CME 2201 - Assignment 2

Journey Planner for Izmir Public Transportation

A journey planner (or trip planner) is a specialized electronic search engine that finds one or more journey (trip) suggestions between an origin and a destination. This system assists travelers for planning their journey.

Izmir is Turkey's the third largest city and has four existing public transport companies including ESHOT for bus transportation, Izmir Metro Inc. for metro, IZBAN Inc. for the rail system and IZDENIZ for maritime transport. All of these systems form a complete transportation network with 319 lines and 6708 stops/stations.

Dijkstra's algorithm forms the basis of modern journey planner search algorithms and provides an optimal solution to simple searches. While planning routes in such a combined network, some constraints as switching the mode of transportation frequently or unacceptable transfer counts must be considered. For instance, Table 1 shows the result path produced by the original Dijkstra's algorithm for a query from the origin stop '10036-Konak' to the destination stop '40120-Tınaztepe Kampüs Son Durak'. The path consists of 23 stops and 12 different lines, thus 11 transfers are required to complete the journey. Producing such an unefficient path should be avoided.

Table 1. The shortest path produced by the original Dijkstra's algorithm.

Step	Stop Id	Stop Name	Line No	Line Name
1	10036	Konak	72	İşçievleri-Konak
2	10023	Bahribaba Alt	7	Sahilevleri-Konak
3	10015	Bahribaba	152	Gaziemir-Konak
4	10241	Kız Yurdu	43	Yapıcıoğlu-Konak
5	12061	Eşref Paşa	23	Uzundere-Konak
6	12063	Yağhaneler	90	Gaziemir- Halkapınar Metro
7	10617	Elka	870	Hıfzıssıhha- Tınaztepe
8	11873	Köprü	870	Hıfzıssıhha- Tınaztepe
9	40001	Nato	576	Tınaztepe- Halkapınar Metro
10	41177	Şirinyer Aktarma	36	Buca-Gümrük
11	40199	Koşu Yolu	36	Buca-Gümrük
12	40201	İstasyon	36	Buca-Gümrük
13	40207	Vali Rahmi Bey	36	Buca-Gümrük
14	40209	Şehitler Parkı	36	Buca-Gümrük

15	40733	Buca Devlet Hastanesi	36	Buca-Gümrük
16	40735	Çevik Bir	36	Buca-Gümrük
17	40737	Buca Sağlık Ocağı	36	Buca-Gümrük
18	40739	Buca Üçkuyular Meydan	604	Sebze Hali-Ayakkabıcılar Sitesi
19	40079	Hasan Ağa Bahçesi	176	Ufuk Mahallesi- Demirciköy
20	40067	Eski Mezarlık	176	Ufuk Mahallesi- Demirciköy
21	40069	Fabrika	176	Ufuk Mahallesi- Demirciköy
22	40071	Begos	671	Narlıdere- Tınaztepe
23	40120	Tınaztepe Kampüs Son Durak		

1 Neighbor Stops

To make point to point queries in a transportation network, some sort of walk-distance edges are required, so any stage of the journey can be covered by walk or passengers may walk between the stops while transferring between two different lines. Walk-distance edges are also providing to link each of the transportation networks (bus, train, metro, ferry etc.). Two stops *u* and *v* are labeled as neighbor stops by adding walk-distance edges between them, if a road segment is available to pedestrians and $\mathrm{dist}(u, v)$ is less than the maximum allowed walking distance.

2 Representation of the Transportation Graph

In this assignment, you are expected to represent each stop (bus, train, metro, ferry) as a node and to represent each line connecting two consecutive stops in a certain direction as a directed edge to form a transportation graph. This graph is a directed graph as illustrated in Figure 1.

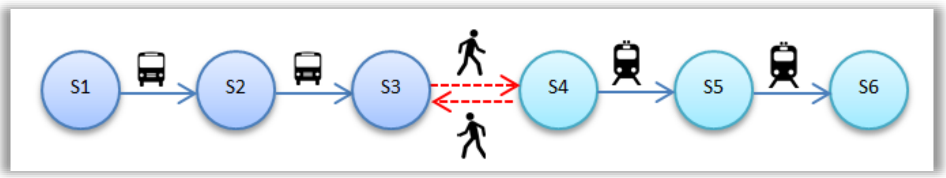


Figure 1. Illustration of a transportation graph.

