Project 2 Report

1. How Algorithm Works

For 5 nodes 10 edges, we assume nodes all work well, each edge is either Up or Down, and we will know the p for each edge. if the system is operational, it means there is no unconnected node.

Exhaustive enumeration:

List all possible states of the system.

Assign "up" and "down" system condition to each state

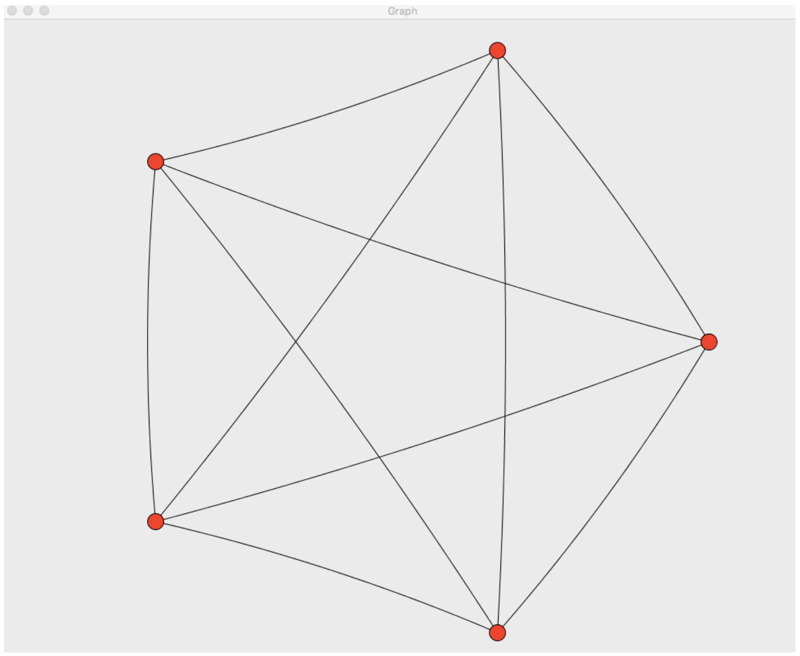
Reliability can be obtained by summing the probability of the "up" states

We are going to calculate all possible combinations, since 10 edges will have 2^{10} combinations, we store each system state in one int array: int[11] , first 10 integer will be represent the components`(edges`s) states, the last integer will represent system condition(Up or Down). All integer arrarys will be combined into one list : List<int[]>, thus if we want choose one random system condition, we just need to generate a random index of the List, then pick the correspond integer array out, that will be one random combination.

While calculating the System Reliability, we first check the last integer in the integer arrays to see if the System will be Up, if it is down, ignore it. We will check all 1024 integer arrays and pick those System Condition is Up, then find the p of each edges to figure out the final system reliabilities.

2. Pseudo Code

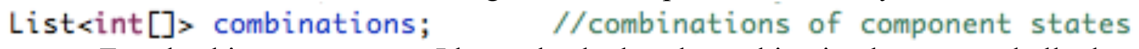
```
SetCombinations{
    List<int[11]> combinations;          //to store 1024 combinations plus
                                         //system condition
    int[] studentID;                     //store my student ID
    Graph graph;
    int d[10]                            //d[i] to store student ID digit

    setCombinations(List<int[11]> combinations){       //set 1024
                                                       //combinations
        for( i = 0; i < 10; i++){
            setCombin(i,0);                            // i decide how many
        }                                              //edges will be Up in
    }                                                  //one combination

    setCombin(int i, int startPoint){           //when edges are Up, get the
                                                //system and component state
        int[11]  combination from List<int[11]> combinations;
        if(i == 0){
            if(System is operational)
                combination[11] = 1;
                else
                        combination[11] = 0;
            store the result combination into the list;
            }
            for(j == startPoint; j < 10; j++){
                combination[j] = 1;
                setCombin(i-1, j+1);
            }
        }
}


CalculateReliability{
        for(every combination[11] in List<combination> == 1){    //for every
                                                                  // combination
                                                              //that system is UP
            result = 1;
            for(every combination[j]){
                if(combination[j] == 1)
                        result *= edge[j].p;
                else
                        result *= (1 - edge[j].p)
            }
        }

        Sum all results to get the System Reliability;

}
```



The Original Graph Generated

3. Checking correctness

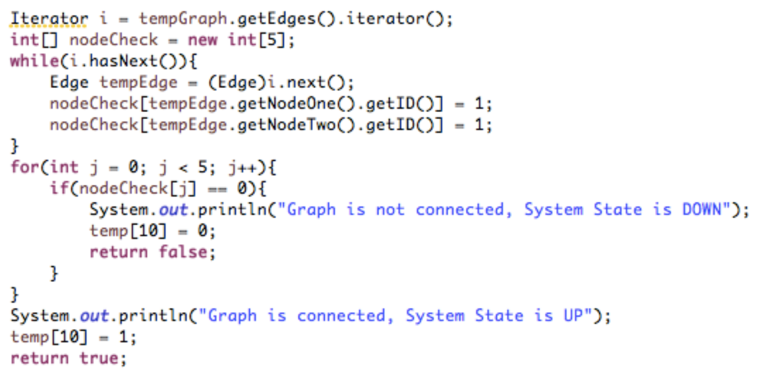
For each combination, an integer array has been created to store all components and System state, if component/system is UP, the represented int in the array will be 1 otherwise 0,each time their state changes, we will update the int array



```
List<int[]> combinations;        //combinations of component states
```

For checking correctness, I have checked each combination by traversal all edges to make sure every node in the graph is connected, means the System State is UP,then added all the UP combinations together via exhaustive enumeration to calculate the final System Reliability.

Figure 1 illustrate the method to traversal edges in the graph of one combination, if the System is Up, return true. If System is Down, return false.



