Assignment 1
Incremental calculation of shortest path in dynamic graphs

Teams Members:

Ashraqat Ahmed Sheta(13)
Marina Zakaria(47)
Fatma Ibrahim Hemeda(42)

Problem Description:-

In graph theory, the shortest path problem is the problem of finding a path between two vertices (or nodes) in a graph such that the sum of the weights of its constituent edges is minimized. This is a fundamental and well-studied combinatorial optimization problem with many practical uses: from GPS navigation to routing schemes in computer networks; search engines apply solutions to this problem on website interconnectivity graphs and social networks apply them on graphs of peoples' relationships.

Functions:-

Query:

Intputs: source node and destination node
Outputs: shortest path between source and destination node
Description: this function implement Breadth First Search(BFS) algorithm to get the shortest path between source and destination
Note: we Choose BFS Because it will get the optimal solution in a few times compared to others algorithms.

```
private String query(int src, int dest) {
    // TODO Auto-generated method stub
    int numberOfNodes = -1;
    boolean visited[] = new boolean[graph.size()];
    ArrayList<Node> queue = new ArrayList<Node>();
    visited[src] = true;
    queue.add(new Node(src, 0));
    while (queue.size() != 0) {

        Node node = queue.remove(0);
        if(node.getnodeNum()>=graph.size()) {
            continue;
        }
        ArrayList<Integer> adjacent = graph.get(node.getnodeNum());
        for (int i = 0; i < adjacent.size(); i++) {
            int adjNode = adjacent.get(i);
            if (adjNode == dest) {
                numberOfNodes = node.getcost() + 1;
                return Integer.toString(numberOfNodes);
            }

            if (!visited[adjNode]) {
                visited[adjNode] = true;
                queue.add(new Node(adjNode, node.getcost() + 1));
            }
        }

    }

    return Integer.toString(numberOfNodes);
}
```

Add Edge

Intputs: source node and destination node
Outputs: No outputs
Description: add edge to current graph and handle if nodes already exist.

```
private void addEdge(int src, int dest) {
    // TODO Auto-generated method stub
    if(src<graph.size()) {
        if(!graph.get(src).contains(dest)) {
            graph.get(src).add(dest);
        }
    }else {
        graph.add(new ArrayList<Integer>());
        graph.get(src).add(dest);
    }
    if(dest==graph.size()) {
        graph.add(new ArrayList<Integer>());
    }

}
```

Delete Edge

Intputs: source node and destination node
Outputs: No outputs
Description: delete edge to current graph.

```
private void deleteEdge(int src, int dest) {
    // TODO Auto-generated method stub
    if(src<graph.size()) {
        for(int i=0;i<graph.get(src).size();i++) {
            if(graph.get(src).get(i)==dest) {
                graph.get(src).remove(i);
                return;
            }
        }
    }

}
```

RMI Implementation:-

The remote interface

The remote interface has two functions.
The first is to check the server is ready and return ‘R’ that indicates the server is ready .

The second is to execute batch queries and implement it's queries, it takes the array list of queries and return array list of output line from query (Q).

```
public interface IRemoteMethod extends Remote {
    public String serverReady()throws RemoteException;
    public ArrayList<String> executeBatch(ArrayList<String> batch)throws RemoteException;

}
```

The Server Program

The server program extends unicast remote object and implement the remote interface (IRemoteMethod)

```
public class Server extends UnicastRemoteObject implements IRemoteMethod{

    ArrayList<ArrayList<Integer>> graph;
    Utiles u=new Utiles();
    private ReadWriteLock rwlock;
    ArrayList<ArrayList<Object>> performance;

    protected Server() throws RemoteException {
        graph= u.parseGraph("graph");
        rwlock = new ReentrantReadWriteLock();
        performance=new ArrayList<ArrayList<Object>>();
        // TODO Auto-generated constructor stub
    }

    /**
     *
     */
    private static final long serialVersionUID = 1L;

    public ArrayList<String> executeBatch(ArrayList<String> batch) throws RemoteException {

    private void Addtoperformance(long id, boolean b, long Time) {

    private void deleteEdge(int src, int dest) {

    private void addEdge(int src, int dest) {

    private String query(int src, int dest) {


    public String serverReady() throws RemoteException {
```

The Client Program

The client program consists of a single class with main() method. Within this method, it takes the hostname of the machine running the RMI registry as command line argument, forms an RMI URL, retrieves the remote

reference of the server RMI class by looking up the registry and invokes the execteBatch() and serverReady() methods.

```
public class Client extends UnicastRemoteObject implements IRemoteMethod , Runnable {
    IRemoteMethod access;
    boolean run=false;
    protected Client(IRemoteMethod access) throws RemoteException {
        // TODO Auto-generated constructor stub
        this.access=access;
    }

    /**
     *
     */
    private static final long serialVersionUID = 1L;

    public void run() {

    private ArrayList<String> generateBatch() {

    public ArrayList<String> executeBatch(ArrayList<String> batch) throws RemoteException {
    public static void main(String[] args) throws MalformedURLException,RemoteException,NotBoundException {
        System.out.println("\nConnecting To RMI Server...\n");
        // lookup method to find reference of remote object
        String URL="rmi://"+args[0]+":"+Integer.parseInt(args[1])+"/shortestPath";

        IRemoteMethod access = (IRemoteMethod) Naming
                .lookup(URL);
        new Thread(new Client(access)).start();

    }
```

Concurrency Control:-

We used A java.util.concurrent.locks.ReadWriteLock is an advanced thread lock mechanism. It allows multiple threads to read a certain resource, but only one to write it, at a time.

