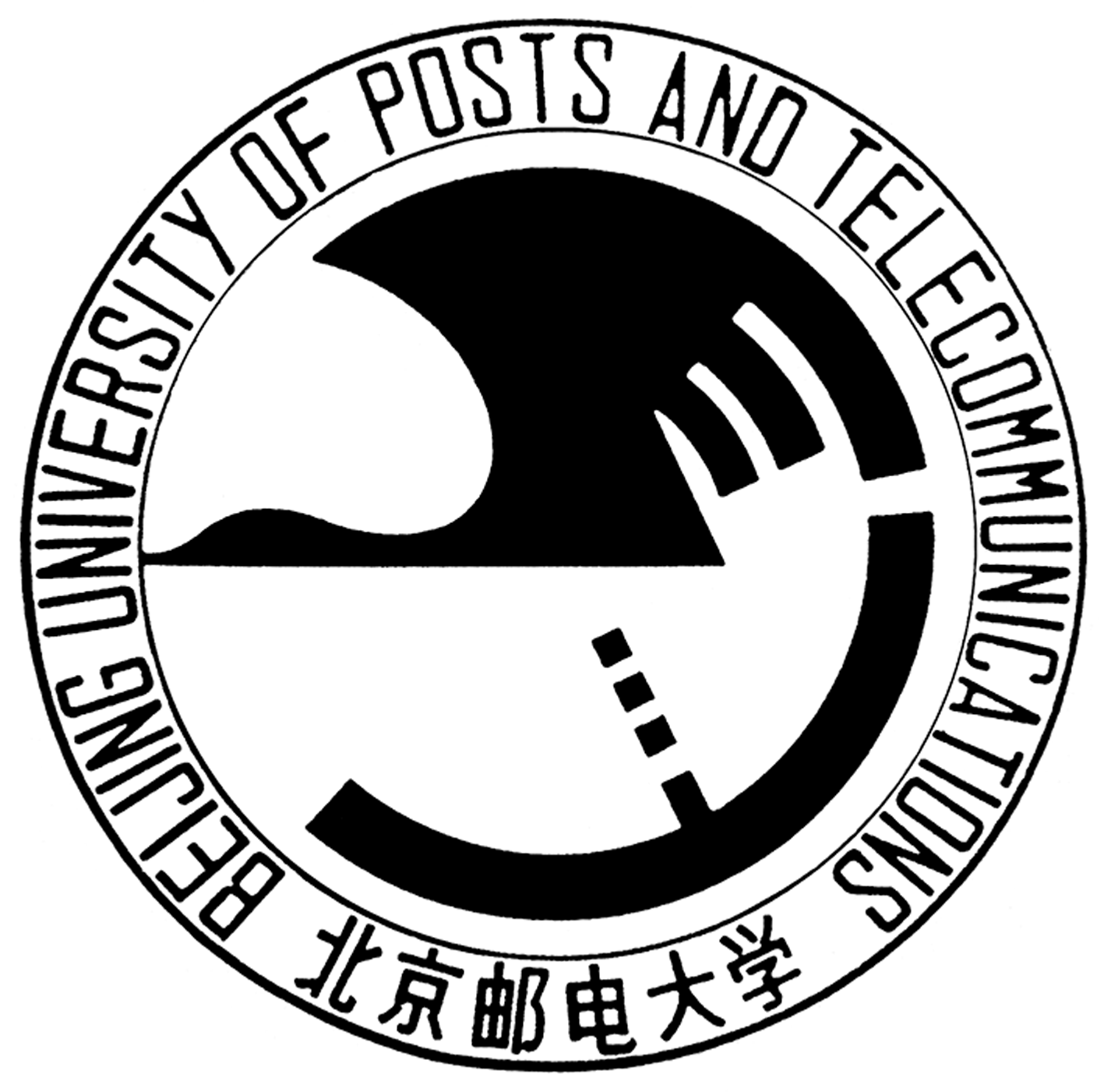


**数据结构课程设计**



**姓 名**  **吴黄璇**

**学 院 人工智能学院**

**专 业 人工智能**

**班 级 11班**

**学 号 2022212043**

**班内序号 4**

**指导教师 刘咏彬**

**2023年 8月**

目录

[题目要求 3](#_Toc145201800)

[本项目亮点 3](#_Toc145201801)

[1.几天快速学会了gradio框架，做了用户友好的WebUI 3](#_Toc145201802)

[查询界面： 3](#_Toc145201803)

[删除界面： 5](#_Toc145201804)

[2.实现了难度较大的最短时间算法 5](#_Toc145201805)

[3.创造性地使用一套算法完成两个任务 5](#_Toc145201806)

[使用的所有数据结构 6](#_Toc145201807)

[所用算法 7](#_Toc145201808)

[整体设计思路 8](#_Toc145201809)

[数据读入模块 8](#_Toc145201810)

[WebUI启动模块 8](#_Toc145201811)

[最短时间查询模块 9](#_Toc145201812)

[最小换乘查询模块 11](#_Toc145201813)

[删除路线模块 14](#_Toc145201814)

[添加路线模块 16](#_Toc145201815)

[个人成长和感想 18](#_Toc145201816)

[开发心路历程（包含遇到的问题和解决方案） 18](#_Toc145201817)

[项目开始时写下的一段话 18](#_Toc145201818)

[机缘与巧合：用Python写项目的原因 19](#_Toc145201819)

[纠结与选择：做最短时间or最短路径？ 20](#_Toc145201820)

[遇到的问题一：读入数据的问题 21](#_Toc145201821)

[遇到的问题二：如何判断这站是否换乘 24](#_Toc145201822)

[遇到的问题三：如何将换乘一边考虑迪杰斯特拉 25](#_Toc145201823)

[灵光一闪：一个算法完成两个任务！ 27](#_Toc145201824)

[用Gradio开发前端 27](#_Toc145201825)

[测试与改bug 28](#_Toc145201826)