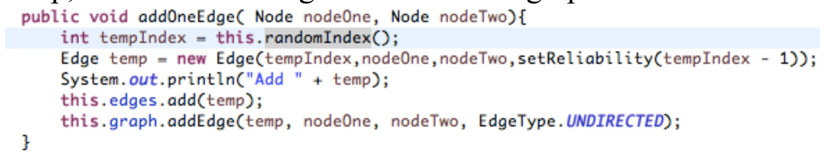
```
Iterator i = tempGraph.getEdges().iterator();
int[] nodeCheck = new int[5];
while(i.hasNext()){
    Edge tempEdge = (Edge)i.next();
    nodeCheck[tempEdge.getNodeOne().getID()] = 1;
    nodeCheck[tempEdge.getNodeTwo().getID()] = 1;
}
for(int j = 0; j < 5; j++){
    if(nodeCheck[j] == 0){
        System.out.println("Graph is not connected, System State is DOWN");
        temp[10] = 0;
        return false;
    }
}
System.out.println("Graph is connected, System State is UP");
temp[10] = 1;
return true;
```

Figure 1

3. ReadMe file

4. Calculate Component Reliability

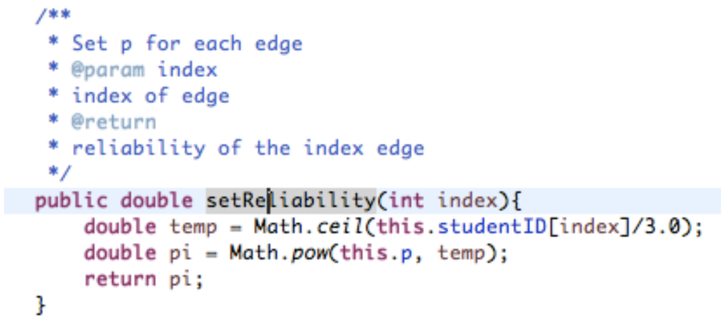
Using randomIndex() we get a non-duplicate index for one edge, with this index we calculate the p, and create the edge added into the graph.



```
public void addOneEdge( Node nodeOne, Node nodeTwo){
    int tempIndex = this.randomIndex();
    Edge temp = new Edge(tempIndex,nodeOne,nodeTwo,setReliability(tempIndex - 1));
    System.out.println("Add " + temp);
    this.edges.add(temp);
    this.graph.addEdge(temp, nodeOne, nodeTwo, EdgeType.UNDIRECTED);
}
```

Figure 2

Figure 3 shows how to calculate the p for each edge. My Student ID is stored in the studentID[] array. $p_i = p^{\lceil d_i/3 \rceil}$.



```
/**
 * Set p for each edge
 * @param index
 * index of edge
 * @return
 * reliability of the index edge
 */
public double setReliability(int index){
    double temp = Math.ceil(this.studentID[index]/3.0);
    double pi = Math.pow(this.p, temp);
    return pi;
}
```

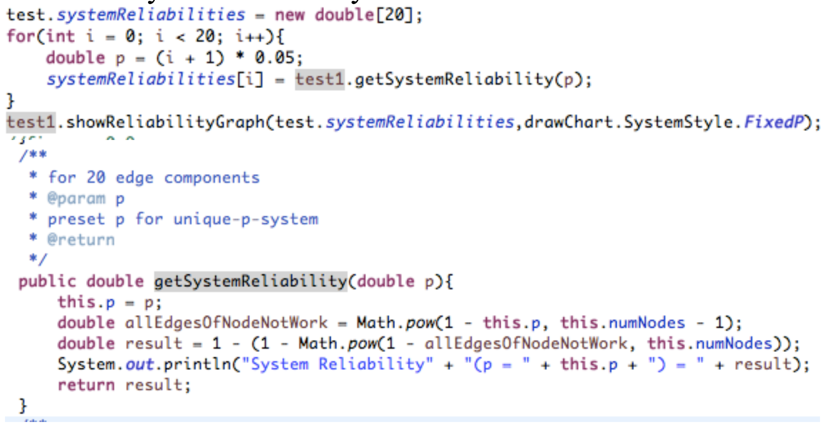
Figure 3

5. Run for different value of p

Let p run over [0.05,1] in steps of 0.05, show how the obtained network reliability values depend on p, since all edges are working, System Reliability can be calculated as follow:

$$
\begin{aligned}
\text{Rsystem} &= 1 - \text{anyNodeNotWork} \\
&= 1 - (1 - \text{allNodeWork}) \\
&= 1 - (1 - \text{oneNodeWork}^{\text{numOfNodes}}) \\
&= 1 - (1 - (1 - \text{allEdgesOfNodeNotWork})^{\text{numOfNodes}}) \\
&= 1 - (1 - (1 - (1 - p)^{\text{numOfNodes - 1}})^{\text{numOfNodes}})
\end{aligned}
$$

Figure 4 is the method using for loop to get all p value from 0.05 to 1, and calculate their relative System Reliability.



```
test.systemReliabilities = new double[20];
for(int i = 0; i < 20; i++){
    double p = (i + 1) * 0.05;
    systemReliabilities[i] = test1.getSystemReliability(p);
}
test1.showReliabilityGraph(test.systemReliabilities,drawChart.SystemStyle.FixedP);

 /**
  * for 20 edge components
  * @param p
  * preset p for unique-p-system
  * @return
  */
 public double getSystemReliability(double p){
     this.p = p;
     double allEdgesOfNodeNotWork = Math.pow(1 - this.p, this.numNodes - 1);
     double result = 1 - (1 - Math.pow(1 - allEdgesOfNodeNotWork, this.numNodes));
     System.out.println("System Reliability" + "(p = " + this.p + ") = " + result);
     return result;
 }
```

Figure 4

Figure 5 shows the result graphically.

