Project title: Graph algorithm research and the path finder

# Contents

[**Contents**](#_amu9bg4ydzl) **1**

[**Analysis**](#_jznpvshs05nx) **4**

[Introduction](#_ow8uo1qwbvov) 4

[Background and basics of learning](#_81e9lsjqz0c) 5

[Problem background](#_gypal71r96jd) 5

[Overview of graph algorithms](#_x55k4wrqhxhx) 7

[Objectives](#_5x29gemywct2) 8

[Data Sources and Destination](#_rvi9eyxsme71) 9

[Graphical Interfaces and Data Dictionary](#_bnl8cf743zo) 10

[Flowchart of steps of research](#_91dcamfmnhad) 12

[Algorithms learning plan](#_o9mpxptyqjqz) 12

[Flowchart of general program module](#_imkog17mpu0l) 13

[Reference](#_nmo641uaac0p) 14

[Reference](#_vc7gctd0rimp) 15

[**Design and implementation**](#_zf4zbl43lsy8) **16**

[Graph Theory](#_fk4ecaxvavew) 16

[Representations of maps](#_ts1zfqpfm65) 18

[Adjacency matrix](#_x0bt3hanli7k) 18

[Adjacency list](#_70915gyq53ce) 19

[Simple recursion idea](#_nm2n2hfqv2yz) 21

[Search algorithm](#_nbhkxb3if49q) 22

[Breadth first search](#_yaeejmtm5i32) 22

[Depth first search](#_xgij1oel0tz6) 22

[The Dijkstra’s algorithm](#_nfkpkn2nvhmu) 24

[The Floyd algorithm](#_4y5dzrca98vj) 29

[Bellman-Ford](#_kdcr1s2ykizt) 33

[Comparison of algorithms](#_lolk4q12asmy) 38

[Time complexity](#_979h8peo9zpp) 38

[Calculation method](#_cwu59w3333fm) 38

[Floyd](#_6hrcco7acmuc) 38

[Bellman-Ford](#_f5w4o85uupkc) 39

[Dijkstra](#_g5m5y4gxkkig) 39

[Overall System Design](#_iamkp9oka6lb) 40

[Description of Modular Structure of System](#_8l5tlshih5co) 41

[Login](#_muznoewnaxy) 41

[Map viewer and User interface](#_5bewv6yxnzqc) 42

[Path finder](#_964kp8rlqbiw) 44

[Flowchart of program](#_uto543gpv9zl) 45

[Algorithm for solve Best route](#_wqslmdw8s71a) 47

[Algorithm for solve Fewer transfer](#_3tva7q5g3292) 47

[**Code**](#_poqjapk2d797) **48**

[login.py](#_yl4o2bvgvtbt) 48

[mapviewer.py](#_y80zw6k9e57o) 55

[path\_find.py](#_9qhf8bg8ou8b) 62

[**Testing**](#_r19iddq2ub52) **67**

[Test strategy](#_ofrdcbjtpkk) 67

[Testing screenshots](#_3xmt0tgus7ym) 71

[Figure 1](#_f7giaqfsoeol) 71

[Figure 2](#_3d67iflsd9nv) 71

[Figure 3](#_1rdvj4oi3we4) 71

[Figure 4](#_oszpcls3zigd) 72

[Figure 5](#_navd9dx4zwkf) 73

[Figure 6](#_gu8xvv8n5gmh) 73

[Figure 7](#_p10a6tnq2i5c) 73

[Figure 8](#_fwwpbd8d46vd) 74

[Figure 9](#_fl1vtxieg78r) 75

[Figure 10](#_iq4zqipg5xl2) 75

[Figure 11](#_4p66u9gnhwa) 76

[Figure 12](#_z4vcv5xm4pa1) 76

[Figure 13](#_bjymkhjj76zq) 77

[Figure 14](#_mu0ec7p9m8zf) 77

[Figure 15](#_y86143dd0rdc) 78

[Figure 16](#_opg7746lfc8v) 78

[Questionnaire](#_k50g7nt52jem) 79

[Evaluation](#_pyyslkugereu) 86

# 

# Analysis

## Introduction

This coursework is mainly divided into three parts.

Researching algorithms

Comparing and choosing differences

Programming in real case.

At the first part, the project will systemic introduce the necessary theory and related algorithms. The Dijkstra algorithm will be programmed as a starter. The learning steps of more algorithms about shortest paths will be shown.

In the second part, I will make the decision of the most suitable algorithm from the algorithms I researched in first part. The evaluation criteria are the time complexity and the space complexity. There is a table report of advantages and disadvantages of each algorithm at the end of part two.

Finally, Use the knowledge to design and implement a route finder, and create a storage system with Input/Output file and Database. Save and link information like maps names, points names and time cost of each path in database. Also, storing the text file of character image will provide maps more visualizations. The pseudo code and python code will be shown in section six and seven. Some of the testing in this program, as well as specific future improvements, are listed in the seventh Testing and Evaluation section.

## 

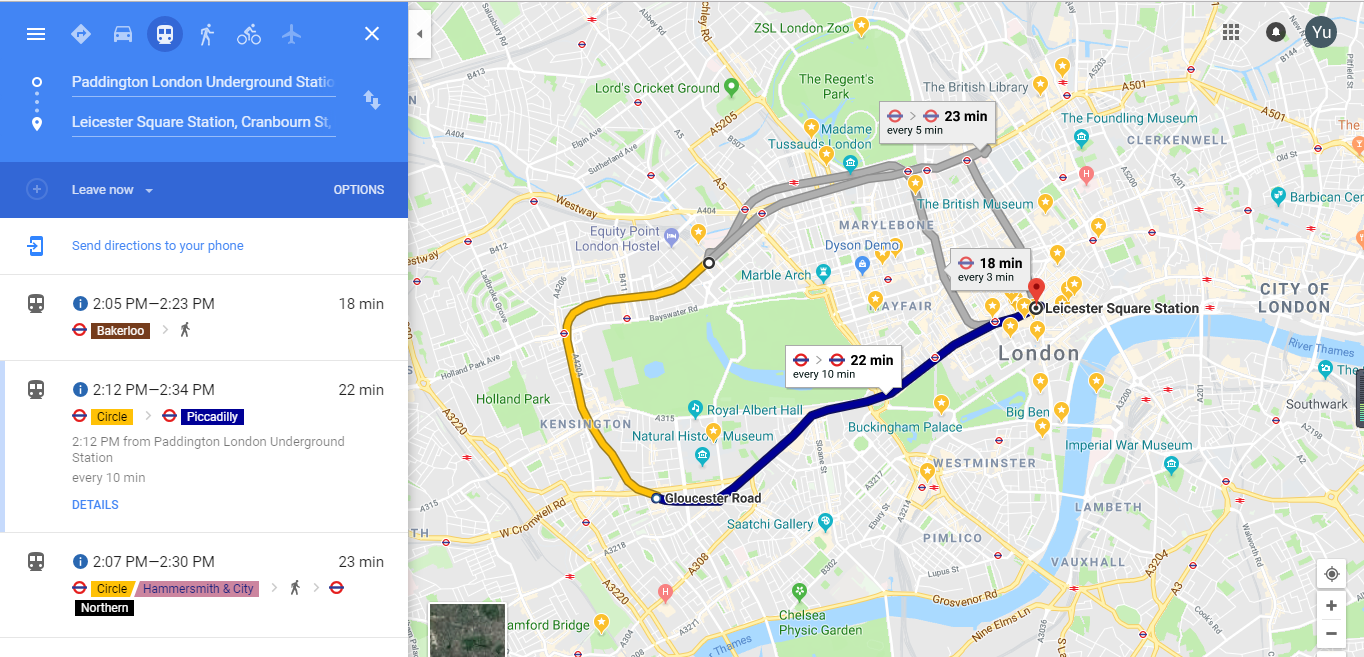
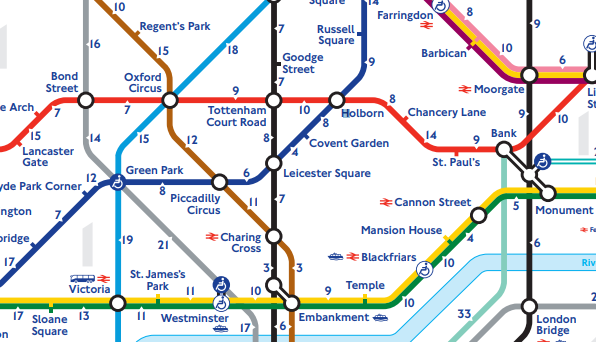
## 

## Background and basics of learning

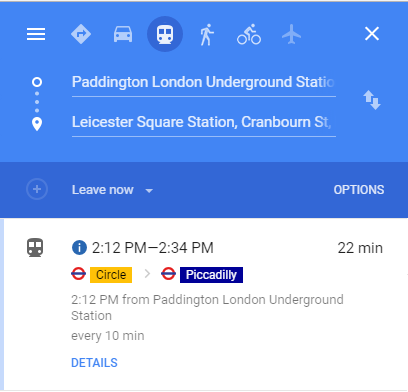
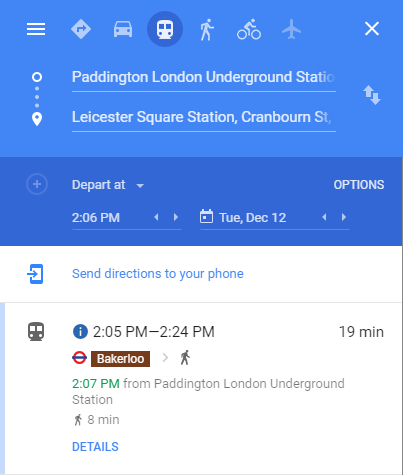
### Problem background

With the development of the public transport system, the tube map of a city will become more and more complicated.The world's first underground railway, The Metropolitan railway, which opened in 1863, is now part of The Circle, Hammersmith & City and Metropolitan lines.

The first line to operate underground electric traction trains, the City & South London Railway in 1890, is now part of the Northern line.The network has expanded to 11 lines, and in 2016-17 carried 1.379 billion passengers, making it the world's 11th underground system.

For a foreign traveler, or a new migrant, there are usually a series of questions about the time and line of the underground. Below is a screenshot of a Google map showing the subway ride from London Paddington to Leicester square. Although the Bakerloo Line can't get to the terminus directly, it can arrive faster than usual and do not need to change a car, by walking on a very short route.Sometimes, it’s quicker to walk, Don’t be fooled by the Tube map. Accomplished commuters will know that the geographically-distorted lines can take you on unnecessarily longer journeys, and often walking between station is actually quicker than changing trains.

Above is the walking time between two subway stations from the London Transport network. You can see Piccadilly walking to Leicester for six minutes.

It can be found that there are many different kinds of itineraries, among which, there are those who take the Bakerloo Line and walk. Also take a Circle Line and change to Piccadilly Line. 

Different schemes will take different time and will determine the length of the walk, In my program, I will determine the optimal distance, the least walk with different algorithms.

### Overview of graph algorithms

Algorithm is one of the most important cornerstones in computer science, but it has been neglected by some programmers. Many people saw some companies in the recruitment requirements of programming language is multifarious leads to a misunderstanding, think learning computer is learn a variety of programming languages, or believe that learning new languages, paving the way, the standard is the best method. In fact, all of them are misled by these companies. Programming languages though the learning, but learning algorithms and theory is more important, because the computer languages and development platform are changing but the changes of algorithms and theory are tiny, such as data structure and algorithm, compiling principle, computer architecture, relational database theory and so on.

Finding shortest path is widely used in our life, when you send an email, log in to school computer from home or search the Internet, a lot happens behind the scenes. The IP-router has to determine which is the best and fastest way to send your message. Finding the shortest path has applications in circuit board design, route planning, games strategy and project management.

The shortest path problem is a classical algorithm in graph theory, aiming to find the shortest path between two nodes in the graph (composed of nodes and paths).The specific forms of the algorithm include:

-The shortest path problem to determine the starting point is the problem of the shortest path.

-The shortest path problem to determine the destination - instead of the problem of determining the starting point, the problem is known to terminate the node and take the shortest path. In the undirected graph, the problem is exactly the same as the problem of the starting point, which is the same as the starting point for the reversal of all paths in the directed graph.

-The shortest path problem to determine the starting point is the shortest path between the two nodes.

-Global shortest path problem - find all shortest paths in the graph.

