FACULTY OF ENGINEERING

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

1 Problem Definition

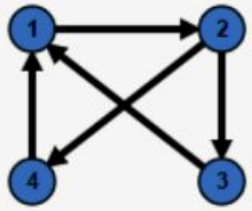
We need to use RMI structure (server and client and remote object) to help clients interacting with a graph on the server and answer some questions from the client like:

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

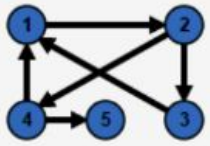
```
1 2
2 3
3 1
4 1
2 4
S
```



The following batches are then added to the graph

Batch 1:

```
Q 1 3
A 4 5
Q 1 5
Q 5 1
F
```

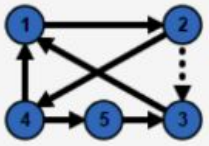


Output:

```
2
3
-1
```

Batch 2:

```
A 5 3
Q 1 3
D 2 3
Q 1 3
F
```



Output:

```
2
4
```

2 Algorithms

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/

- BellMan - Ford Algorithm

```
function bellmanFord(G, S)
    for each vertex V in G
        distance[V] <- infinite
        previous[V] <- NULL
    distance[S] <- 0
    for each vertex V in G
        for each edge (U,V) in G
            tempDistance <- distance[U] + edge_weight(U, V)
            if tempDistance < distance[V]
                distance[V] <- tempDistance
                previous[V] <- U

    for each edge (U,V) in G
        If distance[U] + edge_weight(U, V) < distance[V}
            Error: Negative Cycle Exists

    return distance[], previous[]
```

- BFS Algorithm

```
BFS (G, s)              //Where G is the graph and s is the source node
    let Q be queue.
    Q.enqueue( s ) //Inserting s in queue until all its neighbour vertices are marked.


    mark s as visited.
    while ( Q is not empty)
        //Removing that vertex from queue,whose neighbour will be visited now
        v  =  Q.dequeue( )


       //processing all the neighbours of v
       for all neighbours w of v in Graph G
           if w is not visited
                    Q.enqueue( w )          //Stores w in Q to further visit its neighbour
                    mark w as visited.
```

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.

Graph Generator :

- We used graph generator to construct test case with different number of nodes from size =2 to size = 15
- To use these test cases and track the response time of each graph.

5 Results

We use initial graph

1	2
2	3
4	5
5	1
4	1

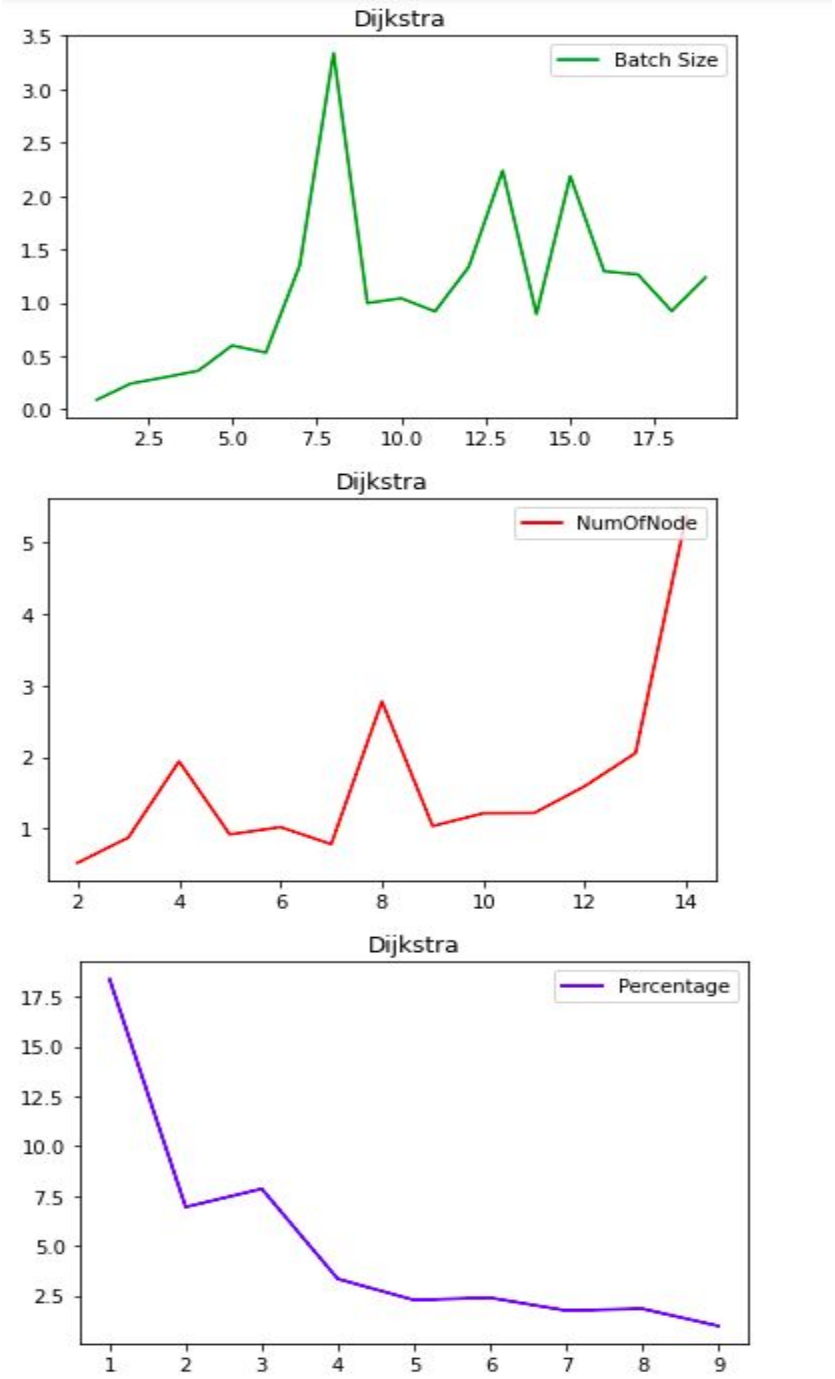
For performance analysis:

We used 3 algorithms for shortest path to be our variants

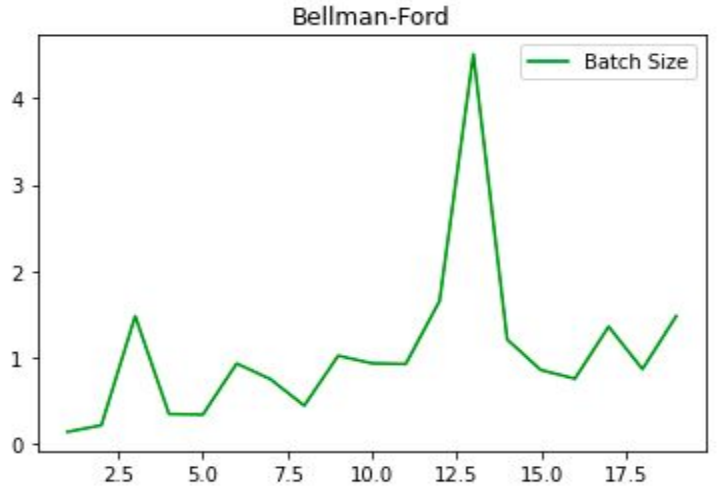
for each algorithm we tried to change some parameter and monitor the response time:

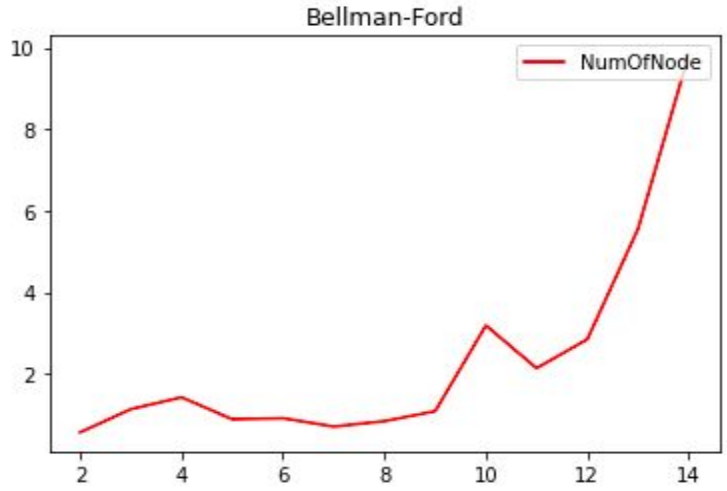
- Batch Size: with Number of nodes = 5 and Percent = 0.3
- Number of Nodes: With Batch size = 10 and Percent = 0.3
- Percent of Writes: with Number of nodes = 5 and Batch size = 20
-

