CS 6385
Algorithmic Aspects of Telecommunication Networks
Spring 2012

Project #2 Report
Due on Wednesday, March 20, 2012

Instructor: Andras Farago

Emrah Cem (exc103320)
Contents

1. Introduction 3
2. Problem Description 3
3. Solution Approach 4
4. Implementation Details 6
   4.1 Technologies Used in Implementation 6
   4.2 Implementation Detail 6
5. Instructions for Users 6
6. Experimental Results 7
7. Conclusion 9

Appendix 11

1. ApplicationFrame.java 11
2. ComputingThread.java 17
3. CustomVertex.java 18
4. CustomVertexFactory.java 18
5. DotFileCreator.java 19
6. IGraphExperiment.java 20
7. NagamochiIbarakiExperiment.java 21
8. RandGraphGenerator.java 23
9. IOrdering.java 24
10. MAOrdering.java 24
11. NagamochiIbaraki.java 25
12. IncrementalFreq.java 26
13. IncrementalStats.java 32
14. MedianStats.java 35
1. Introduction

In this project, the famous algorithm found by Hiroshi Nagomachi and Toshihide Ibaraki [1], known as Nagomachi-Ibaraki algorithm, has been implemented with a user friendly GUI. Firstly, the relationship between edge connectivity and average node degree in a graph has been analyzed. Moreover, effect of number of nodes (with fixed edge density) on edge connectivity has been presented. Lastly, the relationship between critical edges and average degree has been analyzed.

In the next section, the description of the problem will be given, including the definition of edge connectivity and critical edge. In section 3, pseudocode of the algorithm is given and explained. Then, in section 4, technologies used in implementation is given, as well as the explanation of the algorithm in detail. In section 5, basic instructions for users is given. Then, in section 6, experimental results are presented and finally in section 7, report is concluded.

2. Problem Description

In this section, formal definition of the problem is given, as well as some diagrams to clarify the problem. Lets first start with the definition of edge connectivity.

**Definition.** Edge connectivity between nodes x and y, $\lambda(x,y)$, is the minimum number of edges that is needed to be deleted to disconnect x and y, where $x \neq y$. Edge connectivity of a graph, $\lambda(G)$, is the minimum number of edges that is needed to be deleted to disconnect the graph.

Note that there is the following relation between $\lambda(x,y)$ and $\lambda(G)$,

$$\lambda(G) = \min_{x,y} \lambda(x,y) \quad (1)$$

**Definition.** An edge is called a critical edge if its removal from the graph decreases edge connectivity of the graph. In other words, an edge is critical if $\lambda(G - e) < \lambda(G)$, where $G - e$ represents the graph G with edge e is removed.

Figure 1 shows a sample graph with edge connectivity 2. Note that we can not disconnect the graph by removing a single edge, but if we remove the edges that connect node3 to node2 and node4. Then graph will be disconnected.

![Figure 1: An example graph with edge connectivity 2.](image)

Additionally, in Figure 2, you can see the critical edges of the same graph.
3. Solution Approach

Nagamochi-Ibaraki algorithm [1] makes use the maximum adjacency (MA) ordering of the vertices while computing the edge connectivity. So, first, the definition of MA ordering should be clear before going into the details of the algorithm.

**Definition.** An ordering \( v_1, v_2, \ldots, v_n \) of vertices is called an MA ordering if an arbitrary vertex is chosen as \( v_1 \), and after choosing the first \( i \) vertices \( v_1, \ldots, v_i \), the \( (i+1) \)th vertex \( v_{i+1} \) is chosen from the vertices \( u \) that have the largest number of edges between \( v_1, \ldots, v_i \) and \( u \).

Figure 3 shows the MA ordering of Figure 1 (assuming the algorithm started from node 1):

![Figure 3: MA ordering of vertices in Figure 1.](image)

Note that MA ordering depends on the initial vertex and it is not necessarily unique for a given graph. For instance, order 146325 is also an MA ordering for Figure 1.

MA ordering in multigraphs is also possible. You simply count all the edges separately while finding the node which has the maximum number of edges with the already ordered set or alternatively we can treat multigraph as a weighted simple graph. If there are \( n \) parallel edges between \( x \) and \( y \), then we add a single edge with degree \( n \) and self-loops are simply ignored. Figure 4 shows an example multigraph and the corresponding weighted simple graph.

Now we can start to describe the solution approach. Given a multigraph, we first convert it to a weighted simple graph as described above. Note that, if the original graph is already a simple graph, then we get a graph with all edge weights equal to 1 after conversion. After that, we find the MA ordering of the vertices. However, note that while finding the MA ordering, we need to consider the weights of edges since they are originally different edges. Instead of using the maximum number of edges, we use the total weight of edges connecting the node to an already ordered set. After these steps, we will make use of some theorems to compute the edge connectivity.