3 Path Finding

3.1 Direct Routes

Direct paths start with the origin stop and reach to the destination stop with no transfer.

3.2 Routes Containing One Transfer

These paths start with an origin line (a line use the origin stop) and ends up in a destination line (a line use the destination stop). Origin and destination lines must be connected in a transfer stop or a walk must exist between two lines as illustrated in Figure 2. Two consecutive walk is not allowed. A path can contains max. three walks (at the beginning, in middle, and at the end of the path).

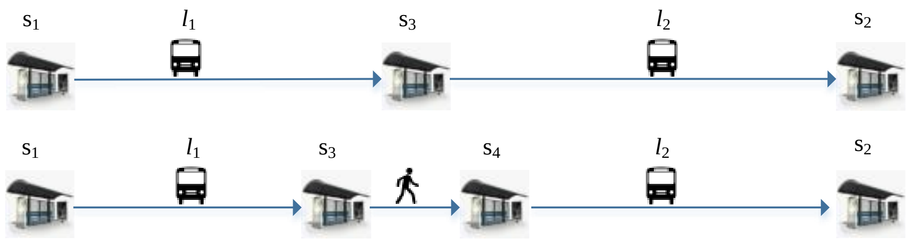


Figure 2. Illustration of the routes containing one transfer.

3.3 Journey Planner Search Engine

Develop a modified Dijkstra's algorithm to find alternative journeys between the given origin and destination stops. The alternative paths must include only direct routes and the routes containing one transfer. You should modify the original Dijkstra's algorithm to limit transfers. Several runs of the algorithm could be required to obtain enough alternative journeys.

You should consider two optimization criteria: ***fewer stops*** and ***minimum distance***. In the first case, you should use equal edge weights in the graph. In the second case, you should use the given distance values (in meters) for the consecutive stops.

4 Experiments

4.1 Sample Results

A sample query from the origin stop “40015-Buca Belediye Sarayı” to destination stop “40124-Tınaz Tepe” retuns following two paths (first one is a direct path, second one contains one transfer).

Table 2. Sample Results

Path 1. 104 - Walk
Line: 104-TINAZTEPE – KONAK (Direction - 0) Origin Stop: 40015 - Buca Belediye Sarayı Destination Stop: 40121 - Tınaz Tepe Stop Count: 11
Line: Walk Origin Stop: 40121 - Tınaz Tepe Destination Stop: 40124 - Tınaztepe Son Durak Walk-distance: 38 m
Path 2. 805 - 878
Line: 805-ÇAMLIKULE - ŞİRİNYER AKTARMA (Direction - 1) Origin Stop: 40015 - Buca Belediye Sarayı Destination Stop: 40737 - Buca Sağlık Ocağı Stop Count: 5
Line: 878-TINAZTEPE - ŞİRİNYER AKTARMA (Direction - 1) Origin Stop: 40737 - Buca Sağlık Ocağı Destination Stop: 40124 - Tınaz Tepe Stop Count: 6

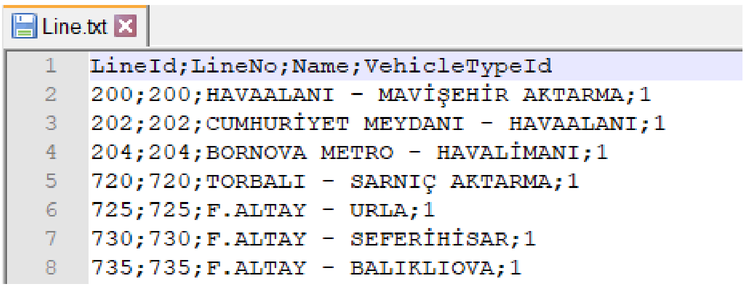
There could be so many alternative paths as a result of a query. It is not necessary to report all of them to the user. These paths should be filtered according to the preferred criteria and only *k*-shortest-paths (*k* is a parameter, but you can assume *k* is 5) should be returned to the user.

4.2 Test Queries

You should test your algorithm with the given origin-destination stop list. Report the average query time and result count in your project report. In addition, explain the details of your algorithm.

