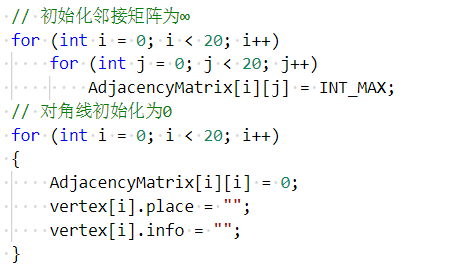
1. **Graph类.h文件中的函数**

* 基本的函数有初始化函数，设定景点信息函数，读取景点信息函数，设定两顶点之间路径函数，获取最短路径函数，获取所有路径函数。由于程序需要，我还使用了DFS深度优先遍历以及清空堆栈的函数。其中深度优先遍历实际上可以获取所有路径信息。

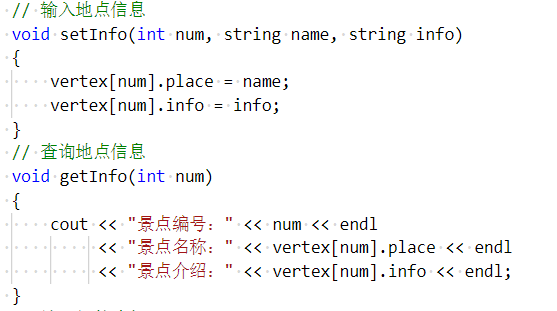


1. **简单函数具体实现**

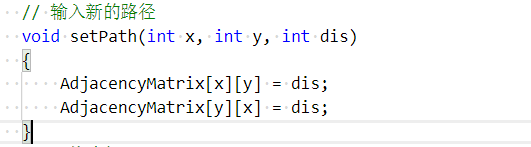
* **初始化函数**，初始化的时候，我只将邻接矩阵进行了初始赋值，即只把未标记的路径置为正无穷大，所有的对角线变为0。



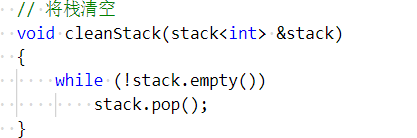
* **输入读取地点信息**，输入景点信息的函数需要三个参数，分别为顶点编号，顶点名称和顶点简介。读取地点信息的函数只有一个参数num，为顶点编号，返回字符串顶点的所有信息。



* **输入路径函数**，只需要将adj[x][y]和adj[y][x]赋值为同一个值即可。

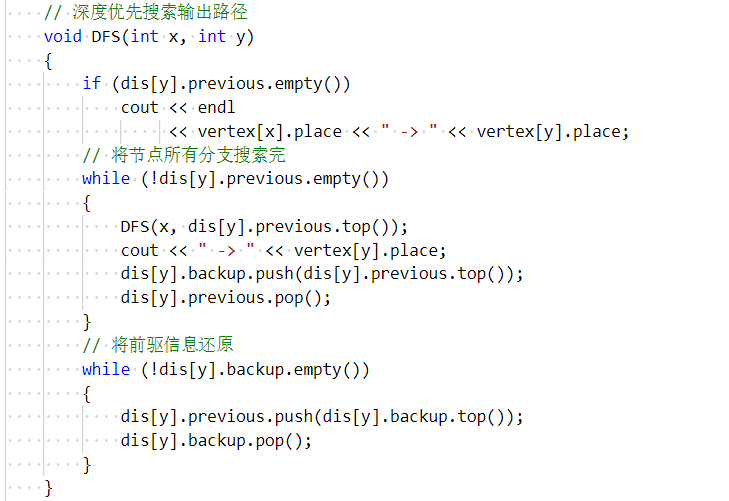


* **清空堆栈函数**，在遍历的时候会使用到这个函数来将堆栈内所有内容清除。



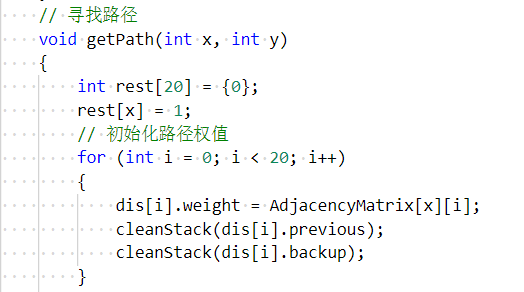
1. **深度优先遍历所有路径信息**

* **深度优先遍历**，假设初始状态是图中所有顶点均未被访问，则从某个顶点v出发，首先访问该顶点，然后依次从它的各个未被访问的邻接点出发深度优先搜索遍历图，直至图中所有和v有路径相通的顶点都被访问到。 若此时尚有其他顶点未被访问到，则另选一个未被访问的顶点作起始点，重复上述过程，直至图中所有顶点都被访问到为止。
* 通过这样的递归过程，就可以把两点之间所有的路径信息全部遍历出来。

