北京工业大学

2023- 2024学年 第1学期

信息学部 物联网工程专业

|  |  |  |  |
| --- | --- | --- | --- |
| 课程名称： | 数据结构课程设计 | | |
| 报告性质： | 课程设计报告 | | |
| 学号： | 19074220 | 姓名： | 汪航 |
| 任课教师： | 苏 航 | 课程性质： | 必 修 |
| 学分： | 3 | 学时： | 60 |
| 班级： | 200731 | 成绩： |  |
| 教师评语： |  | | |

年 月 日

目录

[1. 需求分析 3](#_Toc1488135658)

[1.1程序功能 3](#_Toc1501451044)

[1.1.1题目要求功能 3](#_Toc1954844258)

[1.1.2控制管理功能 3](#_Toc715128753)

[1.2需要处理的数据 3](#_Toc1850463059)

[1.2.1 原始数据 3](#_Toc874456759)

[1.2.2 中间数据 4](#_Toc1764152092)

[1.3程序开发运行选用的环境 5](#_Toc1944979762)

[1.4用户界面的设计 5](#_Toc278785300)

[1.4.1 地铁线路图 5](#_Toc1882702993)

[1.4.2 用户控制区 6](#_Toc1565148453)

[1.4.3 结果显示抽屉 6](#_Toc922857468)

[1.4.4 特定线路展示 6](#_Toc1338566042)

[2. 数据结构设计 7](#_Toc240781922)

[2.1 所定义主要的数据结构 7](#_Toc962572106)

[2.1.1 Subway 地铁类 7](#_Toc955072691)

[2.1.2 Station类 8](#_Toc1613939959)

[2.1.3 Line类 8](#_Toc622945656)

[2.1.4 重载原始数据并创建地铁Map结构 9](#_Toc864861267)

[2.2 程序整体结构以及各模块的功能描述。 10](#_Toc1553991573)

[2.2.1 程序整体结构 10](#_Toc240252597)

[2.2.2 模块功能描述 10](#_Toc656141419)

[3. 详细设计 12](#_Toc440301788)

[3.1 主要函数的流程图 12](#_Toc2070987001)

[3.2 局部的数据结构说明 12](#_Toc663575231)

[3.2.1 intersectionMatrix相交矩阵 12](#_Toc826328546)

[3.2.2 solutions数组 13](#_Toc327127473)

[3.2.3 queue队列 13](#_Toc473302391)

[4. 测试 14](#_Toc513857049)

[4.1正确运行程序的用例 14](#_Toc1363677956)

[4.1.1 换乘最少 14](#_Toc1389925708)

[4.1.2 用时最短 14](#_Toc154262290)

[4.2 导致程序运行错误的用例 15](#_Toc673546101)

[4.3 边界数据的用例 15](#_Toc903016170)

[5. 总结与提高 16](#_Toc725835841)

[5.1 体会与收获 16](#_Toc1415864727)

[5.2开发中遇到的问题与解决情况 16](#_Toc172174282)

# 需求分析

1.1程序功能

### 1.1.1题目要求功能

通过读取百度地图提供的北京地铁json数据文件信息，加载地铁网络信息或者拉取到本地作为模块导入。配置文件内容根据设计自己组织，但需要合理。

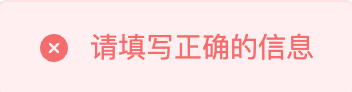
通过软件界面，可以输入起点（对于每一个地铁站，可以设置其各自本站为默认起始站）、和终点，以及选用乘车策略功能。

根据输入数据，通过相应模块进行计算处理后，给出合适的乘车路线，线路提示可以分为“用时最少”和“换乘最少”，并在地铁线路图上提示（包括提示换乘站），票价，历程/时长。

### 1.1.2控制管理功能

程序通过vite进行打包构建，在本地nodejs环境下的web服务器运行，可以在vite config文件修改项目公共配置，比如公共模块前缀，开发服务器代理配置，样式抽离等等，可以自己通过插件的形式，在整个项目层去配置。

用户异常行为合理反馈功能，比如站点没有选择，站点重复等等异常行为，或者代码层的异常，需要拦截给出日志反馈，用户视图上通过UI组件提示，比如当输入重复站点时查询线路，给出如下错误警告。



