海 南 大 学

计算机科学与技术学院

**《软件工程》课程设计报告**

课 程： 软件工程

题 目： 海口公交出行路线规划程序

学号姓名： 20203105185 崔庆轩

徐俊 my teammate

专业年级： 2020级计算机科学与技术

学 院： 计算机科学与技术学院

指导教师： 杨厚群

完成日期： 2022 年 10 月 9 日

# 需求概述

1.输入线路名称,按顺序返回站点名称

2.输入-b station1 station2 返回最短路线

3.输入-c sation1 station2 返回换成最少的最短路线

4.函数可处理格式正确的文件，并处理异常情况

5.加入帮助提示及代码详细注释，改进代码中的问题

# 二、分工和规范

## 分工

本项目中我组两位成员分工明确，关于输入线路名称按顺序返回站点名称，和返回最短路线及途径的所有站点和返回最少换乘的路径代码设计和编写，以及项目结构，具体实现功能测试及代码注释、调试和改进均由崔庆轩完成，徐俊参与到最少换乘的代码设计中，完成了性能分析工具的使用，PSP表格，测试案例。

我们共同参与到了需求文档的编写和程序报告的编写，代码规范的限制也是合作完成

## 代码规范

1.变量名英文全名用来提高代码可读性, 缩写时注释全称。

2.所有变量需要初始化

3.所有可能会被判定为空的存储空间(数组，指针，字符串)都需要额外的空语句，并且规范返回的错误形式，比如固定的输出

4.浮点数操作均使用double数据类型

5.创建的类都是private类型，成员变量必须要有get和set函数，还要保证其中的数值在合理的范围内

# 三、PSP表格

|  |  |  |  |
| --- | --- | --- | --- |
| **PSP 2.1** | **Personal Software Process Stages** | **估计的时间(分钟)** | **实际花费的时间 (分钟)** |
| **Planning** | **计划。把工作细化并大致安排次序** | **50** | **73** |
| · Estimate | · 估计这个任务需要多少时间 | **20** | **18** |
| **Development** | **开发** |  |  |
| · Analysis | · 需求分析 (包括学习新技术) | **150** | **196** |
| · Design Spec | · 生成设计文档 | **60** | **82** |
| · Design Review | · 设计复审 (和同事审核设计文档) | **50** | **51** |
| · Coding Standard | · 代码规范 (为目前的开发制定合适的规范) | **120** | **162** |
| · Design | · 具体设计 | **150** | **224** |
| · Coding | · 具体编码 | **140** | **185** |
| · Code Review | · 代码复审 | **120** | **167** |
| · Test | · 测试（自我测试，修改代码，提交修改） | **240** | **329** |
| **Reporting** | **报告** |  |  |
| · Test Report | · 测试报告 | **120** | **109** |
| · Size Measurement | · 计算工作量 | **20** | **31** |
| · Postmortem & Process Improvement Plan | · 事后总结, 并提出过程改进计划 | **120** | **194** |
| **Total** | **合计** | **1360** | **1821** |

# 四、技术栈

|  |  |
| --- | --- |
| 编程语言 | C++ |
| IDE | Microsoft Visual Studio |
| 测试工具 | Microsoft Visual Studio |

# 项目结构

## 结构

**struct line** {  
string name;  
vector<string> stations;  
vector<int> index;  
};  
  
**struct station** {  
string name;  
vec\_sz index;  
};  
  
**class File\_Editor** {  
private:  
ifstream bus\_lines;  
vector<line> lines;  
//所有线路  
vector<station> all\_stations;  
//所有的站点  
public:  
File\_Editor();  
  
void split(vector<string>& ret, string str, char sparator);  
//字符串分割函数,i,j索引分隔符，中间的字段是要被分割的字符串 need to verify  
  
int get\_string\_length(string str);  
//计算含汉字的字符串长度need to verify  
  
void separate\_line(string line, string& name, string& stations);  
//将获取的字符串分为线路名称，和站点名称大集合,这样或许就不用两个文件了，可以理解为初始化线路结构体  
  
vector<line>& get\_lines();  
  
vector<station>& get\_all\_stations();  
  
void treat\_stations(vector<string>& stations);  
  
void set\_lines\_index(line& line, vector<station> all\_stations);  
};

**class Guide** {  
private:  
int transfer\_times;  
int distance;  
public:  
Guide();  
  
bool search\_for\_fewest\_stations(string start, string end, File\_Editor editor, Map map);  
  
bool search\_for\_fewest\_transfer\_times(string start, string end, File\_Editor editor, Map map);  
  
/\*void demostrate\_line(string line\_name, vector <line> lines);\*/  
  
bool str\_trans\_line(line& lin, string line\_name, vector <line> lines);  
  
bool str\_trans\_sta(station& sta, string sta\_name, vector<station> all\_stations);  
  
void output(vector<station> route);  
  
void search\_line(vector<line> lines, station sta, line& lin);  
  
void search\_crossing(line lin, vector<station>& crossing, vector<line> lines, vector<station> all\_stations);  
  
bool search\_avalivable\_line(vector<line> lines, station sta, vector<bool> visit, line& avaliable\_line);  
  
bool station\_on\_line(station sta1, station sta2, vector<line> lines, line& ob\_line);  
  
void search\_lines\_index(vector<line> lines, line\_size& index, line lin);  
  
bool demostrate\_line(string line\_name, File\_Editor editor);  
  
void measure\_distance(int& distance, station start, station end, line ob\_line);  
  
};  
  
**struct mat** {  
vector<int> element;  
vector <vector<int>> matrix;  
};  
  
#define \_Map\_h\_  
#ifdef \_Map\_h\_  
**class Map** {  
private:  
mat adjacency\_matrix;  
  
public:  
Map(File\_Editor editor);  
  
void Dijkstra\_algorithm(mat adjacency\_matrix, station start, station end, vector<station> all\_stations, int& min\_dist, vector<station>& min\_station);  
  
mat& get\_adjacency\_matrix();  
};