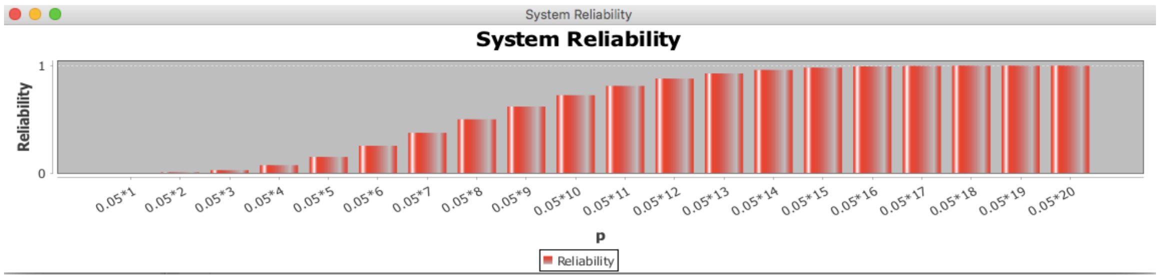


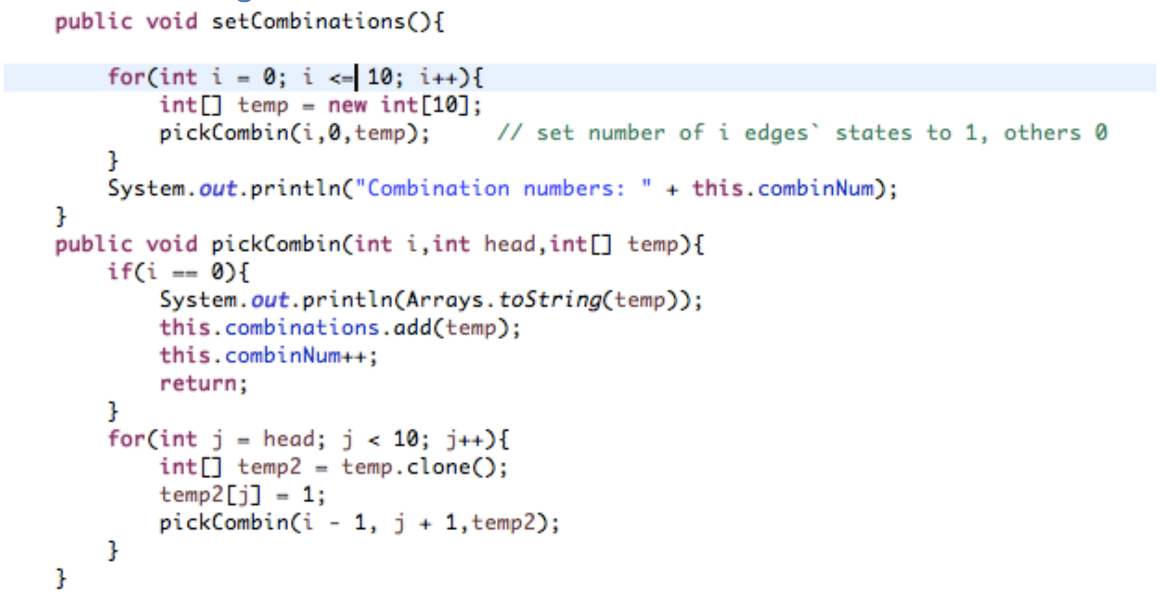
Figure 5

6. Fix p Parameter at 0.9

Then fix p = 0.9, among 2^{10} combination states pick k of combinations randomly and flip the corresponding system condition, show in diagram how the reliability of the system changes due to this alteration. Show in diagram how the change depends on k range 0,1,2,3….20, run several experiments for each k, and average them out. Give several paragraph explanation.

First, calculate all 1024 possible original combinations.

Combination algorithm:



```
public void setCombinations(){

    for(int i = 0; i <= 10; i++){
        int[] temp = new int[10];
        pickCombin(i,0,temp);     // set number of i edges` states to 1, others 0
    }
    System.out.println("Combination numbers: " + this.combinNum);
}
public void pickCombin(int i,int head,int[] temp){
    if(i == 0){
        System.out.println(Arrays.toString(temp));
        this.combinations.add(temp);
        this.combinNum++;
        return;
    }
    for(int j = head; j < 10; j++){
        int[] temp2 = temp.clone();
        temp2[j] = 1;
        pickCombin(i - 1, j + 1,temp2);
    }
}
```

Figure 6

We create a temporary array ‘temp[]’ which stores all outputs one by one. The idea is to start from first index (index = 0) in temp[], one by one fix elements at this index and recur for remaining indexes. We first fix 1 number in data[] as 1, then return and recur for remaining indexes, then we fix 2 number in data[] as 1, return and recur. Finally, we will fix 1 for all remaining indexes. Following diagram shows recursion tree, number in the block represent the index of array to be set as 1.

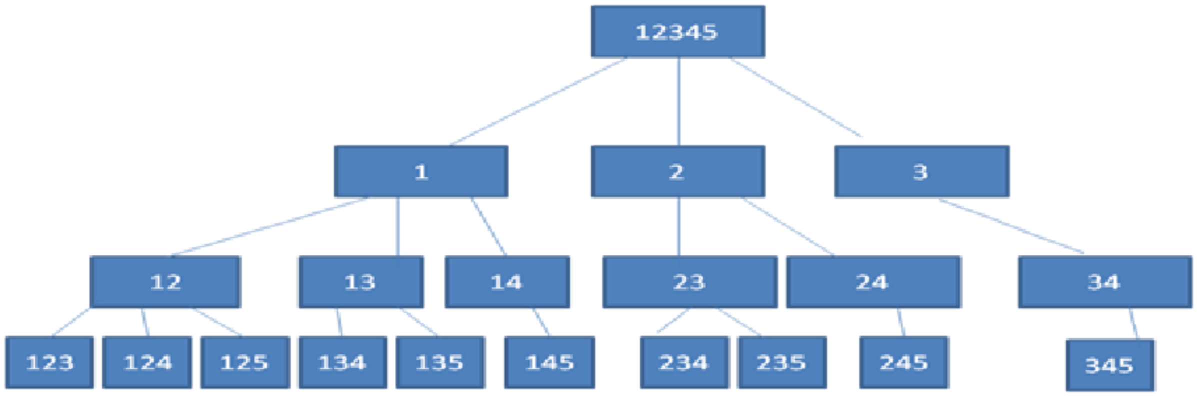
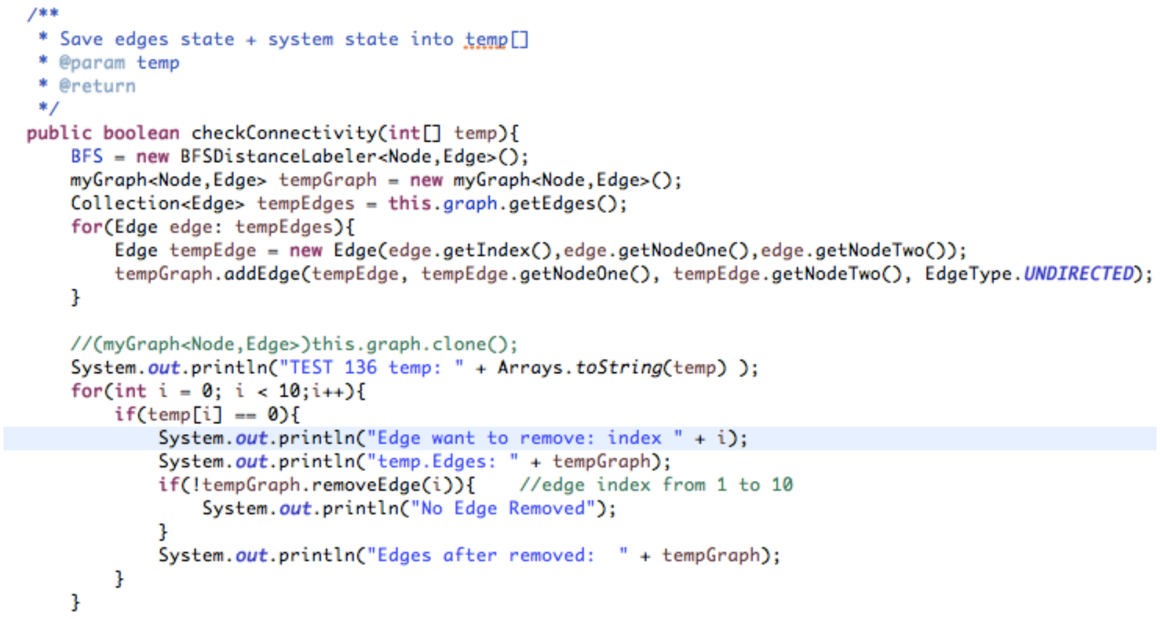