The algorithm used to solve the shortest path problem is called "shortest path algorithm", sometimes referred to as "path algorithm". The most commonly used path algorithms are:

Dijkstra algorithm

bellman-ford algorithm

Floyd algorithm

## Objectives

1. Studying and implement different algorithms from Math decision lesson, for example Dijkstra and Floyd.

2. Tracking each steps of algorithms and understand them sufficiently.

3. Compare and discuss with their time complexity, space complexity and their relative merits.

4. Create a route finder for London underground, to offer travellers a route between two different station.

5. Each user able to have log on and new user able to sign in.

6. User can create other map from other city, and it will not only for London underground.

7. Users have different account and they can add maps of other cities.

8. It is able to find a route between two place.

9. Different algorithms are used to calculate the minimum walking distance, minimum transfer.

10. The image viewer must work well, and able to be scaled.

11. The system must be able to run quickly and efficiently.

## 

## 

## Data Sources and Destination

|  |  |  |  |
| --- | --- | --- | --- |
| **Data in the system** | | | |
| **Data** | **Source** | **Description** | **Destination** |
| Username | userinput or database | username of account | [login.py](http://login.py) |
| Password | userinput or database | password of account | [login.py](http://login.py) |
| Tube image | google image | A jpg file which will be display in application | QtGraphicViews(Label) |
| Tube station list | [gov.uk](http://gov.uk) | Got list of stations in each line | [path\_finder.py](http://path_finder.py) |
| Time between two station | google map | Time cost between every two stations | [path\_finder.py](http://path_finder.py) |

# 

# 

## Graphical Interfaces and Data Dictionary

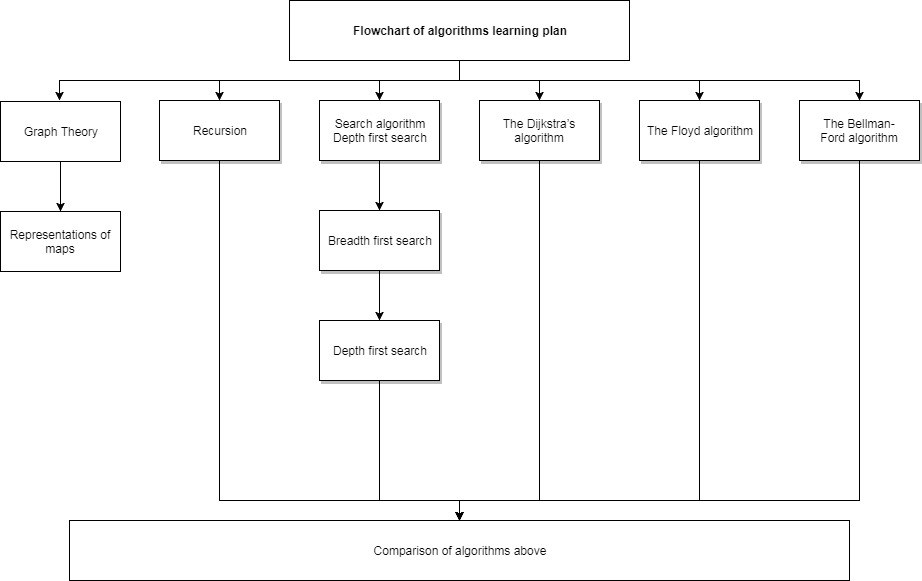
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Graphical Interfaces Data Dictionary** | | | | |
| **Field name** | | **Data type** | **Example** | **Description** |
| layout | | GridLayout |  | Layout is the tool when the user enlarges the interface window, the size of the window will not be affected  GridLayout can make sure the image and result path contains the most area of screen |
| Button | | PushButton |  | It is the button that can call the path finder function, also if there is any text in tabWidget, this button can clear the text. |
| Listcity | | ComboBox |  | The combobox(listbox) is used for select city. |
| Textcity | | Label |  |  |
| Textfrom | | Label |  |  |
| start | | LineEdit | This is used for the start point. |
| Textto | | Label |  |
| goal | | LineEdit | This is used for the end point. |
| map | | GraphicsView |  | GraphicsView is used for check and scale image of city tube map. |
| tabPath | tab | Widget |  | Widget is used for showing instructions for different requirements ,Best route,Fewer transfer |
| tab\_2 | Widget |
| tab\_3 | Widget |

## 

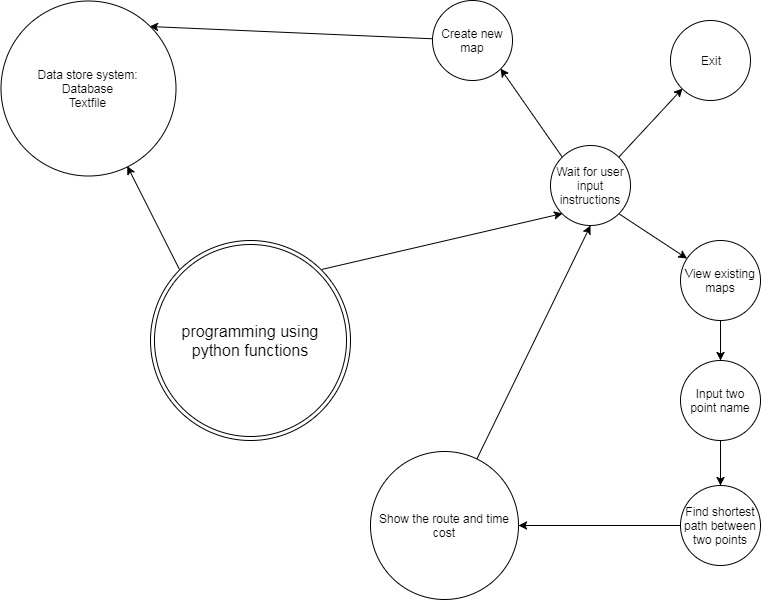
## 

## Flowchart of steps of research

### Algorithms learning plan



### Flowchart of general program module



## Reference

**London Underground**

<https://en.wikipedia.org/wiki/London_Underground>

**13 secrets of the Tube revealed by London Underground driver**

<http://www.independent.co.uk/travel/news-and-advice/13-secrets-of-the-tube-revealed-by-london-underground-driver-a6838976.html>

**Walking time between two station**

<http://content.tfl.gov.uk/walking-tube-map.pdf>

## Reference

Cormen, T.H., 2009. *Introduction to algorithms*. MIT press.

Wilson, G. and Oram, A., 2007. *Beautiful code: leading programmers explain how they think*. " O'Reilly Media, Inc.".

Russell, S., Norvig, P. and Intelligence, A., 1995. A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, *25*

Dijkstra, E.W., 1959. A note on two problems in connexion with graphs. *Numerische mathematik*, *1*(1)

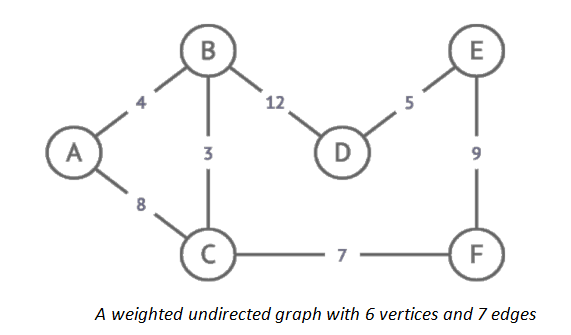
**Shortest Path Algorithms TUTORIAL**

<https://www.hackerearth.com/practice/algorithms/graphs/shortest-path-algorithms/tutorial/>

# Design and implementation

## Graph Theory

In computer science, graph theory is concerned with the study of graphs and is a concept that was introduced by the Swiss mathematician Leonhard Euler. Graphs are used to represent relationships within a set of objects. A graph is a collection of vertices (points) that are connected by edges (lines).

As I mentioned in last section, formally, a graph G is an ordered pair G = (V, E) comprising a set of V (nodes) and a set E (edges).

There is a significant amount of vocabulary associated with graph theory, however, I only list few below:

Undirected graph: is a graph in which edges have no orientation, therefore the edge (x, y) is identical to the edge (y, x). This means that the edges can be traversed in both directions.

Weighted graph: is a graph where edges have an associated numerical value called the weight. As an example, the weight might refer to the length of a route in the context of route planning applications.

Simple graph: is an unweighted, undirected graph containing no graph loops, an edge of a graph that joins a vertex to itself, or multiple edges, two or more edges connecting the same two vertices.

Path: is a finite sequence of edges which connect a sequence of vertices. In a weighted graph, the weight/cost/length of a path is the sum of the weights of the traversed edges.

Connectivity: if there is at least one path between all pairs of vertices, the graph is said to be connected.

## 

## 

## Representations of maps

There are two ways to represent a graph G=(V,E), with adjacency matrix or adjacency list. Either way applies to both directed and undirected graphs. In this coursework, we will mainly use undirected graphs.

### 

### Adjacency matrix

Each row and column represents a node. The item at [row, column] indicates a connection

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **A** | **B** | **C** | **D** | **E** |
| **A** |  | **7** |  | **3** |  |
| **B** | **7** |  | **3** | **2** | **6** |
| **C** |  | **3** |  | **4** | **1** |
| **D** | **3** | **2** | **4** |  | **7** |
| **E** | **6** |  | **1** | **7** |  |

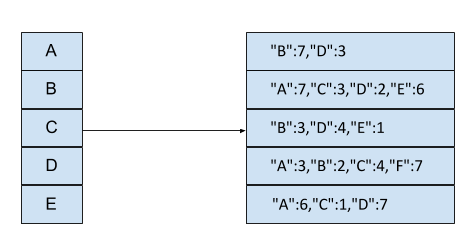
Advantage: an adjacency matrix is convenient to work with, and adding an edge is simple

Disadvantage: a sparse graph with not many connections (edges) will leave most of the cells empty, wasting a lot of memory space

### Adjacency list

An adjacency list is an alternative way of representing a graph. A list of nodes is created, and each node points to a list of adjacent nodes

A weighted graph can be represented as a dictionary of dictionaries, with each key in the dictionary being the node, and the value being a dictionary of adjacent nodes and edge weights



The following Python code represents a graph as a dictionary, with the key being a node and the value being a list of adjacent nodes.

**nodes = ('A', 'B', 'C', 'D', 'E')**

**distances = {**

**'A': {"B":7,"D":3},**

**'B': {"A":7,"C":3,"D":2,"E":6},**

**'C': {"B":3,"D":4,"E":1},**

**'D': {"A":3,"B":2,"C":4,"F":7},**

**'E': {"A":6,"C":1,"D":7},**

**}**

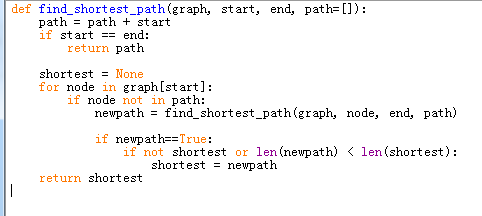
Advantage: It only uses storage for the connections that exist, so it is more space-efficient. It is a good way of representing a large, sparsely connected graph. So in this coursework I will choose adjacency list to represent the graph.

## 

## 

## Simple recursion idea

Graphs are networks consisting of nodes connected by edges or arcs. In directed graphs, the connections between nodes have a direction, and are called arcs; in undirected graphs, the connections have no direction and are called edges. We mainly discuss directed graphs. Algorithms in graphs include finding a path between two nodes, finding the shortest path between two nodes, determining cycles in the graph (a cycle is a non-empty path from a node to itself), finding a path that reaches all nodes (the famous "traveling salesman problem"), and so on.



## 

## 

## Search algorithm

Searching for a graph means accessing its vertices in one particular order.For multigraph algorithms, both breadth and depth priority search algorithms are important because they provide a systematic way to access graph data structures.

### Breadth first search

Breadth-first search is one of the simplest algorithms for searching a graph and the archetype for many important graph algorithms. It is also known as breadth first search (BFS), is a kind of graph search algorithm, the shortest path is appropriate to discuss the first model.

BFS is a blind search method to systematically expand and check all nodes in the diagram to find results.In other words, it does not consider the possible address of the result, and thoroughly searches the whole picture until it finds the result. The steps are as follows:

· first put the root node into the queue.

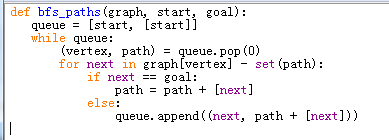
· remove the first node from the queue and verify that it is the target.

· if the target is found, the search is completed and the results are returned.

· otherwise, all of the untested direct child nodes (adjacent nodes) are added to the queue.

· if the queue is empty, the entire graph is checked - that is, there is no target to be searched for.End the search and return to "no target".

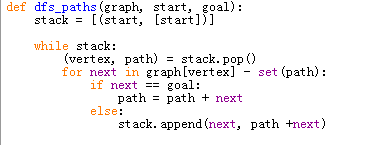
**Code:**



### Depth first search

Depth-first search algorithm (DFS) is an algorithm for traversing or searching trees or graphs.Traverse the tree's nodes at the depth of the tree and search the tree's branches as deep as possible.When the side of the node v has been explored, the search will go back to the beginning node of the side where the node v is found. This process continues until all nodes that have been discovered from the source node can be reached.