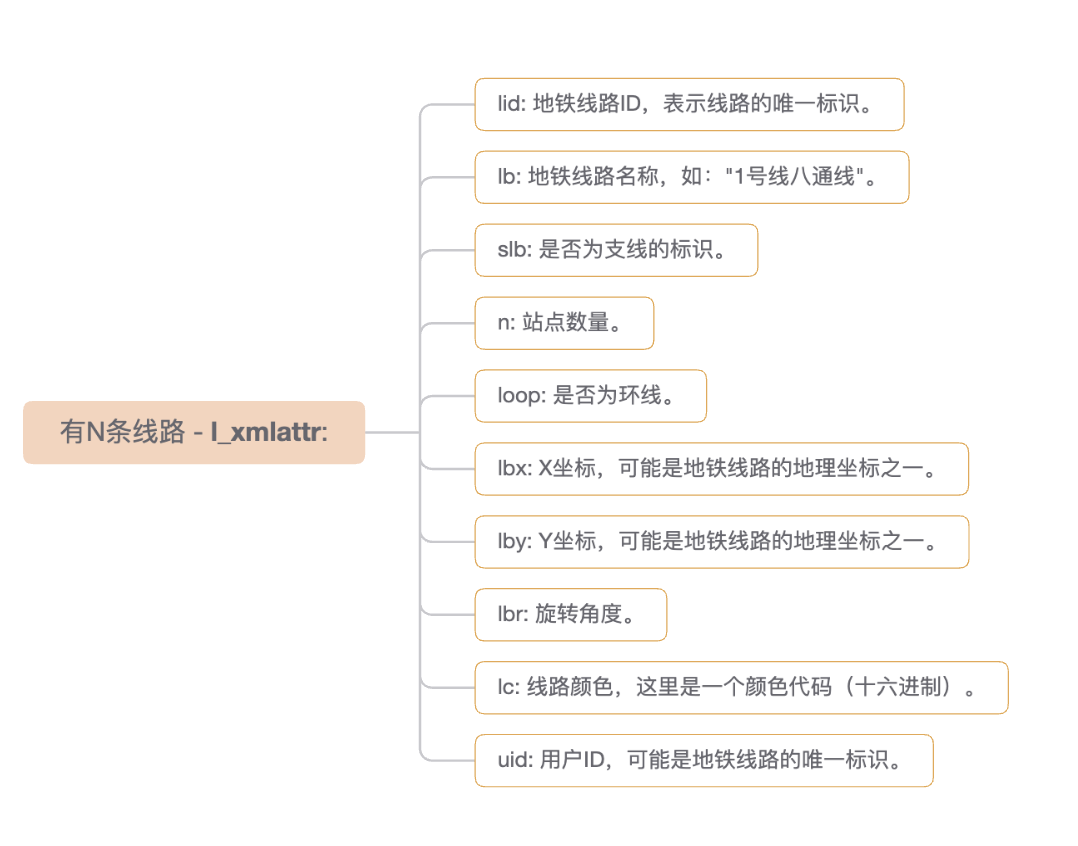
程序版本控制方面，使用Git控制，以便跟踪程序的变化、管理代码库，并轻松进行版本回滚或合并。

## 1.2需要处理的数据

### 1.2.1 原始数据

北京地铁信息subways

* 有N条线路



* 每条线路下有N个站点：



### 1.2.2 中间数据

预处理一遍源数据，将其存储到Map里，该map表是后续重载数据到核心类数据结构里的重要依据，它在项目中的角色也是原始数据。



最终结构如下：

Map: {

line\_name: station\_array [[station\_name, isEx, weight], [...], ...]

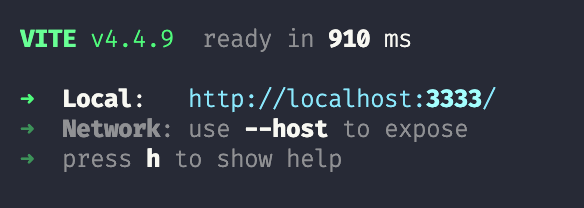
// ...