## 功能说明

File\_Editor类：

负责将.txt格式的文件读取，并初始化线路和站点的相关信息。

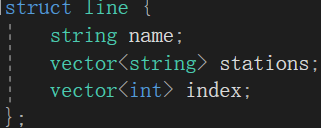
Map类：

使用初始化好的结构体信息，构建站点之间的邻接矩阵，并实现迪杰斯特拉算法。

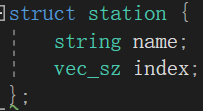
Guide类：

实现最短路径查询和换成最少次数路径的查询。

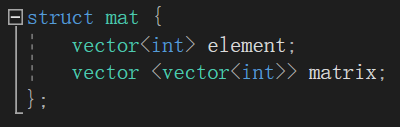
## 自定义复合数据类型



Line 结构，用于存放线路，一个线路的属性有名字，按照顺序排好的所有站点名称，和每个站点的索引(用于快速找出某个站点)



Station 结构体，属性有站点名称，站点的索引。



Mat结构体，以matrix简写命名(matrix命名空间似乎已被占用), 属性有一个vector容器，用来初始化邻接矩阵，还有一个二位vector容器，代表邻接矩阵。

# 具体实现

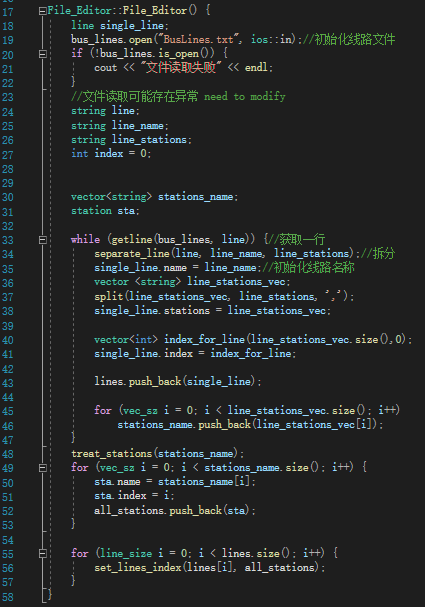
## 读取文档

文档的结构采用基本的文本文档作为被读取的文件，先将文件预处理变为站点+线路，再以逗号分隔的格式。

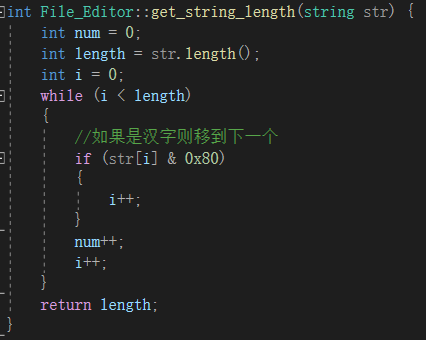


预处理完成后，通过打开文件流，以行为单位读取后，分成两部分(线路名称和站点集合)。

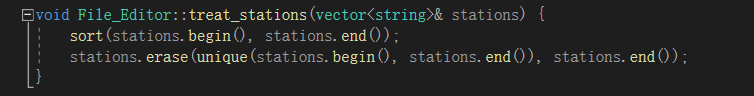
站点集合通过字符串分割函数，和线路名称一起，初始化线路结构，将所有站点集中在stations\_name容器中，通过去重排序处理，给每个站点分配索引，之后用他们初始化站点结构体集合all\_stations。



在构造函数中，因为中文字符编码比较特殊，而且在字符串分割函数中，必须明确字符串的大小，基于此，添加了返回中文字符串长度函数。

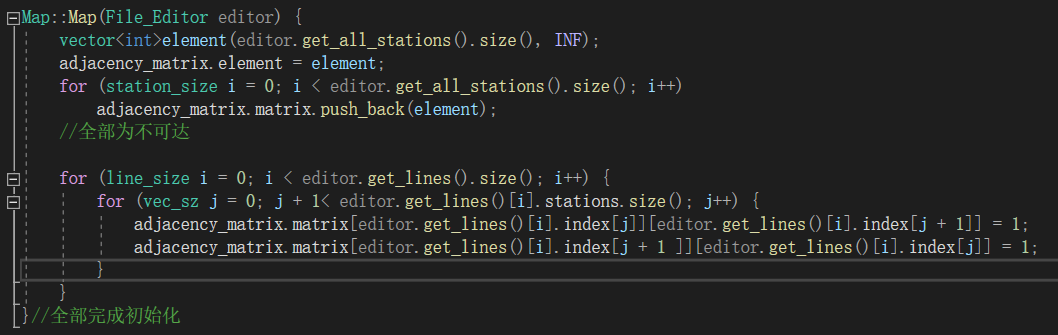


Vector版本的字符串去重也是比较巧妙的地方，由于vector的特性，删除中间的元素效率十分低下，所以采取了排序再删除结尾的元素来进行去重复(用于初始化File\_Editor中的属性all\_stations)。