**Theorem.** Let \( G_{xy} \) be the graph obtained from \( G \) by contracting nodes \( x \) and \( y \). In this operation we omit the possibility of arising loop (if \( x \) and \( y \) are connected in \( G \)), but keep the parallel edges. Then for any two
Theorem. In any MA ordering $v_1, \ldots, v_n$ of the nodes

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

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

Using above theorems, we can continue to our algorithm description. We had a weighted simple graph at hand. We first find the MA ordering of the graph by also considering the edge weights and then take the last two nodes in this ordering for $x$ and $y$. Then edge connectivity of $x$ and $y$ will be equal to the degree of the last node in the MA ordering. Then, we contract $x$ and $y$ and apply the same algorithm recursively to the resulting graph. Note that resulting contracted graph is smaller than the original graph at each step, so eventually the graph will end up with two nodes. At this point, the recursion ends.

Additionally, to compute the number critical edges, I basically removed each edge from the weighted graph seperately and checked if the edge connectivity has decreased. If the removal of an edge causes a decrease in edge connectivity, this edge is selected as a critical edge.

Algorithm 1 Pseudocode for Nagamochi-Ibaraki algorithm

```
INPUT : G := (V, E) //possibly a multi-graph
OUTPUT : $\lambda(G)$

Initialization:
G= MULTIGRAPH-TO-WEIGHTED-SIMPLE-GRAPH(G)
Nagamochi-Ibaraki-Recursive (G)
if $|V| == 2$ then
    return weight($E_G(v_1, v_2)$)
end if
vertList:=MA-ORDER(G)
(x, y) := vertList.lastTwo()
$\lambda_G(x, y) := d_G(y)$
G-contracted:=CONTRACT(G, x, y)
return min($\lambda_G(x, y)$, NAGAMOCHI-IBARAKI-RECURSIVE(G-contracted))
```
4. Implementation Details

4.1 Technologies Used in Implementation

To implement the proposed solution Java programming language was preferred since it has a very well known graph library, namely JGrapht\[2\], with various kinds of graph algorithms. As a development environment, Netbeans IDE\[3\] has been used since developing a GUI is fairly easy (simply by drag-and-drop). Ubuntu 11.10 operating system has been used. Note that, the operating system and IDE used in development does not affect the end user, the only requirement to run the program is Java Runtime Environment (JRE) which comes with Java Virtual Machine (JVM) (See Table 1).

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Ubuntu 11.10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming Language</td>
<td>Java</td>
</tr>
<tr>
<td>External Library</td>
<td>JGrapht</td>
</tr>
<tr>
<td>Development Environment</td>
<td>NetBeans</td>
</tr>
<tr>
<td>Graph Visualization Tool</td>
<td>Graphviz[4]</td>
</tr>
</tbody>
</table>

Table 1: Technologies used for implementation

4.2 Implementation Detail

Source codes can be found in the appendix. Brief descriptions of each class is given in Table 2.

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApplicationFrame.java</td>
<td>used to develop a GUI</td>
</tr>
<tr>
<td>ComputingThread.java</td>
<td>used to compute the result in a thread</td>
</tr>
<tr>
<td>CustomVertex.java</td>
<td>used to represent a vertex</td>
</tr>
<tr>
<td>CustomVertexFactory.java</td>
<td>used to create a vertex</td>
</tr>
<tr>
<td>DotFileCreator.java</td>
<td>used to create a graph file with dot format</td>
</tr>
<tr>
<td>IGraphExperiment.java</td>
<td>interface for the experiment</td>
</tr>
<tr>
<td>NagamochiIbarkiExperiment.java</td>
<td>used to define an experiment</td>
</tr>
<tr>
<td>RandGraphGenerator.java</td>
<td>used to generate random graphs</td>
</tr>
<tr>
<td>IOrdering.java</td>
<td>interface for vertex ordering</td>
</tr>
<tr>
<td>MAOrdering.java</td>
<td>used to compute maximum adjacency ordering</td>
</tr>
<tr>
<td>NagamochiIbaraki.java</td>
<td>implementation of Naramochi-Ibaraki algorithm</td>
</tr>
<tr>
<td>IncrementalStats.java</td>
<td>used to compute graph statistics</td>
</tr>
<tr>
<td>MedianStats.java</td>
<td>used to compute median graph statistics</td>
</tr>
</tbody>
</table>

Table 2: Description of classes

5. Instructions for Users

In order to make it user friendly, a graphical user interface (GUI) has been developed. It has a very simplistic interface. User is asked to enter the number of edges, number of vertices and the number of experiments he wants to perform. Then, program computes different statistics about these experiments. This way, the
results will be more trustworthy since the result of a single experiment might always be misleading due to randomness.