Performance Analysis:-

First Variant :- using read write lock to allow allows multiple threads to read a certain resources

Second variant :- it makes the thread run in sequential when we read or write.

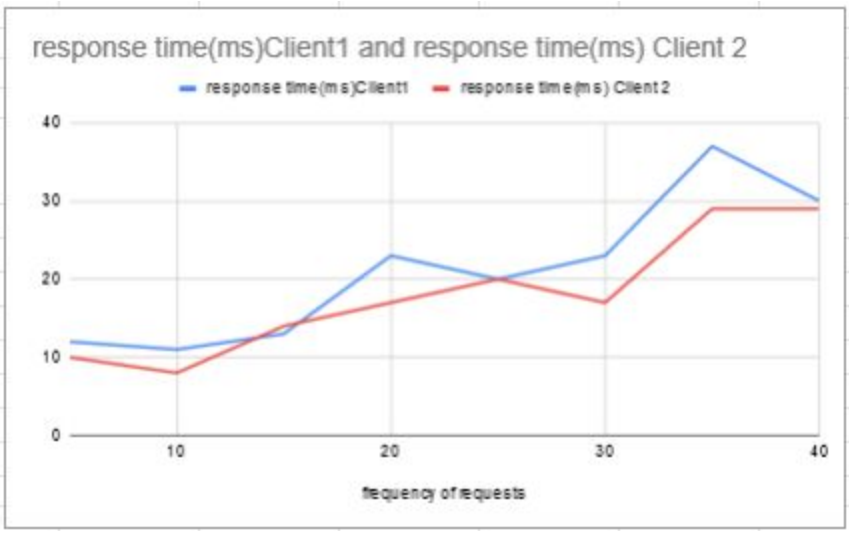
Recording response time Vs frequency of requests

First Variant

2 Clients ,delay between each request =1000

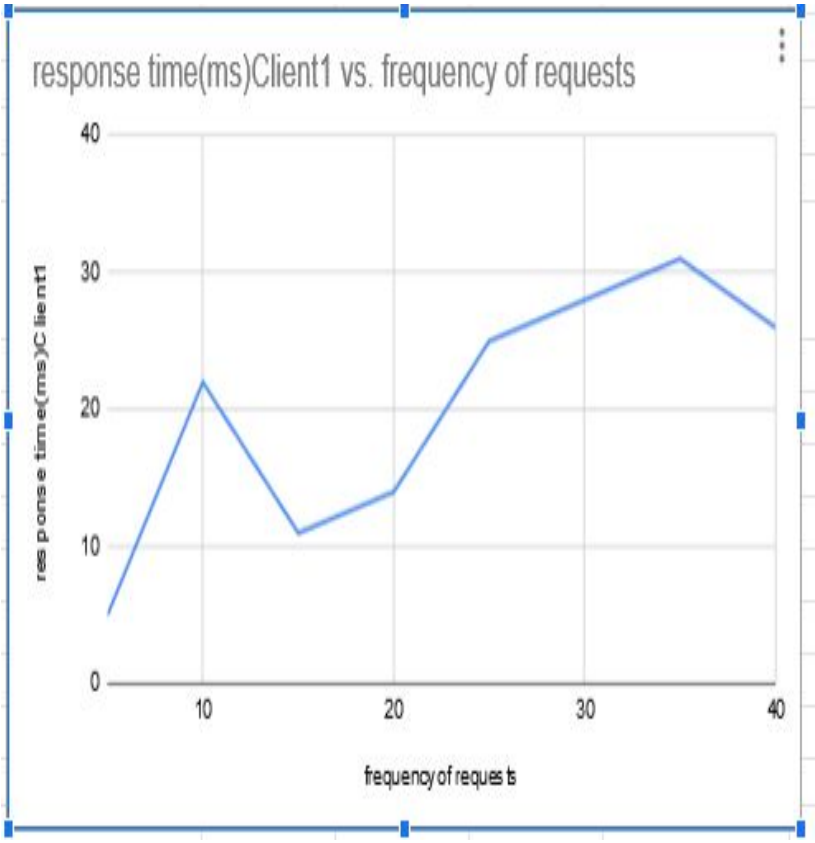
frequency of requests	5	10	15	20	25	30	35	40

response time(ms)	12	11	13	23	20	23	37	30
response time(ms)	10	8	14	17	20	17	29	29



