Assignment 3

Indranil Pradhan
Roll no.- 2019202008

Problem – Using RMI (Remote Method Invocation) in Java implements a sing server architecture of finding the weight of minimum spanning tree with support for multiple clients.

Input

“add_graph g1 n” – Graph identifier, Number of nodes in the graph.

“add_edge g1 u v w” - Graph Identifier, starting node, ending node, edge weight.

“get_mst g1” - Graph Identifier.

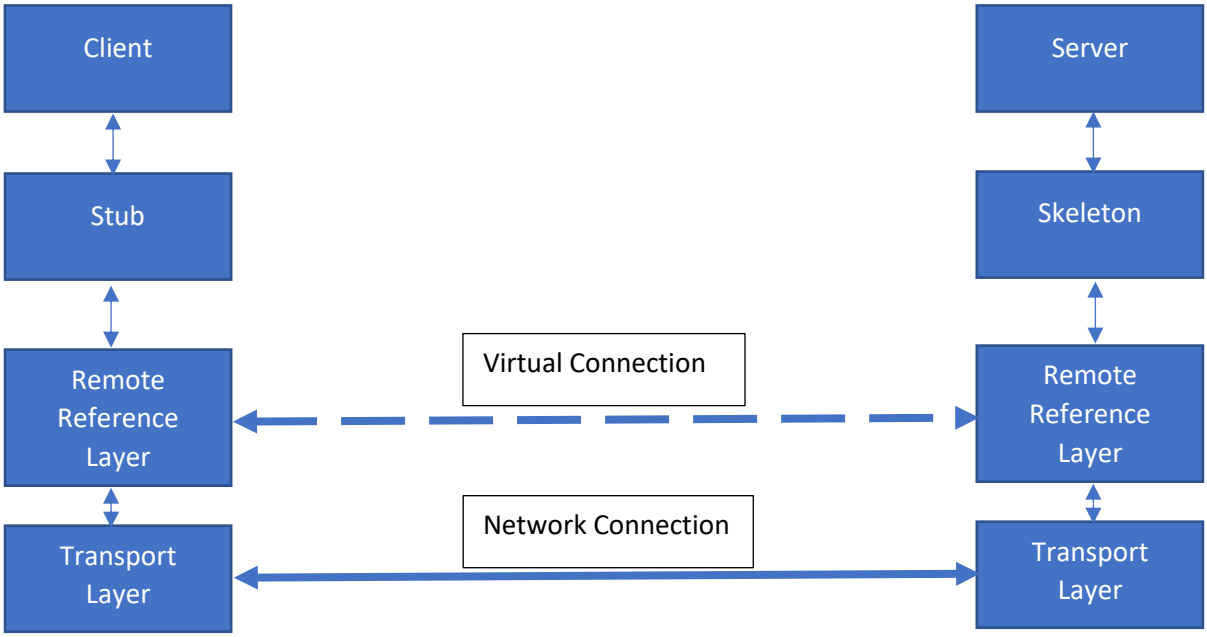
Output-

“add_graph g1 n” – Graph, g1 is added successfully.

“add_edge u v w” – Edge between node u and v with weight w is added successfully in the graph, g1.

