Artificial Intelligence-Homework 3

Name: 张弛（ZHANG Chi）

SID: 12110821

Link of pull request: **Zhang Chi 12110821**

Introduction

In this assignment, we were asked to find the shortest path from given two stations on the London railway by using different algorithms. The algorithms I used consist of `BFS` , `Dijkstra` , `Bellman-Ford` , `Greedy BFS` , `A*,` and `Bidirectional A*` . And I used `number of iterations` and to test the performance of these algorithms. The result shows that `A*` (with cost defined by Euclidean distance and heuristics defined by Euclidean or Haversine distance) is the best algorithm to find the shortest path in the London railway.

- Algorithm description
  - BFS
  - Dijkstra's Algorithm (cost)
    - Initialize distances from the start node to all other nodes as infinity.
    - Set the distance to the start node as 0.
    - For each neighbor of the current node, update their distances if a shorter path (depending on **cost function**) is found.
    - Continue until the destination is reached.
  - Bellman-Ford Algorithm (heuristic)
    - Initialize distances from the start node to all other nodes as infinity.
    - Set the distance to the start node as 0.
    - Repeat the following process for (number of nodes - 1) iterations: For each edge in the graph, update the distance (calculated in the **heuristic function**) to the destination node if a shorter path is found.
    - Check for negative cycles: For each edge in the graph, If updating the distance to the destination node results in a shorter path, a negative cycle exists.
    - The final distances represent the shortest paths from the start node to all other nodes.
  - Greedy Best First Search (heuristic)
    - Instead of searching in all directions, in each step, it will take a step in the direction where the **heuristic function** is the smallest.
  - A* Algorithm (cost, heuristic)
    - Similar to Dijkstra's algorithm but uses a **heuristic function** to estimate the cost to reach the destination from the current node.
    - The A* algorithm combines the actual cost (from Dijkstra's) and the heuristic cost to make better decisions.
  - Bidirectional_A* Algorithm (cost, heuristic)
    - Similar to A*, but it can search for the shortest path from start_station and from end_station
    - Searching until two paths meet
- For cost and heuristic functions, I have 3 different measurement methods and 6 distinct combinations
  - Functions

Name	Description
Euclidean Distance (km)	Straight-line distance between two stations.
Haversine Distance (km)	Spherical distance or great circle distance: In a spherical coordinate system, latitude and longitude indicate the position of points on the earth's surface. To calculate the actual distance between two points, I use the function `geodesic` in package `geopy`
1（only for cost)	The time between the two subway stations is short, and there is little difference. However, the subway has experienced a process of acceleration-uniform speed-deceleration between two stations, and it takes a lot of time to stop and go at each station. So it is set to 1 to reduce the number of sites.

```python
def function(station1: Station, station2: Station, type: str) -> float:
    if type == "1":
        return 1.0
    if type == "Haversine":
        return geodesic(station1.position, station2.position).kilometers
    elif type == "Euclidean":
        # Convert latitude and longitude to kilometers
        lat_km = geodesic(
            (station1.position[0], station1.position[1]),
            (station1.position[0], station2.position[1]),
        ).kilometers
        lon_km = geodesic(
            (station1.position[0], station1.position[1]),
            (station2.position[0], station1.position[1]),
        ).kilometers
        euclidean_dist = sqrt(lat_km**2 + lon_km**2)         # Calculate Euclidean distance
        return euclidean_dist
```

- Combinations

Cost function type	Heuristic type	Weight for heuristic function (to make cost and heuristic in the same order of magnitude)
1	Euclidean	The actual distance between London Underground stations varies depending on the location of the stations. Some stations may be closer to each other, while others may be farther away. According to the general situation, the distance between London subway stations can be between 1 and 10 kilometers. (So the weight is set to 0.001)
1	Haversine	Set to 0.001
Euclidean	Euclidean	
Euclidean	Haversine	
Haversine	Euclidean	
Haversine	Haversine	

- Besides finishing the required .py file, I also created a .py file called `functions.py` , with functions `heuristics, cost,pathLength` and `random_choice` in it.