Figure 7

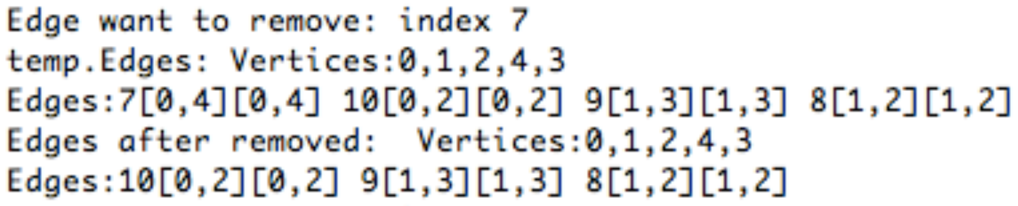
The result format is as Figure 8, after we get all edges` condition in one combination, we could calculate the System Condition by the method checkConnectivity() to check if the System is operational, if yes , we add 1 to the end of the array, else add 0 as Figure 8



```
/**
 * Save edges state + system state into temp[]
 * @param temp
 * @return
 */
public boolean checkConnectivity(int[] temp){
    BFS = new BFSDistanceLabeler<Node,Edge>();
    myGraph<Node,Edge> tempGraph = new myGraph<Node,Edge>();
    Collection<Edge> tempEdges = this.graph.getEdges();
    for(Edge edge: tempEdges){
        Edge tempEdge = new Edge(edge.getIndex(),edge.getNodeOne(),edge.getNodeTwo());
        tempGraph.addEdge(tempEdge, tempEdge.getNodeOne(), tempEdge.getNodeTwo(), EdgeType.UNDIRECTED);
    }

    //(myGraph<Node,Edge>)this.graph.clone();
    System.out.println("TEST 136 temp: " + Arrays.toString(temp) );
    for(int i = 0; i < 10;i++){
        if(temp[i] == 0){
            System.out.println("Edge want to remove: index " + i);
            System.out.println("temp.Edges: " + tempGraph);
            if(!tempGraph.removeEdge(i)){    //edge index from 1 to 10
                System.out.println("No Edge Removed");
            }
            System.out.println("Edges after removed:  " + tempGraph);
        }
    }
```

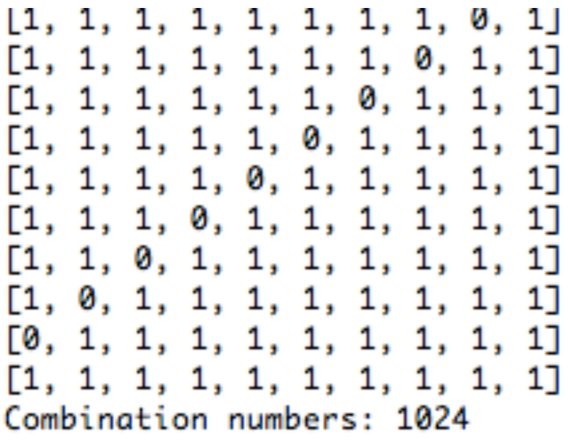
Figure 8

checkConnectivity(int[] temp){} will be used to check system state if components` state have changed, we first copy a new graph from the original graph, if an edge is DOWN(not working), we will delete the edge from the new graph, The running sample is as follow:



```
Edge want to remove: index 7
temp.Edges: Vertices:0,1,2,4,3
Edges:7[0,4][0,4] 10[0,2][0,2] 9[1,3][1,3] 8[1,2][1,2]
Edges after removed:  Vertices:0,1,2,4,3
Edges:10[0,2][0,2] 9[1,3][1,3] 8[1,2][1,2]
```

Edge with index 7 is not working, the current graph has edges 7, 10, 9 , 8, we are going to delete edge 7, after delete it, the graph contains edges 10, 9 ,8



```
[1, 1, 1, 1, 1, 1, 1, 1, 0, 1]
[1, 1, 1, 1, 1, 1, 1, 0, 1, 1]
[1, 1, 1, 1, 1, 1, 0, 1, 1, 1]
[1, 1, 1, 1, 1, 0, 1, 1, 1, 1]
[1, 1, 1, 1, 0, 1, 1, 1, 1, 1]
[1, 1, 1, 0, 1, 1, 1, 1, 1, 1]
[1, 1, 0, 1, 1, 1, 1, 1, 1, 1]
[1, 0, 1, 1, 1, 1, 1, 1, 1, 1]
[0, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Combination numbers: 1024
```

