Hacettepe University
Department of Computer Engineering
BBM204 Programming Lab.
Spring 2021
Programming Assignment 3

Subject: Directed Graphs and Graph Search
Deadline: 05.05.2021 23:59
Programming Language: Java

1. Introduction

In this experiment, you are going to practice on directed graphs and graph search by using java programming language. Also, you will perform file I/O operations.

2. Background

A Graph is a non-linear data structure consisting of nodes and edges as shown in Figure 1. The nodes are sometimes also referred to as vertices and the edges are lines or arcs. They are data structures used to model pairwise relations between objects. The vertices are used to model the objects and edges connecting two vertices that are used to model the pairwise relation between those objects. They are used to solve many real-life problems. Graphs are used to represent networks. The networks may include paths in a city or telephone network or circuit network.

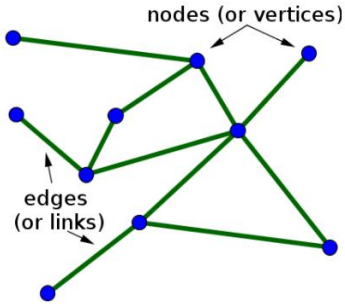

Figure 1. Example graph with 10 nodes and 11 edges.

3. Experiments

In this experiment, you are expected to develop a flight search engine (such as skyscanner (www.skayscanner.com/) or e-dream (www.edreams.com/)) that enables users to make flight plans. The flight database will be provided via two input files. Your program should read these input files and create a directed graph structure to store the information. Finally, your program should create output files according to information given two input files. In this section, details will be given about the input files to be used and the output file to be created.

4. Input Files

In this section, details will be given about the input files to be used in this experiment. All the information about the flight is stored in three input files, which are described below: *airportList.txt*, *flightList.txt* and *commandList.txt*.

4.1 Airport List

The first input file contains a list of cities along with aliases of the airports located at those cities. Each line of this file contains the name of a city and the airport aliases which are separated by tab character. A city can have more than one aliases. The city names and airport aliases will be unique among the system.

Structure of airportList.txt

[city_name] *tab* **[airport_alias]** *newline*
[city_name] *tab* **[airport_alias]** *tab* **[airport_alias]** *newline*
…….

Example of airportList.txt

```
Vienna      VIE
Brussels    BRU     CRL
Sofia       SOF
Prague      PRG
Copenhagen          CPH
Helsinki    HEL
```

4.2 Flight List

The second input file contains a list of all the available flights to be included in the database. Each line represents an individual flight containing the flight id, departure and arrival airport aliases, the departure date and time (as hours and minutes), flight duration (again, as hours and minutes) and the price respectively.

The flight ids are composed of two uppercase characters that represent one of the airlines companies that are:

- Turkish Airlines (TK)
- Royal Dutch Airlines (KL)
- Lufthansa Airlines (LH)
- Delta Airlines (DL)
- American Airlines (AA)

and a four-digit number to ensure that the flight ids are unique among the system. You are expected to create a graph according to airportList.txt and flightList.txt files as example one shown in Figure 2 (example graph).

Structure of flightList.txt

[flight_id] *tab* **[dept]** -> **[arr]** *tab* **[dept_date]** *tab* **[duration]** *tab* **[price]** *newline*
[flight_id] *tab* **[dept]** -> **[arr]** *tab* **[dept_date]** *tab* **[duration]** *tab* **[price]** *newline*
…….

Example of flightList.txt

KL6805	FCO->TXL	30/04/2020	21:40	Thu	02:25	75
LH6171	SOF->FCO	01/05/2020	19:00	Fri	01:55	135
TK2133	IST->FCO	01/05/2020	16:20	Thu	02:25	125
LH5680	IST->SOF	30/04/2020	17:50	Thu	01:20	110
KL3142	FCO->TXL	01/05/2020	21:40	Fri	02:25	75
KL0135	FCO->TXL	30/04/2020	12:45	Thu	02:15	75

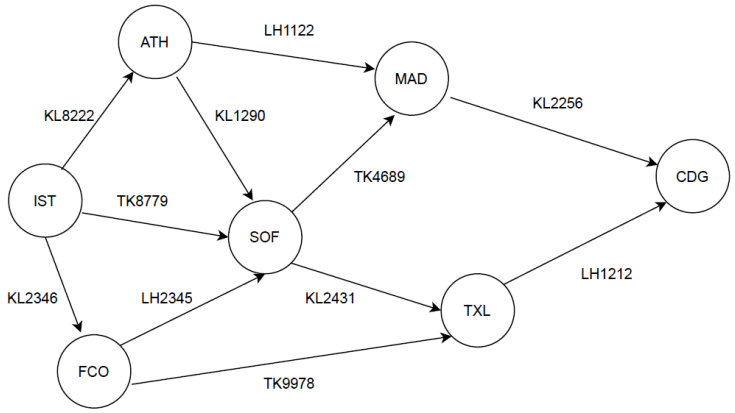

Figure 2. Example Graph of flights between cities

4.3 Command List

After reading the two input files mentioned above, your program should read from a third input file that contains the commands which are essentially flight search queries. A flight search query yields a flight plan which is either a single flight or a series of connected flights from the departure point to the arrival point. There are two rules for two flights to be eligible to be connected in a flight plan:

- Arrival point of the first flight and departure point of the second flight should be the same airport.
- Departure time of the second flight should be later than arrival time of the first flight.