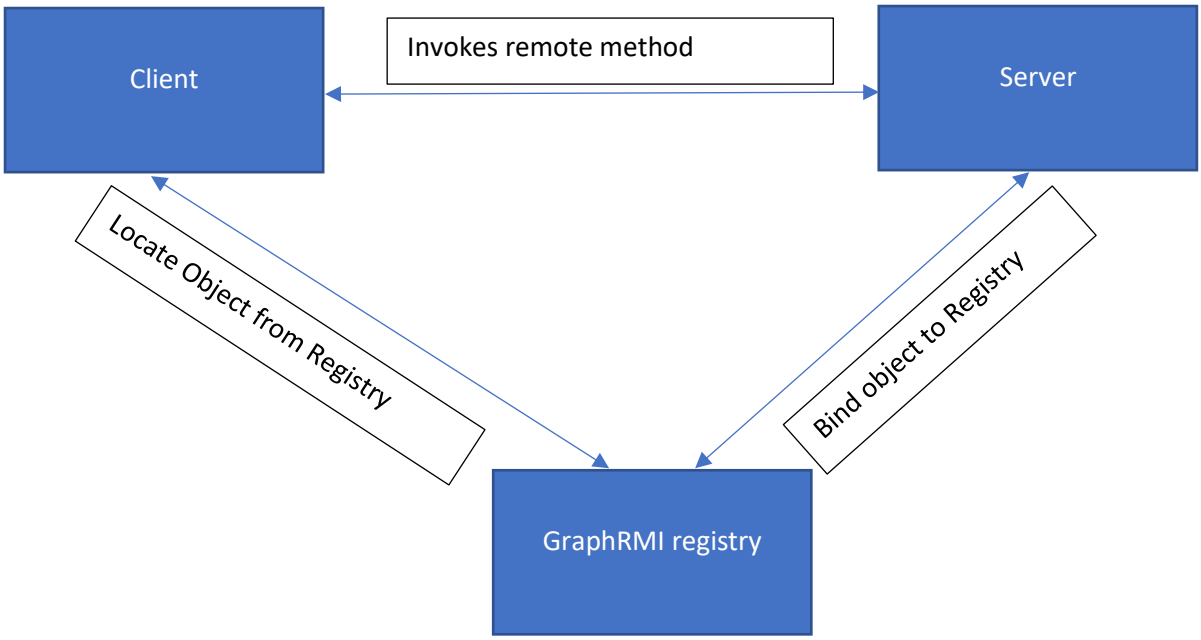
“get_mst g1” – Get weight of minimum spanning tree of the graph g1. If minimum spanning tree doesn’t exist, then return -1.

Architecture



Inside the server remote object of GraphRMI is created. This object is rebind to the name “GraphRMI” and made available to the client. The server is also bind to the port number provided in the aurgument sothat the server is available to thar particular port.

Client register server id address and server port from argument to get the remote registry. It looks for “GraphRMI” registry and creates object of the remote registry on successful lookup. Using the remote object it accesses the method of remote server.



Within the RMI registry all the server objects resides in. The server each time runs by registering the object to the registry using rebind. When Client needs to access the object it uses look up to find the object from the registry.

Algorithm

"add_graph g1 n" –

A global map is used through out the program which keep record of graph created against each identifier. Identifier is unique for every graph. This method creates graph with n number of nodes and make an entry for the graph identifier with newly created graph. The key is graph identifier and value is the starting memory of the graph created. If the graph is already created, then don't allow the client to create new graph with same identifier.

"add_edge g1 u v w" –

1. Fetch the starting memory from the map from the map against the identifier.
2. Creates edge between the nodes u and v with weight w in the graph with identifier g1.

"get_mst g1" –

1. Fetch the starting memory from the map from the map against the identifier.
2. Check if the graph is connected. If graph isn't connected return mst as -1.
3. A set is used to keep track of all the vertices included in the graph.
4. Take the first vertex as the source and assign value 0 to it. And the other vertices are initialized with key value of infinity.
5. Step 6,7,8 loop through until the set doesn't include all the vertices.
6. Pick a vertex u which is included in the set till now.
7. Include u to the set.
8. Update the key value of all the adjacent vertices of u. If the adjacent vertex v has larger weight then the weight edge of u to v then update the key value of v.

Command to run

1. javac *.java
2. java Server 12345
3. java Client 127.0.0.1 12345 < input.txt

Sample Test Cases

Input

```
add_graph G4 4
get_mst G4
add_edge G4 1 2 10
add_edge G4 1 1 11
add_edge G4 3 4 4
get_mst G4
add_edge G4 1 4 5
get_mst G4
add_edge G4 1 3 3
get_mst G4
add_edge G4 2 4 2
get_mst G4
```

Output

```
Graph has been created successfully.
The weight of the MST is -1.
Edge added successfully.
Edge added successfully.
Edge added successfully.
The weight of the MST is -1.
Edge added successfully.
The weight of the MST is 19.
Edge added successfully.
The weight of the MST is 17.
Edge added successfully.
The weight of the MST is 9.
```

Input

```
add_graph g1 5
add_edge g1 1 2 2
add_edge g1 1 4 6
add_edge g1 2 3 3
add_edge g1 2 4 8
add_edge g1 2 5 5
add_edge g1 3 5 7
add_edge g1 4 5 9
get_mst g1
```

Output

Graph has been created successfully.
Edge added successfully.
Edge added successfully.
Edge added successfully.
Edge added successfully.
Edge added successfully.
Edge added successfully.
Edge added successfully.
The weight of the MST is 16.