[最后，撰写使用手册和录制demo视频 28](#_Toc145201827)

[文档末尾，我要感谢各位提供数据，探讨思路，帮我改代码等等等等的各位大佬以及耐心解答同学问题的老师和助教！ 28](#_Toc145201828)

**数据结构课程设计大作业**

# 题目要求

当一个用户从甲地到乙地时，由于不同需求，就有不同的交通路线，有人希望以最短时间到达，有人希望用最少的换乘次数等。请编写一北京地铁线路查询系统，通过输入起始站、终点站，为用户提供两种决策的交通咨询。

设计要求：

1.提供对地铁线路进行编辑的功能，要求可以添加或删除线路。

2.提供两种决策：最短时间，最少换乘次数。

3.中途换乘站换乘耗时为5分钟，地铁在除始发站外每一站停留1分钟。

4.按照始发站时间、地铁时速及停留时间推算之后各个线路的地铁到站时间。

5.该系统以人机对话方式进行。系统自动获取当前时间，用户输入起始站，终点站以及需求原则（需求原则包括最短时间，最少换乘次数），系统输出乘车方案：乘几号线，距离，时间，费用，换乘方法等相关信息。

# 本项目亮点

## 1.几天快速学会了gradio框架，做了用户友好的WebUI

我基于python的gradio框架开发了一个WebUI。上面有四个选项卡，分别可以实现最短时间方案查询，最少换乘方案查询，删除线路以及添加项目功能。

### 查询界面：

我在网页左侧做了地铁线路图，右侧是查询栏。这样用户在使用的时候边可以将查询到的方案和地图对照，更加直观。

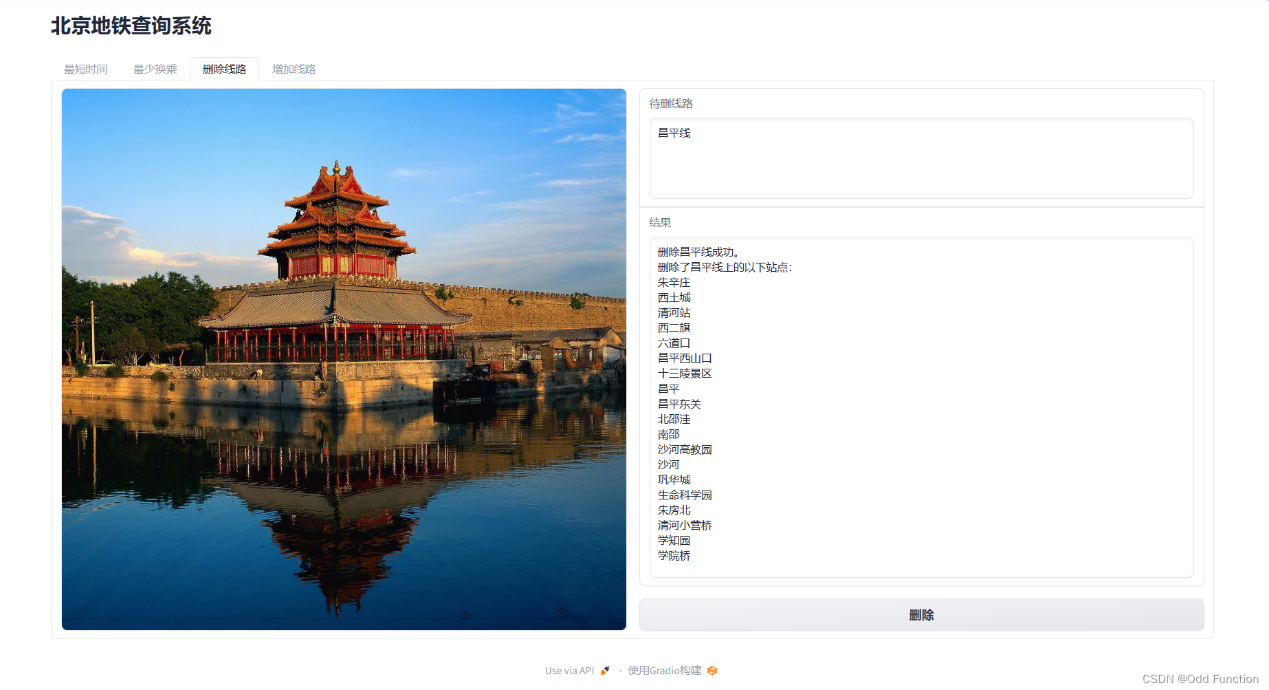
此外，右侧还输出了票价，距离，乘车时间等信息，帮助用户更好规划路径。



此外，我还做了良好的报错机制。



### 删除界面：



## 2.实现了难度较大的最短时间算法

相比最短路径，最短时间算法的难点在于：

**1.数据量增多。**需要爬取线路运行速度，以及注意一些细节，例如一号线的八通线段和原一号线段的运行时速是不一样的。

**2.要对是否换乘修改逻辑。**最短路径的方案不一定是最短时间的方案。由于还要考虑换乘的五分钟，所以还要判断在换乘站是换乘还是绕远路时间更合算。我对这个问题的解决思路是：记录下换乘站点，换乘站点前面的站点和换乘站之后的站点。之后，对前面和后面站点经过的路线依次进行比较：如果都没有相同的路线，就必须要换乘。否则，就按照不换乘的方案计算，最后由迪杰斯特拉算法取最短路径。

## 3.创造性地使用一套算法完成两个任务

写完最短换乘时间的算法之后，我对最少换乘站这个任务进行了思考：我要用BFS还是DFS，还是其他方法？

在这个过程，我突然灵光一闪！——其实求最短换乘站的方案也可以用迪杰斯特拉算法。唯一需要更改的就是：将换乘时间调的非常非常大！这样子最短时间的方案就是最小换乘站的方案了。