There are also two more points that should be taken into consideration while forming a flight plan that are:

- Transferring between two airports at the same city is not allowed.
- A flight plan may not pass through the same city more than once. This rule applies to all the airports at a city.

Below is a list of 10 different commands. Each command includes a departure point and an arrival point that are cities. The start date parameter of the commands indicates the earliest date (starting from midnight, 00:00) the flights may start and it is possible for a flight plan to start any time after the start date.

4.3.1 List All

List all command is used for listing all the possible flight plan(s) from departure point to the arrival point.

Structure of command

[listAll] *tab* [***dept***]-> [***arr***] *tab* [***start_date***]

4.3.2 List Proper

List proper command is used for listing all the proper flight plan(s) from departure point to the arrival point which means a flight plan should be removed from the resulting flight plan set if there is another flight plan which is both quicker (that arrives at the final destination earlier) and cheaper.

Structure of command

[listProper] *tab* [***dept***]-> [***arr***] *tab* [***start_date***]

4.3.3 List Cheapest

List cheapest command is used for listing the cheapest possible flight plan(s) from departure point to the arrival point. Keep in mind that there may be more than one possible cheapest flight plans.

Structure of command

[listCheapest] *tab* [***dept***]-> [***arr***] *tab* [***start_date***]

4.3.4 List Quickest

List quickest command is used for listing the quickest possible flight plan(s) from departure point to the arrival point. Keep in mind that there may be more than one possible quickest flight plans.

Structure of command

[listQuickest] *tab* [***dept***]-> [***arr***] *tab* [***start_date***]

4.3.5 List Cheaper

List cheaper command is used for listing all the ***proper*** flight plans from departure point to the arrival point that are cheaper than given parameter.

Structure of command

[listCheaper] *tab* [***dept***]-> [***arr***] *tab* [***start_date***] *tab* **[max_price]**

4.3.6 List Quicker

List quicker command is used for listing all the ***proper*** flight plans from departure point to the arrival point that arrive to the destination earlier than given parameter.

Structure of command

[listQuicker] *tab* [***dept***]-> [***arr***] *tab* [***start_date***] *tab* **[*latest_date_time*]**

4.3.7 List Excluding

List excluding command is used for listing all the ***proper*** flight plans from departure point to the arrival point that do not involve a flight from given airlines company.

Structure of command

[listExcluding] *tab* [***dept***]-> [***arr***] *tab* [***start_date***] *tab* **[*company*]**

4.3.8 List Only From

List excluding command is used for listing all the ***proper*** flight plans from departure point to the arrival point that consists of flights only from the given airlines company.

Structure of command

[listOnlyFrom] *tab* [***dept***]-> [***arr***] *tab* [***start_date***] *tab* **[*company*]**

4.3.9 Diameter of Graph

The diameter of graph is the maximum distance between the pair of vertices. It can also be defined as the **longest shortest path**. An example calculation of diameter of graph is given in Figure 3. When calculating the diameter of graph, consider the graph that you created by using input files and flight **price** will be cost of edges.

Figure 3. Example of diameter

Diameter (δ)=5+6+8+3+2+5+8+3=40

Structure of command

[diameterOfGraph]

4.3.10 PageRank of Nodes

PageRank (PR) is an algorithm used by Google Search to rank web pages in their search engine results. PageRank is a way of measuring the importance of website pages. The basic idea of PageRank is that the importance of a web page depends on the pages that link to it. For instance, we create a web page *i* that includes a hyperlink to web page *j*. If there are a lot pages also link to *j*, we then consider *j* is important on the web.

When calculating the pageRank of nodes, consider each nodes of graph as web pages and calculate rank for each nodes according to incoming and outgoing edges. You can see formulation of pageRank algorithm in Equation 1. Here, PR(A)=page rank of page A (kind of recursion formula because it depends on other pages' page rank), $PR(T_i)$=page rank of pages T_i which link to page A, $C(T_i)$ =number of out bounds links on a given T_i page d=jump factor in the range 0 and 1 (usually set d to 0.85). You are free to develop your algorithm.

$$PR(A) = (1-d) + d\,(PR(T1)/C(T1) + \ldots + PR(Tn)/C(Tn))$$

Equation 1. PageRank Formulation

Here you can see example simulator:
(https://computerscience.chemeketa.edu/cs160Reader/_static/pageRankApp/index.html)

Structure of command

[pageRankOfNodes]

Example input file for all commandList.txt

```
listAll      Istanbul->Madrid   30/04/2020
listProper   Istanbul->Madrid   30/04/2020
listCheapest Istanbul->Madrid   30/04/2020
listQuickest Istanbul->Madrid   30/04/2020
listCheaper  Istanbul->Madrid   30/04/2020   340
listQuicker  Istanbul->Madrid   30/04/2020   01/05/2020 09:00 Fri
listExcluding Istanbul->Madrid  30/04/2020   LH
listOnlyFrom Istanbul->Madrid   30/04/2020   TK
diameterOfGraph
pageRankOfNodes
```