5 Provided Resources

- Line lists (line.txt)

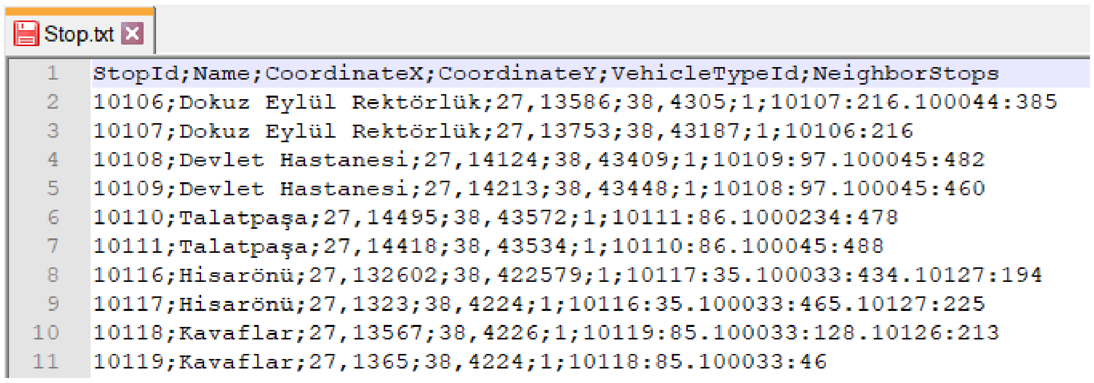


```
LineId;LineNo;Name;VehicleTypeId
200;200;HAVAALANI - MAVİŞEHİR AKTARMA;1
202;202;CUMHURİYET MEYDANI - HAVAALANI;1
204;204;BORNOVA METRO - HAVALİMANI;1
720;720;TORBALI - SARNIÇ AKTARMA;1
725;725;F.ALTAY - URLA;1
730;730;F.ALTAY - SEFERİHİSAR;1
735;735;F.ALTAY - BALIKLIOVA;1
```

Table 3. Vehicle Types

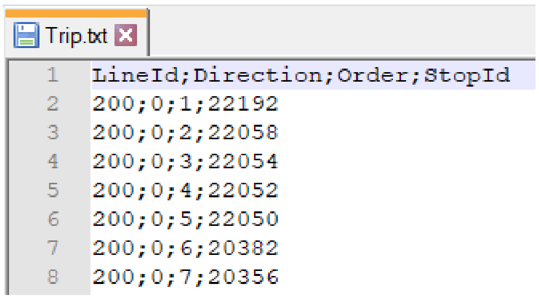
Vehicle Type Id	Vehicle Type
1	Bus
2	Rail
3	Metro
4	Ferry

- Stop list (stop.txt)



```
StopId;Name;CoordinateX;CoordinateY;VehicleTypeId;NeighborStops
10106;Dokuz Eylül Rektörlük;27,13586;38,4305;1;10107:216.100044:385
10107;Dokuz Eylül Rektörlük;27,13753;38,43187;1;10106:216
10108;Devlet Hastanesi;27,14124;38,43409;1;10109:97.100045:482
10109;Devlet Hastanesi;27,14213;38,43448;1;10108:97.100045:460
10110;Talatpaşa;27,14495;38,43572;1;10111:86.1000234:478
10111;Talatpaşa;27,14418;38,43534;1;10110:86.100045:488
10116;Hisarönü;27,132602;38,422579;1;10117:35.100033:434.10127:194
10117;Hisarönü;27,1323;38,4224;1;10116:35.100033:465.10127:225
10118;Kavaflar;27,13567;38,4226;1;10119:85.100033:128.10126:213
10119;Kavaflar;27,1365;38,4224;1;10118:85.100033:46
```

- Trip list (trip.txt)

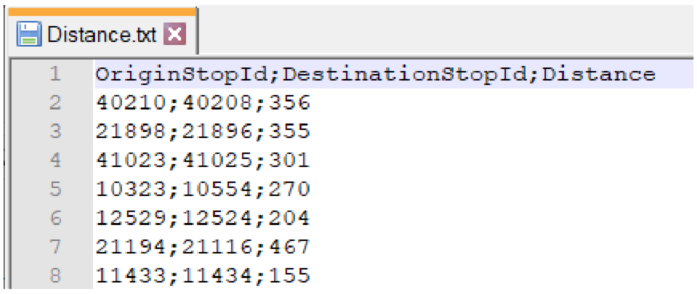


```
LineId;Direction;Order;StopId
200;0;1;22192
200;0;2;22058
200;0;3;22054
200;0;4;22052
200;0;5;22050
200;0;6;20382
200;0;7;20356
```