于是，我将迪杰斯特拉算法加以修改，并且取得了很好的成效。

# 使用的所有数据结构

import math  
import gradio as gr  
import datetime  
# 获取当前时间, 其中中包含了year, month, hour, 需要import datetime  
  
  
# 定义节点的结构体：  
class Node:  
 def \_\_init\_\_(self, num, name):  
 self.num = num # 记录站点的编号  
 self.name = name # 记录站点的名字  
 self.lines = ["" for i in range(5)] # 属于几号线(由于可能有换乘站，所以要用数组存)  
 self.line\_num = 0 # 有几条线通过这个站点  
  
  
class LaunchTime: # 记录每一个发车时间  
 def \_\_init\_\_(self, line\_name, is\_forward):  
 self.line\_name = line\_name # 记录是第几号线的发车时间  
 self.launch\_time = [0.0]  
 self.is\_forward = is\_forward # 记录是往哪个方向发车  
  
  
inf = 100  
total\_station\_num = 0 # 记录全部的站点数  
station = [Node(0, "") for i in range(500)] # 记录各个站点的列表  
time\_map = [[inf for i in range(500)] for j in range(500)] # 建立一个500\*500的二维数组，来存储时间的邻接矩阵.初始化为无穷大（都不相连）  
dist\_map = [[inf for i in range(500)] for j in range(500)] # 建立一个500\*500的二维数组，来存储距离的邻接矩阵.初始化为无穷大（都不相连）  
S = [0 for i in range(500)] # 记录下第i个节点是否有被添加  
Time = [inf for i in range(500)] # 记录下起点到顶点i的时间  
Dist = [inf for i in range(500)] # 记录下起点到顶点i的路径长度  
Path = [0 for i in range(500)] # 记录下起点到顶点i的路径  
whether\_to\_transfer = [False for i in range(500)] # 用来记录在某条特定线上，哪些站需要换乘  
have\_been\_deleted = [[False, ""] for i in range(500)] # 用来记录某个站点是否被删除过.记录下：是否被删除，从哪条线被删除

# 所用算法

在求最短时间和最少换乘站的时候，我都用了迪杰斯特拉算法。

迪杰斯特拉算法的主要思想是：

1. 初始化：创建一个集合用于存储已经找到最短路径的节点，以及一个集合用于存储尚未找到最短路径的节点。初始化距离数组，记录从源节点到每个节点的最短距离，初始时源节点距离为0，其他节点距离设为无穷大。
2. 选择起始节点：从尚未找到最短路径的节点中选择一个距离最短的节点作为当前节点，将其标记为已找到最短路径。
3. 更新距离数组：对于当前节点的每个邻接节点，计算从起始节点经过当前节点到达邻接节点的距离，如果这个距离小于邻接节点当前记录的最短距离，更新邻接节点的最短距离。
4. 重复步骤2和步骤3，直到所有节点都被标记为已找到最短路径，或者找到目标节点的最短路径。
5. 输出最短路径：通过距离数组，可以得到从源节点到所有其他节点的最短距离。如果需要找到具体的最短路径，可以从目标节点开始，回溯距离数组，沿着最短路径逐步反向查找，直到达到源节点为止。

**此外，我在更新距离数组的时候加了我自己设计的判断是否换乘的算法。**详细来说，整个算法是这样的：记录下换乘站点，换乘站点前面的站点和换乘站之后的站点。之后，对前面和后面站点经过的路线依次进行比较：如果都没有相同的路线，就必须要换乘。否则，就按照不换乘的方案计算，最后由迪杰斯特拉算法取最短路径。