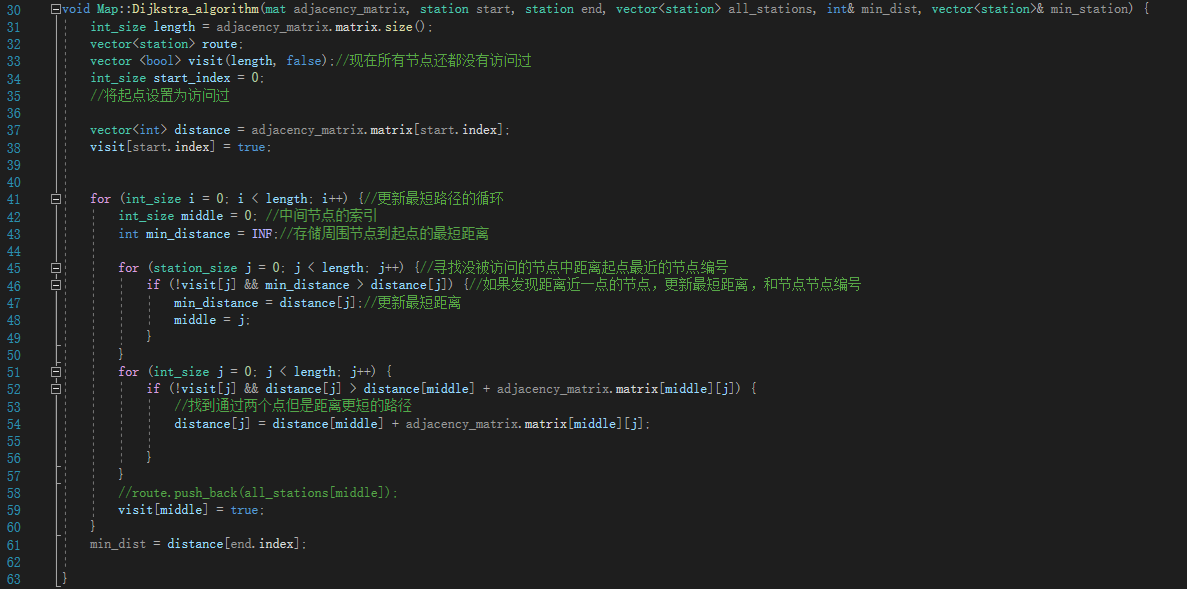


## 构建地图

先创建all\_stations.size()大小的vector容器，依次作为初始化的工具，后遍历每个线路的每两个相邻节点，并在对应的矩阵位置赋值距离1，其余默认不可达。虽然程序采用的是无向图存储结构，但并没有使用压缩矩阵来存储，所以赋值必须赋值两个以对角线对称的位置都赋值1。

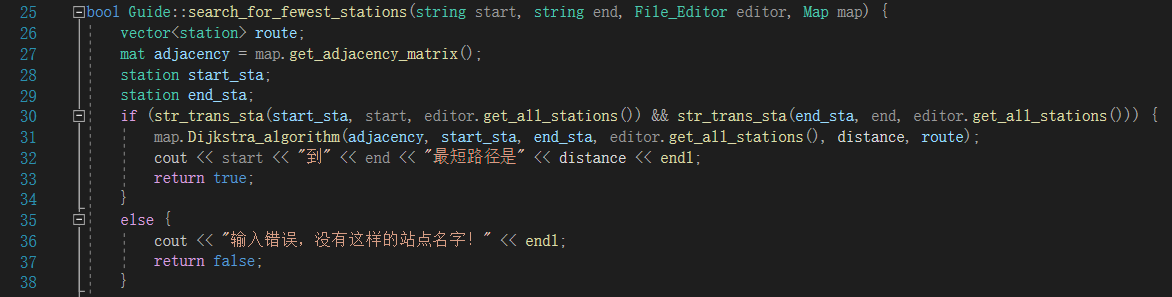


迪杰斯特拉算法的实现也是Map类的主要功能之一，迪杰斯特拉算法是基于贪心算法的一种最短线路搜寻算法，利用邻接矩阵探寻可到达且未到达的节点，以最小路径到达目标节点，再将目标节点设置为中间节点，直到所有节点都被遍历并被更新节点。具体实现如下：



## 最短路径

最短路径实现主要是将输入的字符串转换成为结构体然后输送参数到迪杰斯特拉算法，再加上防止错误输入的功能，比较简单。



## 最少换乘

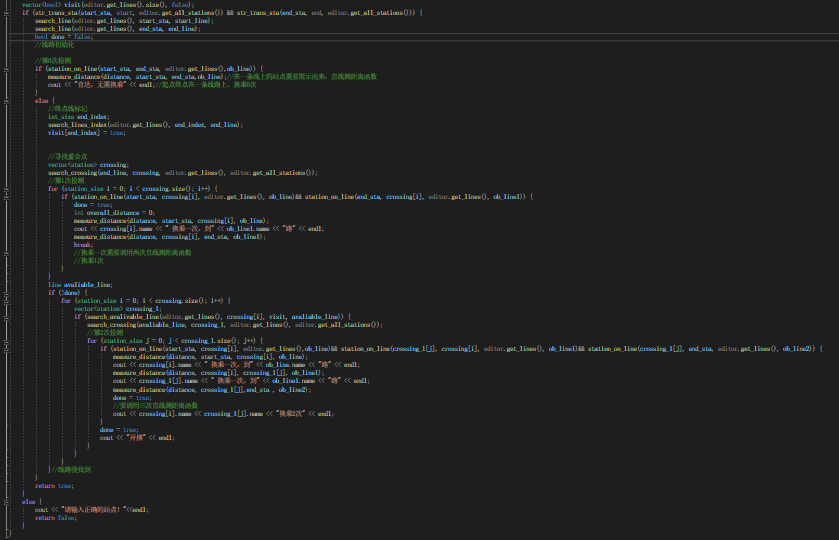
最少换乘的思路需要解释一下：

我们经过讨论决定采用如下方法，

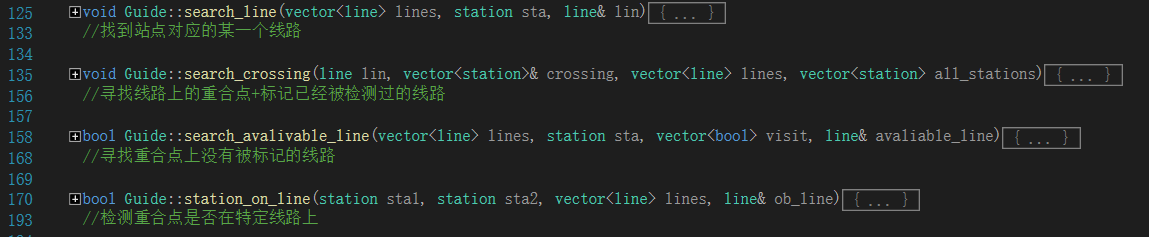
1. 寻找起点所在的线路
2. 判断终点和起点是否一条线(正确返回路径，错误继续)
3. 寻找线路换乘点
4. 判断换乘点和终点是否一条线(正确返回路径，错误继续到3)