Second Variant

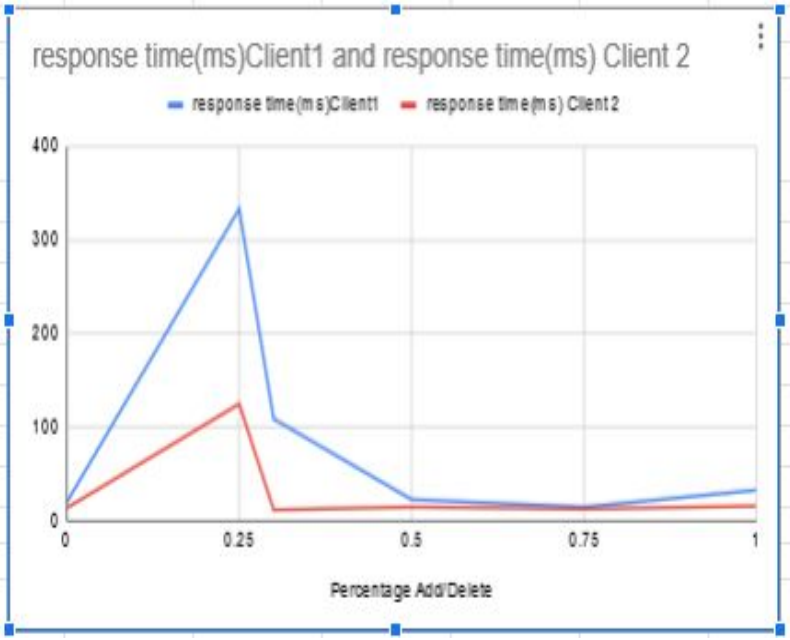
frequency of requests	response time(ms)Client1
5	5
10	22
15	11
20	14
25	25
30	28
35	31
40	26



Recording response time Vs percentage of add/delete operations

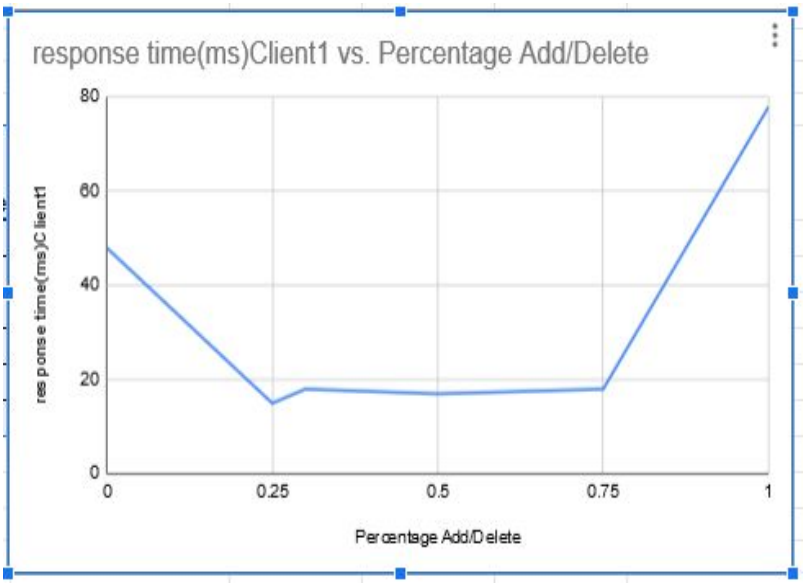
First Variant

Percentage Add/Delete	response time(ms)Client1	response time(ms) Client 2
0	20	14
0.25	333	125
0.3	109	12
0.5	23	15
0.75	15	13
1	33	16



Second Variant

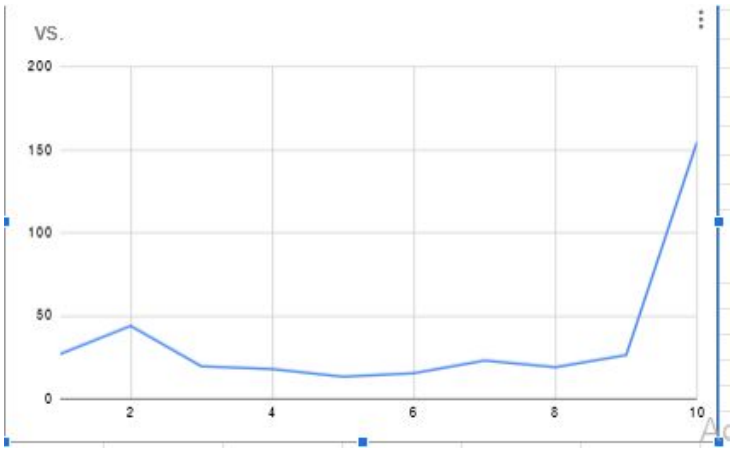
Percentage Add/Delete	response time(ms)Client1
0	48
0.25	15
0.3	18
0.5	17
0.75	18
1	78



Recording response time Vs number of nodes

First Variant

Number Of Nodes	Average Response Time
1	27
2	44
3	19.66666667
4	18
5	13.4
6	15.5
7	23.14285714
8	19.125
9	26.44444444
10	154.8



Second Variant

Number Of Nodes	Average Response Time
1	1515
2	2154
3	2592.666667
4	3046.75
5	4586.2
6	5396.666667
7	5837.571429
8	6238.5625
9	6445.222222
10	6128.7