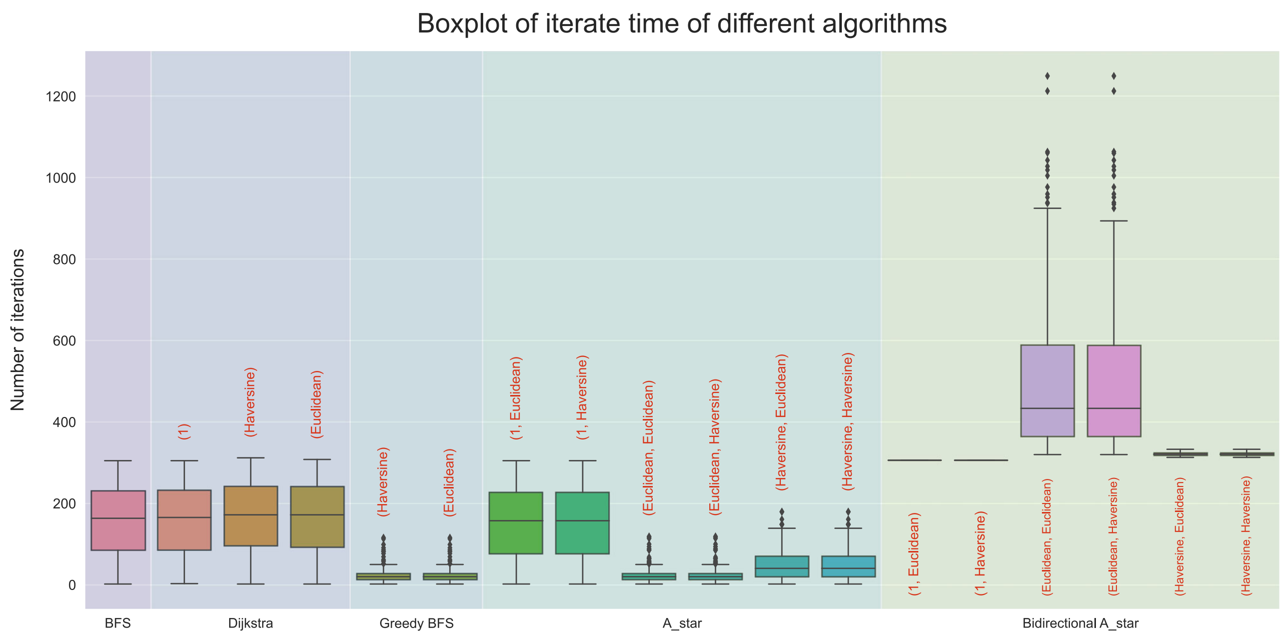
Experiment

Time test

In this part, I randomly choose 250 pairs of stations by using function `random_choice` . For each pair, I compute its `number of iterations` in all algorithms and get a table in size 250*21. Here is the boxplot.