相当于，假设不换乘成立，验证，假设换乘一次成立，验证，这样依此判断，最先获取的线路一定是最小换乘次数。

因为实现并没有采用数据结构的知识(树)，所以实现比较麻烦，而且假设换乘的次数有限。



函数主要由下面几个函数构成：

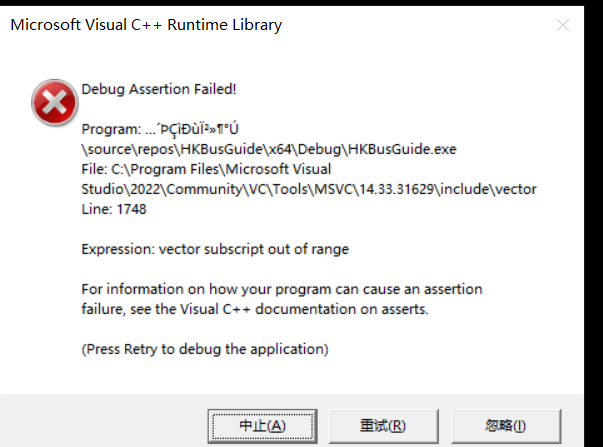


根据注释可以得到他们的功能。

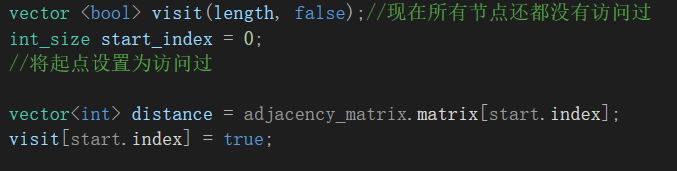
# 功能测试和修改

## 异常处理和逻辑错误的修改

**异常处理：**



这是本次程序唯一且多次出现的异常，由于程序大量使用vector容器存储数据。该异常触发有两个原因：一是没有初始化vector就采用索引访问，二十索引超出界限，这里举个例子。



如果没有第一行的定义，只是简单的声明vector<bool> visit; 那么就会在最后一个语句产生异常。

**逻辑错误：**

Editor类的函数和Guide类的函数出现逻辑错误的次数是最多的。

Editor:

字符串分割函数只能识别左右都有分隔符的字符子串

所有站点的集合all\_stations的没有去重

忽略了vector::push\_back函数的累计效果

Guide:

判断两个站点是否在同一条线路上，并返回线路时，没有用break关键字

没有考虑到起点终点的索引大小可能颠倒这件事

除了这些以外还有很多，但是当时疏忽没有选择记录，今后会注意这个问题。

## 性能测试和分析

# 改进方案

## 文件格式改进

文件格式采用最简单的txt文本文件，但是通过程序分析发现，尽管从程序上来讲，这段代码已经没有已知的优化空间，但是执行效率依然低下。

经过查阅和讨论，.json(一种轻量级的数据交换格式)更适合读写，常见的.json格式由一些键值对组成

{"key1":"value1","key2":"value2"} 由多个key:value键值对组成

{"key":["a","b","sojson.com"]} value是一个array的JSON格式

## 地图构建改进

地图的构建过两次重构：

第一次采用换乘点地图，我们认为这样极大简化了整个地图，但在最后的实现的过程的中，发现不仅要去初始化所有站点，还需要在最后采用暴力方法选出最短路径，例如：起点附近有三个换乘点，终点附近有三个换乘点，最后要在3\*5个选择中找出最短路径，相比于直接调用迪杰斯特拉算法，这样明显麻烦许多。

第二次决定，采用构建所有站点的地图，但是受到第一种观念的影响，初始化邻接矩阵时，试图大跨步前进，失败。

最后一次采用站点地图，而且每个点之和附近的几个点的距离是可以到达，这样就方便许多，而且对效率的影响忽略不计。

## 最少换乘改进

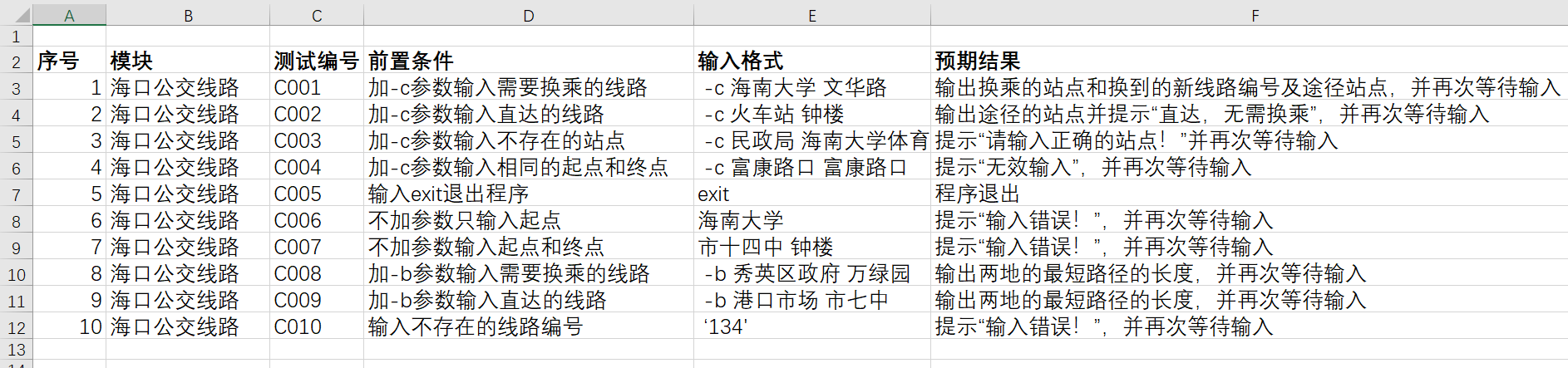
最少换乘原先采取暴力算法+深度优先遍历寻找所有的线路，不仅效率低下而且在每次检测到有多个方向时需要标记哪个已经是走过的路线，不容易实现。后面采用假设的方法直接寻找最短换乘，程序通过假设第n次换乘到达终点，然后验证，如果成功就返回路线，如果失败就继续第n+1次，具体的实现上文有描述。原定计划采用树来存储这种结构，但是考虑到换乘次数不会特别大，就直接设置了假设上限，目前测试案例没有可以超过上限的路线。