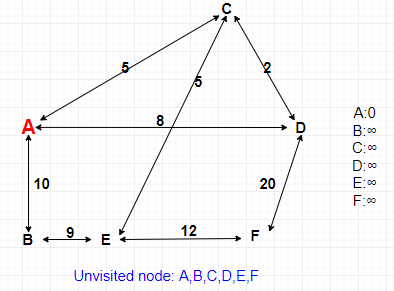
**Code:**



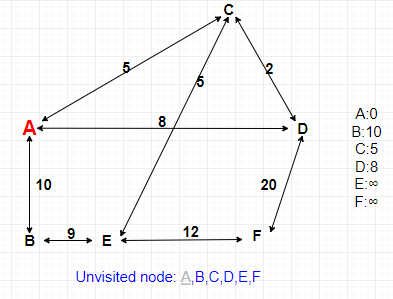
## The Dijkstra’s algorithm

The implementation of the algorithm is similar to a breadth first search, It uses a priority queue as the supporting data structure to keep a record of which vertex to visit next

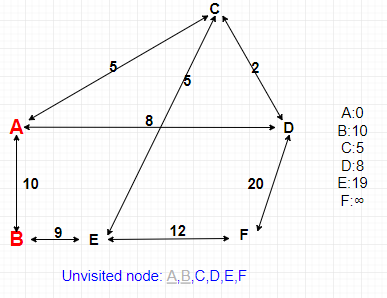
It starts by assigning a temporary distance value to each node. The temporary distance is 0 at the start node, and ∞ at every other node. The graphs below is an example of how dijkstra algorithm working.



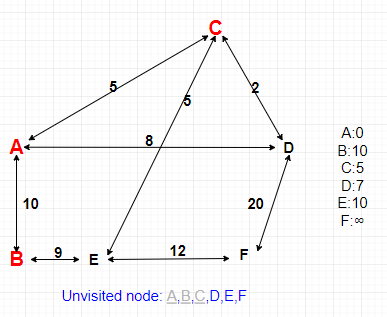
The temporary distance is 0 at the start node, and ∞ at every other node. Also, there is a list named unvisited node at bottom of our first step is to picking a start point. The table right is to keep track of the distance.



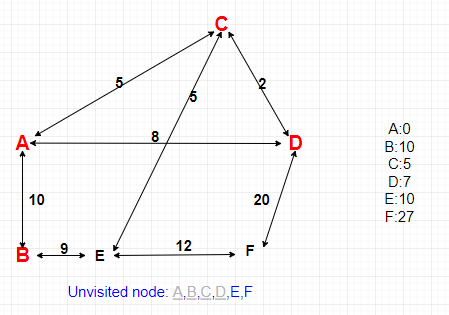
The next step is to examine the edges leaving A, we can reach B C D, so update the table with corresponding costs. Cross off A in unvisited node list.

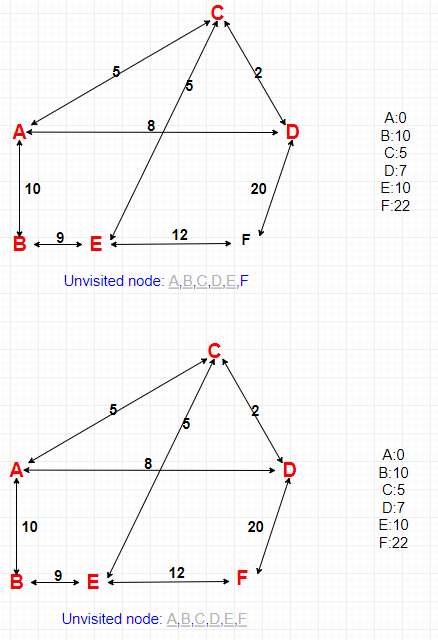


As we reach the B, the point E becomes reachable.



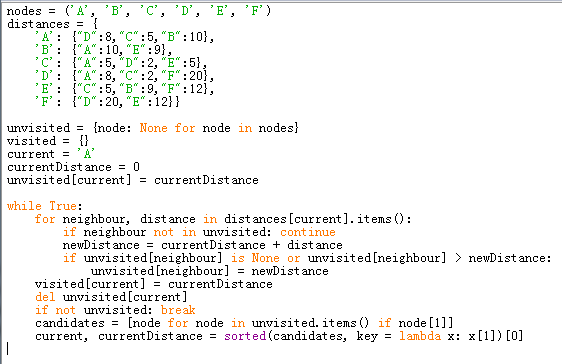
Do the same thing, choosing the closed node C and updating the costs.





Finally, we have chosen all points and there is no updates. All the nodes and edges in graph have now been visited.

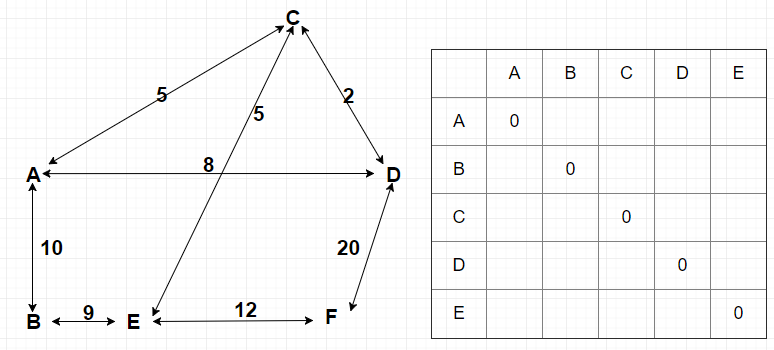
Code:

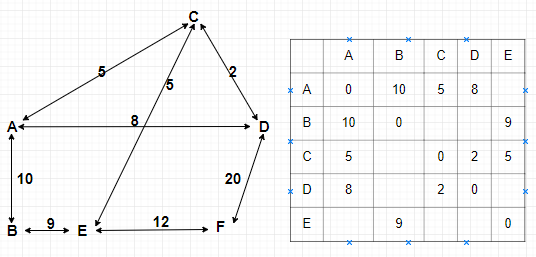


## 

## 

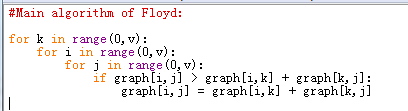
## The Floyd algorithm

Floyd algorithm is also named Floyd-warshall algorithm, it is different from Dijkstra’s and Bellman-ford, it discovers shortest path between all pairs of vertices. First, we need a graph, we create a distance array that keeps track of the shortest path between nodes, for example row 1 column 2 will store the shortest path from node A to node B.

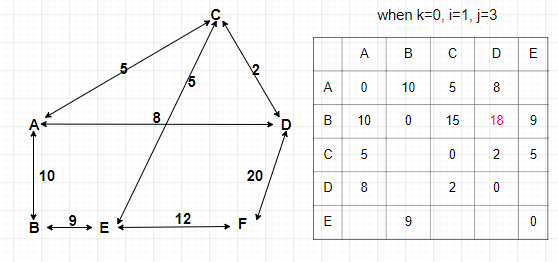
Initialize each nodes to itself to 0, next step is loop through the edges of our graph and fill the table with corresponding weight.

Start with them all equal 0, graph[0][0]>graph[0][0]+graph[0][0], the if condition isn’t met so we don’t update our table.

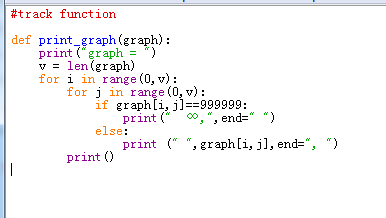
Code:

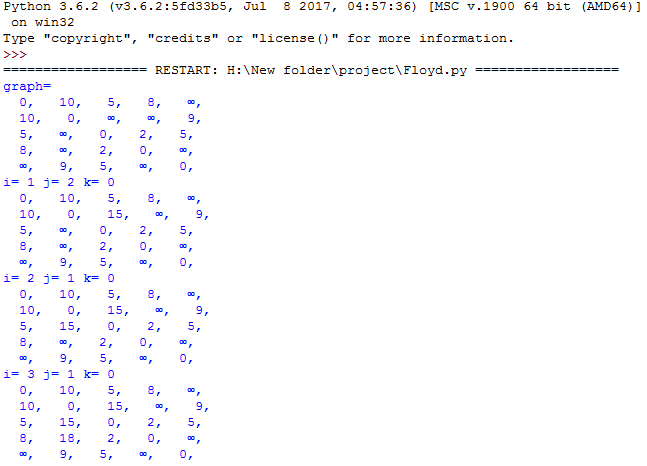
******

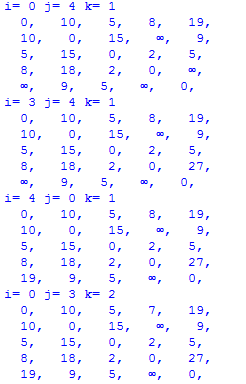
Follow the loop, and track the values of k,i,j. When k=0, i=1, j=2, the if condition is met(graph[1][2]>graph[1][0]+graph[0][2], 15>∞), therefore we update the path from B to C with a smaller weight which is 15.

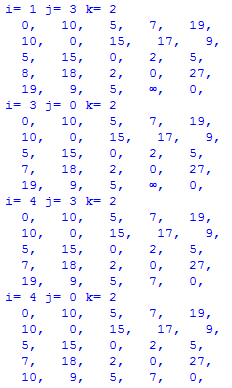
****

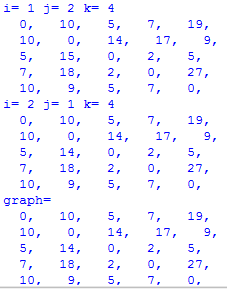
I add the print\_graph function to track iteration, I skip the iterations that have no effect on table, I show the code and result below.



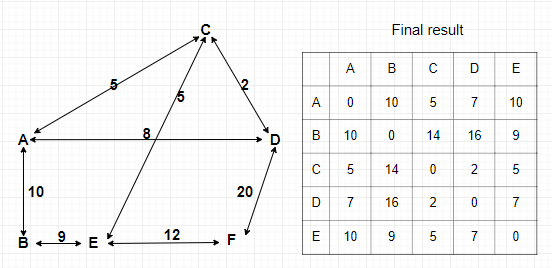








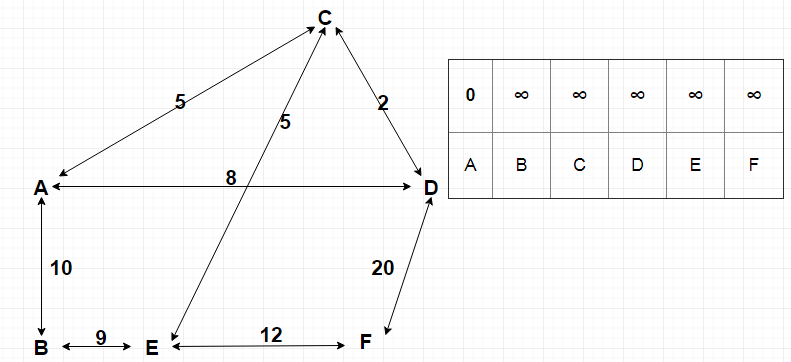
The final result is below.