}

站点数组是一个3\*n的二维数组。

## 1.3程序开发运行选用的环境

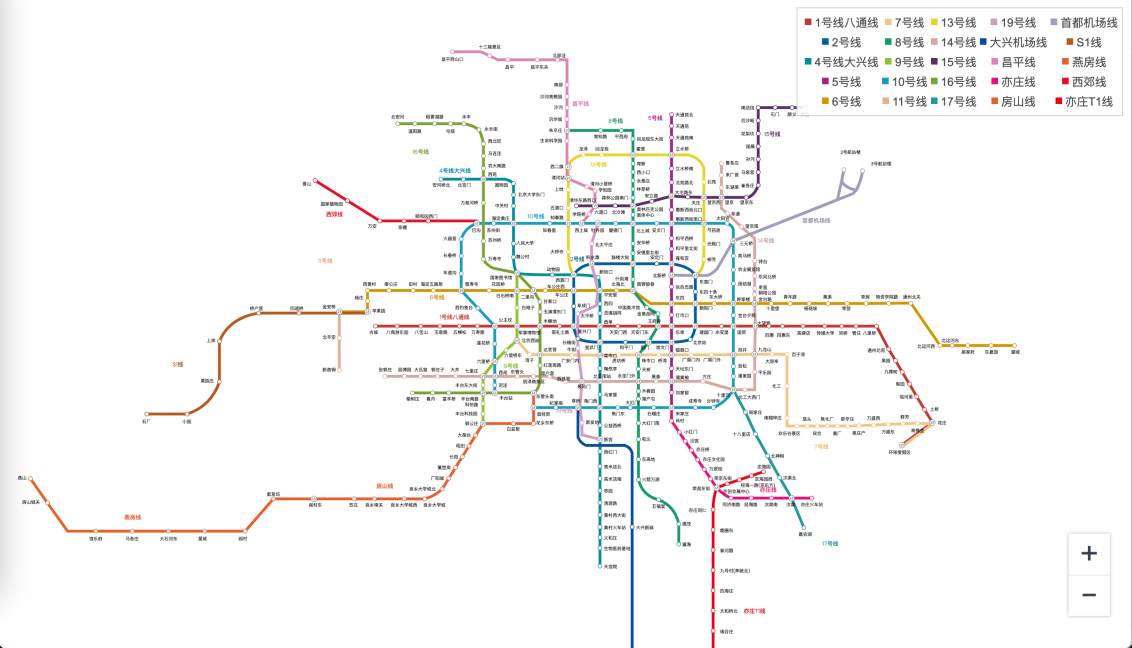
运行环境：使用vite在nodejs环境下启用本地开发服务器，在web浏览器下访问本地开发服务器并运行程序。