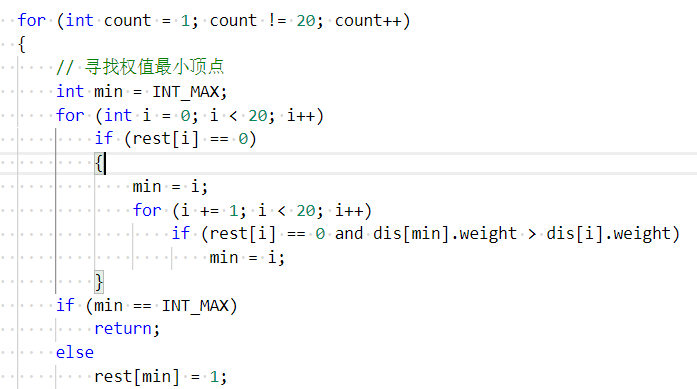


1. **迪杰斯特拉算法查询两点之间最短路径**

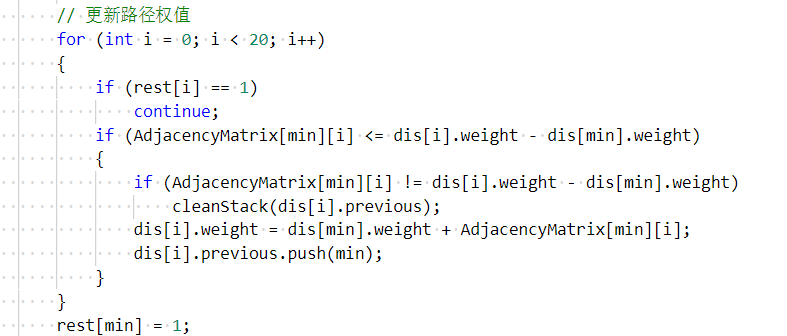
* 其基本思想是，设置顶点集合S并不断地作贪心选择来扩充这个集合。一个顶点属于集合S当且仅当从源到该顶点的最短路径长度已知。
* 初始时，S中仅含有源。设u是G的某一个顶点，把从源到u且中间只经过S中顶点的路称为从源到u的特殊路径，并用数组dist记录当前每个顶点所对 应的最短特殊路径长度。Dijkstra算法每次从V-S中取出具有最短特殊路长度的顶点u，将u添加到S中，同时对数组dist作必要的修改。一旦S包含了所有V中顶点，dist就记录了从源到所有其它顶点之间的最短路径长度。
* 迪杰斯特拉算法初始化路径权值