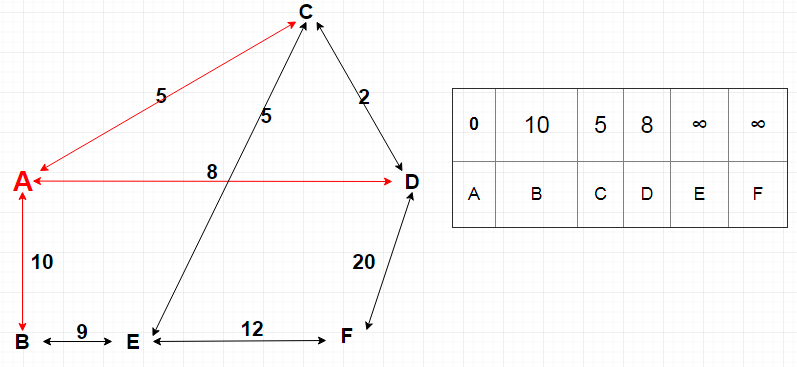


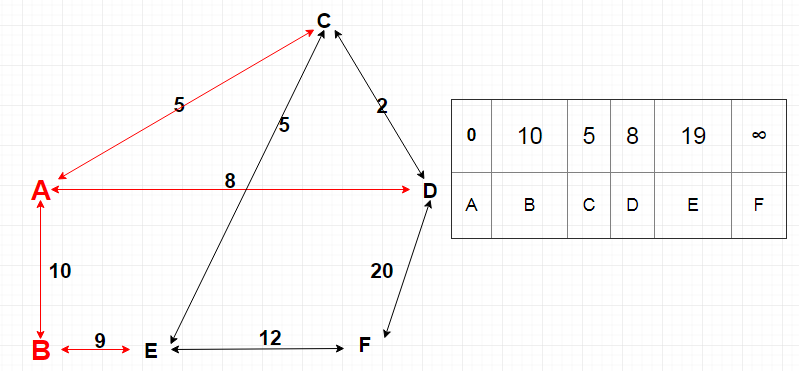
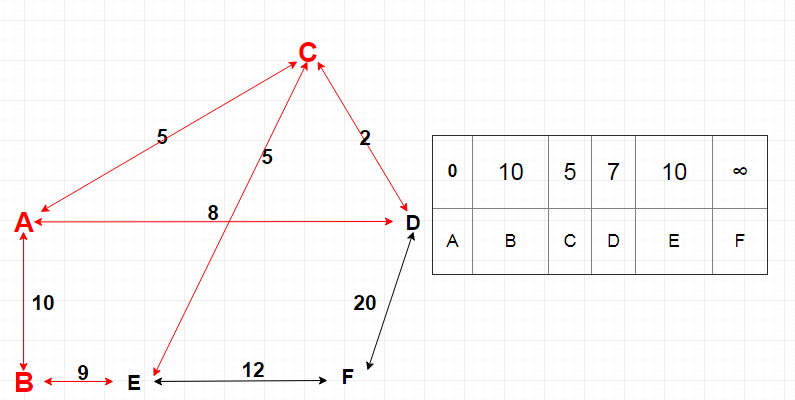
## Bellman-Ford

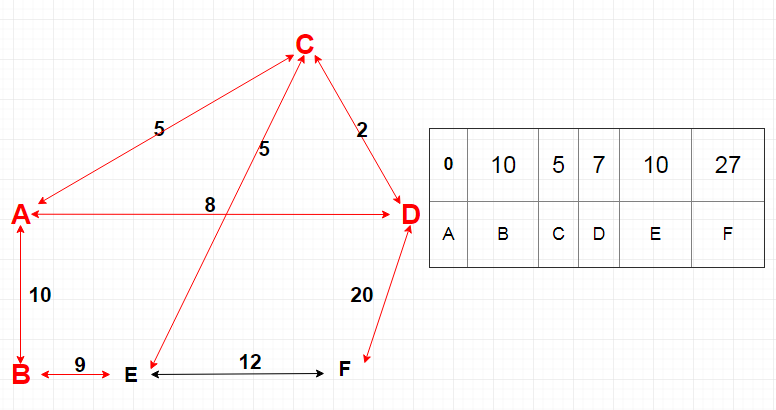
Bellman-ford algorithm is an algorithm to find the shortest path of single source with negative power graph, which is less efficient and less difficult to code.

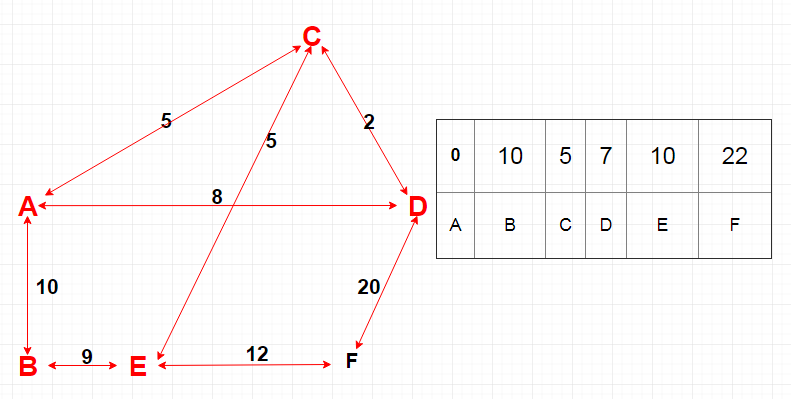
The principle for continuous relaxation, at every time of relaxation to each edge is update, if the relaxation in the n - 1 times can update, then there are negative ring in the figure, so doesn't give a result, otherwise we are finished.

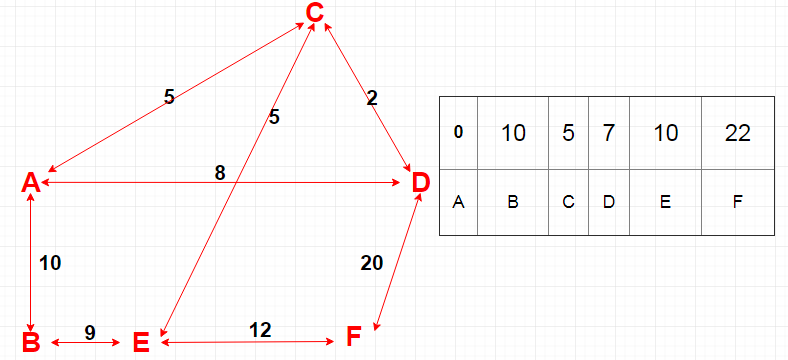
I'll use the following graph for our example notice there are 6 vertices sowe'll need to do 5 iterations A will be our starting node to begin. We'll make a table on the right to keep track of distances we set 0 as the distance to A and infinity for the rest of the nodes we're ready for our first iteration. At each iteration we need to examine all the edges.

I think an organized way of doing this is to look at each node one by one and examine its outgoing edges when we've done this for every node we're guaranteed to have seen all the edges in our case. Looking at A we see we can reach B with the cost of 10, C with the cost of 5 and D with a cost of 8 so we'll update the table next up is a we now know that we can reach a with the cost of 10 looking at its outgoing edges we can reach E with the cost of 9 or a total cost of 19 getting to E.

 we notice we don't even know how to reach it yet so we can skip this note moving on to see we know that we can get to see with the cost of 19 

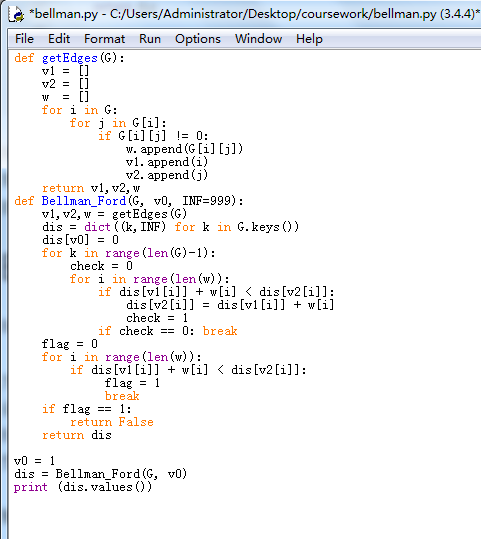






The nodes notice that none of the values in our table are changing you we finished iteration 2 and all the values are the same why is this well the algorithm takes at most V minus one iterations but we can make it more efficient by stopping early if nothing improves during an iteration so after 2 iterations were finished our table shows the shortest path from A to all other nodes don't be worried if your values at each iteration are different a lot depends on the implementation and the order in which the edges are visited you may even need five iterations but we could end up with the same result.

Code:



## Comparison of algorithms

### Time complexity

Time complexity is the same problem can be solved by different algorithms , and the quality of an algorithm will affect the efficiency of algorithms and procedures . The purpose of algorithm analysis is to select the appropriate algorithm and improve the algorithm.

In Computer science, the time complexity of the algorithm is a function, which qualitatively describes the algorithm's running time. This is a function of the length of the string representing the input value of the algorithm . Time complexity commonly used large O symbol expression, does not include the low order terms and the first coefficient of this function. With this approach, the time complexity can be said to be asymptotic, looking at the situation as the input value approaches infinity.

### Calculation method

1. In general, the number of iterations of the basic operations in the algorithm is a function of the problem size n, denoted by T(n). If there is some auxiliary function f(n), when n approaches to infinity, The limit value of T(n)/f(n) is a constant not equal to zero, then f(n) is the same order of magnitude of T(n). Denoted as T(n) = O(f(n)), which is called as O(f(n)) is the asymptotic time complexity of the algorithm.

Analysis: With the increase of module n, the growth rate of algorithm execution time is proportional to the growth rate of f(n), so the smaller f(n) is, the lower the time complexity of the algorithm is and the higher the efficiency of the algorithm is.

2. When calculating the time complexity, find the basic operation of the algorithm, and then determine the number of its execution according to the corresponding statement, and then find the same order of magnitude of T (n). After finding the constant, can get a constant c, then the time complexity T(n) = O(f(n))

3 In python more easy to understand, easy to calculate the method is to see if there are several for for loops, only one time while the time complexity O (n), double is O (n ^ 2), and so on, If there are dichotomous O (log n), dichotomy For example, rapid power , binary search, if a for loop sets a dichotomy, then the time complexity is O (n log n).

### Floyd

Seeking the most source, no negative side of the shortest way. Record the graph with matrix. Timeliness is poor, time complexity O (V ^ 3). Floyd-Warshall algorithm is an algorithm to solve the shortest path between two arbitrary points, which can correctly deal with the shortest path problem of directed graphs or negative weights.

The time complexity of Floyd-Warshall algorithm is O (V ^ 3), and the space complexity is O (E ^ 2).

### Bellman-Ford

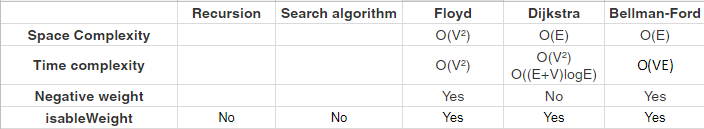
Find the single-source shortest path, you can determine whether there is a negative return (if there is no shortest path), timeliness is better, the time complexity O (VE).

### Dijkstra

Find a single source, the shortest path without negative weight. Good timeliness, the time complexity of O (V \* V + E).

If the source is reachable, O (V \* lgV + E \* lgV) => O (E \* lgV).

In the case of a sparse graph, E = V \* V / lgV, so the time complexity of the algorithm can be O (V ^ 2) . If the Fibonacci heap as a priority queue, then the algorithm time complexity, then O (V \* lgV + E).



## Overall System Design

|  |  |  |
| --- | --- | --- |
| **Inputs** | **Processes** | **Outputs** |
| Username, Password | Try to find username in database | Log in successfully,  Close login windows, and open main window. |
| Log in unsuccessfully |
| Register a new account |
| Select name of city | load image and station list from picture files and text files | Show the pictures and transfer Line list become Station list |
| Start, Goal | Import path\_finder.py algorithm | Route |

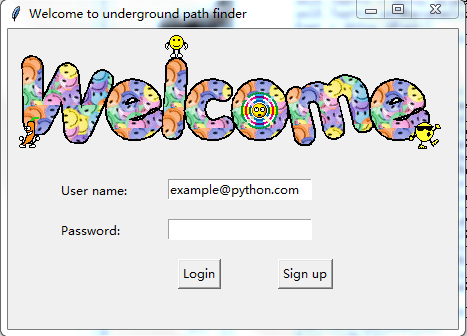
## 

## 

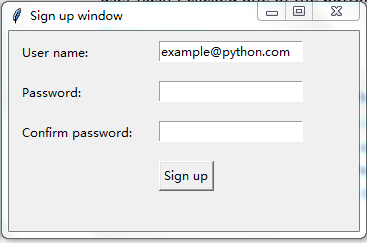
## Description of Modular Structure of System

