CS 6385

Algorithmic Aspects of Telecommunication Network

Spring 2017

Project 2

An implementation of Nagamochi-Ibaraki Algorithm

Submitted by:

Ankita S. Patil

(**Net ID:** asp160730)

Instructor:

Andras Farago

Table of Contents

Introduction

In this project, a simple deterministic algorithm for finding minimum cuts in an undirected graph, discovered by Nagamochi and Ibaraki has been implemented. The algorithm is known as **Nagamochi – Ibaraki** algorithm.

In the next section, the goal or objective of an algorithm is explained, then pseudo code for Nagamochi-Ibaraki algorithm is provided followed by explanation of how implementation works. For better understanding experimental results are included along with the analysis. Appendix contains the source code.

Problem Description

In the design of a network topology, the main concern is vulnerability i.e. how vulnerable is the network, how easy it is to disconnect the network.

Therefore, to check vulnerability of a network we find a *minimum cut* in the entire graph (Undirected graph models the network topology) because the size of minimum cut tells us that how many links have to fail to disconnect the network assuming that it was originally connected. Hence, we often want to find a minimum cut in the entire graph not just between a specified source and destination. The minimum cut between two specified nodes can be obtained as a byproduct of the maximum flow computation. If, however, we want an overall minimum cut in the whole graph, then a single maximum flow computation does not suffice.

Interestingly, one can find an overall minimum cut directly, without using anything about maximum flows. We can determine the minimum cut in the graph by estimating the edge connectivity.

Goal: Given an undirected graph with N nodes and M edges, find the minimum cut. The size of this minimum cut characterizes the connectivity of the graph.

Definition:

Edge-connectivity between two nodes:

$\lambda(x,y)$ = minimum number of edges that need to be deleted to disconnect nodes *x and y.*

Edge-connectivity (of the graph):

$\lambda(G)$ = minimum number of edges that need to be deleted to disconnect G.

Therefore, $\lambda(G)$ is the size of a *minimum cut.*

Solution

A minimum cut for an entire undirected graph can be computed without using anything about maximum flows. We can determine the minimum cut in the graph by calculating the (edge)connectivity of the graph.

A simple deterministic algorithm for minimum cut discovered by Nagamochi-Ibaraki can be used.

Nagamochi – Ibaraki algorithm makes use of **maximum adjacency (MA) ordering** of vertices to compute the edge connectivity.

An MA ordering $v_1,...,v_n$ of the nodes is generated recursively the by the following algorithm:

- Take any of the nodes for v_1.
- Once $v_1,...,v_i$ is already chosen, take a node for v_{i+1} that has the maximum number of edges connecting it with the set $\{v_1,...,v_i\}$
- *In any MA ordering $v_1,...,v_n$ of the nodes $\lambda(v_{n-1},v_n) = d(v_n)$ holds, where $d(.)$ denotes the degree of the node.*

Deterministic algorithm for minimum cut :

Let G_{xy} be the graph obtained from G by contracting (merging) nodes x,y. In this operation we omit the possibly arising loop (if x,y are connected in G), but keep the parallel edges.

We can now use the following result that is not hard to prove:

For any two nodes x,y

$$\lambda(G) = \min\{\lambda(x,y), \lambda(G_{xy})\} \qquad \text{(A)}$$

Thus, if we can find two nodes x,y for which $\lambda(x,y)$ can be computed easily (without a flow computation), then the above formula yields a simple recursive algorithm.

One can indeed easily find such a pair x,y of nodes. This is done via the so-called *Maximum Adjacency (MA) ordering.* An MA ordering $v_1,...,v_n$ of the nodes is generated recursively the by the following algorithm:

- Take any of the nodes for v_1.
- Once $v_1,...,v_i$ is already chosen, take a node for v_{i+1} that has the maximum number of edges connecting it with the set $\{v_1,...,v_i\}$.

Nagamochi and Ibaraki proved that this ordering has the following nice property:

In any MA ordering $v_1,\dots,v_n$ of the nodes

$$\lambda(v_{n-1}, v_n) = d(v_n)$$

holds, where $d(.)$ *denotes the degree of the node.*

Therefore, it is enough to create an MA ordering, as described above, and take the last 2 nodes in this ordering for x,y. Then the connectivity between them will be equal to the degree of the last node in the MA ordering. Then one can apply formula (A) recursively to get the minimum cut. Here we make use of the fact that it reduces the problem to computing $\lambda(G_{xy})$ and G_{xy} is already a smaller graph.

Pseudocode

```
INPUT:    // an undirected graph
            G := (n,m)
             n: number of nodes
             m: number of edges
            M[n][n]: Adjacency Matrix

OUTPUT:   //Connectivity of a graph or minimum cut
            λ(G)

//a recursive function to implement Nagamochi-Ibaraki algorithm
calculateEdgeConnectivity(Graph G)
{
        //Base case
        If( n = = 2 ){
                    // from adjacency matrix of a graph
                    λ (x,y) := M[0][1]
           return λ(x,y)

        }
        else {
               if(G is connected){
                        //call MA order
                        λ(x,y) := MAOrder(G)

                        //contracted graph is obtained by merging G and (x,y)
                        Gxy : := getContractedGraph( )
                  return min( λ(x,y) ,  calculateEdgeConnectivity(Gxy) )
               }
              else {
                            //graph is disconnected
                            return null
                  }
        }

}
```

The entire pseudo code is explained in next section labelled – Implementation details

Implementation Details

Technologies used:

Programming Language	Java
Operating System	Windows 10
Development Environment	Eclipse Neon

Implementation Details:

The project consists of following package and classes.

Package	com.ankita.atn.project2
classes	Graph.java TestProject.java

- The algorithm is implemented with respect to following guidelines –

  Run the program on randomly generated examples. Let the number of nodes be fixed at *n* = 21, while the number *m* of edges will vary between 20 and 200, in steps of 4. For any such value of *m*, the program creates 5 graphs with *n* = 21 nodes and *m* edges. The actual edges are selected randomly. Parallel edges and self-loops are *not* allowed in the original graph generation. Note, however, that the Nagamochi-Ibaraki algorithm allows parallel edges in its internal working, as they may arise due to the merging of nodes.

- The package com.ankita.atn.project2 contains the implementation of Nagamochi – Ibaraki algorithm to find minimum cut.
- TestProject.java is a driver class which contains the main() and Graph.java class contains the entire logic.

  (1) Read the values of number of nodes (n) and number of edges (m). As per the guidelines given, the number of nodes are kept fixed (n = 21). Number of edges (m) vary between 20 and 200.
  For every run of the program, number of edges are incremented by 4.
  (2) For fixed value of n and varying value of m, every time a graph is generated using ***createGraph()*** and the control of the program goes to Graph.java and a graph with m edges is created. The actual edges are selected randomly.
  If we want to see which nodes are randomly connected, its adjacency matrix can be printed using ***printGraph()***

Now to find a minimum cut for this graph ***calculateEdgeConnectivity()*** is invoked from main() and control goes to Graph.java

If number of nodes are equal to 2 then, edge connectivity between the two nodes is returned which is nothing but the degree of second node. This can be obtained from the adjacency matrix.

