CS 251 OutLab7

Java Basics

Instructions

- The submissions will be **autograded**. Please read the problem statement carefully.
- **Your implementation will be run against multiple test cases and marks will be provided based on the fraction of test cases passed.**
  - Zero marks if code does not compile
  - Zero marks if files have improper names
- **Commands to run Main.java: (Ensure that you are in directory just outside Q$).**

  javac Q$/*.java && java Main
- Submission directory must be named - <roll_no>_outlab7

<roll_no>_outlab7

- The unzipped submission directory must look as follows:

```
.
├── Q1
│   ├── Graph.java
│   └── Node.java
└── Q2
    ├── Board.java
    └── Client.java

2 directories, 4 files
```

Q1 Dijkstra's Algorithm (40 point)

Mr. Gokhale, the not-so-generous salesman, wants to save every last penny he holds. He covers one city a day and reports it back to Dr. Ashwin at the end of the day. Exhausted by sleeping on station benches, he requires you to calculate the distance of optimal paths from Braunschweig to every other city.

Dijkstra's Algorithm is used for solving the **Single Source Shortest Path** problem in a **positive** weighted directed graph. Learn about the algorithm and its implementation using the following links.

return it as a Map of node names along with the length of shortest paths using Dijkstra's Algorithm. Also, you need to implement other constructors and member functions in Graph and Node classes.

You can test your implementation by running it against the sample case provided in input.txt by using the command after compilation.

java Q1/main < input.txt

NOTE:

1. **To get full credits, you must provide an implementation using Priority Queue or Ordered Set. O((m+n)log(n)) is the required time complexity where m is # of edges and n is # of nodes.**
2. **You need to make changes in the files where it is mentioned.**
3. **You can assume the longest path to be less than 10^9.**
4. **For nodes unreachable from the source, put 10^9 as the distance.**
5. **When adding edges, check if nodes with the given names exist.**
6. **There won't be multiple edges between the same pair of nodes.**

You are provided with three files - **Node.java**, **Graph.java**, **Main.java**.

Node.java

This file defines the Node class which has two attributes:

CS 251 Outlab7

Graph.java

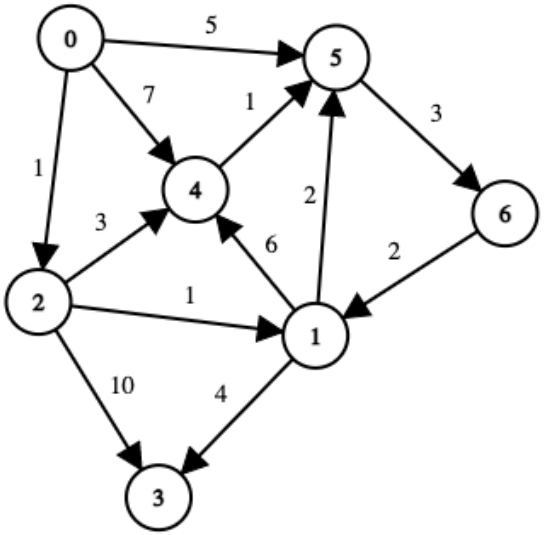
This file defines the Graph class which has a **Map nodeMap** mapping node name to the corresponding node instance. Implement the member functions **addNode**, **addDirectedEdge** and **dijkstraAlgorithm**.

Main.java

This file is provided so that you can test your implementation. You need not make any changes to this file. Don't submit it.

The public main method simply takes the graph as input and prints out all the nodes with their distances from source.

Sample Case



```
1 5 2                6 7
2 3 10
2 4 3
4 5 1
2 1 1
6 1 2
5 6 3
1 3 4
0
```

Q2 Tic-Tac-Toe (60 point)

No stories this time. Post a relevant one on Piazza. We will include it here. Bonus marks for best stories. :P

In this problem we will implement the famous tic tac toe game. You would be learning some basics of inter-process communication, socket programming along with some exception handling.
Our model would be peer to peer. Here the peers (the two players here) act as both server as well as client.

The peer-to-peer model

- Each peer has a ServerSocket and two socket variables.

There are two files **Client.java** and **Board.java**. You are supposed to complete the function as stated in the docstrings of the boilerplate code. Recommended to start with **Board.java** first.

Board.java

The board is a **3x3** array with indices and values both as 0,1,2. Also note that the 2 clients have different board variables so after sharing moves both become identical. We could have used a shared variable but that would have made the implementation complicated.
We have provided a **printBoard** function to print the board.
You have to implement the following functions:

- **public Boolean available:** returns whether a position is available to be filled or not.
- **public void updateBoard:** Make updates in the board after receiving a move.
- **public int checkBoard:** calls the **printBoard** function and returns different integers depending on the status of the game.

Client.java

Here we implement running of the game as well as inter process communication.
Follow the comments to complete the code

Evaluation:

For evaluation we would compare the output of your code using **diff** command with our original implementation. Some sample test cases with input and output are provided.

5000,8000 specifies the ports, use any other if the port is unavailable.

Command to run the game from testcase:

After compiling as above run both the player simultaneously using

java Q2.Client 1 5000 8000 < in1.txt > out1.txt & java Q2.Client 2 8000 5000 < in2.txt > out2.txt

Use **diff** to compare your output with given **out*.txt**. **(Should match exactly)**

Output:

There will be 2 output files. Display the board status after each move (the print function and where it needs to be called is already present in helper code). The output file should contain the final results of the game for the corresponding player. (refer to the output files provided in the resources)

We will release more test cases soon.

Points to Note:

- Don't worry about cases like a player closing the game without completing.
- The input files contain the sequence of moves for each player (imagine having a fixed set of moves for both player)
- It does not matter which player you start first but the **ID** of both should not be the same and there won't be any test cases with both players having the same ID.

CS 251 Outlab7