### Login

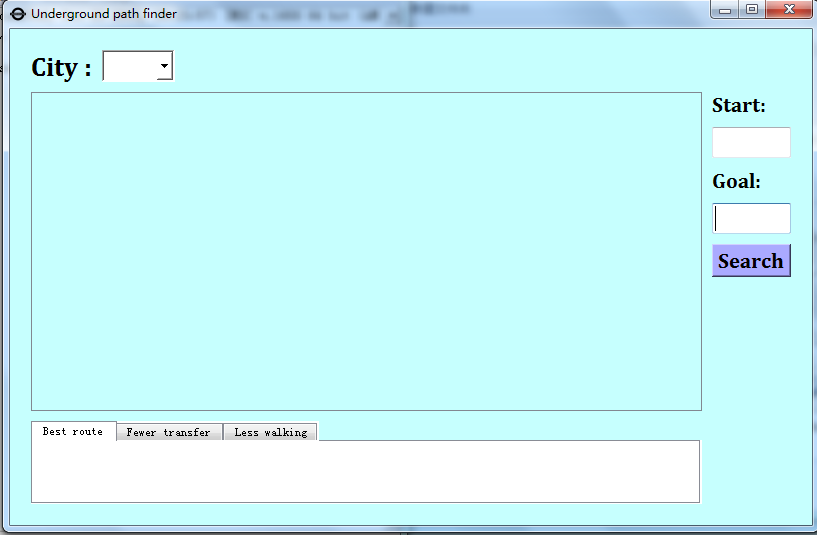
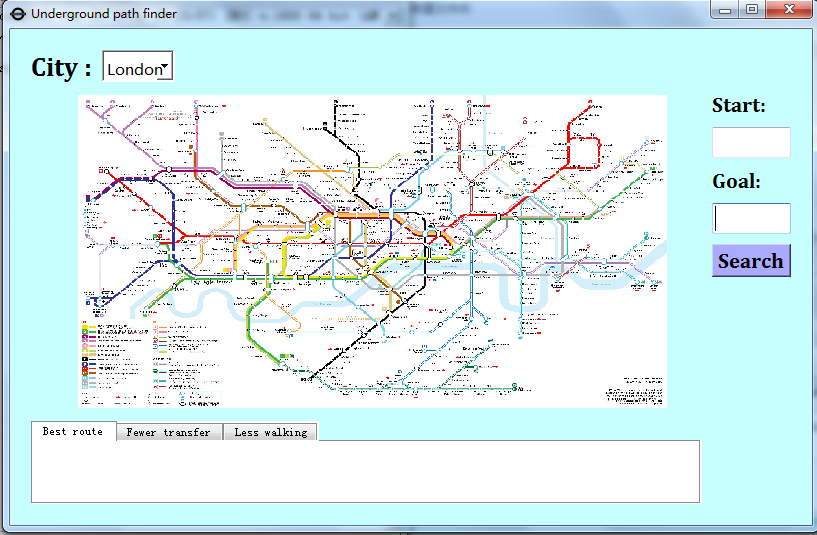
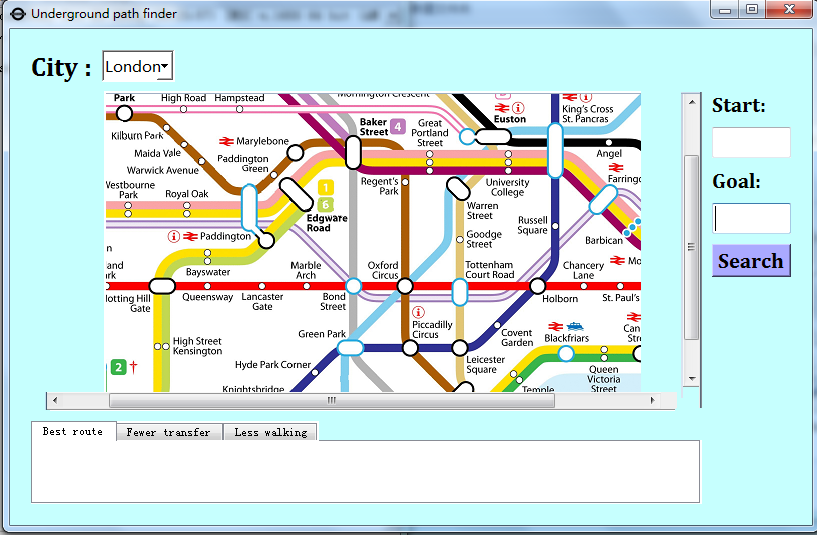
This login window is the first page which user will see when program ran. On this page there will be a welcome image, also there will be two input boxes, one to put username, and one which will keep the password. Above this, there will be the title of the program, “Welcome to underground path finder”.



There will be two buttons below, one of them is for user to login if username and password are filled, the second one is for new user to sign up and it will give user a new sign up window to fill.



### Map viewer and User interface

This picture below is the idea structure of map viewer.For example, when user choose london, the graphics view will illustrate the tube map of London. However the image is not clear enough because the pixel is too large.So it is very important to add a scrollbar, like picture below. It is controlled by mouse wheel.

### Path finder

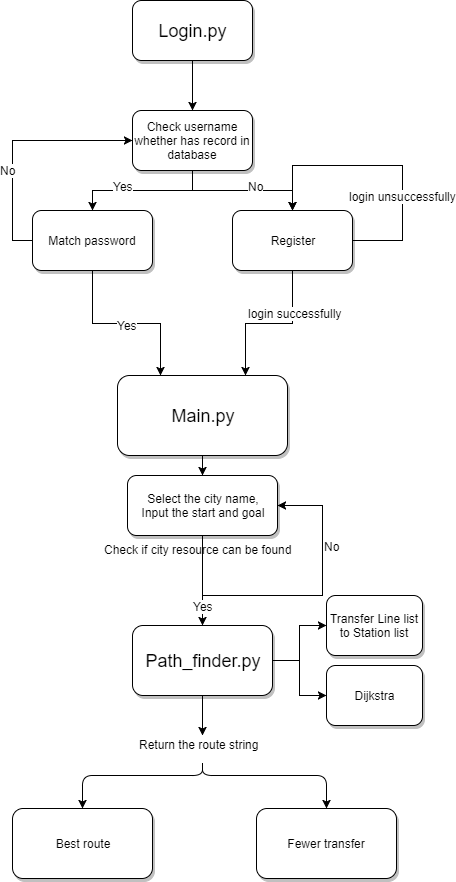
## 

|  |  |  |
| --- | --- | --- |
| **Inputs** | **Variable name** | **Outputs** |
| combobox.Listcity.text | Cityname | Path |
| start.label | Start |
| goal.label | Goal |

## 

## 

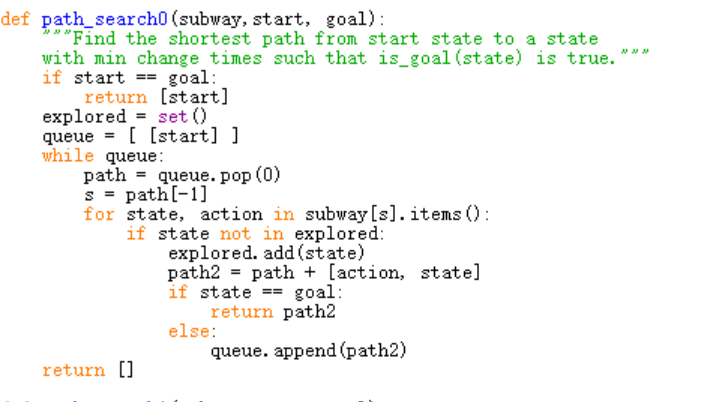
## Flowchart of program

****

## Algorithm for solve Best route

The best route is also the shortest time, which means that the route can be turned or walked. Dijkstra is the algorithm I'm going to use, and I've compared several other methods in the Algorithms studies, which are the fastest and least wasteful of the space.

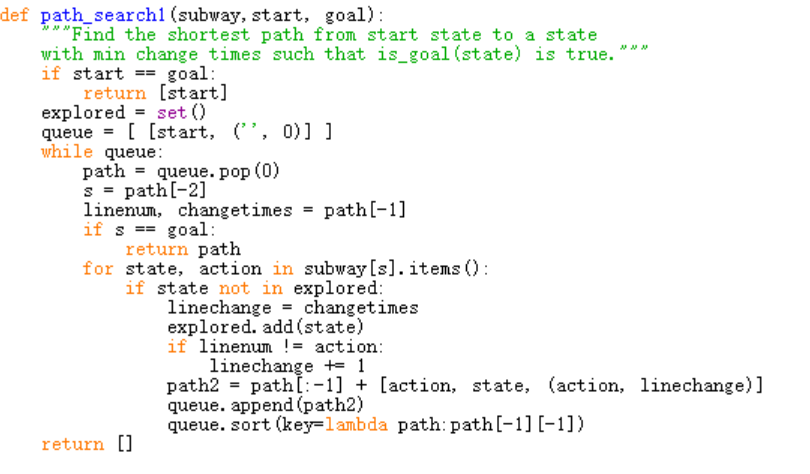
Code Snippet：



## Algorithm for solve Fewer transfer

Fewer transfer is simpler than the other two, and there are two ways to do this. The first method is to find all the paths, sorted, to find the route with the least number of times. Another way to do this is to add the idea of a depth first search algorithm, which, when looked up, would be as good as possible to avoid change.

Code Snippet：



# Code

## login.py

import sys

import tkinter as tk

from tkinter import messagebox # import this to fix messagebox error

import sqlite3

import mapviewer

# initialize

def create\_table():

with sqlite3.connect("userinfo.db") as db:

sql = """create table Userinfo

(UserID integer,

Username text,

Password text,

Map text,

primary key(UserID))"""

cursor = db.cursor()

cursor.execute(sql)

db.commit()

def create\_currentuser\_table():

try:

with sqlite3.connect("Userinfo.db") as db:

cursor = db.cursor()

sql = "DROP TABLE Current;"

cursor.execute(sql)

except:

pass

with sqlite3.connect("userinfo.db") as db:

sql = """create table Current

(Username text,

Map text)

"""

cursor = db.cursor()

cursor.execute(sql)

db.commit()

def usr\_login():

usr\_name = var\_usr\_name.get()

usr\_pwd = var\_usr\_pwd.get()

with sqlite3.connect("userinfo.db") as db:

cursor = db.cursor()

cursor.execute("PRAGMA foreign\_keys = ON")

cursor.execute("select UserID, Username, Password, Map from Userinfo where Username=?",(usr\_name,))

userinfo = cursor.fetchone()

db.commit()

if userinfo is None:

is\_sign\_up = tk.messagebox.askyesno('Welcome',

'You have not signed up yet. Sign up today?')

if is\_sign\_up:

usr\_sign\_up()

else:

if usr\_pwd == userinfo[2]:

tk.messagebox.showinfo(title='Welcome', message=usr\_name)

with sqlite3.connect("userinfo.db") as db:

cursor = db.cursor()

sql = "insert into Current (Username, Map) values (?,?)"

values=(userinfo[1], userinfo[3])

cursor.execute(sql,values)

db.commit()

window.destroy()

mapviewer.main()

else:

tk.messagebox.showerror(message='Error, your password is wrong, try again.')

def select\_all\_users():

with sqlite3.connect("userinfo.db") as db:

cursor = db.cursor()

cursor.execute("select \* from Userinfo order by UserID")

users = cursor.fetchall()

db.commit()

return users

def usr\_sign\_up():

def sign\_to\_Python():

np = new\_pwd.get()

npf = new\_pwd\_confirm.get()

nn = new\_name.get()

with sqlite3.connect("userinfo.db") as db:

if np != npf:

tk.messagebox.showerror('Error', 'Password and confirm password must be the same!')

elif np==npf:

with sqlite3.connect("userinfo.db") as db:

cursor = db.cursor()

cursor.execute("PRAGMA foreign\_keys = ON")

cursor.execute("select UserID, Username, Password, Map from Userinfo where Username=?",(nn,))

userinfo = cursor.fetchone()

db.commit()

if userinfo == None:

with sqlite3.connect("userinfo.db") as db:

cursor = db.cursor()

cursor.execute("PRAGMA foreign\_keys = ON")

sql = "insert into Userinfo (UserID, Username, Password, Map) values (?,?,?,?)"

users=select\_all\_users()

values=(len(users)+1, nn, np, "")

cursor.execute(sql,values)

db.commit()

tk.messagebox.showinfo('Welcome', 'You have successfully signed up!')

else:

tk.messagebox.showerror('Error', 'The user has already signed up!')

window\_sign\_up.destroy()

window\_sign\_up = tk.Toplevel(window)

window\_sign\_up.geometry('350x200')

window\_sign\_up.title('Sign up window')

new\_name = tk.StringVar()

new\_name.set('example@python.com')

tk.Label(window\_sign\_up, text='User name: ').place(x=10, y= 10)

entry\_new\_name = tk.Entry(window\_sign\_up, textvariable=new\_name)

entry\_new\_name.place(x=150, y=10)

new\_pwd = tk.StringVar()

tk.Label(window\_sign\_up, text='Password: ').place(x=10, y=50)

entry\_usr\_pwd = tk.Entry(window\_sign\_up, textvariable=new\_pwd, show='\*')

entry\_usr\_pwd.place(x=150, y=50)

new\_pwd\_confirm = tk.StringVar()

tk.Label(window\_sign\_up, text='Confirm password: ').place(x=10, y= 90)

entry\_usr\_pwd\_confirm = tk.Entry(window\_sign\_up, textvariable=new\_pwd\_confirm, show='\*')

entry\_usr\_pwd\_confirm.place(x=150, y=90)

btn\_comfirm\_sign\_up = tk.Button(window\_sign\_up, text='Sign up', command=sign\_to\_Python)

btn\_comfirm\_sign\_up.place(x=150, y=130)

try:

create\_table()

except:

pass

try:

create\_currentuser\_table()

except:

pass

window = tk.Tk()

window.title('Welcome to underground path finder')

window.geometry('450x300')

# welcome image

canvas = tk.Canvas(window, height=200, width=500)

image\_file = tk.PhotoImage(file='welcome.gif')

image = canvas.create\_image(0,0, anchor='nw', image=image\_file)

canvas.pack(side='top')