开发技术环境：vuejs；element-plus；d3.js；

## 1.4用户界面的设计

1.4.1 地铁线路图



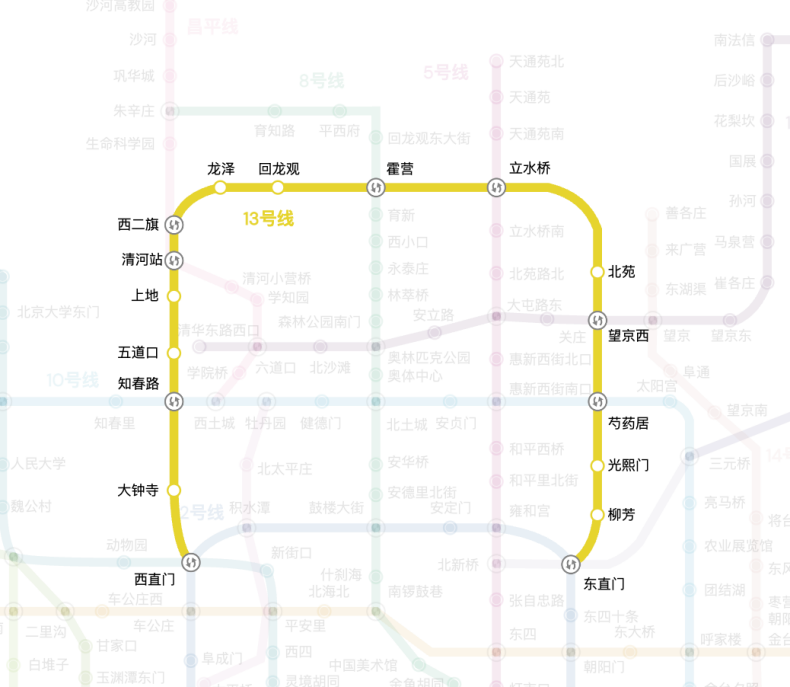
1.4.2 用户控制区



1.4.3 结果显示抽屉



1.4.4 特定线路展示



# 数据结构设计

## 所定义主要的数据结构

### Subway 地铁类



subway是承载地铁图起到引导作用的数据结构，主要是直接操作网络加载的网络原始数据。这里之所以使用原始数据，而后面的Line、Station还有算法都用的是二次处理过的数据，是因为d3绘制svg图像需要原始数据很多图形属性，而算法需要的数据结构只需要很简洁的id标识，倘若保留太繁琐的属性，那么后续数据结构的处理会变得非常麻烦，当时想的是权衡的结果，当然这样的坏处就是后面，如果需要这些属性就无法使用已经写好的数据结构，需要完全设计成一致的结构。