在最后计算换乘站数量的任务中，我同样采用了这一套算法，取得了准确获得换乘站的效果。

# 整体设计思路

### 数据读入模块

首先，我将所有数据都存在source//subway.txt文件中，并且调用load\_station\_data函数从数据集中读取所有数据，存储到了：

以Node为类型的station列表

表示时间的邻接矩阵time\_map

表示距离的邻接矩阵dist\_map

表示所有站点数的total\_station\_num

加载完所有数据后，屏幕上会显示：

​

具体代码见main.py文件。

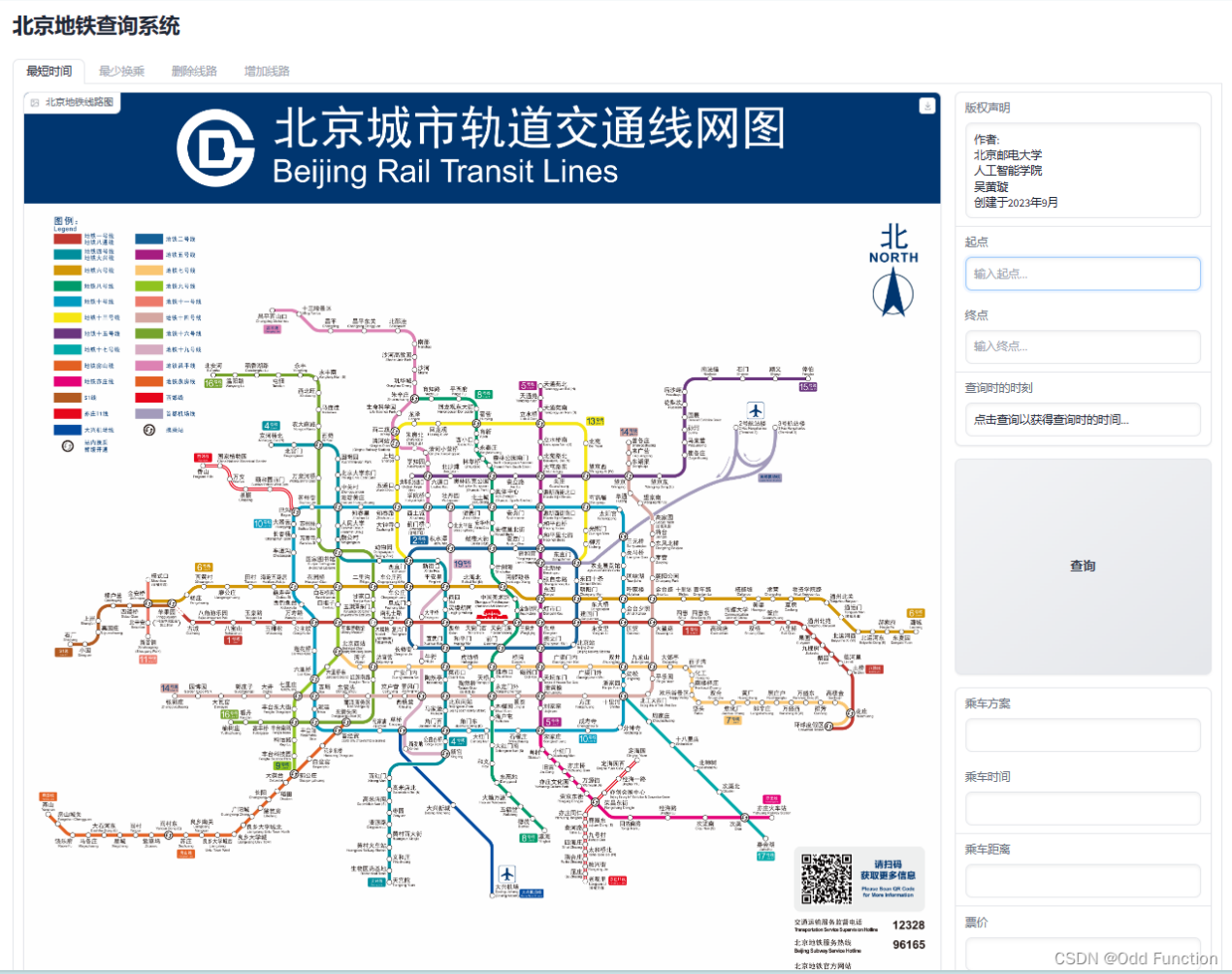
### WebUI启动模块

接着，会调用visualize函数，来基于gradio框架生成一个课用于实现各种功能的WebUI。当生成完成的时候，会出现如下界面：



此时，点击本地的URL，就可以启动WebUI了。

启动后的示例：



具体代码见main.py文件。

### 最短时间查询模块

在页面上方选择最短时间：

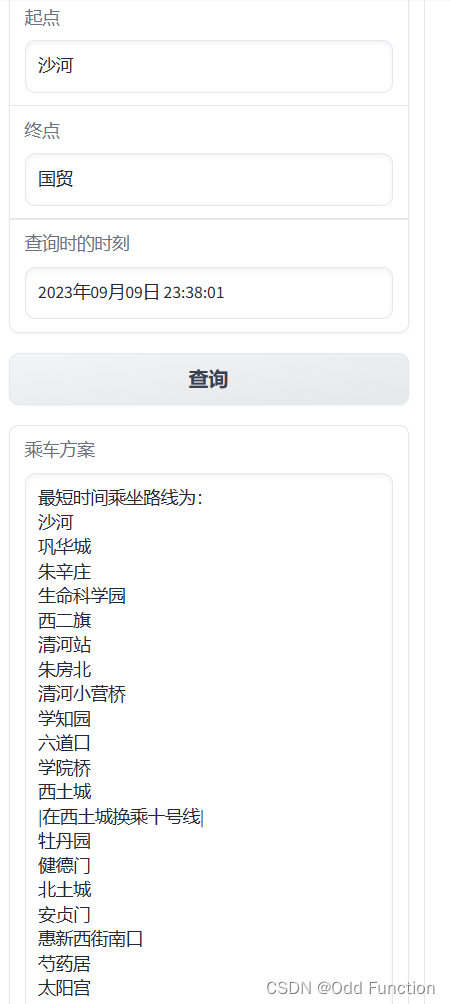


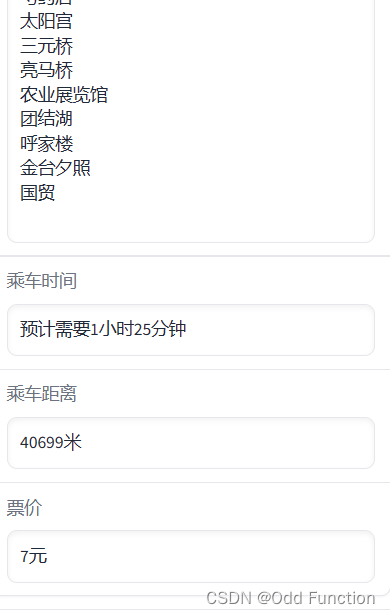
输入起点和终点，并且点击“查询”：

此时，后端首先会调用get\_time函数，自动读取系统的时间，然后将结果返回前端。

之后，后端将会调用short\_time函数。在这个函数中，主要执行迪杰斯特拉算法和判断换乘算法，接着在函数内调用print\_way\_min\_time函数来得到乘车方案，乘车时间，乘车距离，票价等信息，以string类型返回到前端的相应框内。在print\_way\_min\_time函数内，会同时调用count\_transfer\_station函数，计算出换乘几站的同时，给出在哪个站点换乘的提示。

结果演示：

****​

****​

具体代码见main.py文件。

### 最小换乘查询模块

在页面上方选择最少换乘：

​编辑

输入起点和终点，并且点击“查询”：

此时，后端首先会调用get\_time函数，自动读取系统的时间，然后将结果返回前端。

之后，后端将会调用min\_transfer函数。在这个函数中，主要执行迪杰斯特拉算法和判断换乘算法，接着在函数内调用print\_way\_min\_transfer函数来得到乘车方案，乘车时间，乘车距离，票价等信息，以string类型返回到前端的相应框内。在print\_way\_min\_transfer函数内，会同时调用count\_transfer\_station函数，计算出换乘几站的同时，给出在哪个站点换乘的提示。

