# Subway Route Planning

## 1、Participants and assignment

曹宏(Hong Cao, 20213802033)

Responsible for shortest route query function and processing of data.

廖若凡(Ruofan Liao, 20213802008)

Responsible for subway line map drawing and UI interface.

## 2、Problem description and basic requirements

Please describe which basic tasks are finished and which extended tasks are further accomplished.

**Basic Tasks:**

According to the assignment, the program should make a UI interface for users to check subway information. When users enter the names of two sites, the program should return the shortest route and between them and show it on the subway map.

**Further Accomplished:**

All the requirements in assignments have been finished, by using **tkinter** and **ttkbootstrap**, the program will create the UI interface for users, if the entered site name did not exist in the subway map, the interface will show the warning that site do not exist (Figure 1).


(Figure 1)

The basic tasks only ask to consider line1, line2 and line3, and the route length between each station can be self-defined. However, we use the api of Baidu Map to get the longitude and latitude of all the metro stations in Guangzhou, calculate the distance between each site to get the approximate route length and save those information into a .pkl document by **pickle** in python.

Additionally, not only can it provide the shortest route between two sites, it also provide a button to check the route of least number of subway changes.

What's more, consider the time it takes to get from Baidu Map api, create subway map and generate the .html document by **plotly**.

Last, before the program open the html document, we edit it in Python. We add some **html** and **css** segments in it, to show the path of subway and beautify the web interface (Figure 2).

```
# 添加网页css样式
styles=['border-radius:12px;',
        'padding:6px 10px;',
        'margin:8px 10px;',
        'background-color:#84AF9B;',
        'font-weight:500;',
        'font-family:幼圆;',
        'font-size:1.3em;',
        'color:#fff;',
        ]
css="style='{0}'".format(''.join(styles))
# 在html中插入如下html标签
content_add = '<div {0}>推荐路线：{1}</div><pre><div {0}">{2}</div></pre>'
# 插在<body>标签后
pos = content.find( "<body>")+18
# css全局样式设置
body_css='style="height:92vh;"'
```

(Figure 2)

## 3、Tools and the knowledge used

2.1 Introduce the data structures and algorithms knowledge used in this experiment.

We used **undirected weighted graph**, **adjacency list** data structure and **dijksrta** algorithm in the program.

2.2 Development based on Visual Studio Code.

## 4、Analysis and implementation

4.1 Please analyze the implementation method for the experiment (based on the basic task requiment).

**Task 1: graph (which can be simulated by yourself), with the vertices indicating each station of the Guangzhou metro, the edges indicating the roads between the stations, and the weights on the edges indicating the distances or cost.**

The graph dictionary is used in conjunction with the adjacency class to store the graph in the form of an adjacency list, in which key-value pairs are formatted as {site name: site object}.

The attributes of the **adjacency class** include: site name (name), the dictionary of the adjacency list (subgraph), and the list of the lines to which it belongs (line).

The **get_graph()** function is used to traverse the preData dictionary containing Guangzhou Metro data to store the graph in the form of adjacency list (the graph stored in the form of adjacency matrix takes up large space when there are a large number of Guangzhou Metro stations, so the graph is stored in the form of adjacency list) :

Iterate over the value of 'station' key in the preData dictionary, which is the array of Guangzhou metro stations.

1. Add a key-value pair of the form {site name: site object} to the dictionary when the traversed site does not exist.
2. Add the station's line (the 'line' key stored in the preData dictionary) to the list of the line property of the station object.
3. Determine whether the current station and the next station belong to the same subway line, if so, add the key-value pair of the stations in the format of {station name: station distance} to the adjacency list dictionary to establish the edge between the two points (Figure 3).

```
graph = {}
# 遍历广州地铁站以生成图的邻接表
for i in range(len(preData['Adjacent.Station'])):
    # 若字典中不存在当前键，则增加格式为(站点名:站点对象)的键值对；若存在，则将对应的地铁线路添加到站点对象的line属性的列表中
    if graph.get(preData['Adjacent.Station'][i], None) is None:
        graph[preData['Adjacent.Station'][i]] = Adjacent.Station(preData['Adjacent.Station'][i])
    graph[preData['Adjacent.Station'][i]].line.append(preData['line'][i])
    # 判断当前站点是否与下一个站点属于同一条地铁线，若属于则互相将站点以格式为(站点名:站点距离)的键值对添加到邻接表字典中
    if i != len(preData['Adjacent.Station']) - 1 and preData['line'][i] == preData['line'][i + 1]:
        stationDistance = geodesic((float(preData['latitude'][i]), float(preData['longitude'][i])),
                                   (float(preData['latitude'][i + 1]), float(preData['longitude'][i + 1]))).m
        graph[preData['Adjacent.Station'][i]].subgraph[preData['Adjacent.Station'][i + 1]] = stationDistance
        if graph.get(preData['Adjacent.Station'][i + 1], None) is None:
            graph[preData['Adjacent.Station'][i + 1]] = Adjacent.Station(preData['Adjacent.Station'][i + 1])
        graph[preData['Adjacent.Station'][i + 1]].subgraph[preData['Adjacent.Station'][i]] = stationDistance
```