### Station类



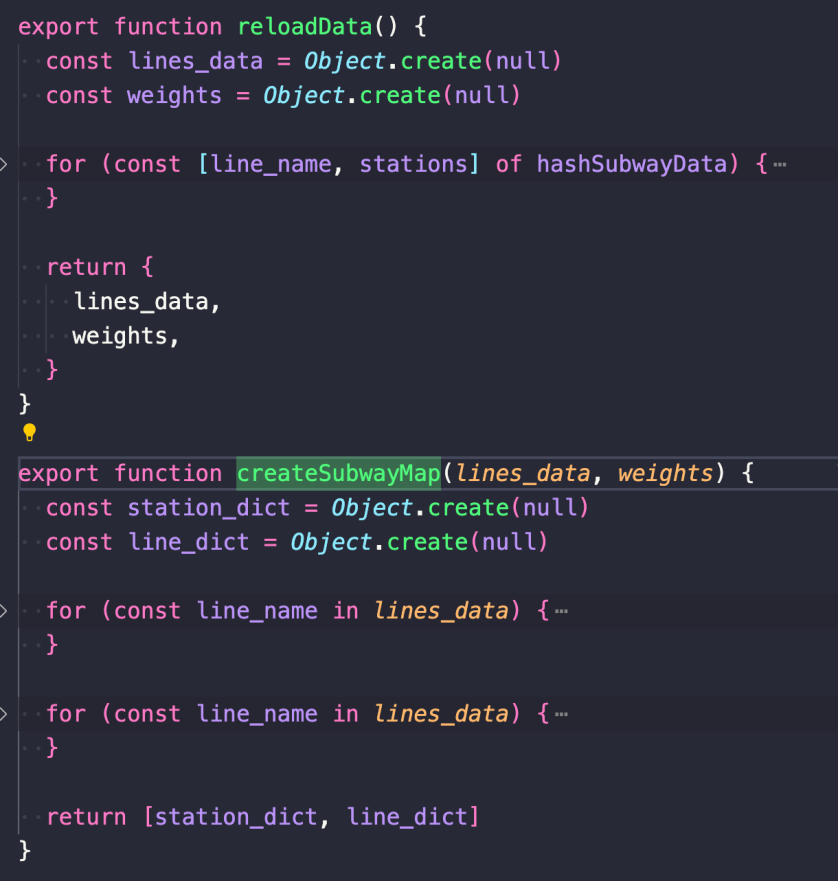
Station是整个地铁带权图的最小单位节点，包含名称，邻居站点等属性。

### Line类



Line是包含线路站点集合以及名称，还有针对特定线路的操作方法。

### 重载原始数据并创建地铁Map结构



1、reloadData函数：

创建两个空对象 lines\_data 和 weights。

遍历名为 hashSubwayData 的迭代对象，该对象可能包含地铁线路和相关站点信息。

对于每个地铁线路，创建一个空数组并遍历其中的每个站点信息。

如果站点名在 weights 对象中不存在，则将其添加，并用相应的站点时间初始化。

将每个站点名添加到该地铁线路对应的数组中。

返回包含 lines\_data 和 weights 对象的结果。



2、createSubwayMap函数：

遍历 lines\_data 对象，其中包含地铁线路及其对应的站点数组。

对于每个地铁线路，创建一个新的 Line 对象，并将该线路的每个站点添加到 subLine 对象中。

如果站点名在 station\_dict 对象中不存在，则创建一个新的 Station 对象。

将该地铁线路和对应的 subLine 对象添加到 line\_dict 对象中。

遍历 lines\_data 对象，处理每个地铁线路的站点信息。

对于每个线路的相邻站点，将它们添加到 station\_dict 对象中，并设置它们之间的权重（时间）。

站点之间的连接是双向的，因此对于每一对相邻站点，都在两个方向上设置了连接。

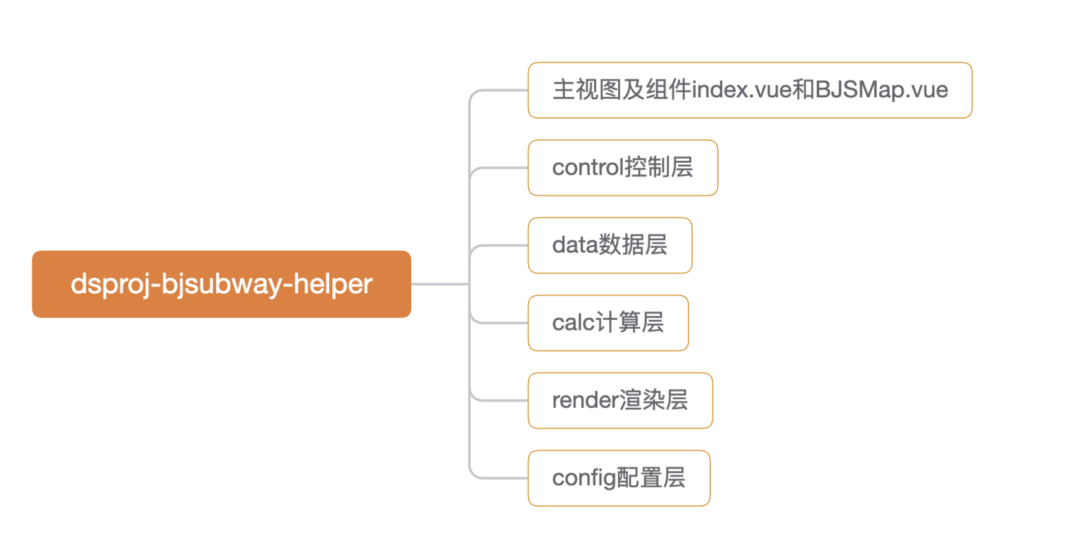
最终返回一个包含 station\_dict 和 line\_dict 对象的数组。

这两个函数一起的作用是将原始的地铁数据 hashSubwayData ，重新整理为两个对象：lines\_data 和 weights。然后，根据这些数据创建一个地铁图，其中包括站点和线路的字典。站点之间的连接包括权重，这样核心的数据结构就搭建起来了。

## 程序整体结构以及各模块的功能描述。

### 程序整体结构

程序整体结构分为6大板块：



### 模块功能描述

1. 视图层

主要作为整个页面的视图框架，BJSMap.vue是地铁线路图子组件，index.vue是主 视图，所有组件最终都会被加载到该页面，形成树形结构最终挂载到浏览器文档对象。

视图层是页面交互逻辑实际展现的地方，用户通过视图结构来与程序进行交互。 比如点击查询之后，会根据输入执行calc层的算法逻辑，最后获得处理后的返回值在 再回显到视图层，这样就完成了视图层的功能。

1. control控制层

控制层是粘合calc层与视图层的胶水层，其中的函数会引用calc层的导出方法， 封装在SideController类中，暴露给视图层去引用，并提供一些响应式数据连接，当无 页面刷新输入数据变更时也能正确执行。

总之，control层提供胶水函数给视图组件，用户通过组件调用这些函数后，就可 以执行算法代码了。

1. data数据层

数据层包含所有数据结构的处理和预处理，以及局部数据结构流转。

该模块供给算法处理模块所依赖的所有数据，比如线路梳理中获取交点换乘站， 站点梳理中获取邻接节点，图heap容器中提供dijkstra算法需要的数据结构，以及制 作核心的subwat map结构等等。

1. calc计算层

calc文件主要负责算法函数的实现，也是整个程序运转的核心逻辑，它接受data 层提供的数据，按照一定格式的约定将数据处理运算后，返回简洁的计算结果提供给 控制层从而在视图展示渲染。

关键函数功能包含以下几个方面，dijkstra算法：function dijkstra(station\_dict, start, end)，接受station字典运算返回weight和路径元组。

leastExchange函数：function leastExchange(intersectionMatrix, line\_dict, station\_dict, start, end)使用相交矩阵运算出最佳换乘策略，返回weight和路径元组。

