ALGORITHMIC ASPECTS OF TELECOMMUNICATION NETWORKS

PROJECT – 2

"AN IMPLEMENTATION OF NAGAMOCHI-IBARAKI ALGORITHM"

Submitted By

Navaneeth Venugopala Rao

CONTENTS

ABSTRACT

- Based on the Number of nodes (n) = 20 and Number of edges (m) taken from the user we design network topology.
- The essence of this project is to design various network topologies with varying number of links and analyze the effect of varying number of links on the link connectivity, critical link count of the topologies and average degree of the graph for different edge connectivity.
- Takes as input for each pair of nodes in each topology –the number of links between them, which is randomly generated from a certain range of values
- Makes use of the Nagamochi-Ibaraki algorithm and Maximum Adjacency Ordering to compute the link connectivity of the network topology.
- Gives insight into the reliability of the network topology by computing the number of critical links for varying densities of the network topology.
- Link connectivity and Critical link Count is analyzed and plotted for varying densities of the network topology.
- Average degree can be plotted for based on edge connectivity (lambda).

PSEUDO-CODE

TESTER CLASS: main method

- Step1 : Start
- Step2: Initialize the number of nodes n
- Step3: Initialize range of possible values for connectivity matrix b[][]
- Step4: For M=40 to 400 , iterate with increments of 5
  DO
- Step5: FOR I=0 to N-1 DO
  FOR J=0 to I DO
  Allocate randomly a value for Connectivity matrix from range of values
  END FOR
  END FOR
- Step 6: Find a random node to start with : randomVertex
- Step 7: Find the minimum degree for the original graph
- Step 8: Find edge connectivity by finding lesser of min degree and call method determineAdjacency(b, n , randomVertex)
- Step 9: FOR X=0 to N-1 DO
  FOR Y=0 to X-1 DO
- Step 10: Copy value from b[][] to a matrix z[][]
- Step 11: IF there exists edges between X and Y
  Remove one edge, update z[][]
  Find edge connectivity ƛ by passing z instead of b in the determineAdjacency parameters list.
- Step 12 : if ƛ is found to be less than edge connectivity of original graph , increment critical count edge by number of edges between X&Y originally.

END IF
END IF

- Step 13: Copy (x,y) to a critical edge matrix (say c[][])
- Step 14: Replace removed edges in z[][]

END FOR
END FOR

- Step 15: Display the number of edges, edge connectivity and no of critical edges of graph with m edges

END FOR

- Step 16: Stop

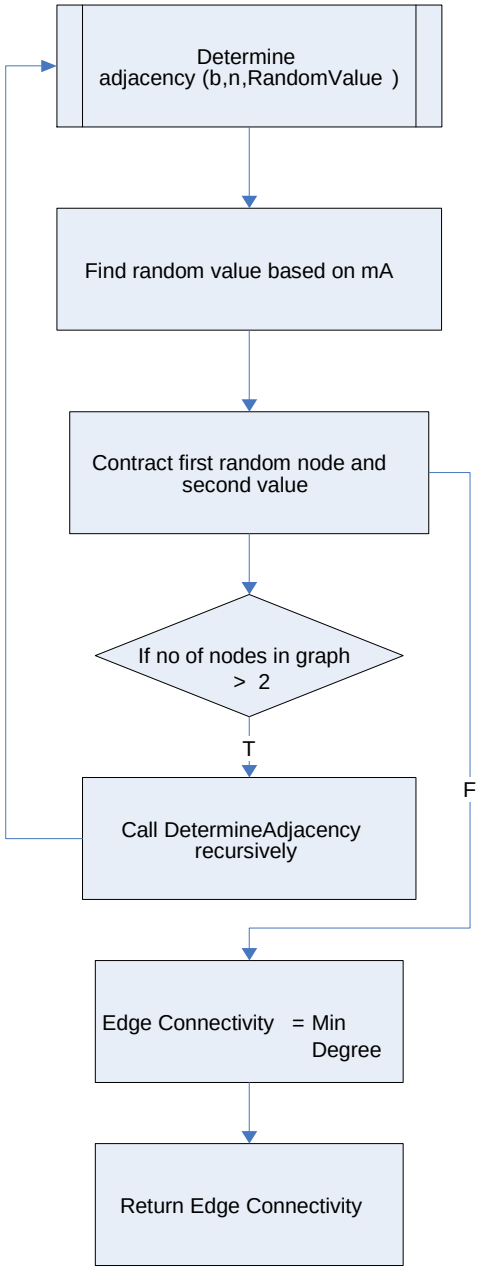
Nagamochi Class:

determineAdjacency(b[ ] [ ], n, randomVertex)