(Figure 3)

**Task 2: The route map formation can be restored in a files for the easy access.**

After completing the traversal of the implementation code of Task 1, the graph dictionary is the adjacency list we need, and we export the graph as a pickle file by importing the pickle module.

Each time the **main()** function is run, it will check whether there is a graph.pkl file in the same directory, if there is one, it will directly import the file to get the graph dictionary, if not, it will call the **get_graph()** function to return the graph and then export it as a pickle file (Figure 4).

```
# 检查当前目录下是否存在广州地铁路线图的pikcle文件，存在则导入以邻接表存储的地铁图(graph)，不存在则调用get_graph()函数生成
if not os.path.exists('./graph.pkl'):
    graph = graphFile.get_graph()
    pklfile = open('graph.pkl', 'wb')
    pickle.dump(graph, pklfile)
    pklfile.close()
else:
    pklfile = open("graph.pkl", 'rb')
    graph = pickle.load(pklfile)
    pklfile.close()
```

(Figure 4)

**Task 3: Enter the names of two sites to get their shortest routes; if they are not accessible, you show "the unreachability of the two sites" .**

Because the subway graph belongs to the **undirected and weighted connected graph**, and only needs the shortest path from the start point to the end point, it belongs to the dense graph, and there are no negative weight edges and negative weight loops, so the **dijksrta** algorithm is used to find the single source shortest path of the graph

1. Initialize a marker dictionary (to indicate whether the shortest path length of the current station is certain), a distance dictionary (to record the shortest path length from the starting

point to each point), and a path dictionary (to record the name of the second-to-last station with the shortest path from the starting point to the current point).

2. Update the value in the distance dictionary of the two points with an edge as the weight of the edge, and the value in the distance dictionary of the two points without an edge as infinity. Mark the value in the distance dictionary of the starting point as 1 (indicating that it has been determined).
3. Find the closest station from the starting point in the undefined (unlabeled) station, mark the value in the distance dictionary of this point as 1 (indicating that it has been determined), relax the path dictionary, if there is an edge from the point to any point, calculate the shortest distance from the starting point to the point plus the weight of the edge, if this number is less than the value of the key in the current distance dictionary, Update the value;
4. Repeat step 3 until all sites are marked.

**Task 4: The route should be displayed with the path length (time OR distance);**

In the case of the shortest path, the **dijksrta()** function corresponds to the end point of the distance dictionary.

When finding the path with the minimum number of transfers, the length of the path with the minimum number of transfers is obtained by the **get_distance()** function. This function traverses the path and accumulates the distance(the value of the adjacency list dictionary) of the two stations before and after to obtain the total length of the path (Figure 5).

```
def getDistance(path, graph):
    disSum = 0
    for i in range(len(path)):
        if i != len(path) - 1:
            disSum += graph[path[i]].subgraph[path[i + 1]]
    return disSum
```

(Figure 5)

**Task 5: In addition to the basic functions mentioned above, extended functions, like routes with the least number of subway changes are also encouraged.**

Implement least number of subway changes algorithm:

1. Taking subway lines as points, the undirected graph of subway lines is obtained by connecting the points of connected subway lines
2. Calculate the shortest path from the starting line to the ending line;
3. According to the shortest path obtained in step 2, find the path from the starting point to the transfer to the next line, and complete the path repeatedly.

**Task 6: Finally, for algorithms, you can try to find the defects of the algorithms and improve, then you will naturally improve your comprehension of the algorithms you have learned, and also easily achieve the ability to create new algorithms.**

**Improvement 1:** The last step of the algorithm for finding the minimum number of transfers in Task 5 needs to repeatedly traverse the stations of the same route, which consumes too much time, so we can improve it.