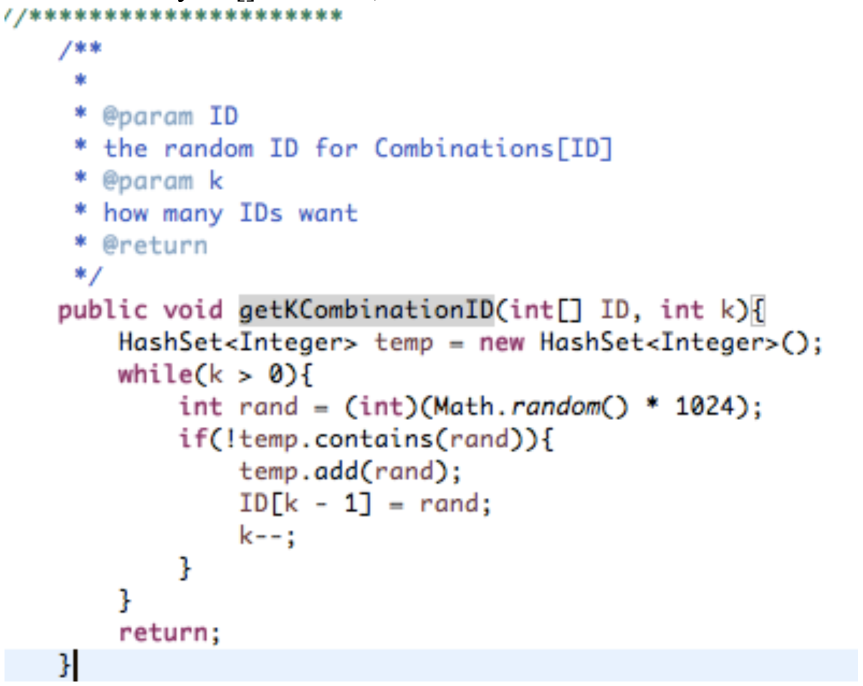
Figure 9

When generating the 1024 combinations, uses variable this.combinNum to count each combination we get for checking correctness, finally get this.combinNum == 1024, and list all of them with no error.

7. Pick k Combinations Randomly

The solution is that we generate k non-duplicate random index for the combination List, and get the k combination arrays.

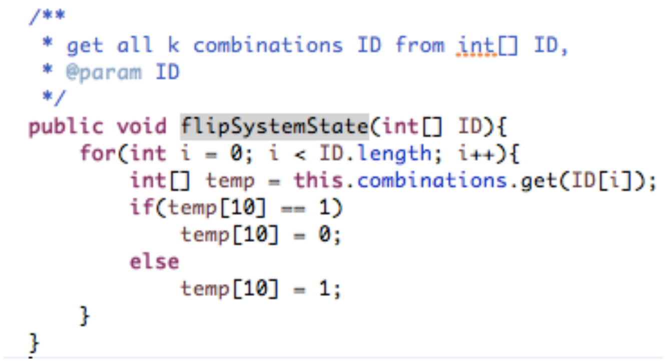
First, generate k random index in the range from 0 to 1023. And put all the k random index in one array ID[] of size k, return it:



```
//**********************
    /**
     *
     * @param ID
     * the random ID for Combinations[ID]
     * @param k
     * how many IDs want
     * @return
     */
    public void getKCombinationID(int[] ID, int k){
        HashSet<Integer> temp = new HashSet<Integer>();
        while(k > 0){
            int rand = (int)(Math.random() * 1024);
            if(!temp.contains(rand)){
                temp.add(rand);
                ID[k - 1] = rand;
                k--;
            }
        }
        return;
    }
```

Figure 10

Then use index in the ID[], we retrieve the System State arrays in the combination list, and flip the 11th integer(the system state integer) ,as Figure 11.



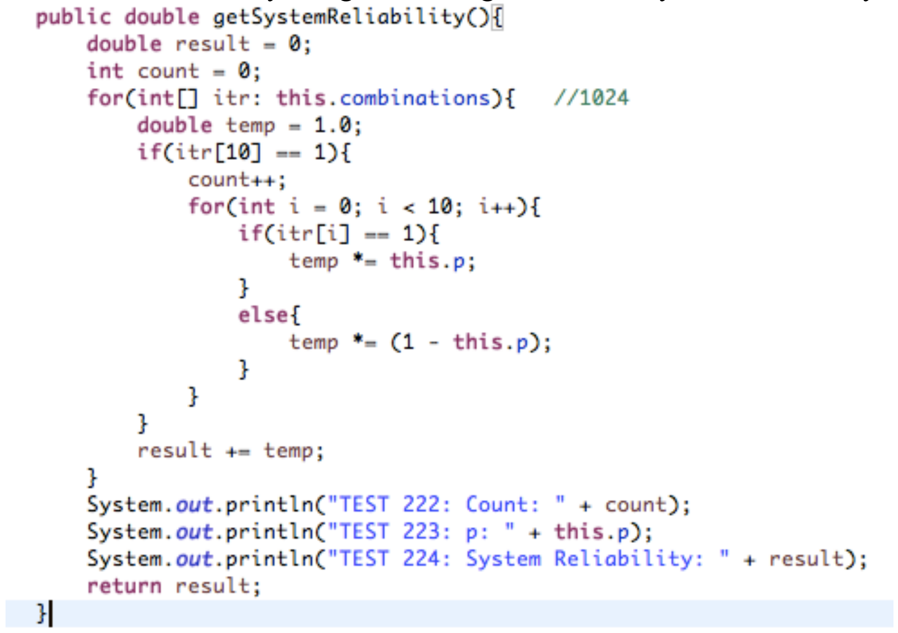
```
/**
 * get all k combinations ID from int[] ID,
 * @param ID
 */
public void flipSystemState(int[] ID){
    for(int i = 0; i < ID.length; i++){
        int[] temp = this.combinations.get(ID[i]);
        if(temp[10] == 1)
            temp[10] = 0;
        else
            temp[10] = 1;
    }
}
```

Figure 11

Then after flip k System states , we get the new combination list.
The following method will calculate the system reliability:

this.combinations will store all 1024 possible combinations, get each system state via int[] itr, itr[10] will indicate if system is UP(1) or DOWN(0). By Exhaustive