If number of nodes are greater than 2 then MA ordering is calculated using ***MAOrder()*** and maximum adjacency of two sets of vertices is returned.

If we get its value to be zero then the graph obtained is disconnected else Graph is contracted with those sets of vertices using ***getContractedGraph ()*** .

Hence, every time graph gets updated because merging of two sets of nodes happen. The connectivity of such a contracted graph is found recursively. The edge connectivity of entire graph is **minimum($\lambda(x,y)$, λ(Contracted graph)).**

The recursion continues till base case is obtained (n = 2).

(3) Step 2 is carried out to obtain 4 more graphs. Hence by the end of this step, we get 5 randomly generated graphs with value of edge connectivity calculated.

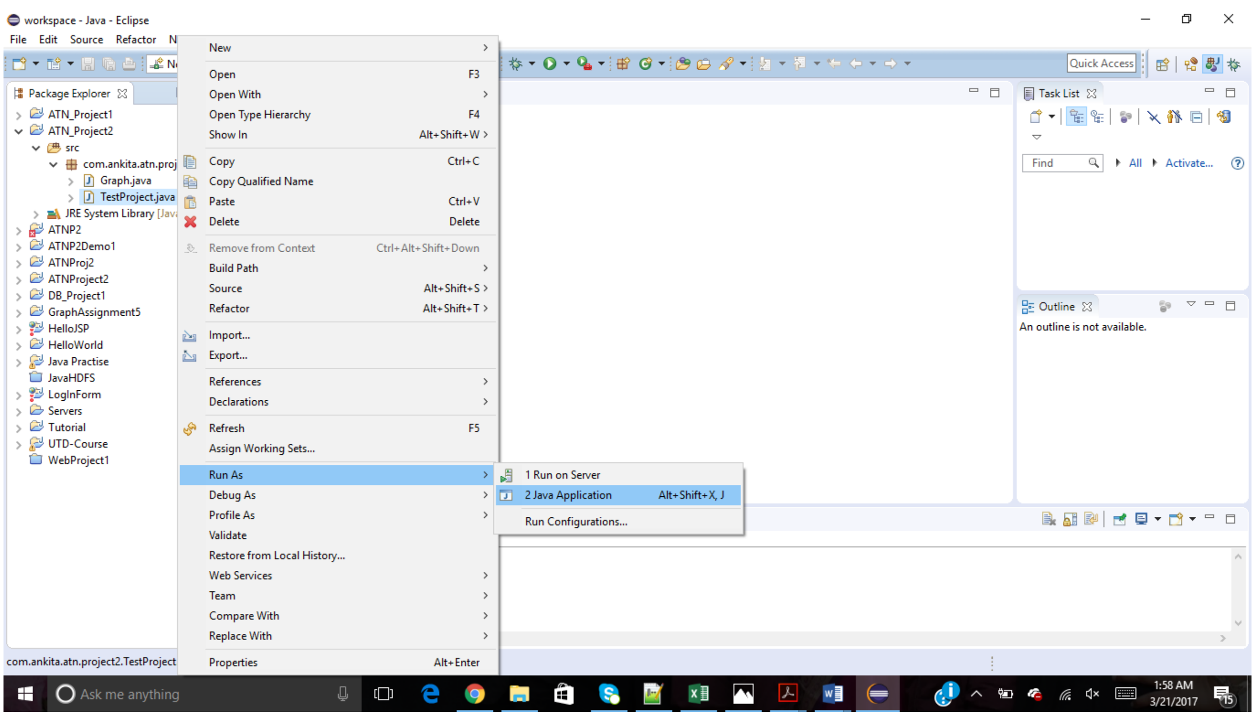
(4) Step (2) and (3) are carried out while number of edges are less than or equal to 200.

ReadMe Section
-Instructions on how to run a program

(1) Import the package com.ankita.atn.project2 in Eclipse IDE.

(2) The package consists of two java files.
  - Graph.java
  - TestProject.java

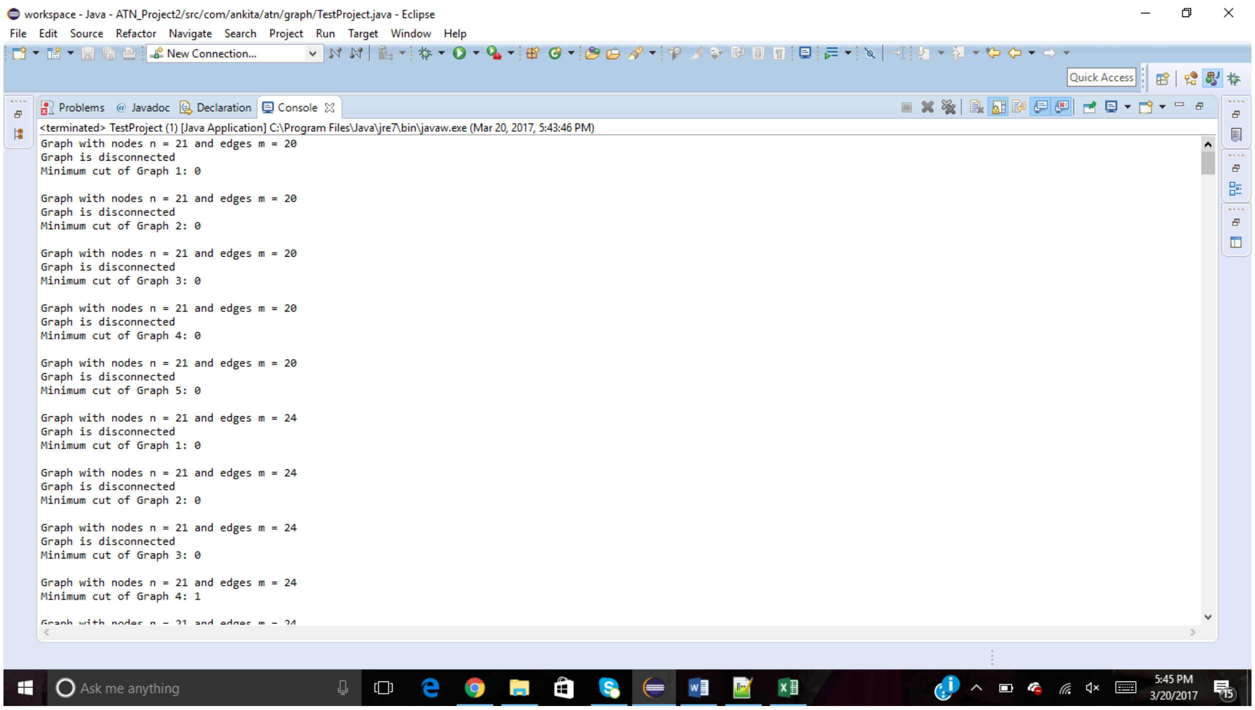
(3) Run TestProject.java from the package com.ankita.atn.project2