- Step 1 : Find 2nd vertex that should be contracted with randomVertex based on Maximum Adjacency Ordering
- Step 2: Contract the two nodes
- Step 3: Find the minimum degree of contracted graph g[][]
- Step 4: IF there are more than 2 nodes in the graph
  Recursively call the same method with g instead of b
  ELSIF
  Set edge connectivity to the min degree of the graph
- Step 5: Return the edge connectivity

FLOWCHART

- The flowchart corresponds to the determineAdjacency method in the Nagamochi Class.



NAGAMOCHI-IBARAKI Algorithm

- Nagamochi-Ibaraki's algorithm is a simple-deterministic algorithm for finding minimum cut in an undirected graph.

- If the edge connectivity of a graph G is denoted by $\lambda(G)$ and $\lambda(x,y)$ denotes the connectivity between two different nodes x,y , then G_{xy} be the graph obtained by contracting nodes x,y,

  Then the following result holds:

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

- In other words, the edge connectivity of a graph is the lesser of the minimum degree of the graph and the edge connectivity of the subgraph of G due to the contraction of the nodes x&y.

- This basically boils down to a recursive algorithm that contracts the graph till only two nodes are left and the degree of nodes at this stage is returned back so that $\lambda(G)$ can be computed at the previous stage.

- Finally $\lambda(G_{xy})$ is found out and its value is substituted in the equation to compute the edge connectivity of the graph.

MAXIMUM ADJACENCY ORDERING

- In the above equation to compute edge connectivity of a graph, choosing of a pair of nodes (x,y) is based on the concept of Maximum Adjacency Ordering.

- Initially, we choose the first node (x) randomly such that each node in the graph has equal probability of getting selected.

- Next, we choose y such that it is the node with maximum connectivity with x.

- We contract the nodes x&y into one node (x) and replace all edges coming into y as going to x, but discarding edges between x and y to avoid self-loops.

- The same procedure is repeated, at each stage for node x (which contains one node more at each stage) till there exist only two nodes in the graph (x and z).

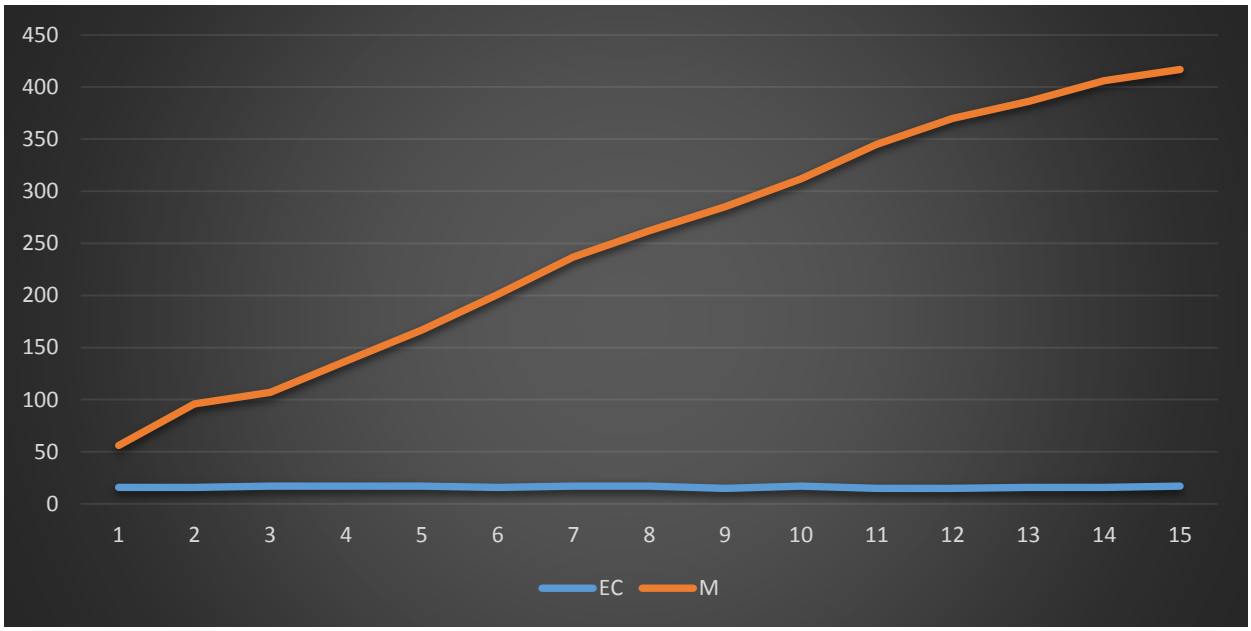
- Now, the degree of the node z gives the edge connectivity of the 2-node graph that is propagated back up the bigger super graphs to G, and hence the edge connectivity of the complete graph may be computed.