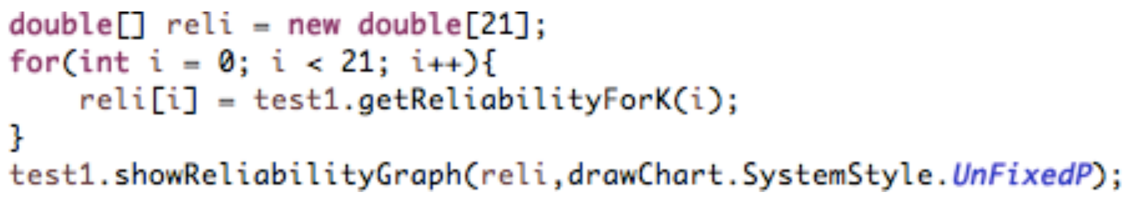
Enumeration method, multiply all components reliability within one combinations, then add all combinations` reliability altogether, we get the final System Reliability.



```
public double getSystemReliability(){
    double result = 0;
    int count = 0;
    for(int[] itr: this.combinations){   //1024
        double temp = 1.0;
        if(itr[10] == 1){
            count++;
            for(int i = 0; i < 10; i++){
                if(itr[i] == 1){
                    temp *= this.p;
                }
                else{
                    temp *= (1 - this.p);
                }
            }
        }
        result += temp;
    }
    System.out.println("TEST 222: Count: " + count);
    System.out.println("TEST 223: p: " + this.p);
    System.out.println("TEST 224: System Reliability: " + result);
    return result;
}
```

Figure 12

Run the program from k = 0 to 20



```
double[] reli = new double[21];
for(int i = 0; i < 21; i++){
    reli[i] = test1.getReliabilityForK(i);
}
test1.showReliabilityGraph(reli,drawChart.SystemStyle.UnFixedP);
```

Figure 13

Fix p = 0.9, System Reliability depends on K as follows:

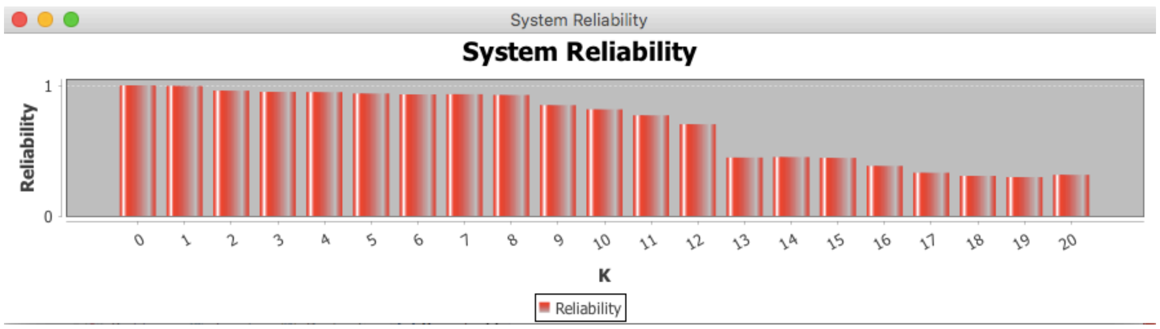
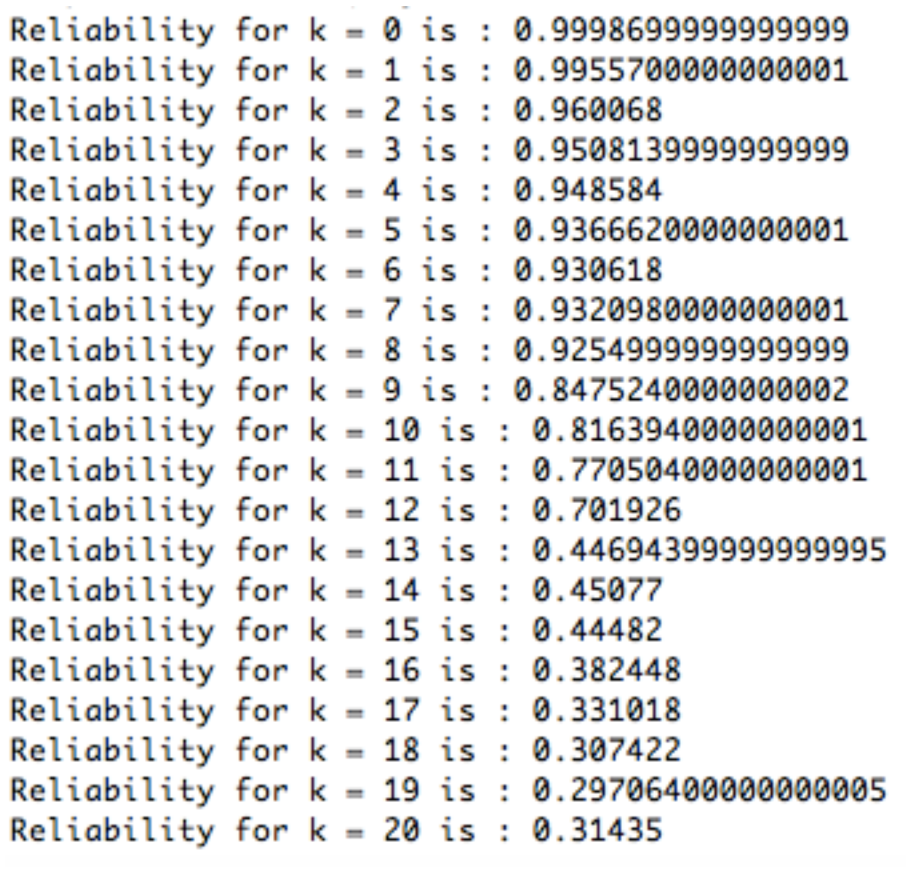