1. render渲染层

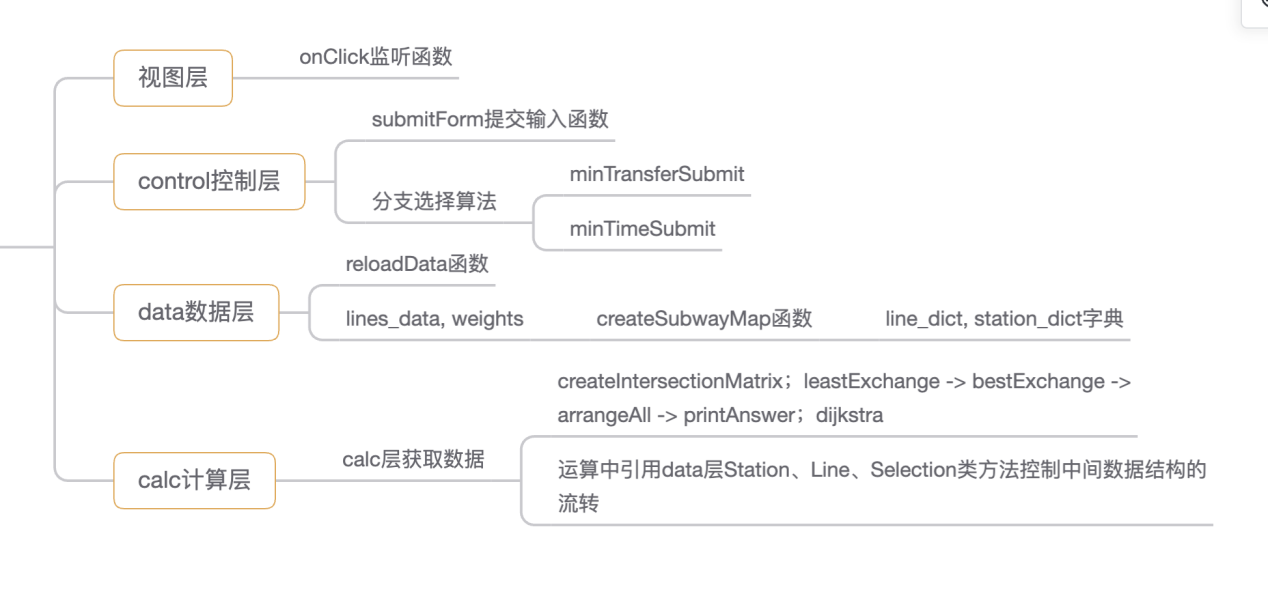
渲染层主要是针对地铁路径图svg矢量图进行初始化，包括d3地铁图初始化，事 件驱动函数。事件驱动包括线路展示，缩放拖动，站点展示，数据-矢量图转换等逻辑。

1. config配置层

如之间所讲，配置层提供全局级别的插件配置函数，可以通过配置插件函数来实 现全局变量改写，模块转译，开发服务器代理配置等功能，总之相当于是一个程序项 目的根管理器。

# 详细设计

## 3.1 主要函数的流程图



## 3.2 局部的数据结构说明

3.2.1 intersectionMatrix相交矩阵

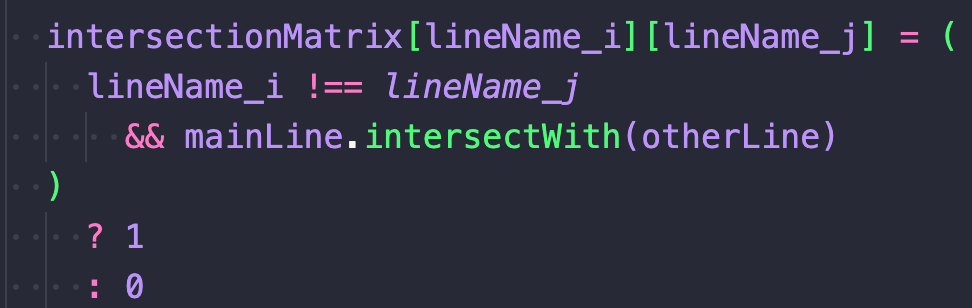
intersectionMatrix 是一个邻接矩阵，用于表示地铁网络中不同站点之间是否可以 换乘。具体来说，这个矩阵的每个元素 intersectionMatrix[i][j] 表示站点 i 和站点 j 之间 是否存在可以进行换乘的连接。