OUTPUTS OF THE ALGORITHM

M (Number of Edges)	EC(Edge Connectivity)	CG(Critical Edge Connectivity)	D(Degree)
40	16	197	4
80	16	215	8
90	17	242	9
120	17	255	12
150	17	246	15
185	16	327	18
220	17	259	22
245	17	288	24
270	15	248	27
295	17	251	29
330	15	200	33
355	15	203	35
370	16	179	37
390	16	317	39
400	17	345	40

GRAPHICAL OBSERVATIONS AND EXPLANATION

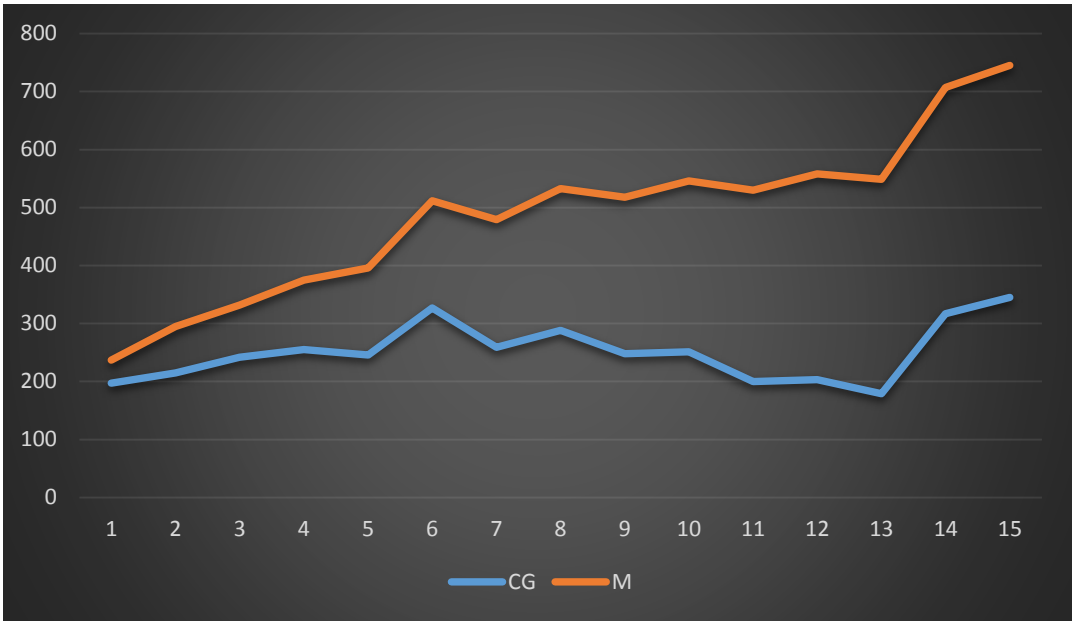
- Edge Connectivity versus Number of Links:



EXPLANATION:-

- The graph of edge connectivity is a rising curve that fluctuates after a rise, which can be attributed to the randomness in the topologies in terms of the number of edges.
- For n = 20 nodes, the edge connectivity increases as the number of edges increases.