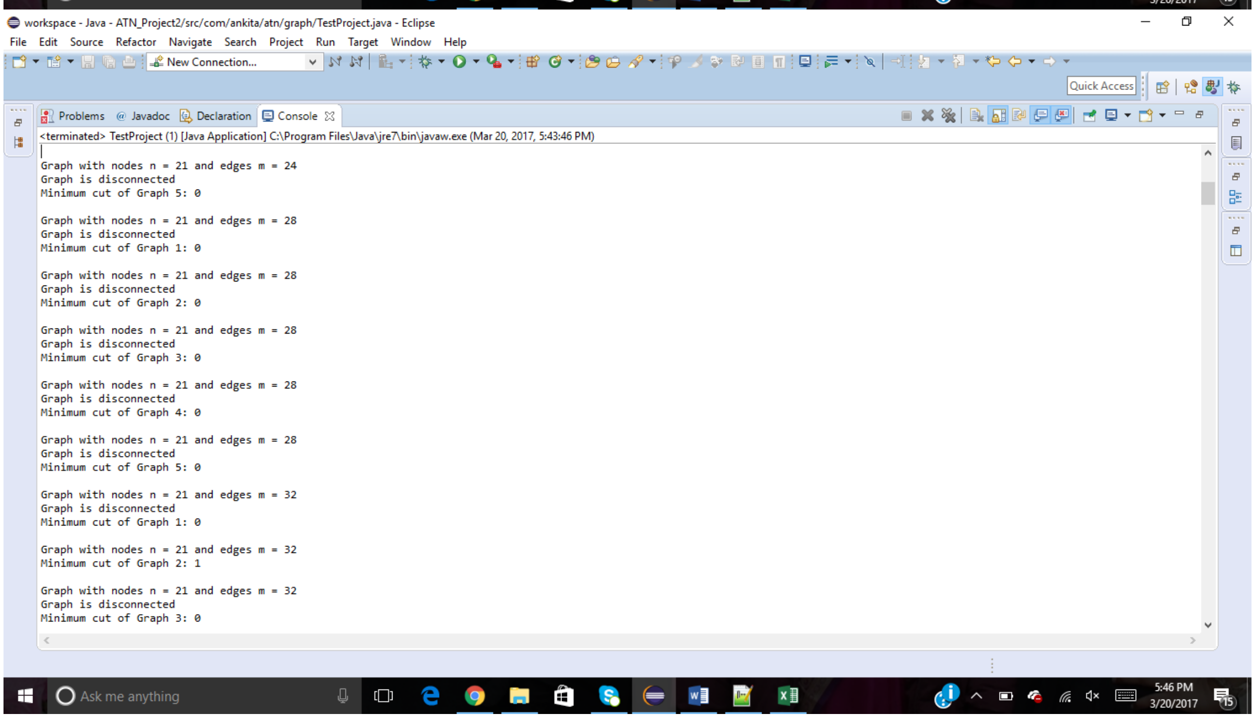


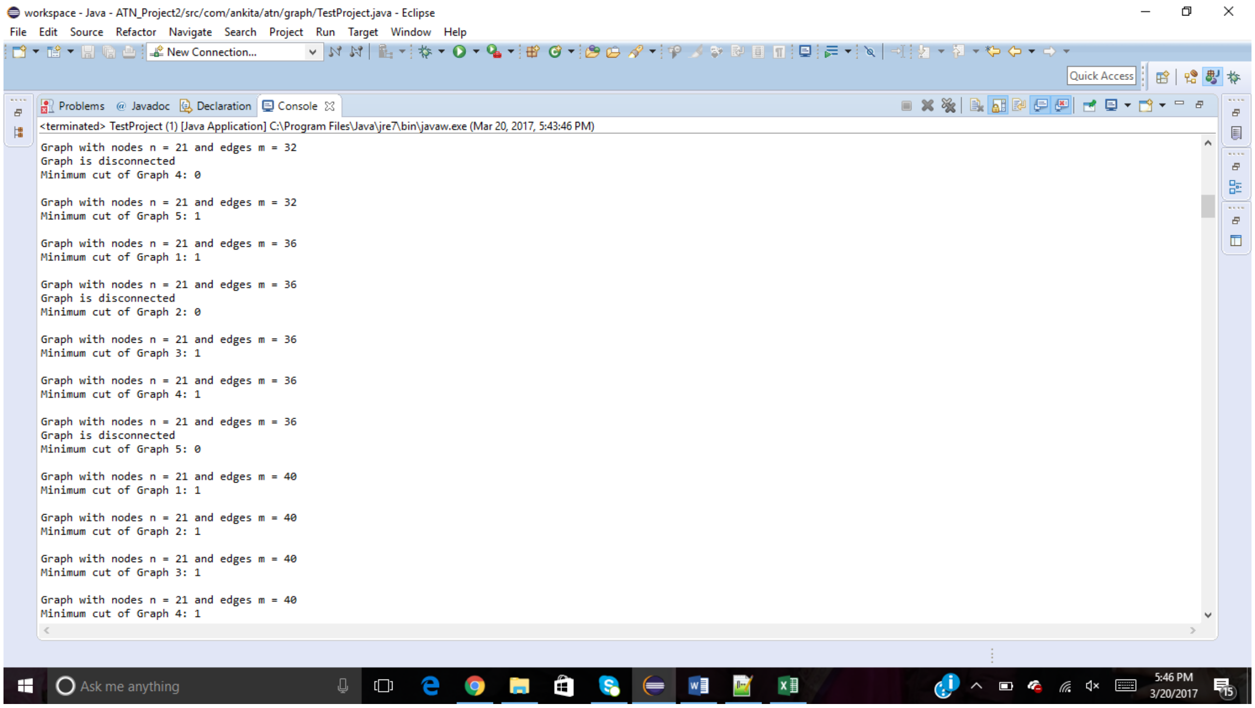
(4) Output can be seen from console.