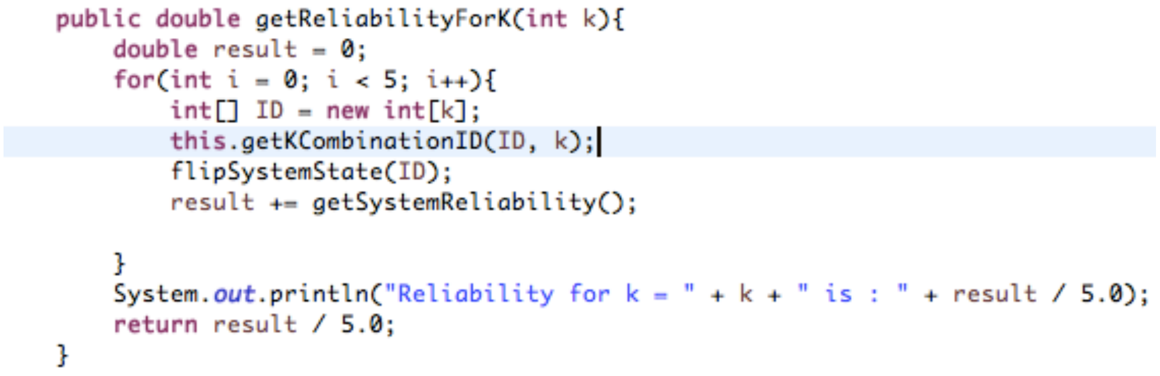
Figure 14



```
Reliability for k = 0 is : 0.9998699999999999
Reliability for k = 1 is : 0.9955700000000001
Reliability for k = 2 is : 0.960068
Reliability for k = 3 is : 0.9508139999999999
Reliability for k = 4 is : 0.948584
Reliability for k = 5 is : 0.9366620000000001
Reliability for k = 6 is : 0.930618
Reliability for k = 7 is : 0.9320980000000001
Reliability for k = 8 is : 0.9254999999999999
Reliability for k = 9 is : 0.8475240000000002
Reliability for k = 10 is : 0.8163940000000001
Reliability for k = 11 is : 0.7705040000000001
Reliability for k = 12 is : 0.701926
Reliability for k = 13 is : 0.44694399999999995
Reliability for k = 14 is : 0.45077
Reliability for k = 15 is : 0.44482
Reliability for k = 16 is : 0.382448
Reliability for k = 17 is : 0.331018
Reliability for k = 18 is : 0.307422
Reliability for k = 19 is : 0.29706400000000005
Reliability for k = 20 is : 0.31435
```

Figure 14

For each k, calculate 5 times then average the result out:



```
public double getReliabilityForK(int k){
    double result = 0;
    for(int i = 0; i < 5; i++){
        int[] ID = new int[k];
        this.getKCombinationID(ID, k);
        flipSystemState(ID);
        result += getSystemReliability();

    }
    System.out.println("Reliability for k = " + k + " is : " + result / 5.0);
    return result / 5.0;
}
```

Figure 15

ReadMe

Project is written by Java
JUNG is used to draw graphs, download from : http://jung.sourceforge.net/
JFree is used to draw charts, download from: http://www.jfree.org/jfreechart/

