

# Assignment #1

---

## Distributed Systems

**Presented by**

- 1- Ahmed Nasser abdelkareem (9)**
- 2- Zeyad Ahmed Elbanna (26)**
- 3- Abdelrahman Ahmed Mohamed Omran (36)**

**04/05/2020**

[This assignment aims at understanding the basics of RMI/RPC implementation and applying it in the shortest path problem where we need to find a path between two vertices (or nodes) in a graph such that the sum of the weights of its constituent edges is minimized.]

# Table of Contents

---

<b><i>1 Problem Definition</i></b>	<b><i>3</i></b>
<b><i>2 Algorithms</i></b>	<b><i>4</i></b>
<b><i>4 Implementation</i></b>	<b><i>5</i></b>
<b><i>5 Results</i></b>	<b><i>6</i></b>
<b><i>6 Conclusion</i></b>	<b><i>7</i></b>

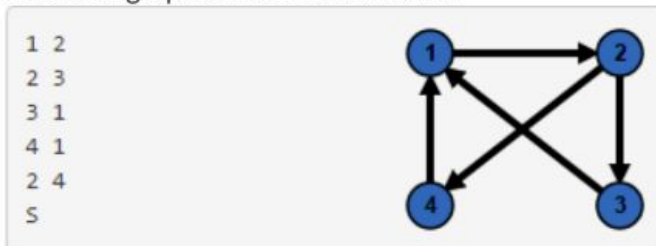
# 1 Problem Definition

We need to use RMI structure (server and client and remote object) to help clients interacting with a graph on the server and answer some questions from the client like:

- 1) Adding Edge to the graph.
- 2) Deleting Edge from the graph.
- 3) Finding the shortest path which is a path between two vertices (or nodes) in a graph such that the sum of the weights of its constituent edges is minimized.

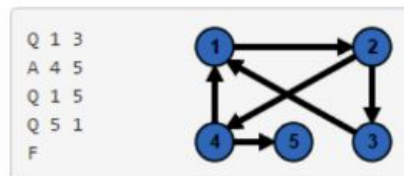
## Examples

The initial graph is entered as follows



The following batches are then added to the graph

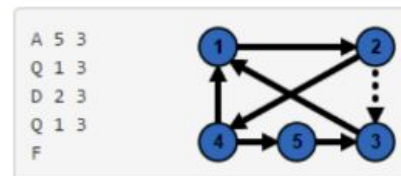
Batch 1:



Output:

```
2
3
-1
```

Batch 2:



Output:

```
2
4
```

## 2 Algorithms

---

In this section you are stating the methods by which you have solved the problem. Revisit the requirements you extracted in section 1 and state what algorithms or method used to fulfill those requirements. Feel free to express your solution in the most elaborate way. (Remember a picture is worth a thousand words).

### Shortest path algorithm:

- Dijkstra's Algorithm

**function** Dijkstra(*Graph*, *source*) :

```
3      create vertex set Q
5      for each vertex v in Graph:
6          dist[v] ← INFINITY
7          prev[v] ← UNDEFINED
8          add v to Q
10     dist[source] ← 0
12     while Q is not empty:
13         u ← vertex in Q with min dist[u]
14
15         remove u from Q
16
17         for each neighbor v of u:           // only v that are still in Q
18             alt ← dist[u] + length(u, v)
19             if alt < dist[v]:
20                 dist[v] ← alt
21                 prev[v] ← u
23     return dist[], prev[]
```

Reasons we used Dijkstra:

- Time Complexity of Dijkstra's Algorithm:  $O(E \log V)$  which is better than most of other techniques
- We don't have negative edges here, so Dijkstra would be perfect
- <https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-in-java-using-priorityqueue/>

# 4 Implementation

---

We are running the client and server on the local machine with this configuration:

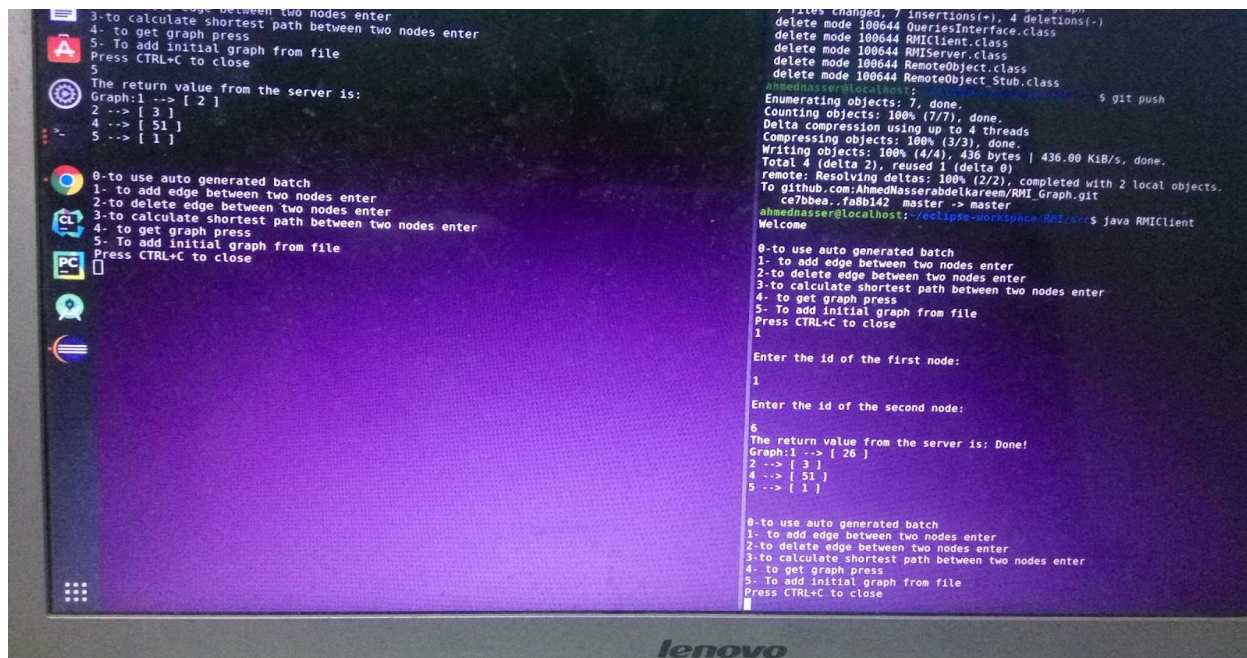
- HOST = localhost
- PORT = 1099

## **Batch Generation :**

- The client will have the option to use the random batch generation
- He will be asked to enter the writing percentage in the batch.
- Then he will be asked to enter the batch size.
- The batch will be generated as required
  - We use `java.util.Random` to generate random numbers -uniformly- from 0 to 100 and if it's less than the percentage entered by the user we add or delete edges with equal probability else we add a Query .
  - the parameters of the add, remove or query are random nodes.
- On the server side the batch is executed line by line on the graph and on finishing it returns the responses of the queries and then the client will have the option to request another batch.

## 5 Results

- Running 2 clients



```
3-to calculate shortest path between two nodes enter
4- to get graph press
5- To add initial graph from file
Press CTRL+C to close
5
The return value from the server is:
Graph:1 --> [ 2 ]
2 --> [ 3 ]
4 --> [ 51 ]
5 --> [ 1 ]

0-to use auto generated batch
1- to add edge between two nodes enter
2-to delete edge between two nodes enter
3-to calculate shortest path between two nodes enter
4- to get graph press
5- To add initial graph from file
Press CTRL+C to close
0

7 files changed, 7 insertions(+), 4 deletions(-)
delete mode 100644 QueriesInterface.class
delete mode 100644 RMIClient.class
delete mode 100644 RMIServer.class
delete mode 100644 RemoteObject.class
delete mode 100644 RemoteObject Stub.class
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 436 bytes | 436.00 KiB/s, done.
Total 4 (delta 2), reused 1 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To github.com:AhmedNasserAbdelkareem/RMI_graph.git
ce7bbea..fa8b142 master -> master
ahmednasser@localhost: /eclipse-workspace/RMI$ java RMIClient
Welcome

0-to use auto generated batch
1- to add edge between two nodes enter
2-to delete edge between two nodes enter
3-to calculate shortest path between two nodes enter
4- to get graph press
5- To add initial graph from file
Press CTRL+C to close
1

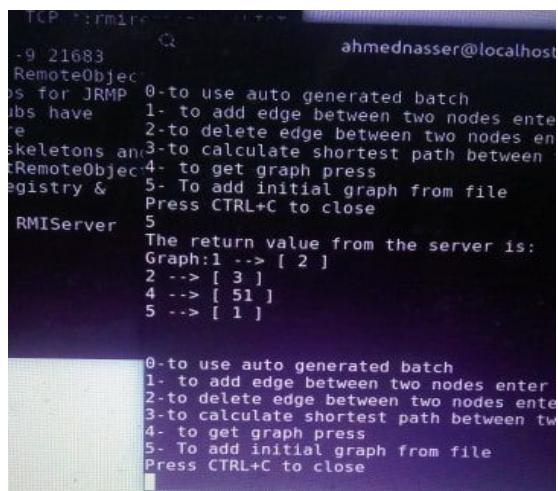
Enter the id of the first node:
1

Enter the id of the second node:
6

The return value from the server is: Done!
Graph:1 --> [ 26 ]
2 --> [ 3 ]
4 --> [ 51 ]
5 --> [ 1 ]

0-to use auto generated batch
1- to add edge between two nodes enter
2-to delete edge between two nodes enter
3-to calculate shortest path between two nodes enter
4- to get graph press
5- To add initial graph from file
Press CTRL+C to close
```

- sample run



```
ahmednasser@localhost: ~$ java RMIClient
-9 21683
RemoteObject
os for JRMP 0-to use auto generated batch
bs have 1- to add edge between two nodes ente
re 2-to delete edge between two nodes ente
skeletons and 3-to calculate shortest path between
RemoteObject 4- to get graph press
registry & 5- To add initial graph from file
Press CTRL+C to close
RMIServer 5
The return value from the server is:
Graph:1 --> [ 2 ]
2 --> [ 3 ]
4 --> [ 51 ]
5 --> [ 1 ]

0-to use auto generated batch
1- to add edge between two nodes enter
2-to delete edge between two nodes ente
3-to calculate shortest path between tw
4- to get graph press
5- To add initial graph from file
Press CTRL+C to close
```

## 6 Conclusion

---

- What you have concluded from working on this lab assignment.
  - We concluded a lot of things concerning distributed systems, and RMI implementation as we used it to control systems in a distributed network
  - That the RMI consisted of three layers which are:
    - Stub
    - Remote reference layer
    - And a transport connection layer
  - Dijkstra works very good in case of unweighted graph by the use of a queue
- What would you have done differently and why.
  - We gave the client the option to use batch generation or to enter a request directly.
  - We would develop the system more to reduce time response