# user information

tk.Label(window, text='User name: ').place(x=50, y= 150)

tk.Label(window, text='Password: ').place(x=50, y= 190)

var\_usr\_name = tk.StringVar()

var\_usr\_name.set('example@python.com')

entry\_usr\_name = tk.Entry(window, textvariable=var\_usr\_name)

entry\_usr\_name.place(x=160, y=150)

var\_usr\_pwd = tk.StringVar()

entry\_usr\_pwd = tk.Entry(window, textvariable=var\_usr\_pwd, show='\*')

entry\_usr\_pwd.place(x=160, y=190)

# login and sign up button

btn\_login = tk.Button(window, text='Login', command=usr\_login)

btn\_login.place(x=170, y=230)

btn\_sign\_up = tk.Button(window, text='Sign up', command=usr\_sign\_up)

btn\_sign\_up.place(x=270, y=230)

window.mainloop()

## mapviewer.py

# -\*- coding: utf-8 -\*-  
import os  
import sys  
import sqlite3  
from PyQt5 import QtCore, QtGui, QtWidgets  
from PyQt5.QtCore import QCoreApplication  
import path\_find  
  
class GRview(QtWidgets.QGraphicsScene):  
 def \_\_init\_\_(self, parent=None):  
 super(GRview, self).\_\_init\_\_(parent)  
  
 def wheelEvent(self,x):  
 factor = x.delta()  
 if x.delta() > 0:  
 factor=2  
 else:  
 factor=0.5  
  
 self.map.scale(factor,factor)  
  
  
class Ui\_Form(object):  
 def setupUi(self, Form):  
 Form.setObjectName("Form")  
 Form.resize(900, 622)  
 Form.setMaximumSize(QtCore.QSize(16777215, 16777215))  
 icon = QtGui.QIcon()  
 icon.addPixmap(QtGui.QPixmap("logo.png"), QtGui.QIcon.Normal, QtGui.QIcon.Off)  
 Form.setWindowIcon(icon)  
 Form.setStyleSheet("background:rgb(255, 255, 255)")  
 self.layout\_2 = QtWidgets.QGridLayout(Form)  
 self.layout\_2.setObjectName("layout\_2")  
 self.layout = QtWidgets.QGridLayout()  
 self.layout.setContentsMargins(10, 10, 10, 10)  
 self.layout.setSpacing(10)  
 self.layout.setObjectName("layout")  
 self.Textcity = QtWidgets.QLabel(Form)  
 self.Textcity.setMaximumSize(QtCore.QSize(900, 600))  
 font = QtGui.QFont()  
 font.setFamily("Cambria")  
 font.setPointSize(20)  
 font.setBold(True)  
 font.setWeight(75)  
 self.Textcity.setFont(font)  
 self.Textcity.setObjectName("Textcity")  
 self.layout.addWidget(self.Textcity, 0, 0, 1, 1)  
 self.goal = QtWidgets.QLineEdit(Form)  
 self.goal.setMaximumSize(QtCore.QSize(900, 600))  
 font = QtGui.QFont()  
 font.setFamily("Cambria")  
 font.setPointSize(16)  
 font.setBold(True)  
 font.setWeight(75)  
 self.goal.setFont(font)  
 self.goal.setStyleSheet("background-color: rgb(255, 255, 255);")  
 self.goal.setObjectName("goal")  
 self.layout.addWidget(self.goal, 4, 3, 1, 1)  
 self.Textfrom = QtWidgets.QLabel(Form)  
 self.Textfrom.setMaximumSize(QtCore.QSize(900, 600))  
 font = QtGui.QFont()  
 font.setFamily("Cambria")  
 font.setPointSize(16)  
 font.setBold(True)  
 font.setWeight(75)  
 self.Textfrom.setFont(font)  
 self.Textfrom.setObjectName("Textfrom")  
 self.layout.addWidget(self.Textfrom, 1, 3, 1, 1)  
 self.start = QtWidgets.QLineEdit(Form)  
 self.start.setMaximumSize(QtCore.QSize(900, 600))  
 font = QtGui.QFont()  
 font.setFamily("Cambria")  
 font.setPointSize(16)  
 font.setBold(True)  
 font.setWeight(75)  
 self.start.setFont(font)  
 self.start.setStyleSheet("background-color: rgb(255, 255, 255);")  
 self.start.setObjectName("start")  
 self.layout.addWidget(self.start, 2, 3, 1, 1)  
 self.Textto = QtWidgets.QLabel(Form)  
 self.Textto.setMaximumSize(QtCore.QSize(900, 600))  
 font = QtGui.QFont()  
 font.setFamily("Cambria")  
 font.setPointSize(16)  
 font.setBold(True)  
 font.setWeight(75)  
 self.Textto.setFont(font)  
 self.Textto.setObjectName("Textto")  
 self.layout.addWidget(self.Textto, 3, 3, 1, 1)  
  
 self.Listcity = QtWidgets.QComboBox(Form)  
 self.Listcity.setMaximumSize(QtCore.QSize(900, 600))  
 self.Listcity.setStyleSheet("background-color: rgb(255, 255, 255);")  
 self.Listcity.setObjectName("Listcity")  
 self.layout.addWidget(self.Listcity, 0, 1, 1, 1)  
 with sqlite3.connect("userinfo.db") as db:  
 cursor = db.cursor()  
 cursor.execute("select \* from Current")  
 currentuser = cursor.fetchall()  
   
 db.commit()  
  
 s=currentuser[0][1]+","  
 start=0  
 end=s.find(",")  
 while len(s)>1:  
 self.Listcity.addItem(s[0:end])  
 start=end+2  
 s=s[start-1:]  
 end=s.find(",")  
 self.Listcity.addItem("ADD MAP")  
  
 self.Listcity.currentIndexChanged.connect(self.selectionchange)  
   
 self.tabPath = QtWidgets.QTabWidget(Form)  
 self.tabPath.setStyleSheet("background-color: rgb(255, 255, 255);")  
 self.tabPath.setObjectName("tabPath")  
 self.tab1 = QtWidgets.QWidget()  
 self.tab1.setObjectName("tab1")  
 self.tabPath.addTab(self.tab1, "")  
 self.tab2 = QtWidgets.QWidget()  
 self.tab2.setObjectName("tab2")  
 self.tabPath.addTab(self.tab2, "")  
   
 self.layout.addWidget(self.tabPath, 7, 0, 1, 3)  
   
 self.map = QtWidgets.QGraphicsView(Form)  
 self.map.setEnabled(True)  
  
 self.viewscene=GRview(self.map)  
 Qpix=QtGui.QPixmap(self.Listcity.currentText()+".png")  
 self.viewscene.addPixmap(Qpix)  
 self.map.setScene(self.viewscene)  
 self.map.scale(0.4,0.4)  
  
  
 sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Expanding)  
 sizePolicy.setHorizontalStretch(0)  
 sizePolicy.setVerticalStretch(0)  
 sizePolicy.setHeightForWidth(self.map.sizePolicy().hasHeightForWidth())  
 self.map.setSizePolicy(sizePolicy)  
 self.map.setMaximumSize(QtCore.QSize(16777215, 16777215))  
 font = QtGui.QFont()  
 font.setFamily("Cambria")  
 font.setPointSize(16)  
 font.setBold(True)  
 font.setWeight(75)  
 self.map.setFont(font)  
 self.map.setMouseTracking(True)  
  
 self.map.setDragMode(QtWidgets.QGraphicsView.ScrollHandDrag)  
 self.map.setObjectName("map")  
 self.layout.addWidget(self.map, 1, 0, 6, 3)  
 self.exit = QtWidgets.QPushButton(Form)  
 font = QtGui.QFont()  
 font.setFamily("Cambria")  
 font.setPointSize(14)  
 font.setBold(True)  
 font.setWeight(75)  
 self.exit.setFont(font)  
 self.exit.setStyleSheet("background-color: rgb(170,0, 0);")  
 self.exit.setObjectName("exit")  
 self.layout.addWidget(self.exit, 7, 3, 1, 1)  
 self.Button = QtWidgets.QPushButton(Form)  
 self.Button.setMaximumSize(QtCore.QSize(900, 600))  
 font = QtGui.QFont()  
 font.setFamily("Cambria")  
 font.setPointSize(16)  
 font.setBold(True)  
 font.setWeight(75)  
 self.Button.setFont(font)  
 self.Button.setStyleSheet("background-color: rgb(170, 170, 255);")  
 self.Button.setObjectName("Button")  
 self.layout.addWidget(self.Button, 5, 3, 1, 1)  
 self.layout.setColumnStretch(1, 10)  
 self.layout.setColumnStretch(2, 90)  
 self.layout.setColumnStretch(3, 10)  
 self.layout.setRowStretch(1, 10)  
 self.layout.setRowStretch(6, 60)  
 self.layout.setRowStretch(7, 40)  
 self.layout\_2.addLayout(self.layout, 0, 1, 1, 1)  
  
 self.retranslateUi(Form)  
 self.tabPath.setCurrentIndex(0)  
 QtCore.QMetaObject.connectSlotsByName(Form)  
 Form.setTabOrder(self.Listcity, self.start)  
 Form.setTabOrder(self.start, self.goal)  
 Form.setTabOrder(self.goal, self.Button)  
 Form.setTabOrder(self.Button, self.tabPath)  
 Form.setTabOrder(self.tabPath, self.exit)  
 Form.setTabOrder(self.exit, self.map)  
  
 def retranslateUi(self, Form):  
 \_translate = QtCore.QCoreApplication.translate  
  
 with sqlite3.connect("userinfo.db") as db:  
 cursor = db.cursor()  
 cursor.execute("select \* from Current")  
 currentuser = cursor.fetchall()  
 db.commit()  
 Form.setWindowTitle(\_translate("Form", "Underground path finder. Welcome "+currentuser[0][0]))  
 self.Textcity.setText(\_translate("Form", "City :"))  
 self.Textfrom.setText(\_translate("Form", "Start:"))  
 self.Textto.setText(\_translate("Form", "Goal:"))  
 self.tabPath.setTabText(self.tabPath.indexOf(self.tab1), \_translate("Form", "Best route"))  
 self.tabPath.setTabText(self.tabPath.indexOf(self.tab2), \_translate("Form", "Fewer transfer"))  
  
 self.exit.setText(\_translate("Form", "Log off"))  
 self.Button.setText(\_translate("Form", "Search"))  
  
  
 self.Button.clicked.connect(self.find\_path)  
 self.exit.clicked.connect(self.logoff)  
  
 def selectionchange(self,i):  
 self.viewscene=GRview(self.map)  
 Qpix=QtGui.QPixmap(self.Listcity.currentText()+".png")  
 self.viewscene.addPixmap(Qpix)  
 self.map.setScene(self.viewscene)  
 self.map.scale(0.4,0.4)  
  
   
 def find\_path(self):  
 command=self.Button.text()  
  
 cityname=self.Listcity.currentText()  
 start=self.start.text()  
 goal=self.goal.text()  
   
   
 if command=="Search":  
  
   
 result = path\_find.search(cityname,start,goal,0)  
 layout = QtWidgets.QFormLayout()  
 while len(result) >4:  
 layout.addRow("",QtWidgets.QLabel(str(result[0:4])))  
 result=result[4:]  
 layout.addRow("",QtWidgets.QLabel(str(result)))  
 self.tab1.setLayout(layout)  
  
 result = path\_find.search(cityname,start,goal,1)  
 layout = QtWidgets.QFormLayout()  
 while len(result) >4:  
 layout.addRow("",QtWidgets.QLabel(str(result[0:4])))  
 result=result[4:]  
 layout.addRow("",QtWidgets.QLabel(str(result)))  
 self.tab2.setLayout(layout)  
  
 self.tab1.setLayout(layout)  
   
 self.Button.setText("Clear")  
 else:  
 layout = QtWidgets.QFormLayout()  
 layout.addRow("",QtWidgets.QLabel(""))  
 self.Button.setText("Search")  
   
 def logoff(self):  
 try:   
 with sqlite3.connect("Userinfo.db") as db:  
 cursor = db.cursor()  
 sql = "DROP TABLE Current;"  
 cursor.execute(sql)  
 except:  
 pass  
   
 os.popen("python login.py")  
 QCoreApplication.quit()  
def main():  
 app = QtWidgets.QApplication(sys.argv)  
 w=QtWidgets.QWidget()  
 main=Ui\_Form()  
 main.setupUi(w)  
   
 w.show()  
 sys.exit(app.exec\_())

## path\_find.py

# -\*- coding: UTF-8 -\*-

import io

def build\_subway(files):

count=0

dictlines={}

sub=[]

subname=""

isCircle= False

for string in files:

if count%3 == 0:

"""

due to some lines are circle lines.