结果演示：

​

​

**测试发现：虽然说最小换乘和最短时间是两种方案，但是最后得出的路线都几乎一样。（目前只发现从大钟寺到东直门方案是不同的）**

具体代码见main.py文件。

### 删除路线模块

点击上方删除线路的模块

****​编辑

输入要删除的线路，点击删除按钮，后端将调用delete\_line函数。这个函数主要执行删除线路功能，并且以string类型返回删除的站点到前端。

在执行删除功能的时候，我将删除站点的信息存储到了

have\_been\_deleted = [[False, ""] for i in range(500)] # 用来记录某个站点是否被删除过.记录下：是否被删除，从哪条线被删除

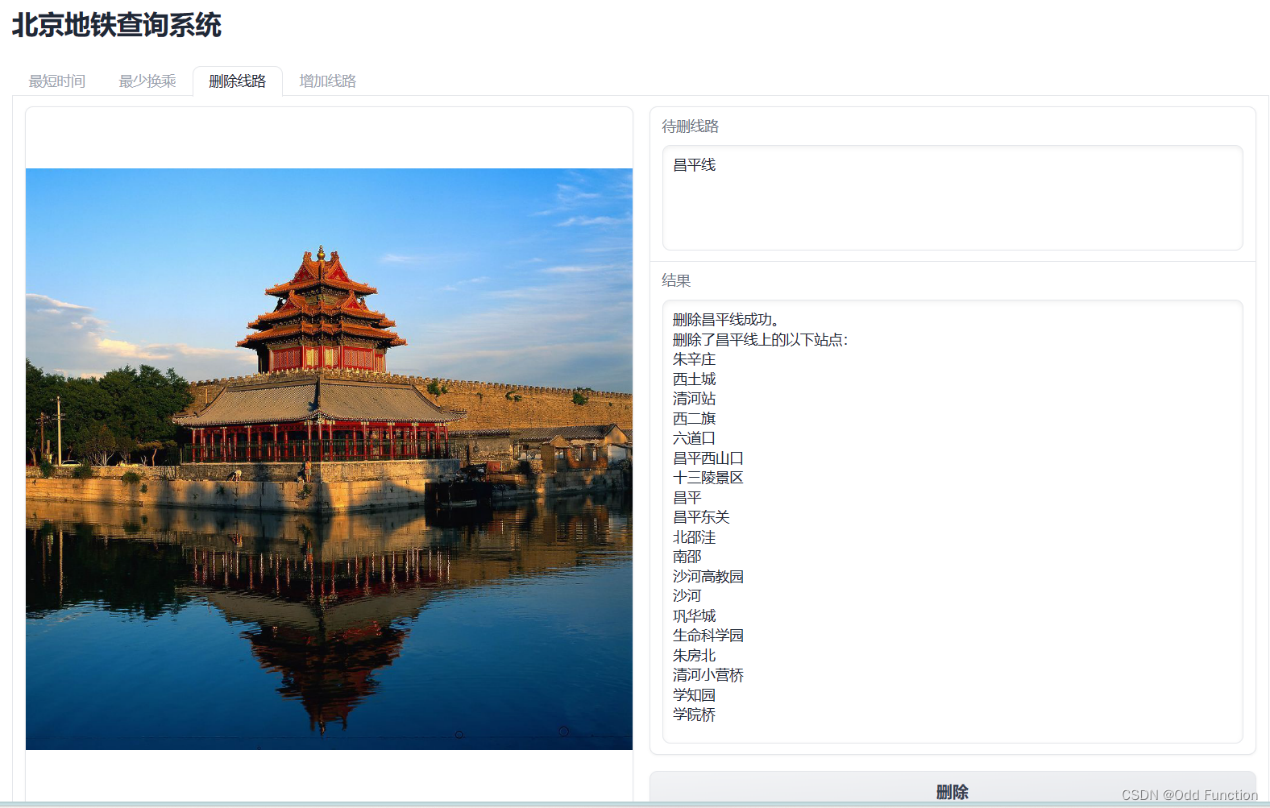
****

这个数组里面。这么做的目的主要是便于后面添加站点的功能。

注意 ：如果某个站点是换乘站，则只会让这个站点没有这条线通过，不会将这个站点彻底删除。

例如 西直门有昌平线和十三号线通过。如果删除昌平线，则西直门还在，只是只有十三号线通过，没有昌平线通过了。

效果演示：



​

具体代码见main.py文件。

### 添加路线模块

点击上方增加线路的模块



输入要添加的线路，点击添加按钮，后端将调用add\_line函数。这个函数主要执行增加线路功能，并且以string类型返回增加的站点到前端。

在执行功能的时候，首先我写了一个判断算法，用来判断要恢复哪些站点。这时候我之前建立的

have\_been\_deleted = [[False, ""] for i in range(500)] # 用来记录某个站点是否被删除过.记录下：是否被删除，从哪条线被删除



数组就派上用场了。关于是否要恢复一个站点，要满足下面两个条件：

1.曾经被删除过

2.曾经是从“待增路线”上删除的

具体代码：

for i in range(total\_station\_num):  
 flag = False # 表示某站点是否应该被恢复  
 if have\_been\_deleted[i][0]: # 如果有被删除过  
 if have\_been\_deleted[i][1] == line\_name: # 并且是从这条线路上删除的  
 flag = True # 确实应该被恢复  
 # 进行恢复：  
 if flag:  
 station[i].lines.append(line\_name)  
 station[i].line\_num += 1  
 added\_station += station[i].name + "\n"  
return f"添加{line\_name}成功。\n添加了{line\_name}上的以下站点：\n" + added\_station

结果演示：





具体代码见main.py文件。

# 个人成长和感想

* 几天速通gradio框架，学会了前端开发，能将自己的作品可视化
* 完整地感受了整个程序从后端到前端开发的历程
* 对迪杰斯特拉算法理解更深了
* 了解了数据的读入和数据的格式化
* 真的感觉到实际项目和抽象算法需要考虑的东西更多了，也更复杂了
* 掉了不少头发hhh