## 部分函数的性能优化

在实现最少换乘次数的几个函数中，可以直观的看到都采用了多层遍历的情况，其实这些遍历大多都是重复的，但是他们被封装的不同的函数中就要重复进行，不仅影响效率，代码复用性受到影响。一方面是存储结构的问题，另一方面是没有做好提前的规划。

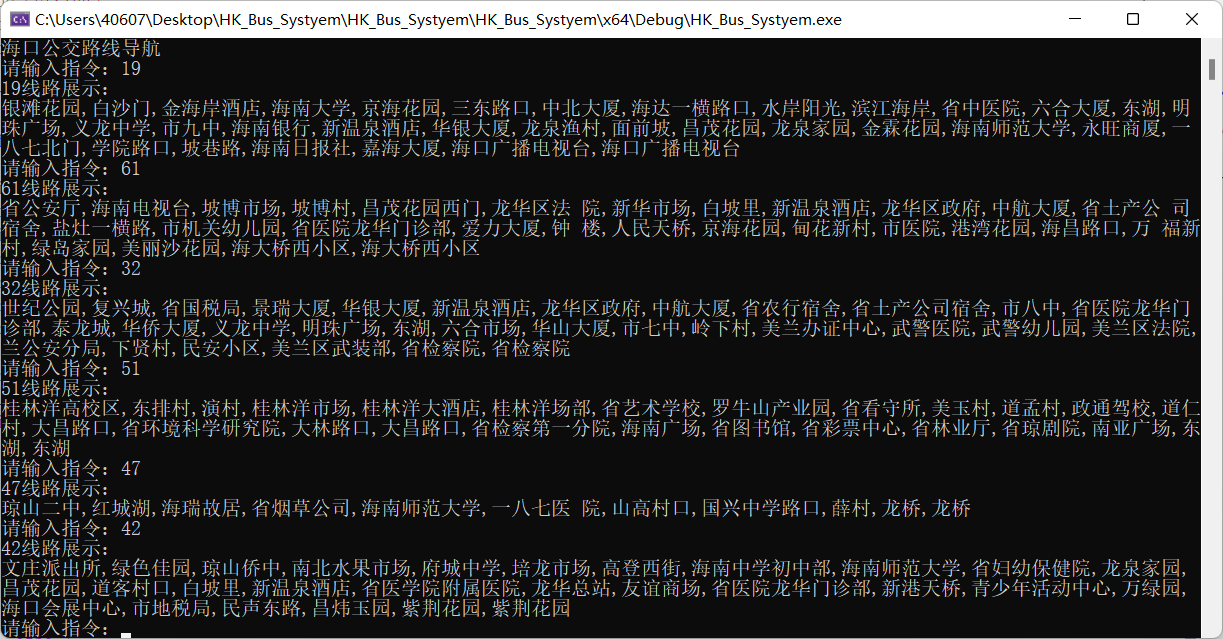


# 测试案例



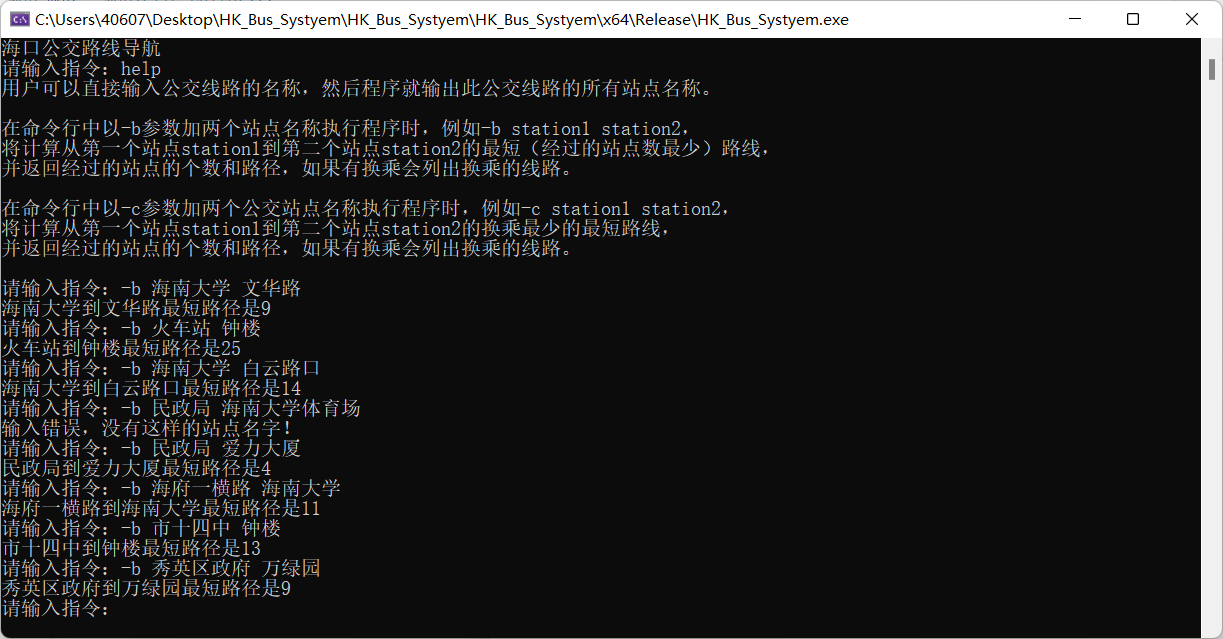