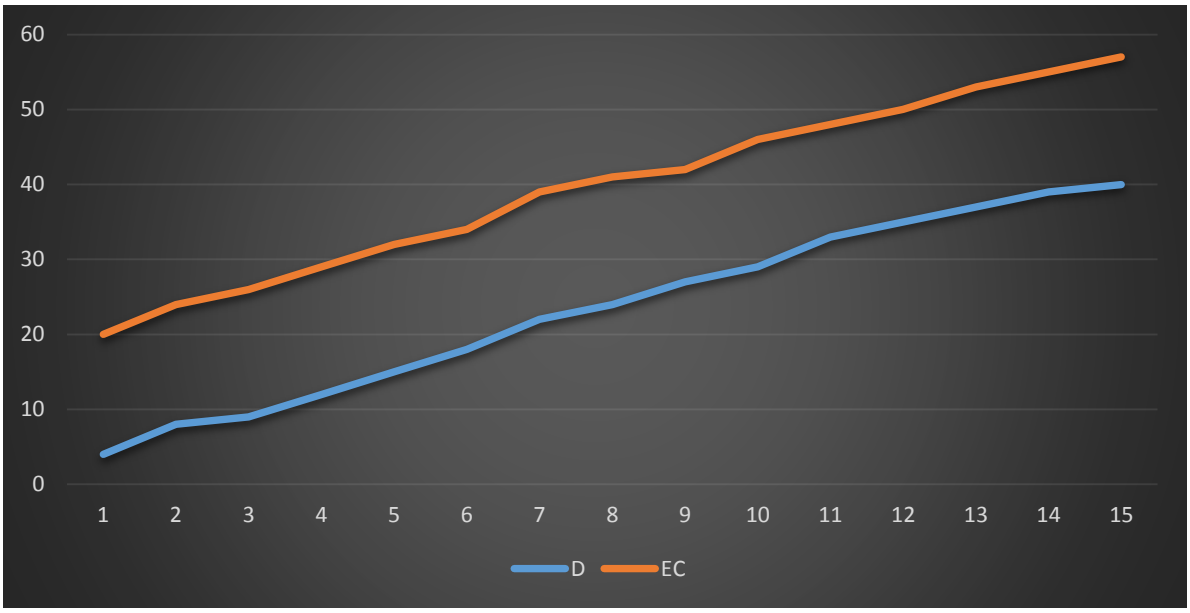
- Critical Edge Count versus Number of Links:



EXPLANATION:-

- The graph of critical edge count versus number of edges is a rising curve that dips at certain intervals, which again can be attributed to the randomness while assigning edges between nodes.

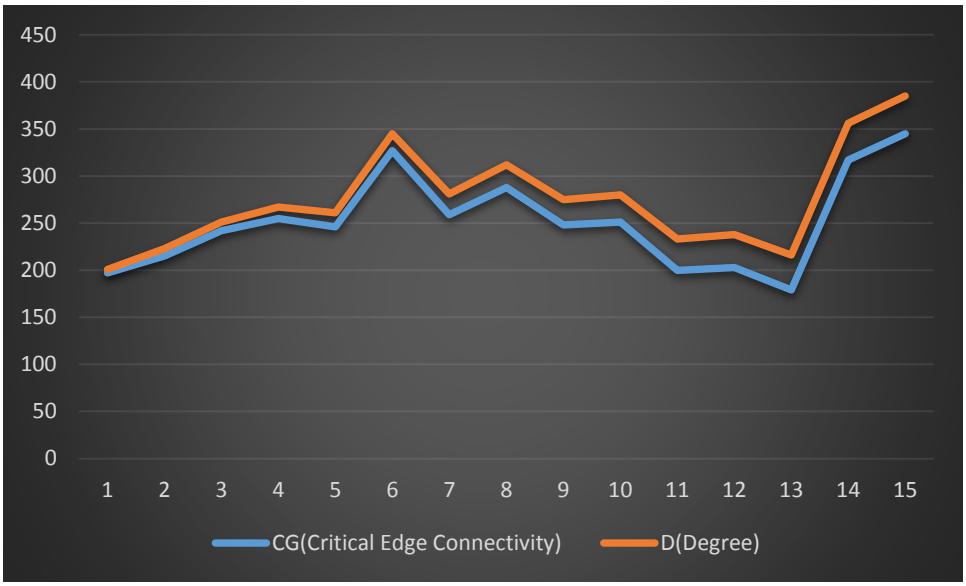
- Average Degree(d = (2*m)/n) vs Edge Connectivity



EXPLANATION:-

- The graph of average degree vs Edge Connectivity is a rising curve that fluctuates at certain intervals, which again can be attributed to the randomness while assigning edges between nodes.
- For n =20 nodes the degree increases as the number of edge increases.

- Critical Edge vs Average Degree:



EXPLANATION:-

- The graph of critical edge count versus average degree curve dips at certain intervals, which again can be attributed to the randomness while assigning edges between nodes.
- For n = 20 the critical edge connectivity dips and rises at certain point as the degree changes.