![Image of Edge Connectivity Calculator]

Software has also a warning mechanism which warns the user in case of invalid entries (See Figure 6).

6. Experimental Results

In this section, experimental results on relationship between edge connectivity and average node degree in a graph is presented, as well as the relationship between critical edges and average degree. Each points in the graph corresponds to the average of 50 experiments.

Figure 7 shows that there is a linear relationship between edge density and edge connectivity. The main reason behind this result is that edges are selected randomly. Selecting edges randomly results in same expected degree for each node. However, this relation may not hold in case of real world topologies following power-law degree distribution. An important property of power-law distribution is that most of the nodes have less degrees, while a few nodes have high degrees. It is much more easier to disconnect this kind of topology than a random topology with the same edge density.

Figure 8 depicts that for a fixed edge density, increase in the network size decreases the edge connectivity. It can be explained by the following analogy: Assume that $X \sim U(0,M)$, where $U$ denotes for discrete uniform distribution. In our context, $X$ corresponds to the node degree of a single node. Then, we have $N$ i.i.d. random variables $X_1, X_2, \ldots, X_N$ (one for each node). Note that expected value of the edge density is $M/2$, since edge density is $\frac{1}{N} \sum_{i=1}^{N} X_i$, and $X_i$'s are i.i.d. Also note that it does not depend on $N$. Let
Figure 6: Graphical user interface warning the user in case of invalid input.

Figure 7: Relationship between edge density and edge connectivity
\[ Y = \delta(G) = \min(X_1, X_2, \ldots, X_N). \] Then, by simple calculations we can find that

\[ F_Y(y) = 1 - \left( 1 - \frac{y}{M} \right)^N \quad 0 < y < M. \]

It is important to note that when \( N \) gets larger, \( F_Y(y) \) converges to 1 in distribution. In other words, \( P(Y=0) \) approaches to 1 in probability. We also know that \( \lambda(G) < \delta(G) = Y \). Combining these two results, the larger the \( N \), the tighter the upper bound on \( \lambda(G) \) is, so increase in \( N \) with same edge density is expected to decrease the edge connectivity of the graph.

Figure 8: Relationship between edge connectivity and the number of nodes for a fixed edge density 10.

Figure 9 shows that the number of critical edges linearly increases with increase in edge density of the graph. This is again due to generating edges randomly.

7. Conclusion

In this project, Nagamochi-Ibaraki algorithm has been implemented to find the edge connectivity of the network. Then, using this algorithm, relationship between different characteristics have been analyzed. First, the relationship between edge connectivity and average node degree has been discussed. It has been observed that there is a linear relationship between them. Then, the effect of number of nodes (with fixed edge density) on edge connectivity has been presented and some technical analysis has been shown. Lastly, the relationship between critical edges and average degree has been analyzed.
Figure 9: Relationship between the number of critical edges and edge density

References


Appendix

1. ApplicationFrame.java

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package atnproject2;

import java.awt.Color;
import java.util.Observable;
import java.util.Observer;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JOptionPane;
import javax.swing JTextArea;
import javax.swing JTextField;

/**
 * @author emrah
 */
public class ApplicationFrame extends javax.swing.JFrame {

/**
 * Creates new form ApplicationFrame
 */

IGraphExperiment experiment;

public ApplicationFrame(IGraphExperiment e) {
    initComponents();
    experiment = e;
    e.addObserver((OutputTextArea) jTextAreaOutput);
}

private void freeze() {
    // TODO add your handling code here:
    jTextFieldNumOfEdges.setEditable(false);
    jTextFieldNumOfNodes.setEditable(false);
    jTextFieldNumOfExperiments.setEditable(false);
    jTextAreaOutput.setText(""");
}

private void unfreeze() {
    // TODO add your handling code here:
    jTextFieldNumOfEdges.setEditable(true);
    jTextFieldNumOfNodes.setEditable(true);
    jTextFieldNumOfExperiments.setEditable(true);
}

/**
 * This method is called from within the constructor to initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is always
 * regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:
    initComponents