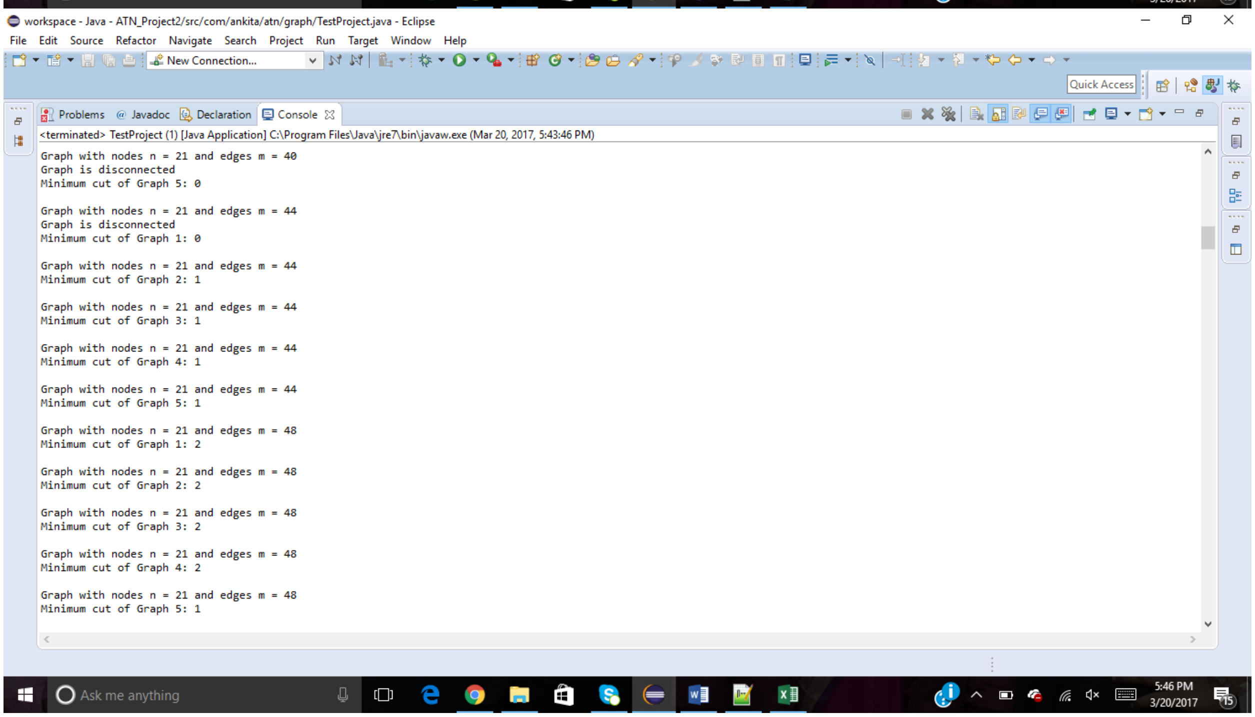
Sample Output

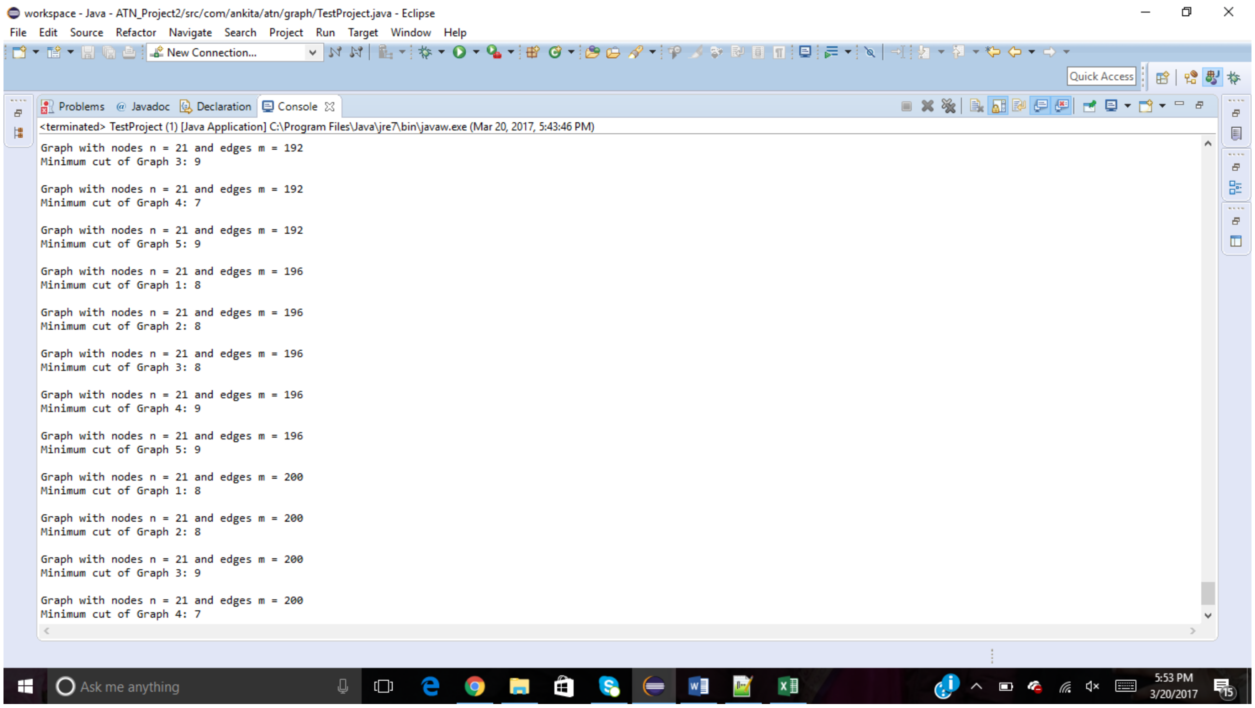
(Only few Screenshots are attached)