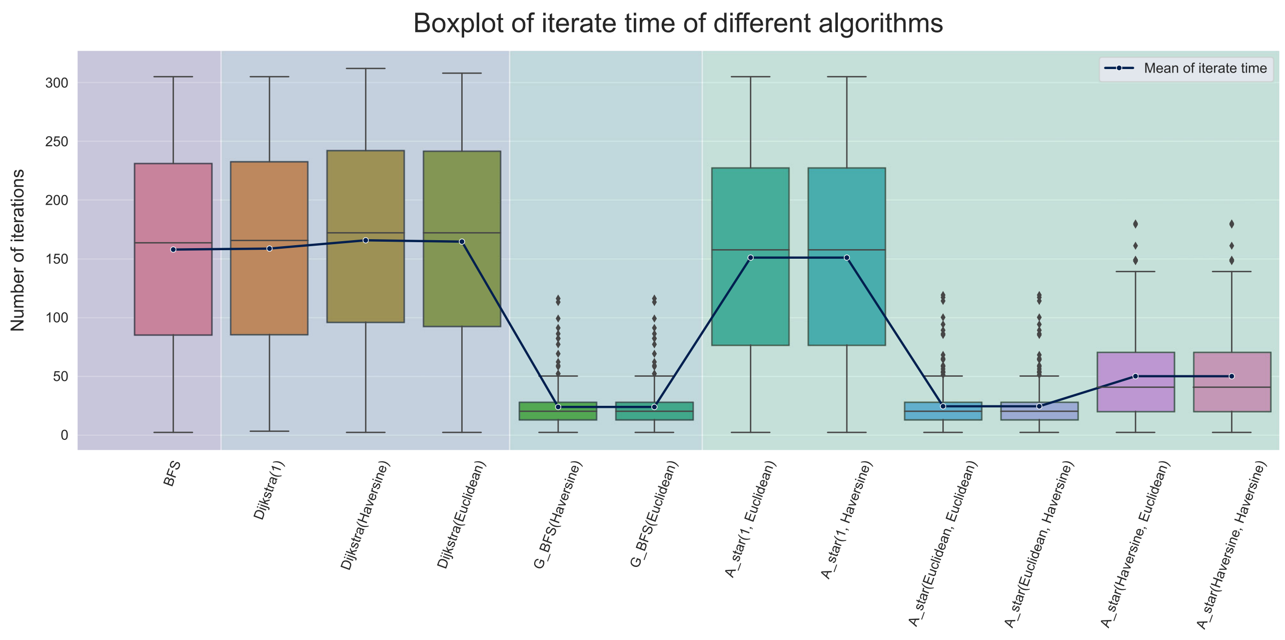
```python
# given a number `n`, randomly choose n pairs of station
def random_choice(station_name: list, number: int) -> list:
    all_pairs = [(a, b) for a in station_name for b in station_name if a != b]
    chosen_pairs = random.sample(all_pairs, number)
    return chosen_pairs
```

Because the amount of computation is small (See Appendix **The number of Time 0 when calculating real time**) in some functions, it is easy to have a time of 0, and is difficult to compare the time gap between algorithms, so the iteration times are used instead.



As the Bellman-Ford Algorithm iterates through all cases, here I just drop it. And we can find that **bi-directional A_star has a high time complexity**, which is unexpected. And I also drop it, then we look at the picture below.

It is apparent that the **overall iteration tie of `A_star` and `G_BFS` are lower than** `BFS` **and** `Dijkstra` , corresponding to lower time complexity . And for `A_star` , we can make a conclusion deciding the **best combinations: (Euclidean, Euclidean) and (Euclidean, Haversine)**.