# 开发心路历程（包含遇到的问题和解决方案）

## 项目开始时写下的一段话

写这篇文章的时候，我还只是北邮大一升大二的学生。大一升大二的这个小学期上数据结构与算法的实践课程，要求10天内开发一个小程序：北京地铁查询系统。

这对我来说有点难度。以前我只会写算法题，但是像开发一个程序，不止要做后面核心的算法，还要做交互界面，最后将交互界面与后面算法的程序连接。好多都是我从来没有接触过的东西。

总之，先写算法吧。前端能写多少算多少。

抱着这样的心态，我开始动手了。

## 机缘与巧合：用Python写项目的原因

一开始，我是用C++写的。写了大概一百多行。

但是我想做个前端，而我不会用Qt。但是正巧这段时间，我参与的实验室项目要我用gradio框架做个前端。

所以我就想：为什么不用gradio做前端呢？这样一来可以为实验室项目做准备，二来可以用于大项目前端，一举两得。

于是我就用Python将已经写的一百多行代码代码重新架构了一遍。

我原本只是学过Python的基础语法，所以用Python重构的过程特别的艰巨折磨。但是也好，让我对Python的语法精进了不少。用一边看书+一边百度+一边问ChatGPT的方法，耗时一两天，我终于重构完了。



## 纠结与选择：做最短时间or最短路径？

求最短时间和最短路径是不同的。最短路径是直接迪杰斯特拉就可以。而最短时间不是简单的迪杰斯特拉。**假如换乘五分钟的话，还要考虑换乘是否值得（因为也许会有换乘虽然少开几站，但是总体时间更长的情况）。所以在用迪杰斯特拉的时候应该把换乘时间考虑进去再迪杰斯特拉。**

**可以看出，最短路径是简单于最短时间的。**

所以很多同学向助教反映做最短时间太难了，申请改需求。

于是助教征求了老师的意见，发起了投票：



结果90%的人选择了更好实现的最短路径（包括我自己选的也是这个hhh）

但是后来我想，既然我抱着学东西和挑战自己的目的来做这个项目，就挑战一下最短时间吧，能做多少做多少吧。

## 遇到的问题一：读入数据的问题

在读入数据的时候，遇到了问题：有些站点由于是换乘站，就意味着它会在很多条线路中都出现。这样每次读入的时候，存在station数组中的位置不同，就会导致算法出错。

我的解决办法：把所有信息都写在了第一次出现的位置。至于如何读取第一次出现的下标，我用的是search\_num函数

def search\_num(station\_name): # 根据站点名称，找出第一次出现的编号。如果没有找到，返回0  
 global total\_station\_num  
 global station  
 global time\_map  
 global dist\_map  
 global Time  
 global S  
 global Dist  
 global Path  
 for i in range(total\_station\_num):  
 if station[i].name == station\_name:  
 return i  
 return 0

然后存入信息的时候，我写了个这样的逻辑：

def load\_station\_data():  
 # 声明全局变量  
 global total\_station\_num  
 global station  
 global time\_map  
 global dist\_map  
 global Time  
 global S  
 global Dist  
 global Path  
 # 建立一个列表，来储存所有单词：  
 words = []  
 with open('source//subway.txt', 'r') as data: # 将文件打开为data  
 words = data.read().split() # 将文件中的所有单词写入words列表  
 for i in range(500):  
 words.append("") # 对words进行一个扩容，防止接下来访问越界  
 line\_num = int(words[0]) # 记录下文件里面的第一个单词（表示总共有几条地铁线）  
  
 word\_index = 1 # 记录下正在读取第几个单词  
 # 对每一条线进行遍历：  
 for i in range(line\_num):  
 # 读取这条地铁线的名字  
 line\_name = words[word\_index]  
 word\_index += 1  
  
 # 读取这条地铁线中有几个站点  
 station\_num = int(words[word\_index]) # 将节点数写入station\_num  
 word\_index += 1  
  
 # 读取地铁线的速度  
 velocity = float(words[word\_index])  
 word\_index += 1  
  
 # 对该线路的每个站点进行遍历  
 for j in range(total\_station\_num, total\_station\_num + station\_num):  
 station\_name = words[word\_index]  
 word\_index += 1  
 station[j].name = station\_name  
 t = search\_num(station\_name)  
  
 if t != 0: # 如果当前站是换乘站  
 station[t].lines[station[t].line\_num] = line\_name  
 station[t].line\_num += 1  
 dist = float(words[word\_index])  
 tt = search\_num(words[word\_index + 1]) # 搜索一下下一站第一次出现的编号,看一下是不是换乘站  
 if tt != 0: # 如果下一站是换乘站  
 time\_map[t][tt] = time\_map[tt][t] = dist / velocity  
 dist\_map[t][tt] = dist\_map[tt][t] = dist  
 if tt == 0: # 如果下一站不是换乘站  
 time\_map[t][j + 1] = time\_map[j + 1][t] = dist / velocity # 在邻接表中存储时间  
 dist\_map[t][j + 1] = dist\_map[j + 1][t] = dist  
  
 if t == 0: # 如果当前站不是换乘站  
 station[j].lines[station[j].line\_num] = line\_name  
 station[j].line\_num += 1  
 dist = float(words[word\_index])  
 tt = search\_num(words[word\_index + 1]) # 搜索一下下一站第一次出现的编号,看一下是不是换乘站  
 if tt != 0: # 如果下一站出现过了（是换乘站）  
 time\_map[j][tt] = time\_map[tt][j] = dist / velocity  
 dist\_map[j][tt] = dist\_map[tt][j] = dist  
 if tt == 0: # 如果下一站没出现过（不是换乘站）  
 time\_map[j][j + 1] = time\_map[j + 1][j] = dist / velocity # 在邻接表中存储时间  
 dist\_map[j][j + 1] = dist\_map[j + 1][j] = dist  
 word\_index += 1  
 total\_station\_num += station\_num  
 print("-----------------")  
 print("所有站点加载完毕!")  
 print("-----------------")