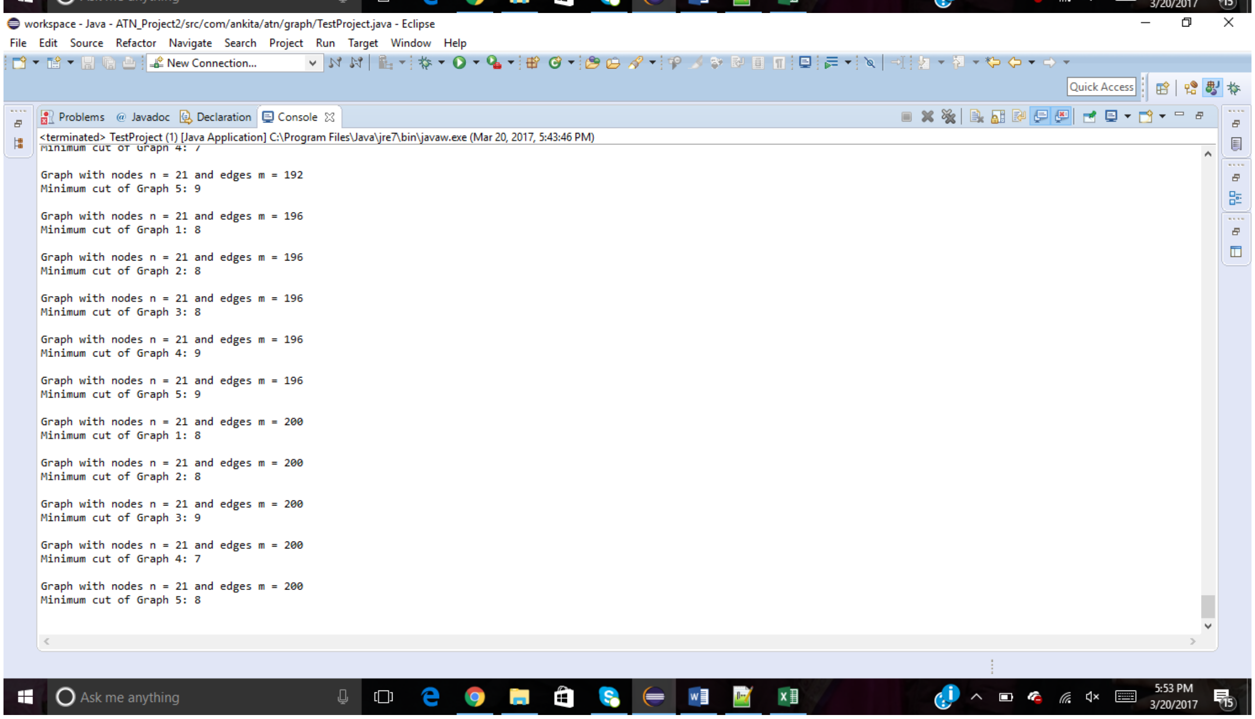





workspace - Java - ATN_Project2/src/com/ankita/atn/graph/TestProject.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Target Window Help
New Connection...
Quick Access
Problems Javadoc Declaration Console
<terminated> TestProject (1) [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Mar 20, 2017, 5:43:46 PM)
Graph with nodes n = 21 and edges m = 32
Graph is disconnected
Minimum cut of Graph 4: 0
Graph with nodes n = 21 and edges m = 32
Minimum cut of Graph 5: 1
Graph with nodes n = 21 and edges m = 36
Minimum cut of Graph 1: 1
Graph with nodes n = 21 and edges m = 36
Graph is disconnected
Minimum cut of Graph 2: 0
Graph with nodes n = 21 and edges m = 36
Minimum cut of Graph 3: 1
Graph with nodes n = 21 and edges m = 36
Minimum cut of Graph 4: 1
Graph with nodes n = 21 and edges m = 36
Graph is disconnected
Minimum cut of Graph 5: 0
Graph with nodes n = 21 and edges m = 40
Minimum cut of Graph 1: 1
Graph with nodes n = 21 and edges m = 40
Minimum cut of Graph 2: 1
Graph with nodes n = 21 and edges m = 40
Minimum cut of Graph 3: 1
Graph with nodes n = 21 and edges m = 40
Minimum cut of Graph 4: 1
Ask me anything
5:46 PM
3/20/2017


workspace - Java - ATN_Project2/src/com/ankita/atn/graph/TestProject.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Target Window Help
New Connection...
Quick Access
Problems Javadoc Declaration Console
<terminated> TestProject (1) [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Mar 20, 2017, 5:43:46 PM)
Graph with nodes n = 21 and edges m = 40
Graph is disconnected
Minimum cut of Graph 5: 0
Graph with nodes n = 21 and edges m = 44
Graph is disconnected
Minimum cut of Graph 1: 0
Graph with nodes n = 21 and edges m = 44
Minimum cut of Graph 2: 1
Graph with nodes n = 21 and edges m = 44
Minimum cut of Graph 3: 1
Graph with nodes n = 21 and edges m = 44
Minimum cut of Graph 4: 1
Graph with nodes n = 21 and edges m = 44
Minimum cut of Graph 5: 1
Graph with nodes n = 21 and edges m = 48
Minimum cut of Graph 1: 2
Graph with nodes n = 21 and edges m = 48
Minimum cut of Graph 2: 2
Graph with nodes n = 21 and edges m = 48
Minimum cut of Graph 3: 2
Graph with nodes n = 21 and edges m = 48
Minimum cut of Graph 4: 2
Graph with nodes n = 21 and edges m = 48
Minimum cut of Graph 5: 1
Ask me anything
5:46 PM
3/20/2017


workspace - Java - ATN_Project2/src/com/ankita/atn/graph/TestProject.java - Eclipse
<terminated> TestProject (1) [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Mar 20, 2017, 5:43:46 PM)
Graph with nodes n = 21 and edges m = 192
Minimum cut of Graph 3: 9
Graph with nodes n = 21 and edges m = 192
Minimum cut of Graph 4: 7
Graph with nodes n = 21 and edges m = 192
Minimum cut of Graph 5: 9
Graph with nodes n = 21 and edges m = 196
Minimum cut of Graph 1: 8
Graph with nodes n = 21 and edges m = 196
Minimum cut of Graph 2: 8
Graph with nodes n = 21 and edges m = 196
Minimum cut of Graph 3: 8
Graph with nodes n = 21 and edges m = 196
Minimum cut of Graph 4: 9
Graph with nodes n = 21 and edges m = 196
Minimum cut of Graph 5: 9
Graph with nodes n = 21 and edges m = 200
Minimum cut of Graph 1: 8
Graph with nodes n = 21 and edges m = 200
Minimum cut of Graph 2: 8
Graph with nodes n = 21 and edges m = 200
Minimum cut of Graph 3: 9
Graph with nodes n = 21 and edges m = 200
Minimum cut of Graph 4: 7


workspace - Java - ATN_Project2/src/com/ankita/atn/graph/TestProject.java - Eclipse
<terminated> TestProject (1) [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Mar 20, 2017, 5:43:46 PM)
Graph with nodes n = 21 and edges m = 192
Minimum cut of Graph 5: 9
Graph with nodes n = 21 and edges m = 196
Minimum cut of Graph 1: 8
Graph with nodes n = 21 and edges m = 196
Minimum cut of Graph 2: 8
Graph with nodes n = 21 and edges m = 196
Minimum cut of Graph 3: 8
Graph with nodes n = 21 and edges m = 196
Minimum cut of Graph 4: 9
Graph with nodes n = 21 and edges m = 196
Minimum cut of Graph 5: 9
Graph with nodes n = 21 and edges m = 200
Minimum cut of Graph 1: 8
Graph with nodes n = 21 and edges m = 200
Minimum cut of Graph 2: 8
Graph with nodes n = 21 and edges m = 200
Minimum cut of Graph 3: 9
Graph with nodes n = 21 and edges m = 200
Minimum cut of Graph 4: 7
Graph with nodes n = 21 and edges m = 200
Minimum cut of Graph 5: 8

Experimental values

Nodes	Edges	Graphs	Minimum cut [λ(G)]
21	20	G1	0
21	20	G2	0
21	20	G3	0
21	20	G4	0
21	20	G5	0
21	24	G1	0
21	24	G2	0
21	24	G3	0
21	24	G4	1
21	24	G5	0
21	28	G1	0
21	28	G2	0
21	28	G3	0
21	28	G4	0
21	28	G5	0
21	32	G1	0
21	32	G2	1
21	32	G3	0
21	32	G4	0
21	32	G5	1
21	36	G1	1
21	36	G2	0
21	36	G3	1
21	36	G4	1
21	36	G5	0
21	40	G1	1
21	40	G2	1
21	40	G3	1
21	40	G4	1
21	40	G5	0
21	44	G1	0
21	44	G2	1
21	44	G3	1
21	44	G4	1
21	44	G5	1
21	48	G1	2
21	48	G2	2