the underground need to update

"""

if string[0]=="\*":

subname=string[1:].strip("\n")

isCircle=True

else:

subname=string.strip("\n")

elif count%3 == 1:

sub = string.strip("\n").split(",")

if isCircle==True:

sub.append(sub[0])

else:

dictlines[subname]=sub

isCircle=False

count+=1

dictlines[subname]=sub

stations = set()

for key in dictlines.keys():

stations.update(set(dictlines[key]))

system = {}

for station in stations:

next\_station = {}

for key in dictlines:

if station in dictlines[key]:

line = dictlines[key]

idx = line.index(station)

if idx == 0:

next\_station[line[1]] = key

elif idx == len(line)-1:

next\_station[line[idx-1]]=key

else:

next\_station[line[idx-1]] = key

next\_station[line[idx+1]] = key

system[station] = next\_station

return (system)

def path\_search0(subway,start, goal):

"""Find the shortest path from start state to a state

with min change times such that is\_goal(state) is true."""

if start == goal:

return [start]

explored = set()

queue = [ [start] ]

while queue:

path = queue.pop(0)

s = path[-1]

for state, action in subway[s].items():

if state not in explored:

explored.add(state)

path2 = path + [action, state]

if state == goal:

return path2

else:

queue.append(path2)

return []

def path\_search1(subway,start, goal):

"""Find the shortest path from start state to a state

with min change times such that is\_goal(state) is true."""

if start == goal:

return [start]

explored = set()

queue = [ [start, ('', 0)] ]

while queue:

path = queue.pop(0)

s = path[-2]

linenum, changetimes = path[-1]

if s == goal:

return path

for state, action in subway[s].items():

if state not in explored:

linechange = changetimes

explored.add(state)

if linenum != action:

linechange += 1

path2 = path[:-1] + [action, state, (action, linechange)]

queue.append(path2)

queue.sort(key=lambda path:path[-1][-1])

return []

def search(cityname,start,goal,index):

try:

file=open(cityname+".txt","r")

except:

return("Can't find map and station data about "+cityname)

subway = build\_subway(file)

if start in subway == False or goal in subway == False:

return ("Can't find station")

file.close()

if index==0:

return(path\_search0(subway,start, goal))

else:

t=path\_search1(subway,start, goal)

t=t[0:len(t)-1]

return(t)

# Testing

## Test strategy

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test No.** | **Data** | **Reasons** | **Expected outcome** | **Actual outcome** | **Evidence** |
| **Login system** | | | | | |
| 1.1 | Username: test@gmail.com  Password: Test1234 | 1.To Check that correct credentials allow Users access to the userinfo database.  2.The password is case sensitive.  3.The password is protected by string "\*" | User will be able to Log in to the database and access data | User could successfully log in.  The welcome window and the mapviewer home page was opened | Figure 1,2 |
| 1.2 | Username: test@gmail.com  Password: 1231231231 | To Check that incorrect credentials do not allow Users access to the database. | User will not be able to Log in to the database and access data | User was not able to log in but was displayed a message box stating incorrect username or password. | Figure 3 |
| 1.3 | Sign up,  Username: signtesting@hotmail.com  Password: 12345678  Confirm password: 12345678 | To Check that user sign up allow write new data into database. | User will able to sign up to write new data in database. | User could successfully log in.  The success window was opened. | Figure 4 |
| 1.4 | Sign up,  Username: signtesting@hotmail.com  Password: 12345678  Confirm password: 123 | In case Password is not equal to the Confirm password | User will not be able to sign up to write new data in database. | User was not able to sign up but was displayed a message box stating already sign up or two password not same. | Figure 5,6 |
| **Map viewer** | | | | | |
| 2.1 | Select the city name | Check the viewer system works well, including the map image and the every button,label,text. | The map image is shown, the title of window shows the information about username | The image loaded successfully,  Underground path finder. Welcome +(username) | Figure 7 |
| 2.2 |  | Map viewer functionality checks, including zooming and dragging | Picture scales with factor, and scroll bar can move the position of image. | Picture enlarge as soon as the wheel goes up, picture shrink as soon as the wheel goes down. | Figure 8,9 |
| 2.3 | Select another city name | Check the viewer system works well, if change the other map | The viewer shows the image of the city. | The viewer shows the image of the city. | Figure 10,11 |
| 2.4 | Select the city name back | Make sure the signal of combobox can be recognize | Picture change back | Picture change back | Figure 12 |
| **Path finder** | | | | | |
| 3.1 | Start: Euston  Goal: Westminster  button clicked:Search | Check whether it links the function of algorithms, and whether it returns the right answers. | As user click the tab, the different solution will show in the different tabwidget. | Two different solution are shown in the tabwidget. One is the Best route with 3 transfers, one is the less transfer with 2 transfers. | Figure 13,14 |
| 3.2 | Start: Marylebone  Goal: Bank  button clicked:Search | Check whether it links the function of algorithms, and whether it returns the right answers. | As user click the tab, the different solution will show in the different tabwidget. | Two different solution are shown in the tabwidget. One is the Best route with 4 transfers, one is the less transfer with 3 transfers. | Figure 15,16 |