Source Code

```java
package TEST;
import NetworkElements.*;
import Drawing.drawChart;

import java.awt.Dimension;
import java.text.DecimalFormat;
import java.util.*;
import java.util.concurrent.TimeUnit;

import javax.swing.JFrame;

import edu.uci.ics.jung.algorithms.layout.CircleLayout;
import edu.uci.ics.jung.graph.*;
import edu.uci.ics.jung.graph.util.EdgeType;
import edu.uci.ics.jung.visualization.*;
import
edu.uci.ics.jung.visualization.renderers.Renderer.VertexLabel.Pos
ition;
import
edu.uci.ics.jung.algorithms.shortestpath.BFSDistanceLabeler;

public class test {
	static double systemReliabilities[];
	int numNodes;
	int numEdges;
	int combinNum;
	double p;				//edge reliability
	myGraph<Node,Edge> graph;
	ArrayList<Integer> indexList;		//d[i] to calculate
p[i]
	List<Node> nodes;
	List<Edge> edges;
	int[] studentID;
	drawChart barChart;	//bar chart to show system
reliabilities depends on p
	List<int[]> combinations;	//combinations of component
```

```
states
	int[] systemState;	//1024 system state;

	BFSDistanceLabeler<Node,Edge> BFS;

	public test(int num, double p){
		numNodes = num;
		this.p = p;
		numEdges = numNodes * (numNodes - 1);
		studentID = new int[]{2,0,2,1,2,2,1,1,3,7};
		graph = new myGraph<Node, Edge>();
		indexList = new ArrayList<Integer>();
		indexList.addAll(Arrays.asList(1,2,3,4,5,6,7,8,9,10));
//index from 1 to 10
		combinations = new ArrayList<int[]>();
		Collections.shuffle(indexList);
		nodes = new ArrayList<Node>();
		edges = new ArrayList<Edge>();
		this.addNumberOfNodes();
		this.addEdges();
		System.out.println(this.graph.getEdgeCount() + "
Undirected Edges Created");
	}
	//di
	public int getIDDigit(int index){
		return studentID[index];
	}

	public int randomIndex(){
		int index =
this.indexList.remove(this.indexList.size() - 1);
		return index;
	}

	//start from 0
	public void addNumberOfNodes(){
		for(int i = 0; i < this.numNodes; i++){
			Node temp = new Node(i);
			//this.graph.addVertex(temp);	if edges be
added, nodes will be added to graph automatically
			this.nodes.add(temp);
		}
		System.out.println(this.numNodes + " Nodes Created");
	}
	/**
	 * Add one edge to this.edges & this.graph
	 * @param nodeOne
	 * @param nodeTwo
	 */
	public void addOneEdge( Node nodeOne, Node nodeTwo){
		int tempIndex = this.randomIndex();
		Edge temp = new 
Edge(tempIndex,nodeOne,nodeTwo,setReliability(tempIndex - 1));
		System.out.println("Add " + temp);
		this.edges.add(temp);
		this.graph.addEdge(temp, nodeOne, nodeTwo, 
EdgeType.UNDIRECTED);
	}

	/**
	 * loop to add all edges
	 */
	public void addEdges(){
		for(int i = 0; i < this.numNodes; i++){
			for(int j = i + 1; j < this.numNodes; j++){
				this.addOneEdge(this.nodes.get(i), 
this.nodes.get(j));
			}
		}
	}

	/**
	 * Set p for each edge
	 * @param index
	 * index of edge
	 * @return
	 * reliability of the index edge
	 */
	public double setReliability(int index){
		double temp = Math.ceil(this.studentID[index]/3.0);
		double pi = Math.pow(this.p, temp);
		return pi;
	}

	/**
	 * Save edges state + system state into temp[]
	 * @param temp
	 * @return
```

```
      */
     public boolean checkConnectivity(int[] temp){
          BFS = new BFSDistanceLabeler<Node,Edge>();
          myGraph<Node,Edge> tempGraph = new
myGraph<Node,Edge>();
          Collection<Edge> tempEdges = this.graph.getEdges();
          for(Edge edge: tempEdges){
               Edge tempEdge = new
Edge(edge.getIndex(),edge.getNodeOne(),edge.getNodeTwo());
               tempGraph.addEdge(tempEdge,
tempEdge.getNodeOne(), tempEdge.getNodeTwo(),
EdgeType.UNDIRECTED);
          }

          for(int i = 0; i < 10;i++){
               if(temp[i] == 0){
                    if(!tempGraph.removeEdge(i)){    //edge
index from 1 to 10
                         //System.out.println("No Edge
Removed");
                    }
               }
          }
          Iterator i = tempGraph.getEdges().iterator();
          int[] nodeCheck = new int[5];
          while(i.hasNext()){
               Edge tempEdge = (Edge)i.next();
               nodeCheck[tempEdge.getNodeOne().getID()] = 1;
               nodeCheck[tempEdge.getNodeTwo().getID()] = 1;
          }
          for(int j = 0; j < 5; j++){
               if(nodeCheck[j] == 0){
                    System.out.println("Graph is not connected,
System State is DOWN");
                    temp[10] = 0;
                    return false;
               }
          }
          System.out.println("Graph is connected, System State
is UP");
          temp[10] = 1;
          return true;
     }
     /**
```

```
	 * for 20 edge components
	 * @param p
	 * preset p for unique-p-system
	 * @return
	 */
	public double getSystemReliability(double p){
		this.p = p;
		double allEdgesOfNodeNotWork = Math.pow(1 - this.p, 
this.numNodes - 1);
		double result = 1 - (1 - Math.pow(1 - 
allEdgesOfNodeNotWork, this.numNodes));
		System.out.println("System Reliability" + "(p = " + 
this.p + ") = " + result);
		return result;
	}
	/**
	 * get System Reliability of Non-unique-p-system
	 * @return
	 * (double)System Reliability
	 */
	public double getSystemReliability(){
		double result = 0;
		DecimalFormat numberFormat = new 
DecimalFormat("#.00000");
		int count = 0;
		for(int[] itr: this.combinations){   //1024
			double temp = 1.0;
			if(itr[10] == 1){
				count++;
				for(int i = 0; i < 10; i++){
					if(itr[i] == 1){
						temp *= this.p;
					}
					else{
						temp *= (1 - this.p);
					}
				}
			}
			else{
				temp = 0;
			}


			temp = 
Double.parseDouble(numberFormat.format(temp));
                result += temp;
                result =
Double.parseDouble(numberFormat.format(result));
                //System.out.println("Current Result: " +
result);

           }
           //System.out.println("TEST 222: Count: " + count);
           //System.out.println("TEST 223: p: " + this.p);
           //System.out.println("TEST 224: System Reliability: "
+ result);
           return result;
      }

      public void showGraph(){
           CircleLayout temp = new CircleLayout(this.graph);
           temp.setRadius(380);
           BasicVisualizationServer vs = new
BasicVisualizationServer(temp,new Dimension(1000,800));

      vs.getRenderer().getVertexLabelRenderer().setPosition(Posit
ion.CNTR);
          JFrame frame = new JFrame("Graph");
          frame.getContentPane().add(vs);
          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
          frame.pack();
          frame.setVisible(true);
      }

      public void showReliabilityGraph(double[]
systemRel,drawChart.SystemStyle systemSty ){
           barChart = new drawChart("System Reliability",
systemRel, systemSty);
           barChart.centerChart();
           barChart.setVisible(true);
      }
//Combination Generating Module
      /**
       * set combinations to Combinations<int[]>, each int[] will
represent one system state
       * for each component 1 flip , 0 remain
       */
      public void setCombinations(){
```

```
		for(int i = 0; i <= 10; i++){
			int[] temp = new int[11];
			pickCombin(i,0,temp);	// set number of i 
edges` states to 1, others 0
		}
	}
	public void pickCombin(int i,int head,int[] temp){
		if(i == 0){
			if(checkConnectivity(temp) == true){
				temp[10] = 1;
			}
			else{
				temp[10] = 0;
			}
			this.combinations.add(temp);
			this.combinNum++;
			return;
		}
		for(int j = head; j < 10; j++){
			int[] temp2 = temp.clone();
			temp2[j] = 1;
			pickCombin(i - 1, j + 1,temp2);
		}
	}
//*********************
	/**
	 * 
	 * @param ID
	 * the random ID for Combinations[ID]
	 * @param k
	 * how many IDs want
	 * @return
	 */
	public void getKCombinationID(int[] ID, int k){
		HashSet<Integer> temp = new HashSet<Integer>();
		while(k > 0){
			int rand = (int)(Math.random() * 1024);
			if(!temp.contains(rand)){
				temp.add(rand);
				ID[k - 1] = rand;
				k--;
			}
		}
```

```
            return;
        }
        /**
         *
         * @param k
         * get System Reliability with K, repeatedly 5 times and
average them
         * @return result
         * return averaged reliability
         */
        public double getReliabilityForK(int k){
            double result = 0;
            for(int i = 0; i < 5; i++){
                int[] ID = new int[k];
                this.getKCombinationID(ID, k);
                flipSystemState(ID);
                result += getSystemReliability();

            }
            System.out.println("Reliability for k = " + k + " is :
" + result / 5.0);
            return result / 5.0;
        }

        /**
         * get all k combinations ID from int[] ID,
         * @param ID
         */
        public void flipSystemState(int[] ID){
            for(int i = 0; i < ID.length; i++){
                int[] temp = this.combinations.get(ID[i]);  //
get int[] components state for one system state
                if(temp[10] == 1)
                    temp[10] = 0;
                else
                    temp[10] = 1;
            }
        }

        public static void main(String args[]){
            test test1 = new test(5,0.85);  //(numOfNodes,p) =
(5,2)
            //test1.showGraph();
            test.systemReliabilities = new double[20];
```

```
		for(int i = 0; i < 20; i++){
			double p = (i + 1) * 0.05;
			systemReliabilities[i] = 
test1.getSystemReliability(p);
		}
		
	test1.showReliabilityGraph(test.systemReliabilities,drawCha
rt.SystemStyle.FixedP);
		//fix p = 0.9
		test1.getSystemReliability(0.9);
		//pick k combinations randomly and fix the 
corresponding system condition
		test1.setCombinations();
		//test1.checkConnectivity(test1.graph);
		//System.out.println("System Reliability: " + 
test1.getSystemReliability());
		  //  pick k of the combinations, flip the system 
condition, then calculate the System Reliability
		double[] reli = new double[21];
		for(int i = 0; i < 21; i++){
			reli[i] = test1.getReliabilityForK(i);
		}
		
	test1.showReliabilityGraph(reli,drawChart.SystemStyle.UnFix
edP);
	}
}
```