21	48	G3	2
21	48	G4	2
21	48	G5	1
21	52	G1	1
21	52	G2	2
21	52	G3	3
21	52	G4	2
21	52	G5	1
21	56	G1	2
21	56	G2	2
21	56	G3	2
21	56	G4	2
21	56	G5	1
21	60	G1	1
21	60	G2	2
21	60	G3	1
21	60	G4	2
21	60	G5	1
21	64	G1	3
21	64	G2	3
21	64	G3	2
21	64	G4	2
21	64	G5	2
21	68	G1	2
21	68	G2	2
21	68	G3	2
21	68	G4	2
21	68	G5	2
21	72	G1	2
21	72	G2	1
21	72	G3	3
21	72	G4	3
21	72	G5	2
21	76	G1	2
21	76	G2	3
21	76	G3	3
21	76	G4	3
21	76	G5	2
21	80	G1	3
21	80	G2	4
21	80	G3	2

21	80	G4	4
21	80	G5	4
21	84	G1	3
21	84	G2	3
21	84	G3	2
21	84	G4	4
21	84	G5	2
21	88	G1	2
21	88	G2	3
21	88	G3	3
21	88	G4	4
21	88	G5	2
21	92	G1	3
21	92	G2	3
21	92	G3	3
21	92	G4	4
21	92	G5	3
21	96	G1	4
21	96	G2	5
21	96	G3	4
21	96	G4	2
21	96	G5	4
21	100	G1	3
21	100	G2	4
21	100	G3	5
21	100	G4	3
21	100	G5	3
21	104	G1	5
21	104	G2	3
21	104	G3	4
21	104	G4	3
21	104	G5	4
21	108	G1	4
21	108	G2	6
21	108	G3	5
21	108	G4	3
21	108	G5	3
21	112	G1	3
21	112	G2	6
21	112	G3	6
21	112	G4	4

21	112	G5	7
21	116	G1	3
21	116	G2	4
21	116	G3	2
21	116	G4	5
21	116	G5	6
21	120	G1	4
21	120	G2	5
21	120	G3	3
21	120	G4	5
21	120	G5	5
21	124	G1	6
21	124	G2	5
21	124	G3	5
21	124	G4	4
21	124	G5	7
21	128	G1	4
21	128	G2	5
21	128	G3	6
21	128	G4	5
21	128	G5	5
21	132	G1	6
21	132	G2	3
21	132	G3	6
21	132	G4	7
21	132	G5	5
21	136	G1	5
21	136	G2	5
21	136	G3	7
21	136	G4	5
21	136	G5	5
21	140	G1	4
21	140	G2	4
21	140	G3	4
21	140	G4	7
21	140	G5	6
21	144	G1	7
21	144	G2	5
21	144	G3	7
21	144	G4	6
21	144	G5	5

21	148	G1	7
21	148	G2	4
21	148	G3	6
21	148	G4	6
21	148	G5	7
21	152	G1	5
21	152	G2	5
21	152	G3	5
21	152	G4	3
21	152	G5	6
21	156	G1	8
21	156	G2	3
21	156	G3	6
21	156	G4	6
21	156	G5	7
21	160	G1	4
21	160	G2	5
21	160	G3	6
21	160	G4	7
21	160	G5	5
21	164	G1	7
21	164	G2	6
21	164	G3	7
21	164	G4	6
21	164	G5	8
21	168	G1	5
21	168	G2	7
21	168	G3	5
21	168	G4	8
21	168	G5	7
21	172	G1	6
21	172	G2	8
21	172	G3	6
21	172	G4	8
21	172	G5	6
21	176	G1	8
21	176	G2	6
21	176	G3	7
21	176	G4	6
21	176	G5	8
21	180	G1	7

21	180	G2	9
21	180	G3	7
21	180	G4	8
21	180	G5	9
21	184	G1	8
21	184	G2	7
21	184	G3	7
21	184	G4	9
21	184	G5	7
21	188	G1	5
21	188	G2	7
21	188	G3	7
21	188	G4	7
21	188	G5	8
21	192	G1	9
21	192	G2	8
21	192	G3	9
21	192	G4	7
21	192	G5	9
21	196	G1	8
21	196	G2	8
21	196	G3	8
21	196	G4	9
21	196	G5	9
21	200	G1	8
21	200	G2	8
21	200	G3	9
21	200	G4	7
21	200	G5	8

Observations and Analysis

(A)Determining the relationship between m and λ(G)

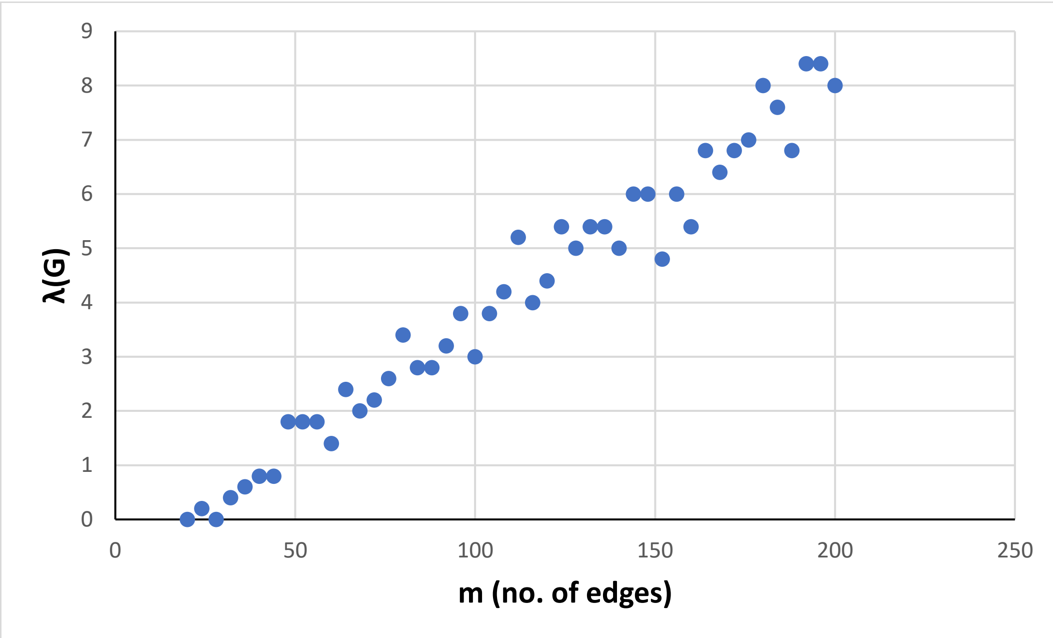
Experimental values for number of edges and connectivity of a graph

Note: Since average of λ values is taken, therefore non-integer value of λ is considered for experimental purpose.