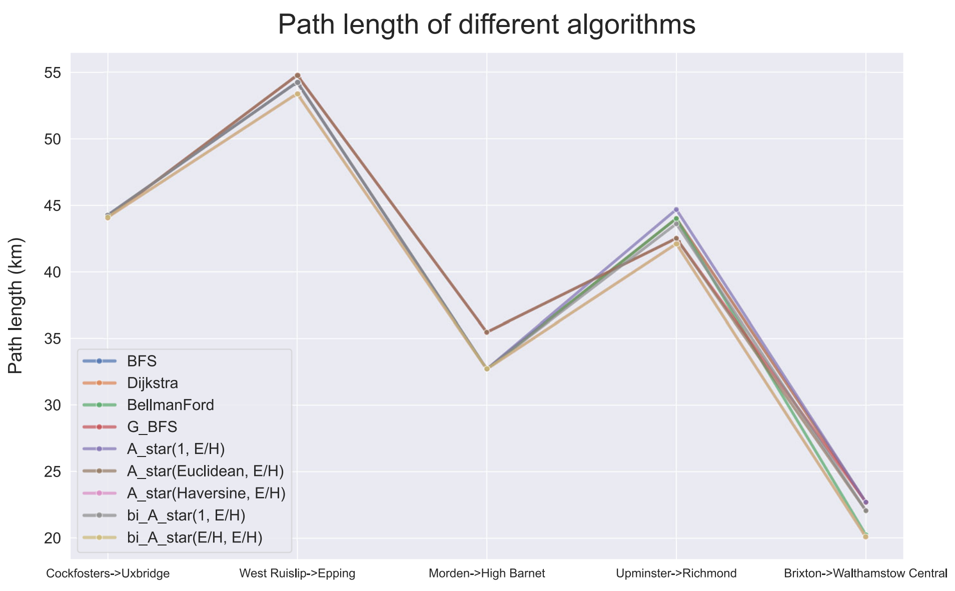


Path length test

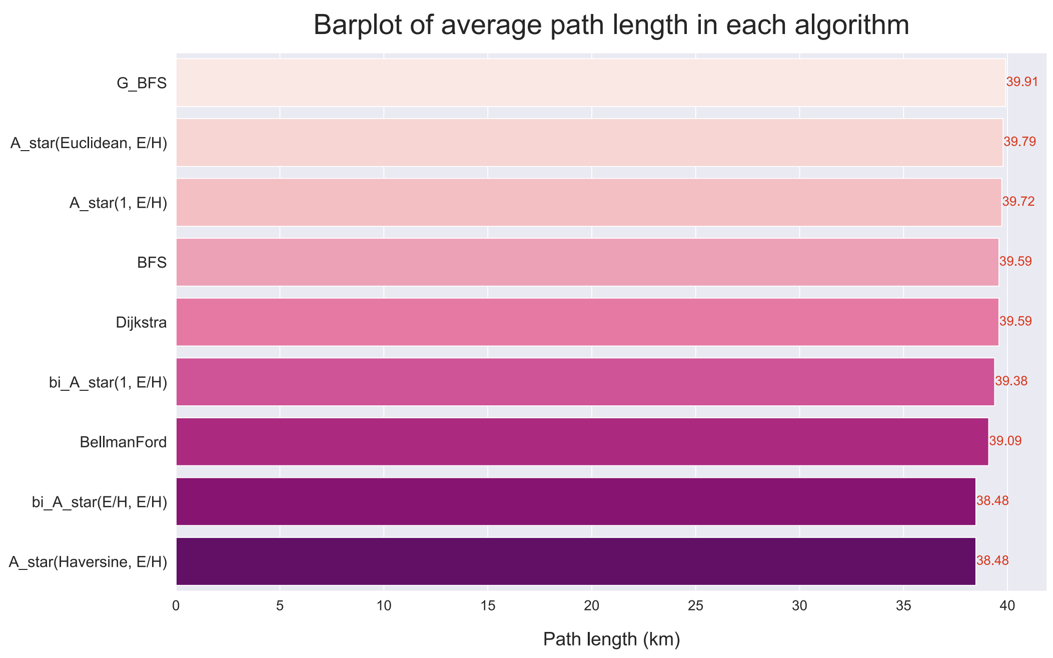
To compare path lengths, I choose 5 underground lines (see Appendix **Routine chosen**) and get the path from each algorithm. Then I compute the **Haversine distance** between every two adjacent stations and add them as the path length.

Particularly, there are algorithms with different costs or heuristic functions that can get the same result (in Appendix **Same path length in the algorithm**), so I combine them together in some cases.

According to the lineplot below (E/H stands for Euclidean or Haversine), for each algorithm, their differences are not large. And `bi_A_star(E/H, E/H)` 's path lengths are the shortest on each underground line.



Because there should be repeated occlusions in the lineplot. Here, the mean value of the path obtained by each algorithm is plotted with a bar graph. From the figure, `A_star(Haversine, E/H)` and `bi_A_star(E/H,E/H)` can get the shortest path. While `greedy_BFS` takes a short time, the distance is the longest. It is also unexpected that `BellmanFord` does not get the shortest as it tries all cases.



Conclusion

Main Findings

The conducted experiments involved testing various pathfinding algorithms, including `BFS, Dijkstra, Bellman-Ford, Greedy BFS, A*` and `Bidirectional A*.` The evaluation criteria focused on the number of iterations and path length, with different cost and heuristic functions. Here are the key observations and comparisons:

- **Time Complexity:**
- A* and Greedy BFS demonstrated lower overall iteration times compared to BFS and Dijkstra, indicating more efficient performance.
    - Bidirectional A* showed unexpectedly high time complexity, which may be an area for further investigation and optimization.
- **Path Length:**
- A* with Euclidean and Haversine heuristics consistently produced the shortest paths, suggesting the effectiveness of these combinations.
    - Greedy BFS, while computationally efficient, tended to generate longer paths, emphasizing the trade-off between computation speed and path optimality.
    - Bellman-Ford, despite exploring all cases, did not consistently yield the shortest paths, highlighting its limitations in certain scenarios.