**测试案例表格**

## 线路展示



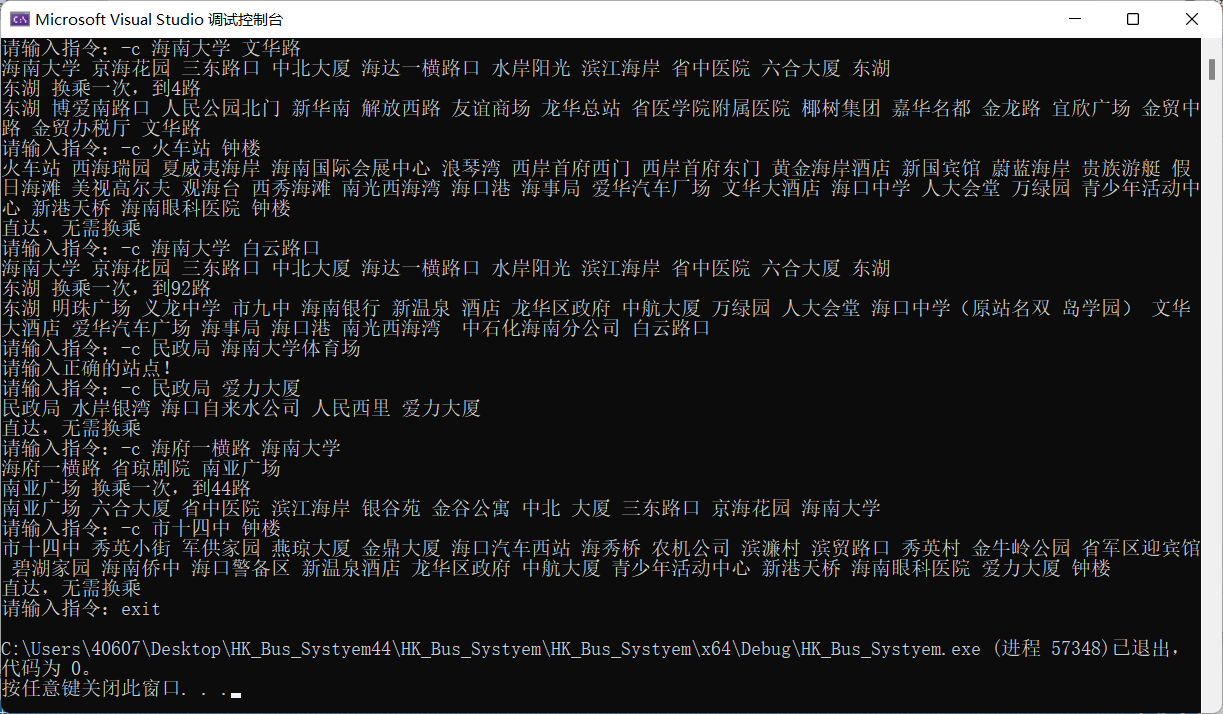
线路展示可以在输入线路编号后直接展示对应线路的所有站点。

## 最短路径搜寻



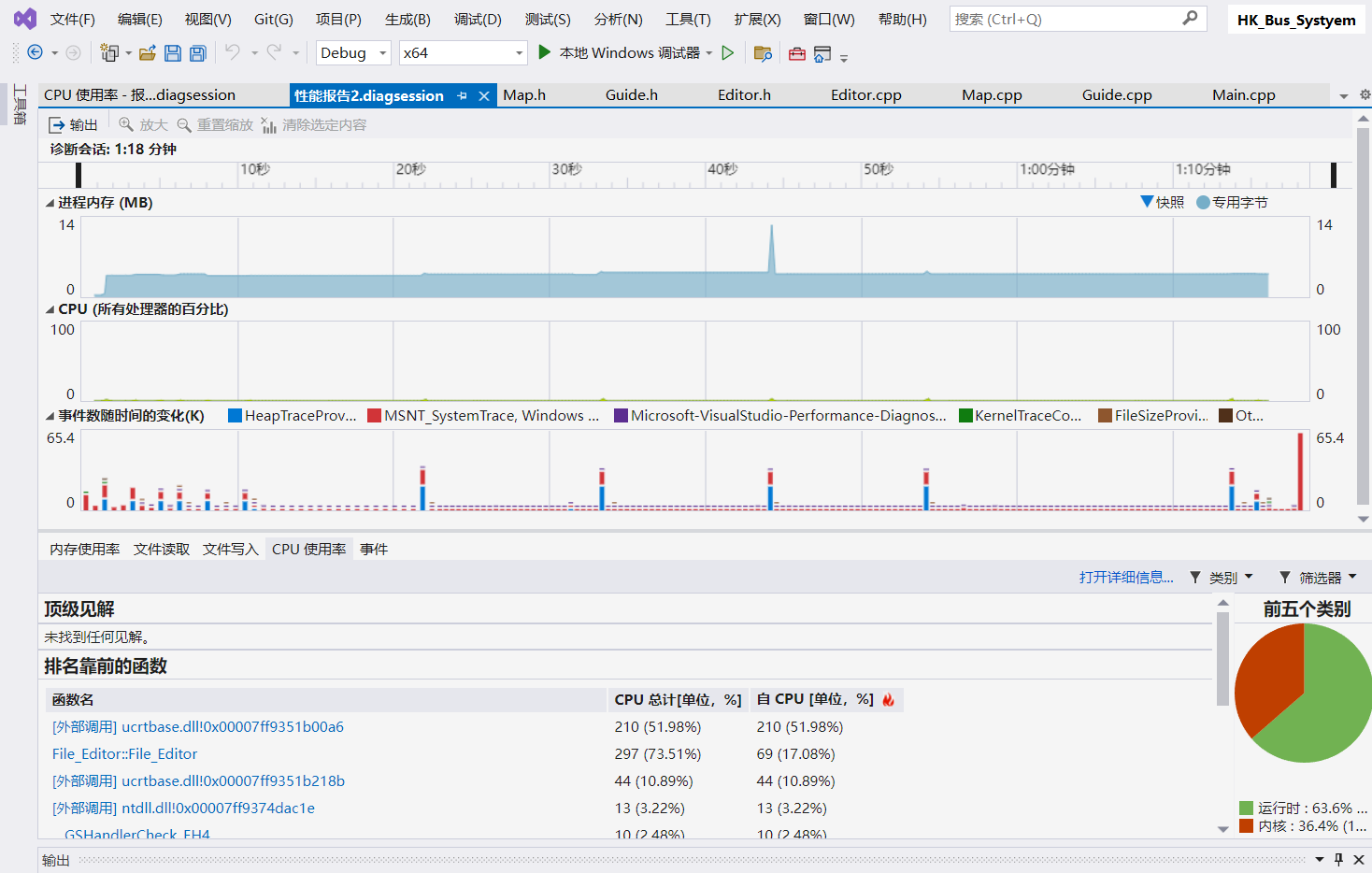
输入help 可以查阅帮助说明，再依次输入-b station1 station2发现在第四次输入时，没有海南大学篮球场这样的公交站，输出提示信息，其他情况输出最短路径长度。