SOURCE CODE

Project2 Class:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Random;


public class Project2 {

	public static void main(String[] args) throws IOException {

		//number of nodes in the topology
		int n=20;

		//number of edges in the topology
		int m = 0;

		//array of size n to store each node's degree
		int sum1[]=new int[n];


		//Obtaining the value of m from the console
		String input = null;
		System.out.print("Enter the value of m =\t");

		try
		{
			BufferedReader bufferedReader = new BufferedReader(new
InputStreamReader(System.in));
			input = bufferedReader.readLine();
			m = Integer.parseInt(input);
	    }
		catch (NumberFormatException ex)
		{
			System.out.println("Invalid Input!!!");
		}


		//matrix to store the topology of network with b[i][j] signifying number of
edges from i to j
		int b[][]=new int[n][n];

		//b matrix is copied and stored in z matrix
		int z[][]=new int[n][n];


		//matrix to store the critical edges of network with b[i][j] signifying
number of edges from i to j
		int critical[][]=new int[n][n];

		//variable to store the number of degrees allocated so far in the matrix b
		int sum=0;
		//possible values to choose from for b matrix values
		int myArray[]=new int[n];
		for(int k=0;k<n;k++)
		{
			myArray[k]=k+1;
		}
```

```
            //creating object of Random library class
            Random generator = new Random();

            for(int i=0;i<n;i++)
            {
                  //System.out.println("In row "+ i);
                  for(int j=0;j<=i;j++)
                  {
                        if(sum<=2*m)
                        {
                              //No loops allowed hence the following statements for the
diagonal elements
                              if(i==j)
                            {
                              b[i][j]=0;
                                b[j][i]=0;
                            }
                              else
                            {
                                    //generating random value from myArray for b[i][j]
                                    b[i][j] = generator.nextInt(myArray.length);
                                    //allocating to b[i][j] only if available number of
degrees is greater than number of degs allocated
                                    if(2*b[i][j]<= 2*m-sum)
                                {
                                        sum+=2*b[i][j];
                                        b[j][i]=b[i][j];
                                }
                                    else
                                {
                                    b[i][j]=b[j][i]=1;
                                  }
                            }

                        }

                  }
             }

            //printing the number of edges in the graph
            System.out.println("The number of edges in the graph = "+ sum/2);

            //printing the number of degrees in the graph
            System.out.println("The sum of degrees = "+ sum);

            //printing out the adjacency matrix b
            System.out.println("The adjacency matrix = ");
            for(int x=0;x<n;x++)
            {
                  for(int y=0;y<n;y++)
                  {
                        System.out.print(b[x][y]);
                        System.out.print("\t");
                  }
            System.out.println("\n");
            }

            //picking a random vertex from the available nodes
            int randomVertex =  (int)(Math.random() * ((n-1) + 1));
            //System.out.println("Calling the Nagamochi's function");
            Nagamochi nagamochi =new Nagamochi();
          int minDeg=999;
```

```
            //finding the minimum degree of the topology
            for(int i=0;i<n;i++)
            {
                    for(int j=0;j<n;j++)
                  {
                          sum1[i]+=b[i][j];
                    }
                    // System.out.println("Row ="+ i + "Sum="+ sum1[i]);
                    if((sum1[i]<minDeg && sum1[i]!=0))
                    {
                        minDeg=sum1[i];
                    }
            }
            System.out.println("The original min degree of the graph is " + minDeg);
            //calling a method of Nagamochi class by passing the matrix b, number of
nodes n and randomVertex as parameters
            //Receiving edge connectivity of recursively called contracted topologies
            int edgeConn=nagamochi.determineAdjacency(b,n,randomVertex);
            System.out.println("Receiving final output");
            //Comparing min Degree with edge connectivity of contracted graph
            edgeConn=Math.min(minDeg,edgeConn);
            System.out.println("The edge connectivity of the graph is " + edgeConn);
          //determining critical edge count of the topology
            int lambda=999;
            int criticalCount=0;
            for(int i=0;i<n;i++)
          {
                    for(int j=0;j<i;j++)
                  {
                    z[i][j]=b[i][j];
                    if(z[i][j]!=0)
                    {
                        //removing an edge, and checking for edge connectivity of
resultant graph
                          z[i][j]-=1;
                        lambda=nagamochi.determineAdjacency(z,n,randomVertex);
                              //if edge connectivity of resultant graph is lesser, then
all the edges between this i,j pair are critical edges
                             if(lambda<edgeConn)
                             {
                                    //incrementing critical edge count
                              criticalCount+=b[i][j];
                             }
                    }
                          critical[i][j]=b[i][j];
                          z[i][j]=b[i][j];
                  }
             }
            //printing out the number of edges, edge connectivity and critical edge count
of the network topologies
            System.out.println("No of edges = " + (sum/2) + "\t" + "Edge Connectivity= "
+ edgeConn +"\t" + "Critical Edge Count= " + criticalCount+"\t" + "Average Degree="
+(2*m)/n);
            }
}
```

Nagamochi Class :