值得注意的是，读取下一个站也要判断下一个站是不是换乘站。这会影响到存入邻接表的位置。

这个时候遇到了问题：读取word\_index+1的时候，words数组会越界。

于是我打了一个小补丁：

words = []  
with open('source//subway.txt', 'r') as data: # 将文件打开为data  
 words = data.read().split() # 将文件中的所有单词写入words列表  
 for i in range(500):  
 words.append("") # 对words进行一个扩容，防止接下来访问越界  
line\_num = int(words[0]) # 记录下文件里面的第一个单词（表示总共有几条地铁线）

这样，这个问题就算顺利解决了

## 遇到的问题二：如何判断这站是否换乘

在判断此站是否换乘的时候，我自己发明了一个算法：

def count\_transfer\_station(end\_num): # 计算一条线路上换乘站的数量  
 transfer\_station\_num = 0  
 stack = [station[end\_num].name] # 用队列来模拟栈,并且先把终点站入栈  
 pre = Path[end\_num]  
 while pre != -1:  
 stack.append(station[pre].name)  
 pre = Path[pre]  
 if len(stack) != 0:  
 pre\_station = stack.pop()  
 if len(stack) != 0:  
 now\_station = stack.pop()  
 if len(stack) != 0:  
 next\_station = stack.pop()  
 while len(stack) != 0:  
 if station[search\_num(now\_station)].line\_num > 1: # 如果当前是换乘站  
 flag = True # 表示是否换乘.True表示换乘  
 # 比较之前和之后的两个站点，看看是不是真的换乘了。如果换乘了，前后两个站点应该完全没有一样的线路  
 for i in range(station[search\_num(pre\_station)].line\_num):  
 for j in range(station[search\_num(next\_station)].line\_num):  
 if station[search\_num(pre\_station)].lines[i] == station[search\_num(next\_station)].lines[j]:  
 flag = False # 一旦有相同的线，就表示没有换乘。旗帜倒下。  
 if flag: # 如果没有倒下  
 transfer\_station\_num += 1 # 换乘站数量+1  
 whether\_to\_transfer[search\_num(now\_station)] = True  
 pre\_station = now\_station  
 now\_station = next\_station  
 next\_station = stack.pop()  
 return transfer\_station\_num

****

## 遇到的问题三：如何将换乘一边考虑迪杰斯特拉

求最短时间和最短路径是不同的。最短路径是直接迪杰斯特拉就可以。而最短时间不是简单的迪杰斯特拉。假如换乘五分钟的话，**还要考虑换乘是否值得（因为也许会有换乘虽然少开几站，但是总体时间更长的情况）。所以在用迪杰斯特拉的时候应该把换乘时间考虑进去再迪杰斯特拉。**

**实现思路：**

我对这个问题的解决思路是：记录下换乘站点，换乘站点前面的站点和换乘站之后的站点。之后，对前面和后面站点经过的路线依次进行比较：如果都没有相同的路线，就必须要换乘。否则，就按照不换乘的方案计算，最后由迪杰斯特拉算法取最短路径。

def short\_time(s, e): # 计算最短时间  
 global station  
 gr.Info("开始查询")  
 start = search\_num(s)  
 end = search\_num(e)  
 if start == 0 or end == 0:  
 raise gr.Error("输入不合法，请重新输入")  
 return  
 if station[start].line\_num == 0 or station[end].line\_num == 0:  
 raise gr.Error("输入不合法，请重新输入")  
 return  
 if start == end:  
 raise gr.Error("起点和终点不能相同")  
 return  
 global total\_station\_num  
 global time\_map  
 global dist\_map  
 global S  
 global Dist  
 global Time  
 global Path  
 # 初始化辅助数组：  
 for i in range(total\_station\_num):  
 S[i] = False # 都标记为没有被添加  
 Time[i] = time\_map[start][i] # 将与起点直接连接的加入Time  
 Dist[i] = dist\_map[start][i] # 将与起点直接连接的加入Dist  
 if Time[i] != inf:  
 Path[i] = start  
 else:  
 Path[i] = -1 # 一开始初始化为无前驱  
 S[start] = True # 起点到起点的最短路径已经被找到  
 Time[start] = 0 # 起点到起点时间为0  
 Dist[start] = 0 # 起点到起点距离为0  
  
 for i in range(total\_station\_num):  
 v = find\_min()  
 if find\_min() == -1: # 如果全是不相邻的  
 return  
 S[v] = True  
 for j in range(total\_station\_num): # 查找到j的距离,并更新  
 pre\_station = Path[v] # 记录下之前的站点  
 now\_station = v # 记录下当前的站点  
 next\_station = j # 记录下下一个站点  
  
 # 接下来这个模块是判断是否换乘的：  
 if station[now\_station].line\_num > 1: # 如果当前是换乘站  
 flag = True # 表示是否换乘.True表示换乘  
 # 比较之前和之后的两个站点，看看是不是真的换乘了。如果换乘了，前后两个站点应该完全没有一样的线路  
 for k in range(station[pre\_station].line\_num):  
 for m in range(station[next\_station].line\_num):  
 if station[pre\_station].lines[k] == station[next\_station].lines[m]:  
 flag = False # 一旦有相同的线，就表示没有换乘。旗帜倒下。  
 if pre\_station == "西二旗" and next\_station == "上地": # 对这两站的特判  
 flag = True  
 if flag: # 如果换乘了，应该考虑进换乘时间0.083小时  
 if S[j] != 1 and Time[j] >= time\_map[v][j] + Time[v] + 0.083: # 如果找到了更短的，更新  
 Time[j] = time\_map[v][j] + Time[v] + 0.083  
 Dist[j] = dist\_map[v][j] + Dist[v]  
 Path[j] = v  
 else: # 如果没换乘，就不考虑换乘的时间。  
 if S[j] != 1 and Time[j] >= time\_map[v][j] + Time[v]: # 如果找到了更短的，更新  
 Time[j] = time\_map[v][j] + Time[v]  
 Dist[j] = dist\_map[v][j] + Dist[v]  
 Path[j] = v  
 else: # 如果当前站不是换乘站，就按照正常的来  
 if S[j] != 1 and Time[j] >= time\_map[v][j] + Time[v]: # 如果找到了更短的，更新  
 Time[j] = time\_map[v][j] + Time[v]  
 Dist[j] = dist\_map[v][j] + Dist[v]  
 Path[j] = v  
 string\_way, string\_time, string\_dist, string\_price = print\_way\_min\_time(end)  
 return string\_way, string\_time, string\_dist, string\_price