1. On the basis of step 1 of **Task 5**, connect all the stations belonging to the corresponding line with the subway line to get a complex undirected graph (you can directly change the original subway diagram, remove the edges between all stations, and add points of subway line to connect the stations with points of the subway line) (Figure 6).

```
def get_lineGraph(graph):
    # 对原图进行深拷贝以初始化地铁线图
    lineGraph = copy.deepcopy(graph)
    # 将原图中所有站点的边删除,新建以地铁线为名的点,将所有站点与所属地铁线的点相连并将边的权设为1
    for i in graph:
        lineGraph[i].subgraph = {}
        for j in graph[i].line:
            if lineGraph.get(j, None) is None:
                lineGraph[j] = Adjacent.Station(j)
            lineGraph[j].subgraph[i] = 1
            lineGraph[i].subgraph[j] = 1
    return lineGraph
```

(Figure 6)

2. Calculate the shortest path from the beginning to the end in this complex graph;

3. The stations in the obtained path are the starting point and ending point of the subpath, and the subway lines in the path are the lines to which the subpath belongs. According to this information, the subpath can be completed quickly, so as to complete the completion of all paths (Figure 7).

```
# 对dijksrta算法处理后的最少换乘次数路径进行分割,得到子路径起点终点的列表和子路径所属地铁线的列表,并初始化完全路径
stationList = path[::2]
lineList = path[1::2]
transferPath = [stationList[0]]

# 遍历子路径起点终点的列表,对子路径进行补全并添加到完全路径列表中
for i in range(len(lineList)):
    startNum = lineStation[lineList[i]].index(stationList[i])
    endNum = lineStation[lineList[i]].index(stationList[i + 1])
    if startNum > endNum:
        transferPath = transferPath + lineStation[lineList[i]][endNum:startNum][::-1]
    else:
        transferPath = transferPath + lineStation[lineList[i]][startNum + 1:endNum + 1]
return transferPath
```

(Figure 7)

**Improvement 2:** It realizes crawling Guangzhou subway data to generate html file of recommended road map, which is convenient for users to consult intuitively

Implementation method

1. Use the **pre_plot()** function to crawl the relevant data of Guangzhou subway station before finding the shortest path/the path with the least number of transfers to prepare for generating html file.

2. The **plot()** function is used to determine the shortest path/least transfer times path according to the path to obtain multiple subpaths.
3. Invoke **plot_graph()** function to crawl the related data of subpath sites (Figure 8).

```
def plotGraph(site1, site2, path, line, distance, ifTest):
    # 根据起点和终点对路径进行切片,得到子路径
    if site2 is None:
        stationList = path[site1:]
    else:
        stationList = path[site1:site2]
    # 针对爬取的地铁线数据与原图数据不一致的情况进行特殊处理
    if 'APM' in line:
        line = 'APM线' + line[line.index('('):]
    # 根据字符匹配获得子路径所属地铁线的uid
    uid, railway = '', {}
    for railwayLine in station_info_json['content']:
        if line[2:] in railwayLine['line_name'] and not ('佛山' in railwayLine['line_name']):
            uid = railwayLine['line_uid']
            railway = railwayLine
            break
            # 通过uid爬取地铁线数据
    railway_json = requests.get(
        'https://map.baidu.com/?qt=bsl&tps=&newmap=1&uid=%s&c=%s' % (uid, 257)
    )
    railway_json = eval(railway_json.content)  # 将json字符串转为python对象
    # 取出地铁线的线段坐标
    trace_mercator = np.array(
        railway_json['content'][0]['geo'].split('|')[2][: -1].split(','),
        dtype=float
    ).reshape((-1, 2))
```

(Figure 8)