```
public class Nagamochi {

	//static variable to hold the number of times the network has been contracted
	static int count=0;


	public Nagamochi() {

		//increments each time this method is called by main or recursively
		count++;
	}

	public int determineAdjacency(int b[][],int n,int randomVertex)
	{
		//System.out.println("In nagamochi function");
		int mat[][]=new int[n][n];

		//variable to store 2nd vertex to contract with random vertex
		int second=0;
		int max=0;
		int edgeConnectivity = 0;
		int[] sum=new int[n];
		for(int j=0;j<n; j++)
		{
			{
				if(b[randomVertex][j]>max)
				{
					max=b[randomVertex][j];
					second=j;
				}
			}

		}

		//System.out.println("The first node selected is " + randomVertex);
	   //System.out.println("The second node selected is " + second);

	   //System.out.println("The max degree is " + max);
	  for(int i=0;i<n;i++)
	  {
		for(int j=0;j<=i;j++)
		{
				if(i==j)
				{
				  mat[i][j]=0;
				}
			else
				if((j==second) || (i==second))
				 {
					mat[i][j]=0;
				 }
			else
			   if((i==randomVertex)  || (j==randomVertex))
			   {
					mat[i][j]=b[randomVertex][j]+b[second][j];
				 }
			else
			{
				 mat[i][j]=b[i][j];
			}
```

```
                mat[j][i]=mat[i][j];
            }
        }

        //System.out.println("Printing mat matrix");
        for(int i=0;i<n;i++)
        {
            for(int j=0;j<n;j++)
            {
                //System.out.print(mat[i][j]+ "\t");
            }

          System.out.println("\n");
        }

        //finding min degree for the contracted matrix
        int minDegree=999;
        for(int i=0;i<n;i++)
        {
                for(int j=0;j<n;j++)
            {
                    sum[i]+=mat[i][j];
            }
              //System.out.println("Row ="+ i + "Sum="+ sum[i]);
              if((sum[i]<minDegree && sum[i]!=0))
            {
                    minDegree=sum[i];
            }

        }
        // System.out.println("The min degree for count =" +count + "is = " + minDeg);
        Nagamochi nagamochi;
        //contracting n-2 times by recursively calling Nagamochi method
        if(count<(n-1))
        {

            nagamochi=new Nagamochi();

            //determining edge connectivity at each stage of contraction

edgeConnectivity=Math.min(minDegree,nagamochi.determineAdjacency(mat,n,randomVertex));
            // System.out.println("The edge connectivity for count="+ (count-1) + "="+
edgeConn);

        }
        else

        //edge connectivity equals min degree when there are just two nodes in the
topology
        edgeConnectivity=minDegree;

        //returns contracted graph edge connectivity to the original graph
        return edgeConnectivity;
        }
}
```

Explanation of Modules

- Nagamochi Class :
  - It determines minimum degree of contracted graph and compares that with recursively computed value of edge connectivity of this graph to return the lesser of the two.
  - It determines the vertex that should be picked for contraction with randomly chosen vertex , based on maximum adjacency ordering

- Project2 Class :
  - It contains the main method.
  - Takes as input the number of edges m from 40 – 400.
  - Constructs the adjacency matrix for the network topology.
  - Determines the edge connectivity of the network topology.
  - Determines the critical link count of the network topology.

CONCLUSION

- We can conclude/infer from the results that as there is an increase in no. of edges in a graph, there is an increase in no. of critical edges as well, which could bring down the network. Edge connectivity also increases as there is an increase in no. of edges. The increment is not linear but it does increase with the number of parallel edges.

REFERENCES

- Nagamochi and Ibaraki Algorithm for minimum cut http://www.kurims.kyoto-u.ac.jp/~kyodo/kokyuroku/contents/pdf/0833-13.pdf
- Definition of terms was borrowed from www.wikipedia.org
- The formulae for computation of the edge connectivity and Maximum Adjacency Ordering (MA) and Degree of connectivity was referred from material posted by Dr Andras Farago in his lecture notes.