Nodes	Edges	Smallest λ(G)	Largest λ(G)	Average λ(G)
21	20	0	0	0
21	24	0	1	0.2
21	28	0	0	0
21	32	0	1	0.4
21	36	0	1	0.6
21	40	0	1	0.8
21	44	0	1	0.8
21	48	1	2	1.8
21	52	2	3	1.8
21	56	1	2	1.8
21	60	1	2	1.4
21	64	2	3	2.4
21	68	2	2	2
21	72	1	3	2.2
21	76	2	3	2.6
21	80	2	4	3.4
21	84	2	4	2.8
21	88	2	4	2.8
21	92	3	4	3.2
21	96	2	5	3.8
21	100	3	5	3
21	104	3	5	3.8
21	108	3	6	4.2
21	112	3	7	5.2
21	116	2	6	4
21	120	3	5	4.4
21	124	4	7	5.4
21	128	4	6	5
21	132	3	7	5.4
21	136	5	7	5.4

21	140	4	7	5
21	144	5	7	6
21	148	4	7	6
21	152	3	6	4.8
21	156	3	8	6
21	160	4	7	5.4
21	164	6	8	6.8
21	168	5	8	6.4
21	172	6	8	6.8
21	176	6	8	7
21	180	7	9	8
21	184	7	9	7.6
21	188	5	8	6.8
21	192	7	9	8.4
21	196	8	9	8.4
21	200	7	9	8

Relationship between number of edges and connectivity of a graph



- Edge connectivity of an undirected graph is the minimum number of edges whose removal disconnects the graph. Hence, expected behavior is- if number of edges are more, the graph is more edge-connected.
- For an undirected graph of size n, there should be at least (n-1) edges to make it connected. Since we are randomly generating edges between two nodes of a graph, it is possible that the graph generated is disconnected when the number of edges are less.
- In the experiment, which we have conducted, the number of nodes are kept fixed and number of edges are incremented by four. For every value of m (number of edges), 5 graphs are generated randomly. To reduce the effect of randomness, average value of edge connectivity is determined.
- From the above **scatter plot**, we observe that for m = 20, it is highly possible that the generated graph resulted in a disconnected graph and therefore the edge connectivity is 0.
- As the number of edges increase, the connectivity of the graph increases. As more number of edges are randomly used between nodes to construct a graph thereby reducing the possibility of generating disconnected graph.
- Therefore, there are minimum chances of getting disconnected graph with more number of edges.
- Hence, number of edges (m) and edge connectivity of a graph λ(G) depend on each other.

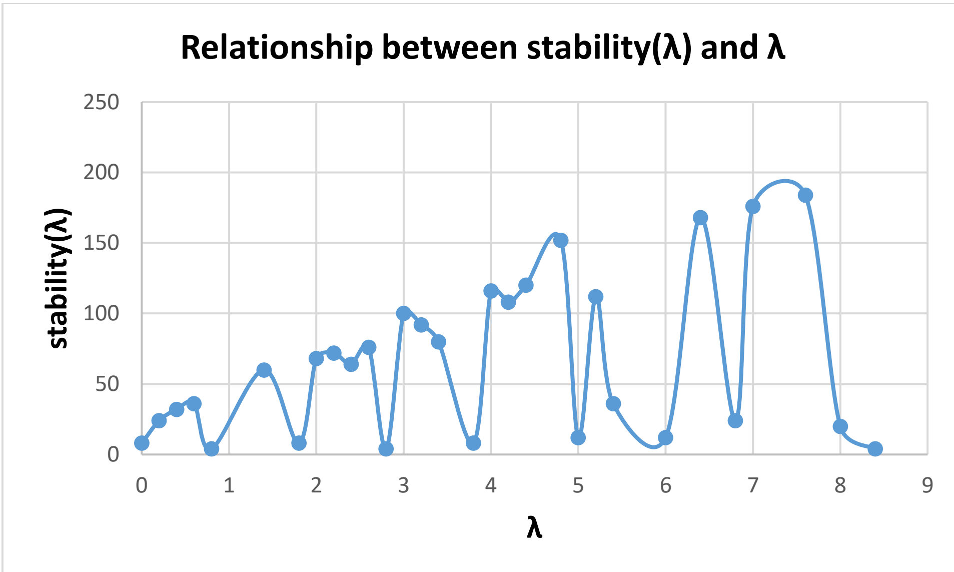
(B)Determining the relationship between λ and Stability(λ)

Experimental values for connectivity and stability

λ(G)	Edges with which λ(G) occurred	Smallest edge	Largest edge	stability(λ)
0	20, 28	20	28	8
0.2	24	0	24	24
0.4	32	0	32	32
0.6	36	0	36	36
0.8	40, 44	40	44	4
1.4	60	0	60	60
1.8	48,52,56	48	56	8
2	68	0	68	68
2.2	72	0	72	72
2.4	64	0	64	64
2.6	76	0	76	76
2.8	84, 88	84	88	4
3	100	0	100	100
3.2	92	0	92	92
3.4	80	0	80	80
3.8	96, 104	96	104	8
4	116	0	116	116
4.2	108	0	108	108
4.4	120	0	120	120
4.8	152	0	152	152
5	128, 140	128	140	12
5.2	112	0	112	112
5.4	124, 132, 136, 160	124	160	36
6	144, 148, 156	144	156	12
6.4	168	0	168	168
6.8	164, 172, 188	164	188	24
7	176	0	176	176
7.6	184	0	184	184
8	180, 200	180	200	20
8.4	192, 196	192	196	4

λ(G)	stability(λ)
0	8
0.2	24
0.4	32
0.6	36
0.8	4
1.4	60
1.8	8
2	68
2.2	72
2.4	64
2.6	76
2.8	4
3	100
3.2	92
3.4	80
3.8	8
4	116
4.2	108
4.4	120
4.8	152
5	12
5.2	112
5.4	36
6	12
6.4	168
6.8	24
7	176
7.6	184
8	20
8.4	4

Relationship between connectivity and stability



- For every connectivity value $\lambda = \lambda(G)$ that occurred in the experiments, we record the largest and smallest number of edges with which this λ value occurred. Let us call their difference the *stability* of λ, and let us denote it by $s(\lambda)$.

- For every connectivity value λ that occurs in some experiment, we check what was the largest number of edges with which this λ value obtained. Let's say this edge number is M. Similarly, you check which was the smallest number of edges with which the same λ value obtained, let's say this edge number is m. Then the stability for this connectivity value λ is defined as $s(\lambda) = M – m$

- Stability determines the how spread the value of λ

- From the experimental values, we observe that for low values of λ, lower values of edges are responsible. That means if there are 21 nodes and 20 edges then it is high possibility that we get a disconnected graph most of the times. But It won't be the same case every time because for this 20 node -graph to be connected we need at least 20 edges.

- Therefore, for a particular value of λ we get may get edge number in specified range. For example,

λ(G)	Edges with which λ(G) occurred
0	20, 28
1.8	48, 52, 56

- But this argument always doesn't hold true if we check some of the experimental values.