4. Then convert the obtained Mercator projection coordinates into wgs84 encoded coordinates for generating html files
5. Add data to html file generation data in unit of the entire subpath;
6. After the **plot_graph()** function has added the last sub-path, call the plotly library to generate and open the html file for the roadmap (Figure 9).

```
# 若当前子路径终点为路径终点,则生成html文件并打开
if stationList[-1] == path[-1]:
    # 生成html文件
    if distance[0] > 19:
        modeStr = '最短路径'
    else:
        modeStr = '最少换乘次数路径'
    fileName = 'Guangzhou_railway_recommend_%s-%s(%s).html' % (path[0], path[-1], modeStr)
    py.plot(fig, filename=fileName, auto_open=False)
    # 向html文件添加推荐路径文字信息
    for j in lineDic:
        if j == path[0]:
            path[path.index(j)] = '(' + lineDic[j] + ')' + path[path.index(j)]
            continue
        path[path.index(j)] = path[path.index(j)] + '(转' + lineDic[j] + ')'
    if distance[0] > 19:
        # path[-1]=path[-1]+' '+'路径长度：{0}km'.format(int(distance[0])/1000)
        path.append('路径长度：{0}km'.format(int(distance[0]) / 1000))
    else:
        # path[-1]=path[-1]+' '+'换乘次数：{0}次. 路径长度：{1}km'.format(distance[0],int(distance[1])/1000)
        path.append('换乘次数：{0}次   路径长度：{1}km'.format(distance[0], int(distance[1]) / 1000))
    htmlChange(path, fileName)
    # 判断是否为测试模式,是则输出最短路径/最少换乘次数路径,否则打开html文件
    if not ifTest:
        webbrowser.open_new_tab(fileName)
```

(Figure 9)

4.2 What kind of data structures and algorithms are applied? Why and how?

**Undirected weighted graph：**

Why：There is a two-way connection between subway stations, but the distance between different stations is different.

How：The subway route map is stored in the form of adjacency list.

**adjacency list：**

Why：Adjacency matrix The graph stored in the form of matrix takes up large space in the case of a large number of Guangzhou metro stations, so the graph is stored in the form of adjacency list.

How：The attributes of the adjacency point are the station name (name), the adjacency list dictionary (subgraph), and the list of lines (line). Edge addition is achieved by adding key-value pairs of the format {connected station name: distance between two stations} to the adjacency list dictionary..

**Dijksrta algorithm**

Why：Because the subway graph is an undirected connected graph with weights, and only needs the shortest path from the starting point to the end point, it is a dense graph with no negative weighted edges and negative weighted loops, so dijksrta algorithm is used to find the shortest circuit of the single source of the graph.

How：Check step 1,2,3,4 in **Task 3** for detail

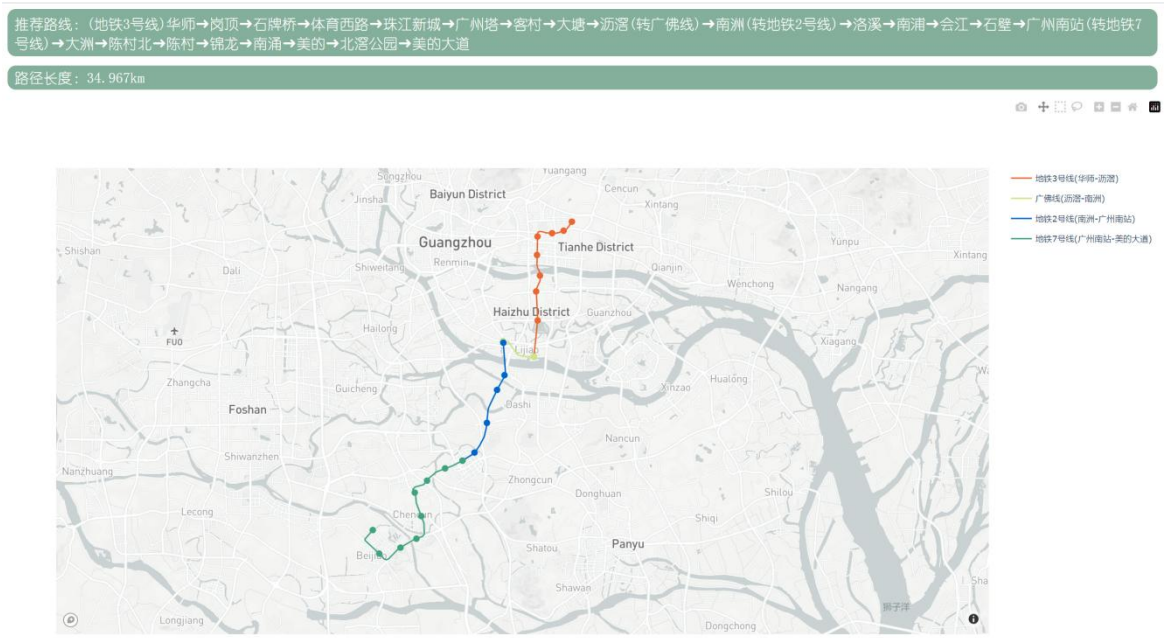
## 5、Testing and conclusion

Please paste images of the running results with a brief text description. To note that the running of program should cover various cases, and finally stating whether the program meets the requirements of this comprehensive experiment.