## 

## Testing screenshots

### Figure 1

### 

### Figure 2

### 

### 

### 

### Figure 3

### 

### Figure 4

### 

### 

### 

### Figure 5

### 

### Figure 6

### 

### Figure 7

### 

### Figure 8

### 

### 

### Figure 9

### 

### Figure 10

### 

### 

### Figure 11

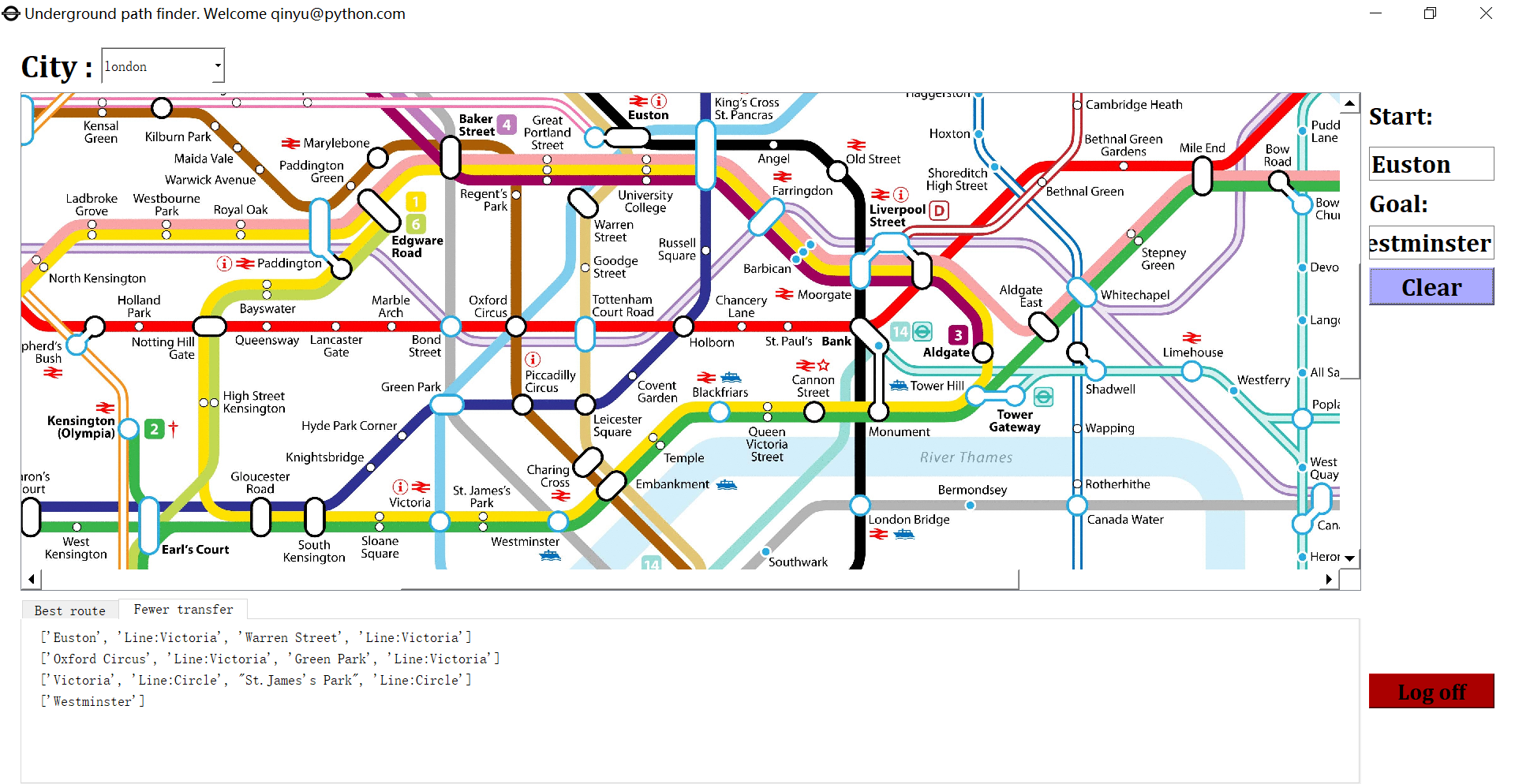
### **Figure** 12

### 

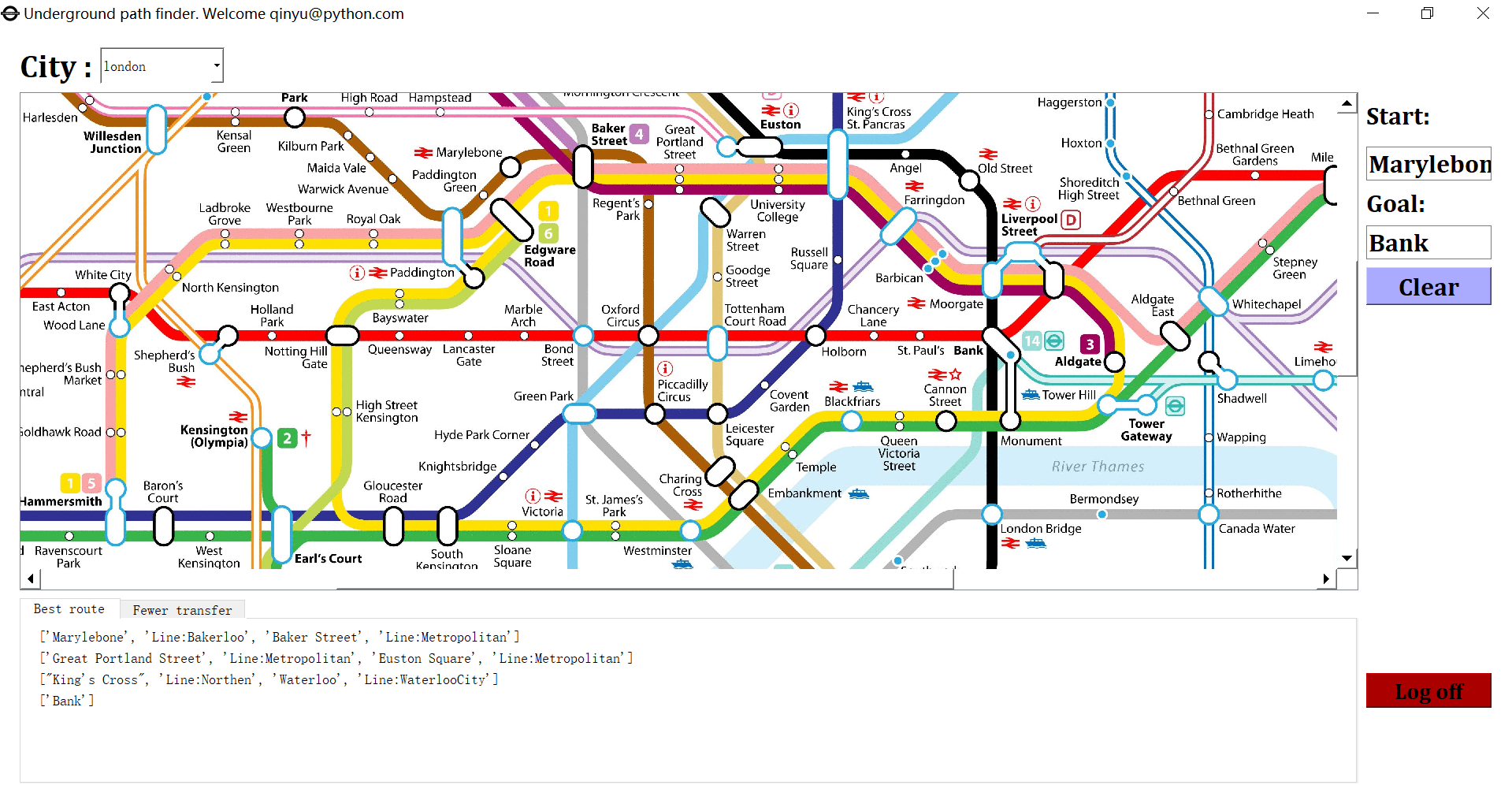
### 

### Figure 13

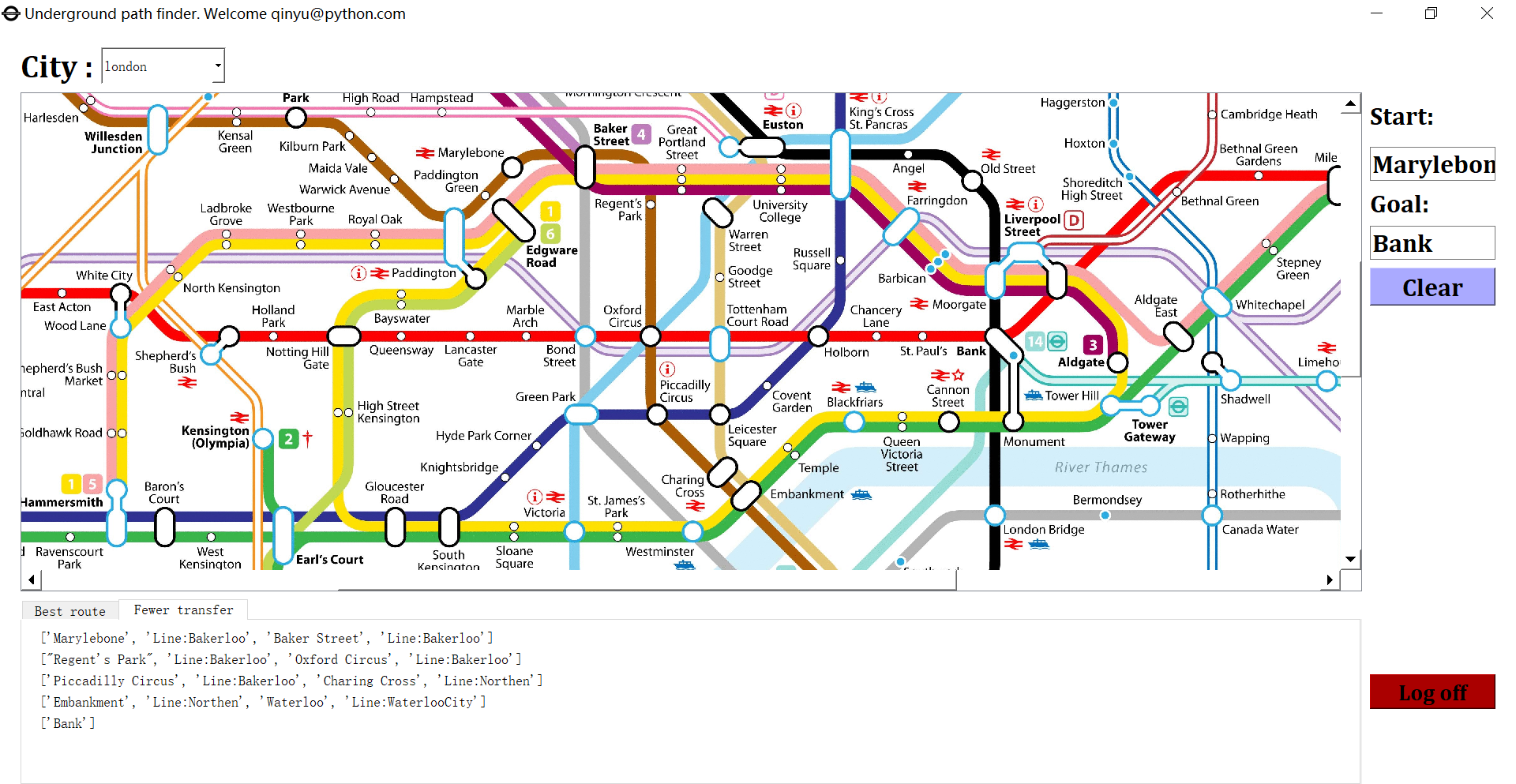
### **Figure** 14



### Figure 15



### Figure 16



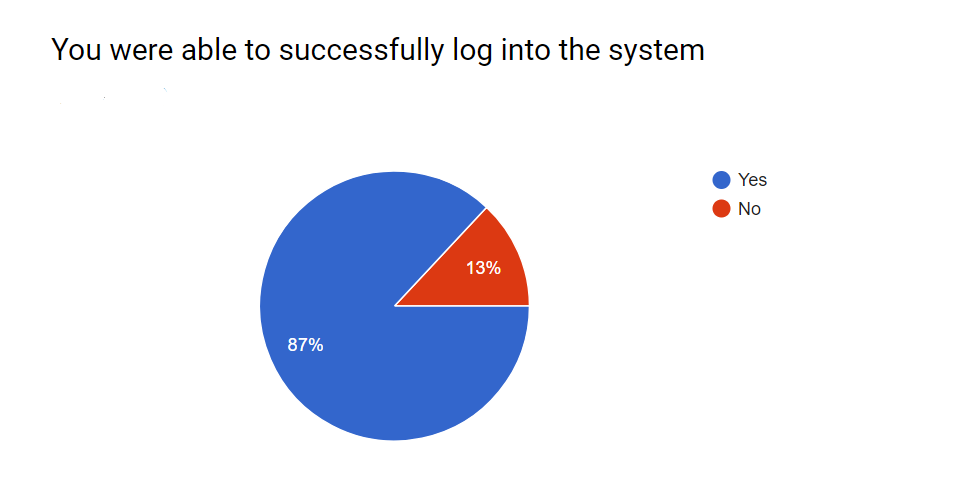
## 

## 

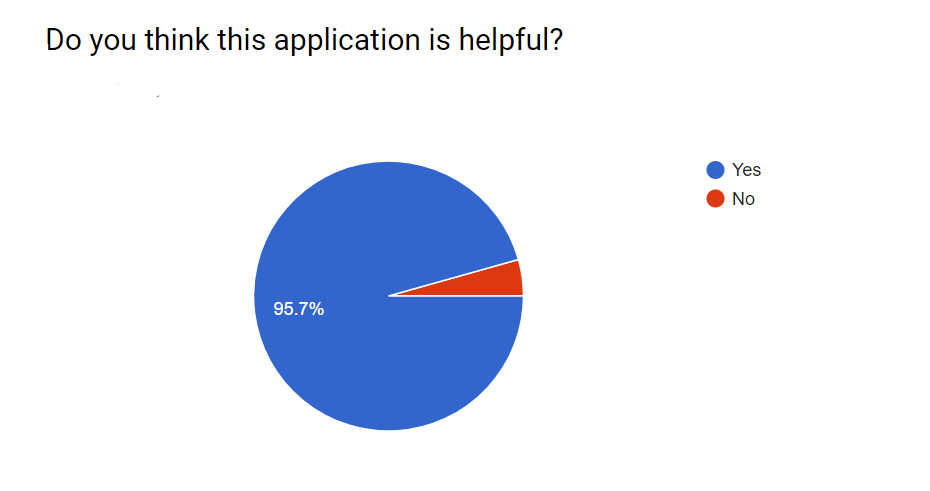
## Questionnaire

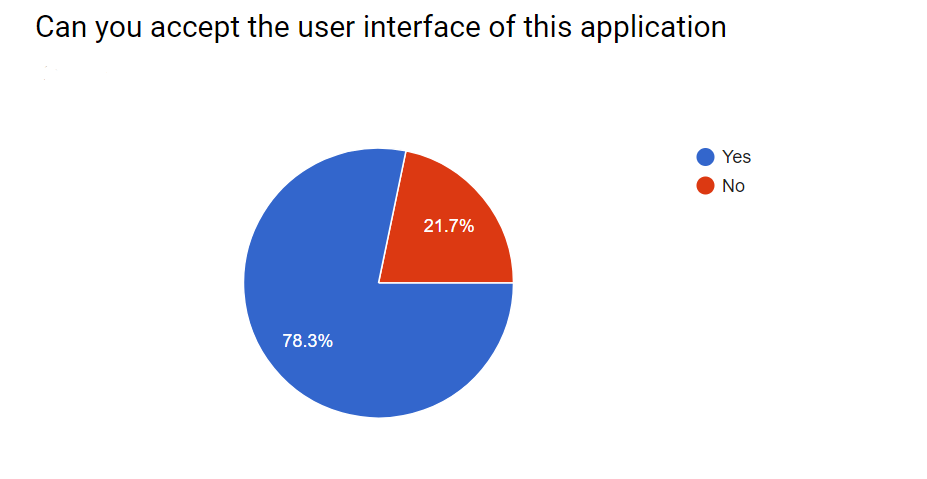
## 

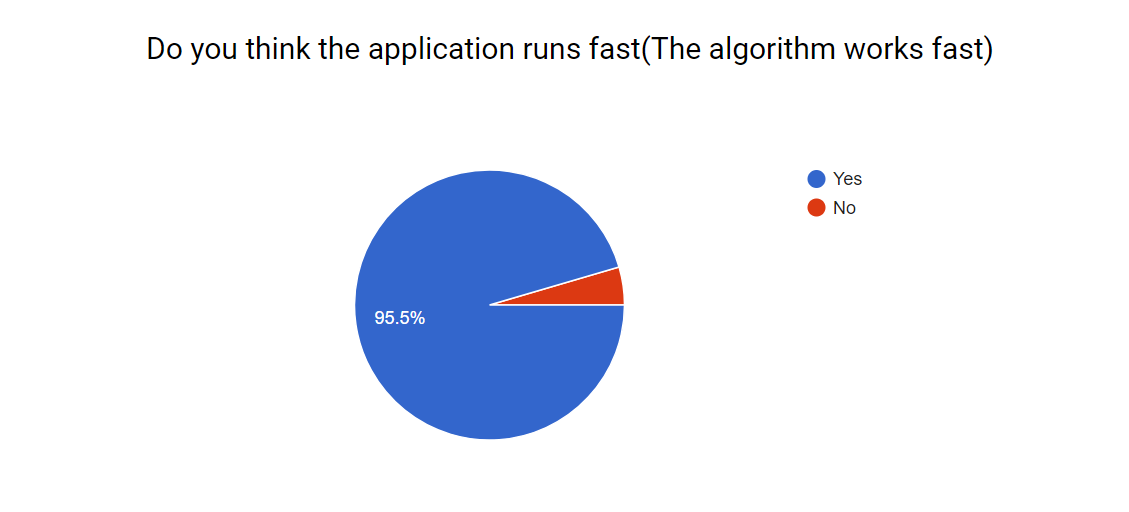
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| You were able to successfully log into the system | You were able to successfully sign up the system | Do you think this application is helpful? | Can you accept the user interface of this application | Do you think the application runs fast(The algorithm works fast) |
| Yes | Yes | Yes | Yes | Yes |
| Yes | Yes | Yes | No | Yes |
| Yes | Yes | Yes | Yes | Yes |
| Yes | Yes | Yes | Yes | Yes |
| No | Yes | Yes | Yes | Yes |
| Yes | Yes | Yes | Yes | No |
| Yes | Yes | Yes | Yes | Yes |
| No | No | Yes | No | Yes |
| Yes | Yes | Yes | Yes | Yes |
| No | No | No | Yes | Yes |
| Yes | Yes | Yes | Yes | Yes |
| Yes | Yes | Yes | Yes | Yes |
| Yes | No | Yes | Yes | Yes |
| Yes | Yes | Yes | Yes | Yes |
| Yes | Yes | Yes | Yes | Yes |
| Yes | Yes | Yes | No | Yes |
| Yes | Yes | Yes | Yes | Yes |
| Yes | Yes | Yes | Yes |  |
| Yes | Yes | Yes | No | Yes |
| Yes | Yes | Yes | Yes | Yes |
| Yes | Yes | Yes | Yes | Yes |
| Yes | Yes | Yes | No | Yes |
| Yes | Yes | Yes | Yes | Yes |











## Evaluation

|  |  |  |
| --- | --- | --- |
|  | Objective | Evaluation of objective |
| 1 | Have an intact login system | This objective was fully met |
| 2 | Be able to see a map viewer, and the map can be zoom with different size. | This objective was fully met |
| 3 | Programs can view maps of different cities. | This objective was fully met |
| 4 | Be able to select different places as start and destination | This objective was fully met |
| 5 | Be able to calculate the shortest path | This objective was fully met. |
| 6 | Show the process of algorithm clearly | This objective was achieved. By showing the graph step by step, the algorithm is shown clearly, though A\* algorithm is not included. |
| 7 | Be able to calculate the cost of time | No, there is no enough data. If I have more time, I probably would study and use web crawlers to get more detailed values. Also the walking time of people are different. |