5. Output File

Results of each command is a list of flight plans that should be printed to an output file which name will be output.txt. The output format of first eight command are similar but for last two (diameterOfGraph and pageRankOfNodes) format are different. Simply, each command should be printed to a line of the output file in the following format:

Structure of output.txt

[command] *:* **[first command in command.txt]** *newline*
[***flight_id]*** *tab* [***dept***]-> [***arr***]||[***flight_id]*** tab [***dept***]-> [***arr***]||…… *tab* [**duration**]/[**total_price**] *newline*
newline
newline
[command] *:* **[second command in command.txt]** *newline*
[**flight_id]** *tab* **[dept]**-> [**arr**]||**[flight_id]** tab [***dept***]-> [***arr***]||…… *tab* [**duration**]/[**total_price**] *newline*
newline
newline
[command] : [third command in command.txt] *newline* // for diameter of graph
[The diameter of graph] : total_price
newline
newline
[command] : [fourth command in command.txt] *newline* // for pageRank of nodes
[node name] = rank value *newline*
[node name] = rank value *newline*
…..

Example of output.txt

```
command : listAll        Istanbul->Madrid        30/04/2020
LH7366    IST->SOF||TK8994     SOF->CDG||TK9156    CDG->MAD          17:45/350
TK8739    IST->FCO||TK4074     FCO->CDG||TK9156    CDG->MAD          13:00/310
KL8222    IST->ATH||LH4696     ATH->FCO||TK4074    FCO->CDG||TK9156  CDG->MAD        13:15/415
KL8222    IST->ATH||LH0893     ATH->TXL||LH9137    TXL->MAD          11:35/480

command : listProper     Istanbul->Madrid        30/04/2020
TK8739    IST->FCO||TK4074     FCO->CDG||TK9156    CDG->MAD          13:00/310
KL8222    IST->ATH||LH0893     ATH->TXL||LH9137    TXL->MAD          11:35/480

command : listCheapest   Istanbul->Madrid        30/04/2020
TK8739    IST->FCO||TK4074     FCO->CDG||TK9156    CDG->MAD          13:00/310

command : listQuickest   Istanbul->Madrid        30/04/2020
Not implemented

command : listOnlyFromIstanbul->Madrid        30/04/2020  AA
No suitable flight plan is found

command : diameterOfGraph
The diameter of graph : 350

command : pageRankOfNodes
IST = 0,226
ATH = 0,321
CDG = 0,123
```

If the search query cannot find any flight plan to list, the output should be "***No suitable flight plan is found***". If you have not implemented any of the commands, print ***"Not implemented"*** as output of that command.

Execution and Test

The input files are going to be given as program arguments. The argument order will be as follow: *airportList.txt, flightList.txt* and *commandList.txt.*

- The file that contains the main method should be named "Main.java".
- Upload your java files to your server account (dev.cs.hacettepe.edu.tr)
- Compile your code (javac Main.java or javac *.java)
- Run your program (java Main *airportList.txt flightList.txt commandList.txt*)
- Control your output data and format.

The Late Submission Policy

You may use up to three extension days for the assignment. For each extension day, you will lose 10 points.

Note and Restrictions

- Your work will be graded over a maximum of 100 points according to the grading policy.
- Do not miss the submission deadline.
- Save all your work until the assignment is graded.
- Name of all files are fixed (*airportList.txt, flightList.txt, commandList.txt* and *output.txt*).
- Your programs will be executed in **DEV** machine, please make it work on dev before submitting (if your code not run on DEV server, it will have graded as zero).
- You have to write your own code for each algorithm you need to use in your assignment. That is, **libraries are not allowed for algorithms** (e.g. JGraphT), but you can use libraries for other implementations such as data structure.
- Regardless of the length, use UNDERSTANDABLE names to your variables, classes and Functions. The names of classes, attributes and methods should obey Java naming convention. This expectation will be graded as Coding standards.
- Source code readability is a great of importance for us. Thus, write READABLE SOURCE CODE, COMMENTS and clear MAIN function. This expectation will be graded as clean code.
- All assignments must be original, individual work. Duplicate or very similar assignments are both going to be considered as cheating.
- Your output for the **pageRank** command may differ from the expected. Therefore, the algorithm (source code) you developed will be checked while grading to this command (if your code produce some output for this). Do not forget to add comments, especially when developing this algorithm. If you don't add comments, you will get 0 from this command.
- You will submit your as following file hierarchy:

<student id.zip
→ src <DIR>
→ Main.java(Required)
→ *.java(optional)

Grading

Task	**Point**
Submitted	1
Coding standards	5
Clean code	10
Outputs (listAll=19, listProper= 12, listCheapest=7, listQuickest=7, listCheaper=7, listQuicker=6, listExcluding=6, listOnlyFrom=6, diameterOfGraph=7, pageRankOfNodes= 7)	84
Total	**100**