The Algorithmic Choices are as follows:

- **A* vs. Greedy BFS:** A* outperformed Greedy BFS in both time complexity and path length, making it a preferred choice for this specific application.
- **Bidirectional A*:** Despite its higher time complexity, Bidirectional A* may still be advantageous in scenarios where finding paths from both start and end points simultaneously is crucial.
- **Heuristic Functions:**
- The combination of Euclidean distance for cost and heuristic, particularly in A*, consistently delivered optimal results.
    - Haversine distance, being more accurate for geographical locations, also performed well.

Answer to Research Question: A* with Euclidean and Haversine heuristics proved to be the most reliable choice.

Future Directions for Improvement

1. **Introducing Noise to Heuristic Functions:** Adding controlled noise to heuristic functions may be explored to assess the algorithms' robustness in real-world, dynamic environments.
2. **Combining Multiple Heuristic Functions:** Experimenting with the combination of multiple heuristic functions, such as averaging their values, could enhance the algorithms' adaptability to diverse scenarios.
3. **Considering Transfer Time and Penalties:** Incorporating transfer time variations (e.g., differentiating between 1, 2, or 5 units) and introducing penalties for transfers could provide a more comprehensive analysis of the algorithms in a practical commuting context.
4. **Sensitivity Analysis:** Conducting sensitivity analysis on key parameters can offer insights into the algorithms' resilience to variations in input data and guide further optimization.