在calc模块中，intersectionMatrix 被用于判断两个站点之间是否可以进行换乘。具体来说：

1. 当 intersectionMatrix[i][j] 为真（非零值）时，表示站点 i 和站点 j 之间存在可以进行换乘的连接。

2. 当 intersectionMatrix[i][j] 为假（0）时，表示站点 i 和站点 j 之间不存在可以进行换乘的连接。

在算法的实现中，通过检查 intersectionMatrix[cur.station\_name][neibor]，确定当前选择的站点 cur.station\_name 和邻接站点 neibor 之间是否可以进行换乘。如果条件满足，算法会在 mayExchanges 中记录一次潜在的换乘路径，并将邻接站点加入到搜索队列 queue 中，以进一步探索可能的路径。



3.2.2 solutions数组

mySolutions 数组的作用是存储可能的换乘方案。在calc模块中，每个 mayex（可 能的换乘路径）都会被转化为一组换乘方案，存储在 mySolutions 数组中。

具体来说，mySolutions 是一个二维数组，其中每个元素是一个数组，表示一种可能的换乘方案。每个数组的结构如下：

mySolutions[0]: 该数组的第一个元素是起始站点（start）的单一元素数组，表示初始状态。

对于 mySolutions[1] 到 mySolutions[exSize]，每个数组包含两个相邻的换乘站点之 间的所有站点，表示在这两个相邻线路之间的可能的换乘方案。

mySolutions[exSize + 1]: 该数组的最后一个元素是终点站点（end）的单一元素数 组，表示最终状态。

这样，mySolutions 中的每个数组都代表了一个完整的可能的换乘路径。整个 mySolutions 数组包含了所有可能的换乘路径。

在接下来的代码中，通过调用 arrangeAll 函数来安排这些可能的换乘路径，以找到最优的换乘方案。这可能涉及到多条线路之间的组合，从而获得最小的总换乘成本。最终，算法会输出最小总成本 min\_total\_cost 和最佳换乘路径 best\_exchange\_way。

3.2.3 queue队列

queue的具体作用是实现广度优先搜索（BFS）的过程。从起始节点开始，逐层遍 历图中的节点，直到找到目标节点或者遍历完整个图。该算法中，queue 被用于存储 待处理的路径选择。

1. starts\_line 数组被初始化为空数组，用于存储起始站点所在的每条地铁线的 Selection 对象。

2. 对于每一条地铁线，检查起始站点是否在该线路上。如果是，就创建一个 Selection 对象，表示选择该线路的起始站点，并将它添加到 starts\_line 数组中。

3. 将 starts\_line 数组的内容复制到 queue 数组中，初始化 minExchange 变量为一个足够大的值（假设为 MAX）。

4. 在一个循环中，从 queue 中取出一个 cur 对象，表示当前的路径选择。如果当前选择的高度（cur.height）超过了当前已知的最小换乘次数（minExchange），则终止循环。

5. 检查当前选择是否在终点所在的线路上，如果是，并且当前选择的高度小于等于当前已知的最小换乘次数，则更新最小换乘次数和可能的换乘路径。

6. 对于当前选择的站点，遍历所有相邻的站点，检查是否可以进行换乘，即检查是否存在一个邻接矩阵中的对应关系，并且该邻接站点不在当前路径中。如果符合条件，则创建一个新的 Selection 对象，表示将当前路径选择扩展到该邻接站点，并将该新选择加入到 queue 中。