**Manual test result：**

Enter any start and end points. Click to find the shortest path.

View the result (Figure 10):

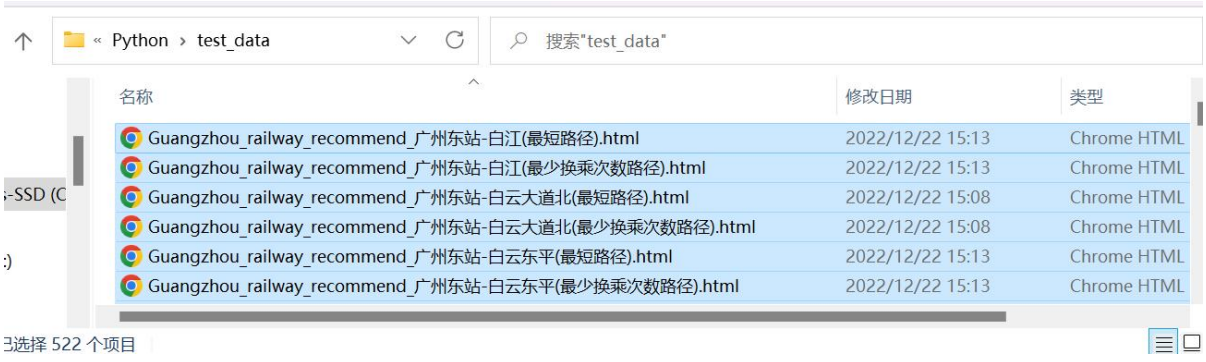



(Figure 10)

Additionally, we wrote a test program to test whether any two paths of Guangzhou subway can generate the shortest path / the path with the least number of transfers and generate html files. The tests all passed and generated the shortest and least transfer routes for 261 sites, with a total of 522 html files (Figure11 Figure 12).

```
正在爬取数据以生成路线图，请稍等....
最短路径为：（地铁3号线北延段）广州东站—>林和西（转APM线）—>体育中心南—>天河南—>黄埔大道—>妇儿中心—>花城大道—>大剧院—>海心沙—>广州塔（转地铁3号线）—>客村—>大塘—>沥滘（转广佛线）—>南洲—>石溪 路径长度：14.409km
正在爬取数据以生成路线图，请稍等....
最少换乘次数路径为：（地铁1号线）广州东站—>体育中心—>体育西路—>杨箕—>东山口—>烈士陵园—>农讲所—>公园前—>西门口—>陈家祠—>长寿路—>黄沙—>芳村—>花地湾—>坑口—>西塱（转广佛线）—>鹤洞—>沙涌—>沙园—>燕岗—>石溪 换乘次数：1次 路径长度：24.868km
测试已完成(261/261)
```

(Figure 11)




(Figure 12)

## 6、Summary

We have done a lot of extensional work, such as getting the longitude and latitude of each site from Baidu Map api, using python to create and edit html and write css to beautify it, provide two options for shortest route and least number of subway changes and write a test

program to make sure the program run as we expect, to make sure everything is clear for user, we also wrote a **README.md** file (Figure 13).

## 如何运行程序

1. 在运行前输入 `pip install --user -r requirements.txt -i https://pypi.douban.com/simple` 检查是否已安装程序相关依赖
2. 打开 `Main.py`，运行程序，程序将自动弹出交互界面
3. 在弹出窗口中进行选择
   a. 查看广州地铁路线图：点击“查看广州地铁路线图”按钮，将自动打开路线图html文件（程序目录下需有 `Guangzhou_railway.html` 文件）
   b. 查询最短路径/最少换乘次数路径：输入起始地铁站和终点地铁站，然后点击“最短路径”或“最少换乘”按钮，将自动打开推荐路线图html文件
   4.查看查询过的推荐路线图：打开程序所在目录，根据文件名找到对应的推荐路线图

## 如何运行测试

打开 `test.py`，运行程序，将自动测试是否广州地铁的任意两条路径都能生成最短路径/最少换乘次数路径并生成html文件，并在终端输出每个测试的结果。

(Figure 13)

## 7、Reference List


[1] https://blog.csdn.net/gdhy9064/article/details/90070793 Python 地铁线路可视化

[2] https://uiverse.io/all Open-Source UI elements made with HTML and CSS

[3] https://blog.csdn.net/AshleyXM/article/details/104484236 tkinter 中 Label 控件动态改变值