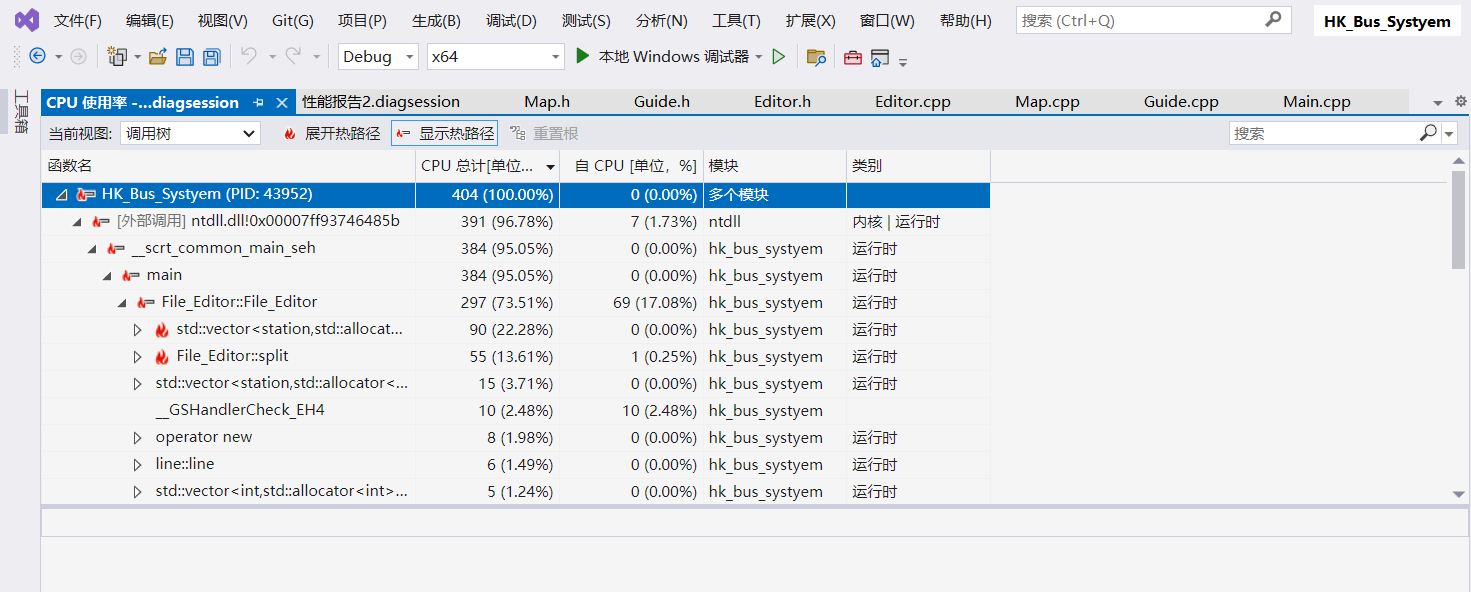
## 最少换乘路径搜寻



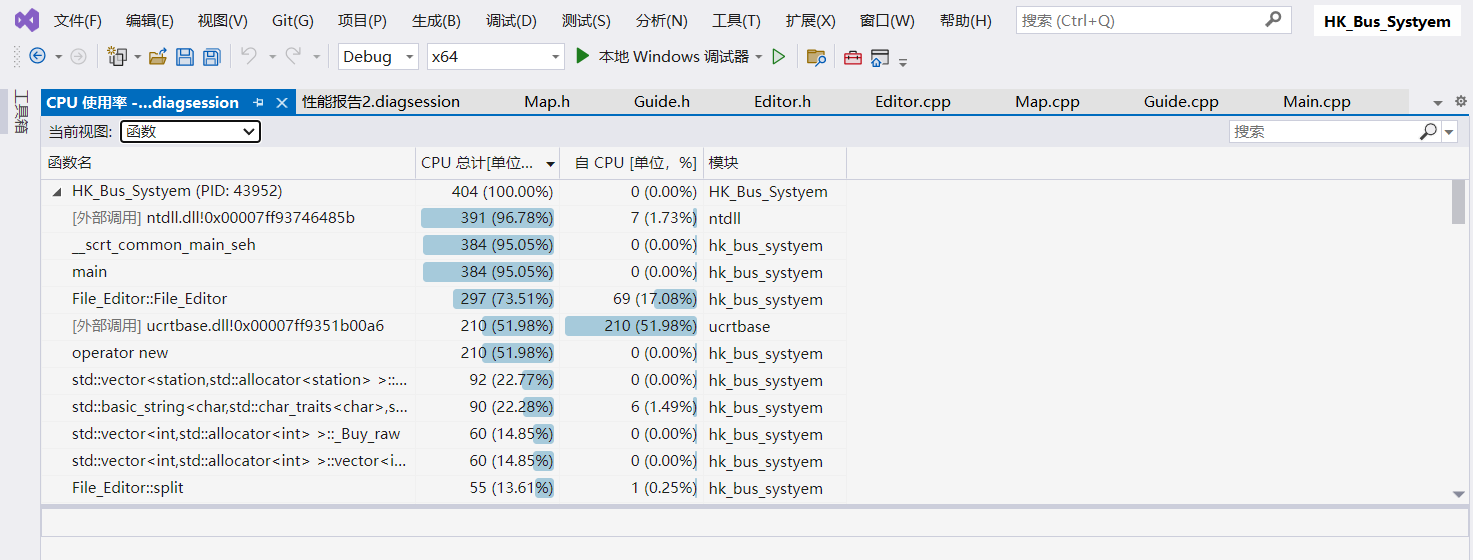
输入-c station1 station2可以得到换乘途径的站点及换乘的线路

**性能分析工具**

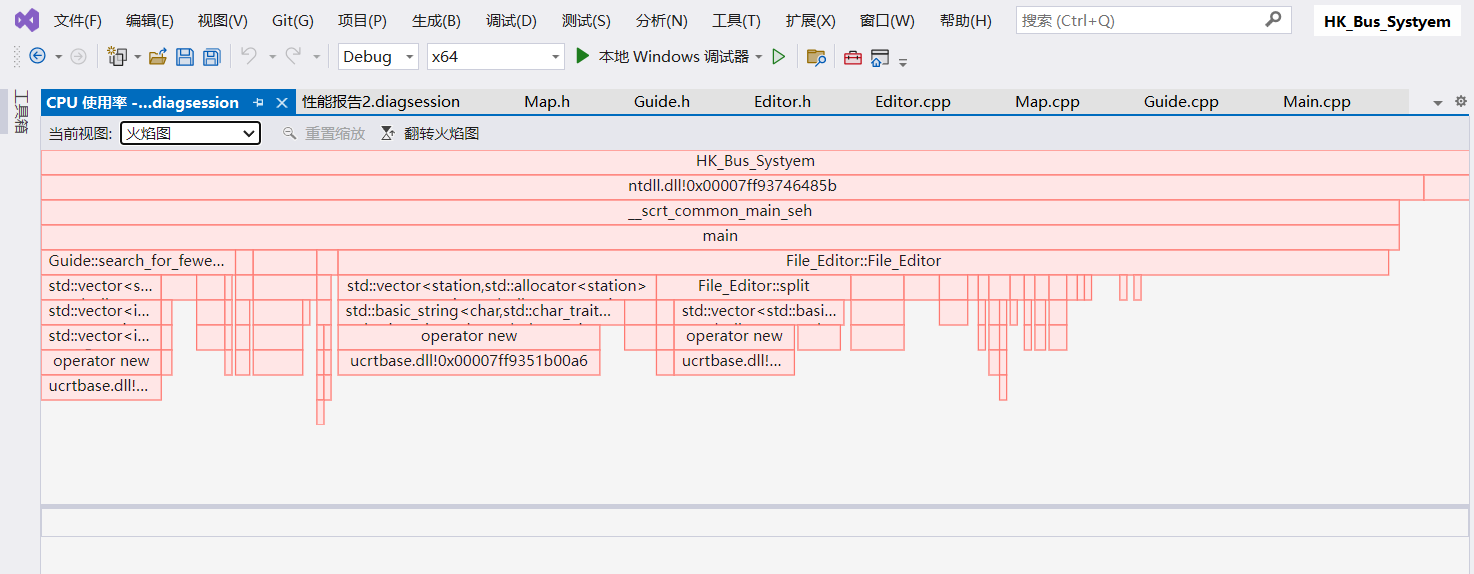
性能报告总览图



CPU使用率调用树图



CPU使用率函数图



CPU使用率火焰图

火焰图纵向（Y 轴）高低不平，表示的是函数调用栈的深度。每一层都是一个函数。调用栈越深，火焰就越高，顶部就是正在执行的函数，下方都是它的父函数。横向（X 轴）表示该函数执行消耗的时间，横向上会按照字母顺序排序，而且如果是同样的调用会做合并（注意：如果一个函数在 X 轴占据的宽度越宽，就表示它被抽到的次数多，即执行的时间长，所以这里不是严格意义上的执行消耗的时间），所以一个横向宽度越大的函数调用，一般很可能是程序的瓶颈。火焰图的颜色是随机分配的，并不是颜色越深就是越瓶颈。因为火焰图表示的是 CPU 的繁忙程度，所以一般都是暖色调。我们需要留意的就是那些比较宽大的火苗。只要有"平顶"，就表示该函数可能存在性能问题。

# 总结

本次结对项目我组两个成员分工明确，协作设计了完整的项目结构和功能性函数实现了对应线路中站点的展示，掌握了最短路径迪杰斯特拉算法的实现，共同思考了最少换乘方法的实现及其改进方法，并且掌握了性能分析工具的使用。在输入前有帮助说明信息可供输入者查阅，得到的最终输出结果有诸多输入异常问题也能够通过程序提示输入错误，例如在函数处理和改进上，Editor其中的treat\_ stations函数的加入使得在读取站点时不会因为站点重复而出现问题，先对站点进行排序后再去除重复站点，而get\_ string\_ length 函数则使得获取中文字符串长度时，不会因为中文而导致编码错误，最终较好地满足了公交线路规划的需求。

整个程序实现过程仍然存在不足，例如部分函数实现思路有些混乱，没有编写详细的伪代码，没有进行单元测试，测试方式单一，只通过测试案例列出了可能出现的几种错误输入情况。