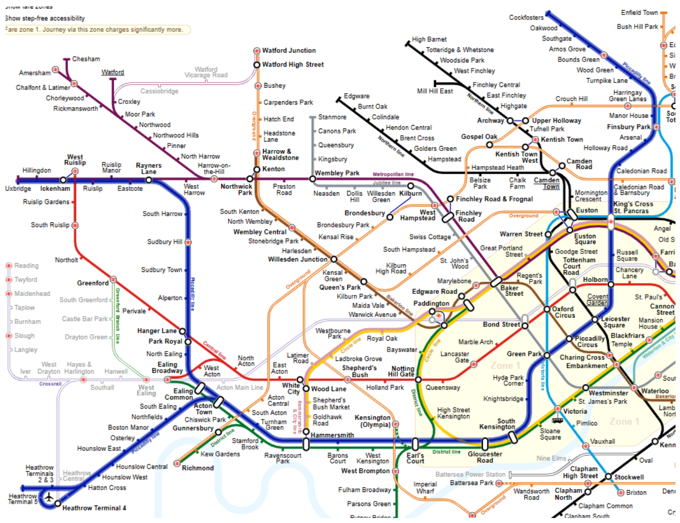
Appendix

The number of Time 0 when calculating real time

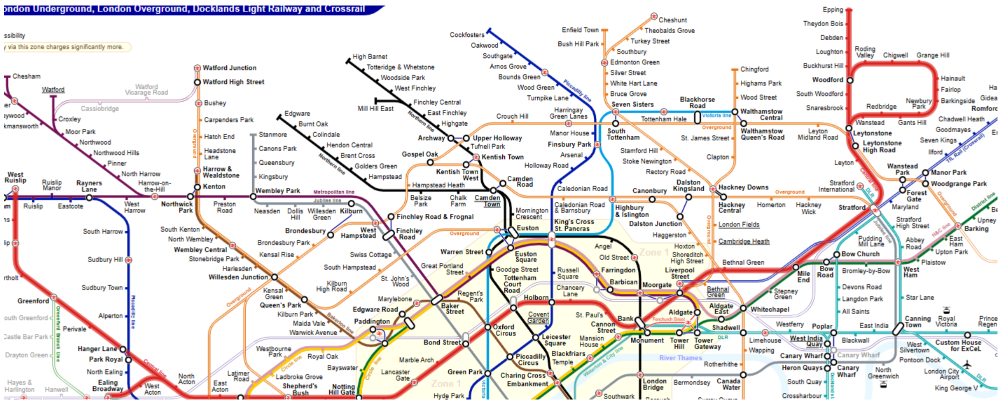
	Number of Time 0
BFS	115
Dijkstra(1)	92
Dijkstra(Haversine)	0
Dijkstra(Euclidean)	75
BellmanFord(1)	0
BellmanFord(Euclidean)	0
G_BFS(Haversine)	3
G_BFS(Euclidean)	214
A_star(1, Euclidean)	98
A_star(1, Haversine)	3
A_star(Euclidean, Euclidean)	165
A_star(Euclidean, Haversine)	1
A_star(Haversine, Euclidean)	2
A_star(Haversine, Haversine)	1
bi_A_star(1, Euclidean)	166
bi_A_star(1, Haversine)	0
bi_A_star(Euclidean, Euclidean)	164
bi_A_star(Euclidean, Haversine)	2
bi_A_star(Haversine, Euclidean)	2
bi_A_star(Haversine, Haversine)	0

Routine chosen for Path length test

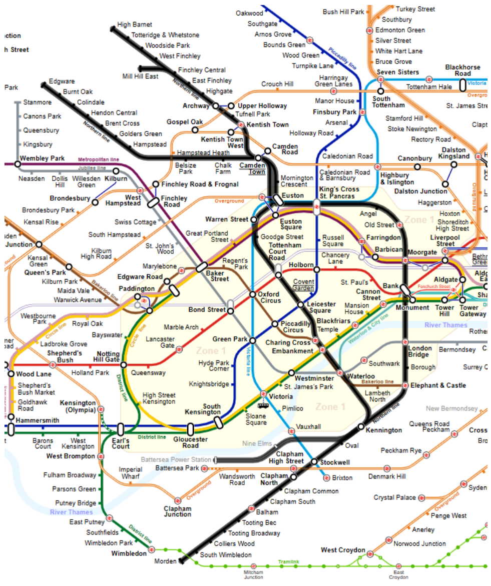
1. Cockfosters -> Uxbridge (in blue)



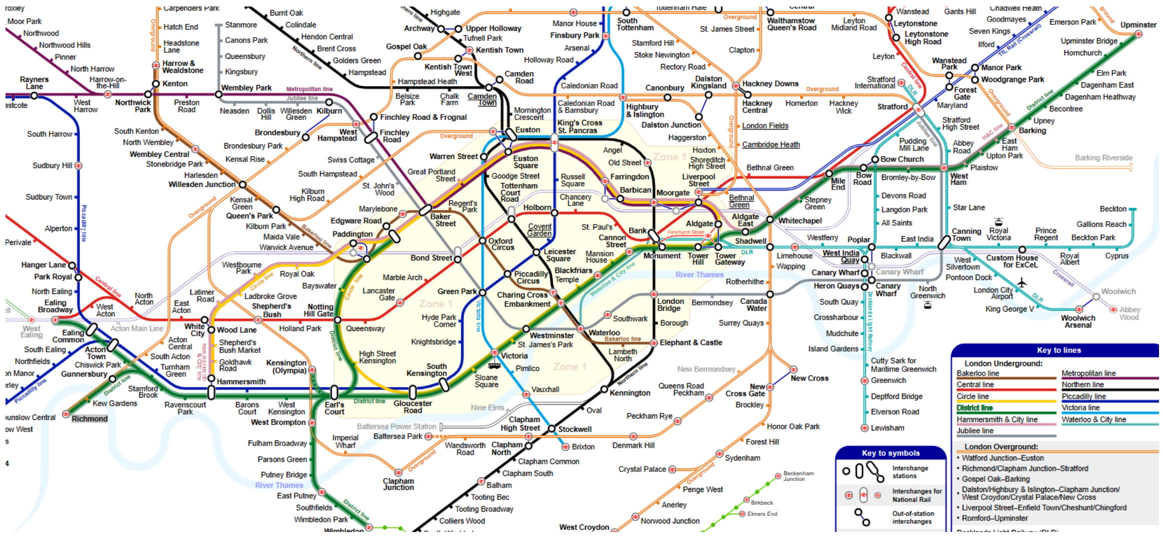
2. West Ruislip -> Epping (in red)