* 寻找权值最短的路径

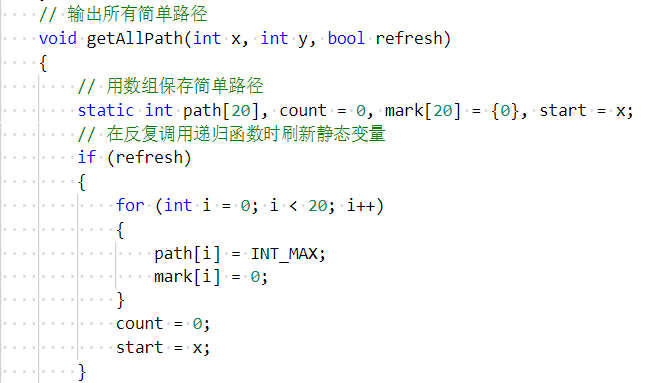


* 每次寻找以后更新权值

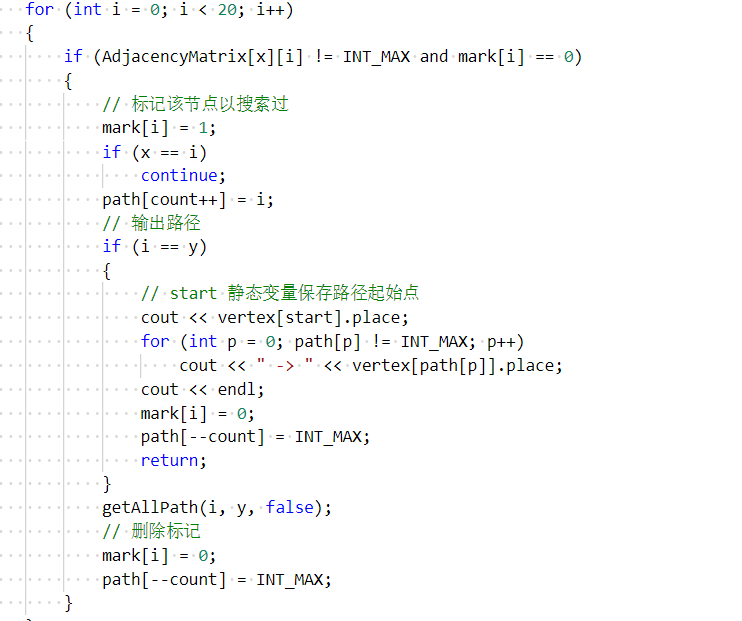


1. **输出所有简单路径**

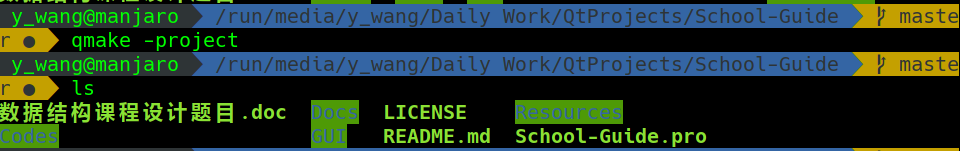
* 在本校园导游系统当中，简单路径是由数组保存的（用迪杰斯特拉算法标记节点），在反复调用递归函数时会刷新静态变量，如果路径已经找完，就会删除标记。



* 对所有的顶点进行遍历，可以输出一条路径。



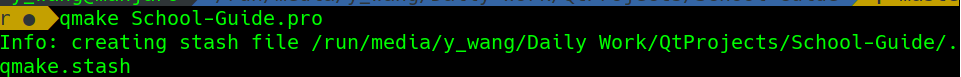
1. **Qt图形界面程序**
2. **软件编译**
3. 使用**qmake -project**命令来生成pro工程文件，这时可以看到Qt工程目录中已经生成了\*.pro文件，此文件与项目文件夹同名。



1. 在有些版本的Qt中，需要添加widgets模块。因为源文件中使用到的类都包含在这个模块中，例如：main函数中用到的QApplication类。如果你的Qt版本是Qt4，包含这个模块可能会出错，所以为了保证与Qt4的兼容性，需要添加代码。



1. 在命令行输入**qmake -工程名.pro**，会自动生成makefile文件以及.stash文件，后续只需要一条命令就可以编译完成。



1. 命令行输入**make**，g++编译器就会自动完成编译生成最终的二进制文件，然后输入./工程名就可以在Linux下运行编译好的Qt程序。
2. **软件测试**
3. **概述**

软件的错误是不可避免的，所以必须经过严格的测试。通过对本软件的测试，尽可能的发现软件中的错误，借以减少系统内部各模块的逻辑，功能上的缺陷和错误，保证每个单元能正确地实现其预期的功能。检测和排除子系统（或系统）结构或相应程序结构上的错误，使所有的系统单元配合合适，整体的性能和功能完整。并且使组装好的软件的功能与用户要求(即常说的产品策划案)保持一致。

1. **测试资源和测试环境**
2. **测试方案**

在不同的计算机操作系统上运行校园导游系统，测试系统的各个功能是否正常。本测试方案的测试数据来源于软件测试需求以及测试用例。

1. **测试执行策略**

①测试程序是否可以完成编译

②测试程序是否可以启动运行

③测试程序运行过程中是否正常工作

④测试程序输入景点信息，查询景点信息，查询路径是否可以产生正常结果

⑤测试程序是否有其他非技术性bug

1. **测试用例**

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 操作步骤 | 紧急程度 | 预置结果 |
| Window 10 version 1809 | ①测试程序是否可以完成编译  ②测试程序是否可以启动运行  ③测试程序运行过程中是否正常工作  ④测试程序输入景点信息，查询景点信息，查询路径是否可以产生正常结果  ⑤测试程序是否有其他非技术性bug | 正常 | 一切正常 |
| Windows 10 version 19h1 | ①测试程序是否可以完成编译  ②测试程序是否可以启动运行  ③测试程序运行过程中是否正常工作  ④测试程序输入景点信息，查询景点信息，查询路径是否可以产生正常结果  ⑤测试程序是否有其他非技术性bug | 正常 | 一切正常 |
| Deepin Linux v15.9 | ①测试程序是否可以完成编译  ②测试程序是否可以启动运行  ③测试程序运行过程中是否正常工作  ④测试程序输入景点信息，查询景点信息，查询路径是否可以产生正常结果  ⑤测试程序是否有其他非技术性bug | 正常 | 一切正常 |
| Manjaro Linux v18 | ①测试程序是否可以完成编译  ②测试程序是否可以启动运行  ③测试程序运行过程中是否正常工作  ④测试程序输入景点信息，查询景点信息，查询路径是否可以产生正常结果  ⑤测试程序是否有其他非技术性bug | 正常 | 一切正常 |

1. **测试报告**

本次测试总体结果还不错，基本达成了预期，但由于软件本身比较简单，因此还需要很多需要提高的地方。

1. **使用说明**
2. **工作总结**