λ(G)	Edges with which λ(G) occurred
5.4	124, 132, 136, 160
6.8	164, 172, 188

- Here we can say that there is variability in the edge numbers. Hence, experimental results contradict the argument that we get edge number in specific range every time for value of λ
- Because of the variability of edges belonging to particular λ, the difference between the smallest and largest edge number fluctuates for every value of λ.
- From experimental results, we observe that for the connectivity 0, stability is 8 but at the same time for connectivity 8.4 the stability is 4. For connectivity 3 the stability is 100. Hence it is complete random.
- Hence, we can conclude that variability in the edge numbers for a particular λ leads to random value of stability and therefore there is no strong relationship between stability and connectivity as both seem random.
  Therefore, stability and edge connectivity does not depend on each other.

References

(1) Lecture notes by Professor Andras Farago
(2) Nagamochi, H.; Ibaraki, T. (2008). *Algorithmic Aspects of Graph Connectivity*. Cambridge University Press

Appendix

--------------------------Source Code--

<u>TestProject.java</u>

```
package com.ankita.atn.project2;
/**
 * A program to find a minimum cut in an undirected graph
 * @author Ankita Patil
 *
 */
public class TestProject {
	static int m = 20;
	public static void main(String[] args) {

		/*
		 * Number of nodes are fixed
		 * Number of edges vary between 20 and 200 in steps of 4
		 */
		while(m <= 200){
		  for (int i = 0; i < 5; i ++){

					Graph g = new Graph(21, m);
					g.createGraph();
					//System.out.println("Adjacency Matrix : Graph " + i + "
with Nodes n = 21 and Edges m = "+ m );
					System.out.println("Graph with nodes n = 21 and edges m =
" + m);
					//g.printGraph();
					System.out.println("Minimum cut of Graph "+ (i+1) +": "
+g.calculateEdgeConnectivity());
					System.out.println();
		  }

				m = m + 4;
		}
	}

}
```

Graph.java

```
package com.ankita.atn.project2;

public class Graph {

	/*
	 * Number of nodes
	 */
	int n;

	/*
	 * Number of edges
	 */
	int m;

	/*
	 * Adjacency Matrix
	 */
	int M[][];


	/*
	 * Edge Connectivity between two sets A and B
	 */
	int edgeConnectivityAB;

	/*
	 * Edge Connectivity of the graph
	 */
	int edgeConnectivityGraph;

	public int a = -1;
	public int b = -1;


	public Graph(int n, int m) {
		this.n = n;
		this.m = m;
		M = new int[n][n];
	}

	/*
	 * Create graph with m edges
	 */
	public void createGraph() {

		//There are m edges
		for (int i = 0; i < m; i++) {

			//Since we need graph with random edges we use random function
			int nodeOne = (int)(Math.random() * n);
			int nodeTwo = (int)(Math.random() * n);
```

```
			//There cannot be self loop in graph
			while (nodeOne == nodeTwo) {
				nodeOne = (int)(Math.random() * 20);

			}

			M[nodeOne][nodeTwo] = 1;
			M[nodeTwo][nodeOne] = 1;
		}
	}

	/*
	 * Print Adjacency Matrix
	 */
	public void printGraph() {
		for (int i = 0; i < M.length; i++) {
			for (int j = 0; j < M.length; j++) {
				System.out.print(M[i][j] + " ");
			}
			System.out.println();
		}
	}

	/*
	 * This method calculates edge connectivity of graph using Nagamochi Ibaraki's
algorithm
	 */
	public int calculateEdgeConnectivity() {

		//Base case
		if(n == 2)
			return M[0][1];
		else {
			// edge connectivity between two sets of nodes
			edgeConnectivityAB = MAOrder();

			if(edgeConnectivityAB == 0) {
				System.out.println("Graph is disconnected");
				return 0;
			} else {
				//Merging the two sets of nodes, therefore we get a
contracted graph
				Graph Gab = getContractedGraph();
				//System.out.println("New Graph (Contracted Graph): ");
				//Gab.printGraph();
				this.edgeConnectivityGraph = Math.min(edgeConnectivityAB,
Gab.calculateEdgeConnectivity());
			}

			return edgeConnectivityGraph;
		}


	}
```

```
/*
 * This method calculates maximum adjacency ordering between verices
 */

public int MAOrder() {
	int maOrderAB = 0;

	//Array to store ordering of nodes
	int local[] = new int[n];

	//First node is selected at random from n nodes
	local[0] = (int) (Math.random() * n);
	//local[0] = 0;

	//This loop finds the ordering of remaining n-1 nodes
	for (int i = 1; i < local.length; i++) {

		int nodeToBeIncludedInSet = -1;
	int sum = 0;

	int nodeToBeTested = 0;
	while (nodeToBeTested < n) {

		int t = 0;
		boolean flag = true;
			for (int j = 0; j < i; j++) {
				if((nodeToBeTested == local[j])) {
					flag = false;
				}
			}
			if(flag) {

					for (int j2 = 0; j2 < i; j2++) {
						t += M[nodeToBeTested][local[j2]];
					}

				}

			if (t > sum)
		{
			nodeToBeIncludedInSet = nodeToBeTested;
			sum = t;
		}
			nodeToBeTested++;
		}

	if(nodeToBeIncludedInSet == -1){

			//System.out.println("Graph is a disconnected graph");
		return 0;
	}

	local[i] = nodeToBeIncludedInSet;
	}
```

```
            //Selecting last two nodes from the MA order in local array

            a = local[n-2];
            b = local[n-1];

            for(int z = 0; z < n; z++){
                  maOrderAB = maOrderAB + M[b][z];
      }

            //System.out.println("a = " + a + " B = " + b + " lambdaAB = " +
maOrderAB);
            return maOrderAB;
      }

      /*
       * This method generates a contracted graph
       */

      public Graph getContractedGraph() {
            Graph newGraph = new Graph(n-1, m);
            newGraph.M = new int[n-1][n-1];

            int tempGraph[][] = new int[n][n];


             //Copying the original graph in a temporary graph

            for (int i = 0; i < M.length; i++) {
                  for (int j = 0; j < M.length; j++) {
                        tempGraph[i][j] = M[i][j];
                  }
            }


              //Merging the values of two sets of nodes, by performing row wise and
column wise addition

            for (int i = 0; i < n; i++) {
                  if(i != a){
                        tempGraph[a][i] = tempGraph[a][i] + tempGraph[b][i];
                        tempGraph[i][a] = tempGraph[i][a] + tempGraph[i][b];

                  }

            }


            // Checking the condition to avoid self loops between two sets of nodes
which are getting merged
            int p=0,q=0;

            for(int i = 0; i < n; i++){
                  for(int j = 0; j < n; j++){
```

```
                    if((i != b)&&(j != b)){

                        newGraph.M[p][q++]= tempGraph[i][j];
                    }
                }

                if(i != b){
                    p++;
                    q=0;
                }
            }


            return newGraph;

        }

}
```