Table 4. Directions

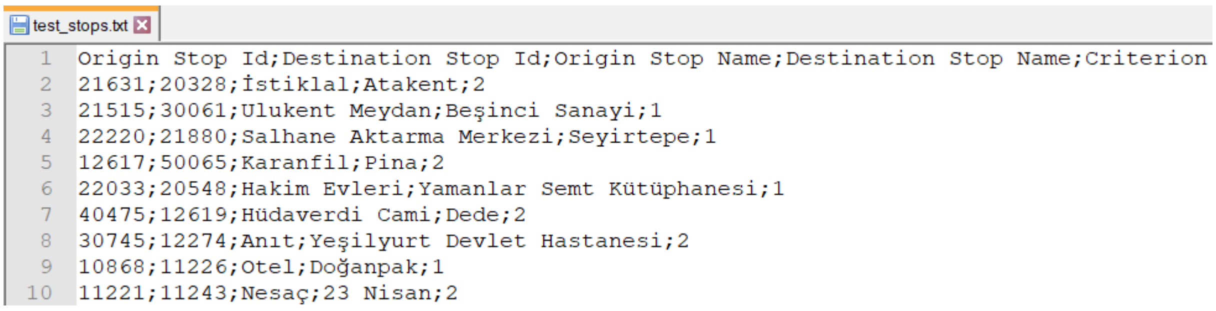
Direction Id	Direction
0	Travel in one direction (e.g. outbound travel)
1	Travel in the opposite direction (e.g. inbound travel)

- Distance list (distance.txt)



```
OriginStopId;DestinationStopId;Distance
40210;40208;356
21898;21896;355
41023;41025;301
10323;10554;270
12529;12524;204
21194;21116;467
11433;11434;155
```

- Origin-Destination Stop List (test_stops.txt)



```
Origin Stop Id;Destination Stop Id;Origin Stop Name;Destination Stop Name;Criterion
21631;20328;İstiklal;Atakent;2
21515;30061;Ulukent Meydan;Beşinci Sanayi;1
22220;21880;Salhane Aktarma Merkezi;Seyirtepe;1
12617;50065;Karanfil;Pina;2
22033;20548;Hakim Evleri;Yamanlar Semt Kütüphanesi;1
40475;12619;Hüdaverdi Cami;Dede;2
30745;12274;Anıt;Yeşilyurt Devlet Hastanesi;2
10868;11226;Otel;Doğanpak;1
11221;11243;Nesaç;23 Nisan;2
```

Table 5. Criteria

Criteria Id	Criterion
1	Fewer stops
2	Minimum distance

Due date

22.12.2019 Sunday 23:59. Late submissions are not allowed.

Requirements

- Usage of *Java* programming language and *Edge List Graph Data Structure* are required.
- *Object Oriented Programming (OOP)* principles must be applied.
- *Exception handling* must be used when it is needed.

Submission

You must upload your all '.java' files as an archive file (.zip or .rar). Your archived file should be named as 'studentnumber_name_surname.rar.zip', e.g., 2007510011_Ali_Yılmaz.rar via Google Classroom.

Prepare and upload a report with descriptions of your data structure, java code, and performance matrix.

Code Control

Control of the project will be on 23-27 of December. Research assistants will control your assignments in their office (door no: 124). Therefore, you should fill in the information on the Google Form to schedule. You will have 15 minutes to show your assignment. Please do not forget to bring your laptops while coming to the assignment control!

Plagiarism Control

The submissions will be checked for code similarity. Copy assignments will be graded as zero, and they will be announced in the Classroom.

Grading Policy

Job	**Percentage**
Usage of and Edge List Graph Data Structure, OOP and Try-Catch	%30
Journey Planner Search Engine implementation by Dijkstra's algorithm	%50
Experimental Results	%20