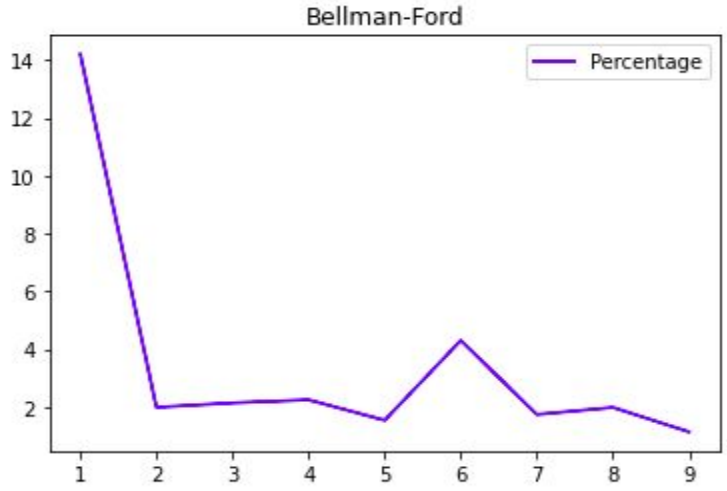
1) Using Dijkstra Algorithm



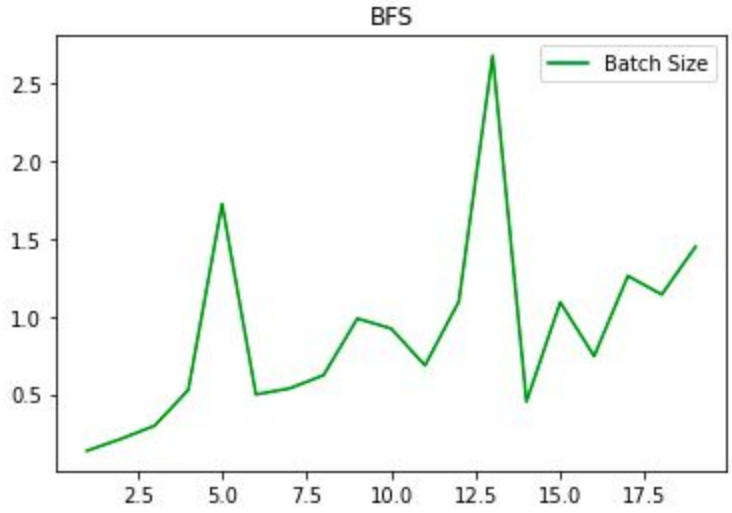
2) Using BellMan-Ford

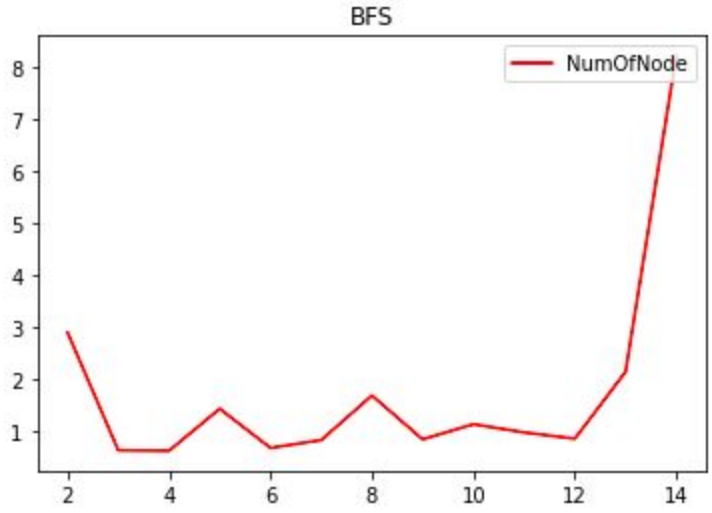


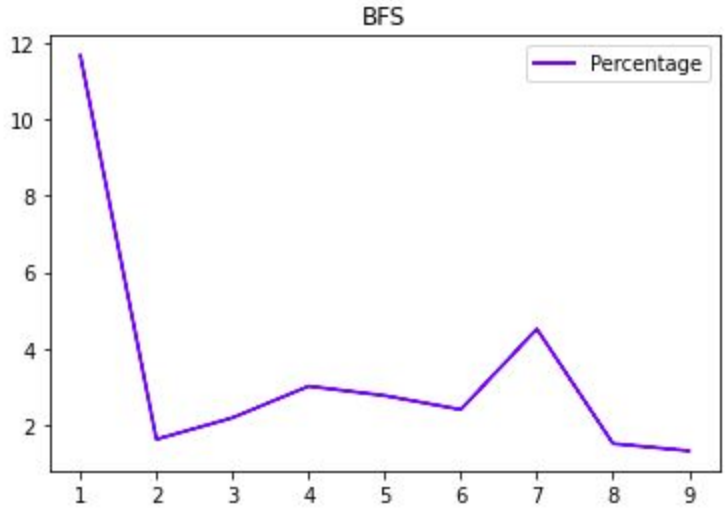




3) Using BFS


BFS
Batch Size
2.5
2.0
1.5
1.0
0.5
2.5
5.0
7.5
10.0
12.5
15.0
17.5


BFS
NumOfNode
8
7
6
5
4
3
2
1
2
4
6
8
10
12
14


BFS
Percentage
12
10
8
6
4
2
1
2
3
4
5
6
7
8
9

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