7. 重复直到 queue 为空或者满足终止条件。

# 测试

## 4.1正确运行程序的用例

### 换乘最少



### 用时最短



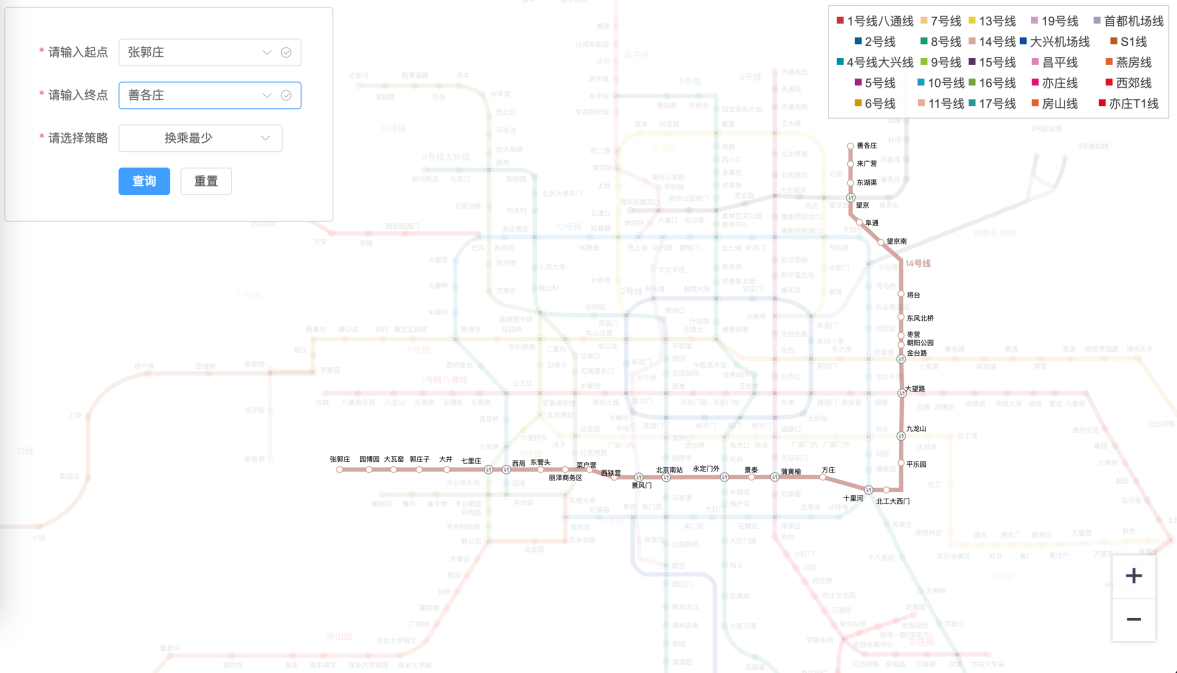
## 4.2 导致程序运行错误的用例





## 4.3 边界数据的用例

点击右上角，单线路所有站点：



# 总结与提高

## 5.1 体会与收获

1. 配置文件的设计：设计合理的配置文件对于程序的灵活性和可扩展性至关重要。通过读取vite全局层级的配置信息，可以更方便地修改获取地铁网络结构，以及处理项目中不同类型的模块，而不必修改源代码。这体现了良好的设计思想和解耦能力。

2. 数据结构选择：选择合适的数据结构至关重要，我使用了带权无向图来表示地铁网络，这种数据结构能够方便地表示站点之间的关系和权重（里程），并为路径计算提供了基础数据。

3. 图形化界面：系统中的图形化界面对使用者来说很重要重要。通过图形界面，用户可以直观地输入起点和终点，查看地铁线路图，并得到路线提示。这为用户提供了便利，增加了系统的易用性。

4. 算法设计：通过实现路径计算的算法，系统可以提供用时最少和换乘最少两种路线提示。这让我对图的遍历和路径搜索算法有了更深入的理解和掌握。

## 5.2开发中遇到的问题与解决情况

1. 数据处理和加载：配置文件的读取和地铁网络的加载可能涉及到数据格式的解析等问题。在开发过程中，遇到了非常多数据格式处理的问题，尤其是百度api提供的原始数据非常繁琐复杂，之后设计了不同的预处理函数，在重新设计和使用适当的数据结构，这些问题都得到了解决。

2. 图形化界面实现：实现图形化界面需要使用图形库或框架，比如项目里的d3，vuejs等，涉及到一些界面元素的布局、事件处理等方面的技术难题，虽然从未接触过d3，但是通过学习参考，最终还是成功加载出地铁线路图。

3. 路径计算算法的设计选择：路径计算涉及到图的遍历和搜索，算法的效率和数据结构的设计选型可能会影响程序的运行效果。在开发过程中，中间使用BFS + 无权图设计后，发现达不到理想的需求，最后重构之后使用邻接矩阵+BFS以及dijisktra成功完成了预期的最少换乘和时间最短功能。