**在此，我要感谢李屹轩大佬对这个算法的指点！**

## 灵光一闪：一个算法完成两个任务！

写完最短换乘时间的算法之后，我准备着手写最少换乘站的算法。

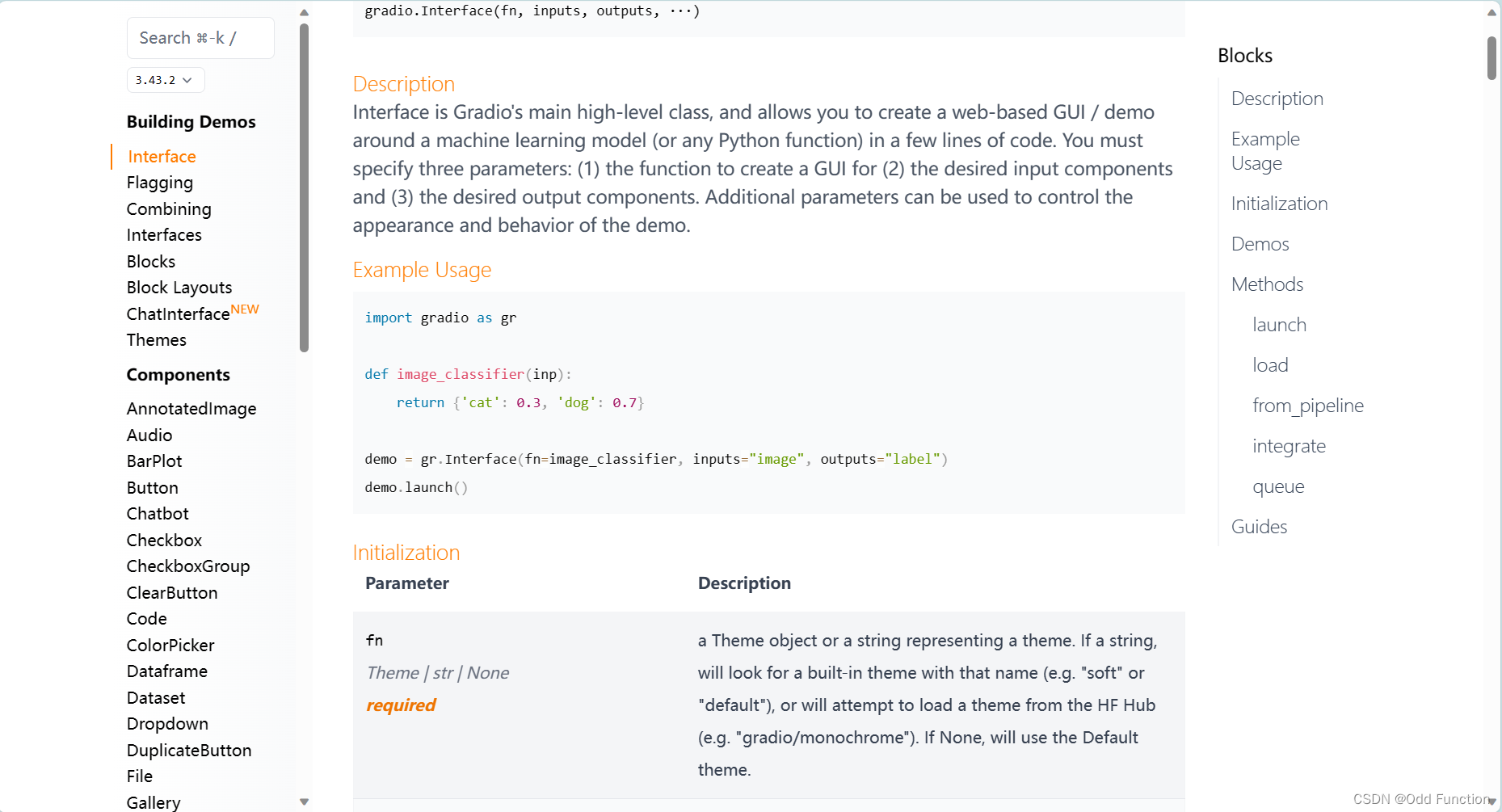
但是动手之前，我先对这个任务进行了思考：我要用BFS还是DFS，还是其他方法？

在这个过程，我突然灵光一闪！——其实求最短换乘站的方案也可以用迪杰斯特拉算法。唯一需要更改的就是：将换乘时间调的非常非常大！这样子最短时间的方案不就是最小换乘站的方案了吗？！

于是我便开始动手，最后取得了很好的效果

## 用Gradio开发前端

在几天速通gradio框架的时候，我一直硬啃英文原版官方文档，然后不断将官网上的demo放在本机跑。还学习了各种方法接口的调用。



感觉这种阅读官方文档的能力在我以后用学习pytorch等框架上也可以用得上。我现在已经信心满满了hhh

## 测试与改bug

这边主要是发现一些小错误，以及利用raise gradio.Error（）方法增加了在前端报错的机制。

## 最后，撰写使用手册和录制demo视频

## 文档末尾，我要感谢各位提供数据，探讨思路，帮我改代码等等等等的各位大佬以及耐心解答同学问题的老师和助教！