3. Morden -> High Barnet (in black)



4. Upminster -> Richmond (in green)



5. Brixton -> Walthamstow Central (in light blue)

High Barnet
Totteridge & Whetstone
Woodside Park
West Finchley
Finchley Central
East Finchley
Highgate
Mill Hill East
Archway
Upper Holloway
Tufnell Park
Kentish Town
Gospel Oak
Kentish Town West
Camden Road
Crouch Hill
Harringay Green Lanes
Manor House
Finsbury Park
Arsenal
Holloway Road
Caledonian Road
Caledonian Road & Barnsbury
Southgate
Arnos Grove
Bounds Green
Wood Green
Turnpike Lane
Edmonton Green
Silver Street
White Hart Lane
Bruce Grove
Seven Sisters
Blackhorse Road
Tottenham Hale
South Tottenham
Stamford Hill
Stoke Newington
Rectory Road
Chingford
Highams Park
Wood Street
Walthamstow Central
Walthamstow Queen's Road
St. James Street
Clapton
Dalston Kingsland
Hackney Downs
Hackney Central
Canonbury
Highbury & Islington
Dalston Junction
Haggerston
Hoxton
London Fields
Cambridge Heath
Bethnal Green
Shoreditch High Street
Liverpool Street
Moorgate
Barbican
Farringdon
Old Street
Angel
King's Cross St. Pancras
Euston
Euston Square
Mornington Crescent
Camden Town
Chalk Farm
Belsize Park
Hampstead Heath
Finchley Road & Frognal
Finchley Road
West Hampstead
Kilburn
Swiss Cottage
Hampstead
Golders Green
Brent Cross
St. John's Wood
Marylebone
Edgware Road
Paddington
Baker Street
Regent's Park
Great Portland Street
Warren Street
Goodge Street
Tottenham Court Road
Russell Square
Chancery Lane
Holborn
Covent Garden
Oxford Circus
Bond Street
Marble Arch
Lancaster Gate
Queensway
Hyde Park Corner
Green Park
Piccadilly Circus
Leicester Square
Charing Cross
Embankment
Temple
Blackfriars
Mansion House
Cannon Street
St. Paul's
Bank
Monument
Aldgate
Aldgate East
Tower Hill
Tower Gateway
Shadwell
Bethnal Green
Whitechapel
Limehouse
Wapping
Rotherhithe
Canada Water
Surrey Quays
Bermondsey
London Bridge
Borough
Southwark
Waterloo
Westminster
St. James's Park
Victoria
Knightsbridge
High Street Kensington
South Kensington
Sloane Square
Pimlico
Vauxhall
Lambeth North
Elephant & Castle
Kennington
Oval
Stockwell
Brixton
Clapham North
Clapham High Street
Wandsworth Road
Nine Elms
Battersea Power Station
Battersea Park
Imperial Wharf
Earl's Court
Gloucester Road
Queens Road Peckham
Peckham Rye
Denmark Hill
New Cross Gate
New Cross
Brockley
Honor Oak
Forest Hill
Sydenham
New Bermondsey
Zone 1
River Thames
Overground
DLR

Same path length in the algorithm

- For BFS
- For Dijkstra, they are the same
- For Bellman-Ford: BellmanFord(Haversine), BellmanFord(Euclidean) same
- For GBFS: GBFS(Haversine), GBFS(Euclidean) same
- For A_star:
- A_star(1,H), A_star(1,E) same
    - A_star(E,H), A_star(E,E) same
    - A_star(H,H), A_star(H,E) same
- For bi_A_star:
- bi_A_star(1,E), bi_A_star(1,H) same
    - bi_A_star(E,H), bi_A_star(E,E), bi_A_star(H,H), bi_A_star(H,E) same