private void initComponents() {

}
```java
jPanel1 = new javax.swing.JPanel();
jSplitPanel1 = new javax.swing.JSplitPane();
jPanel2 = new javax.swing.JPanel();
jLabelNumOfNodes = new javax.swing.JLabel();
jLabelNumOfExperiments = new javax.swing.JLabel();
jTextFieldNumOfNodes = new javax.swing.JTextField();
jTextFieldNumOfEdges = new javax.swing.JTextField();
jButtonCompute = new javax.swing.JButton();
jLabel2 = new javax.swing.JLabel();
jTextFieldNumOfExperiments = new javax.swing.JTextField();
JSeparator1 = new javax.swing.JSeparator();
jScrollPaneOutput = new javax.swing.JScrollPane();
jTextAreaOutput = new OutputTextArea();
setDefaultCloseOperation (javax.swing.WindowConstants.EXIT_ON_CLOSE);
setTitle("Edge Connectivity Calculator");

jSplitPanel1.setDividerLocation(220);
jSplitPanel1.setDividerSize(5);
jSplitPanel1.setOrientation(javax.swing.JSplitPane.VERTICAL_SPLIT);

jLabelNumOfNodes.setFont(new java.awt.Font("Comic Sans MS", 0, 20)); // NOI18N
jLabelNumOfNodes.setText("Number of Nodes (2−500)");
jLabelNumOfExperiments.setFont(new java.awt.Font("Comic Sans MS", 0, 20)); // NOI18N
jLabelNumOfExperiments.setText("Number of Experiments (1−100)");

jTextFieldNumOfNodes.setFont(new java.awt.Font("Comic Sans MS", 0, 20)); // NOI18N
jTextFieldNumOfNodes.setHorizontalAlignment(javax.swing.JTextField.RIGHT);
jTextFieldNumOfNodes.setMinimumSize(new java.awt.Dimension(100, 30));
jTextFieldNumOfNodes.setPreferredSize(new java.awt.Dimension(100, 30));
jTextFieldNumOfNodes.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jTextFieldNumOfNodesActionPerformed(evt);
    }
});

jTextFieldNumOfEdges.setFont(new java.awt.Font("Comic Sans MS", 0, 20)); // NOI18N
jTextFieldNumOfEdges.setHorizontalAlignment(javax.swing.JTextField.RIGHT);
jTextFieldNumOfEdges.setMinimumSize(new java.awt.Dimension(100, 30));
jTextFieldNumOfEdges.setPreferredSize(new java.awt.Dimension(100, 30));
jTextFieldNumOfEdges.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jTextFieldNumOfEdgesActionPerformed(evt);
    }
});

jButtonCompute.setFont(new java.awt.Font("Comic Sans MS", 0, 24)); // NOI18N
jButtonCompute.setText("Compute graph statistics");
jButtonCompute.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButtonComputeActionPerformed(evt);
    }
});

jLabel2.setFont(new java.awt.Font("Comic Sans MS", 0, 20)); // NOI18N
jLabel2.setText("Number of Edges (1−2000)");

jTextFieldNumOfExperiments.setFont(new java.awt.Font("Comic Sans MS", 0, 20)); // NOI18N
jTextFieldNumOfExperiments.setHorizontalAlignment(javax.swing.JTextField.RIGHT);
jTextFieldNumOfExperiments.setMinimumSize(new java.awt.Dimension(100, 30));
```

Appendix [1. ApplicationFrame.java] continued on next page...
jTextFieldNumOfExperiments.setPreferredSize(new java.awt.Dimension(100, 30));
jTextFieldNumOfExperiments.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jTextFieldNumOfExperimentsActionPerformed(evt);
    }
});

javax.swing.GroupLayout jPanel2Layout = new javax.swing.GroupLayout(jPanel2);
jPanel2.setLayout(jPanel2Layout);

jPanel2.setHorizontalGroup(
    jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(jPanel2Layout.createSequentialGroup()
            .addGap(18, 18, 18)
        ));

jPanel2.setVerticalGroup(
    jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(jPanel2Layout.createSequentialGroup())
        .addGap(18, 18, 18)
);
jSplitPane1.setTopComponent(jPanel2);

jScrollPaneOutput.setBorder(javax.swing.BorderFactory.createTitledBorder("Output"));

jTextAreaOutput.setColumns(20);
jTextAreaOutput.setEditable(false);
jTextAreaOutput.setRows(5);
jTextAreaOutput.addPropertyChangeListener(new java.beans.PropertyChangeListener() {
    public void propertyChange(java.beans.PropertyChangeEvent evt) {
        jTextAreaOutputPropertyChange(evt);
    }
});

jScrollPaneOutput.setViewportView(jTextAreaOutput);

jSplitPane1.setRightComponent(jScrollPaneOutput);

javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);
jPanel1.setLayout(jPanel1Layout);
jPanel1Layout.setHorizontalGroup(jPanel1Layout.createComponentParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addComponent(jSplitPane1));
jPanel1Layout.setVerticalGroup(jPanel1Layout.createComponentParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addComponent(jSplitPane1, javax.swing.GroupLayout.DEFAULT_SIZE, 512, Short.MAX_VALUE));

javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
getContentPane().setLayout(layout);
layout.setHorizontalGroup(layout.createComponentParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE));

Appendix [1. ApplicationFrame.java] continued on next page...
private void jTextFieldNumOfEdgesActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}
private void jButtonComputeActionPerformed(java.awt.event.ActionEvent evt) {

    freeze();
    if (isValidInputs()) {
        new ComputingThread(experiment, Integer.parseInt(jTextFieldNumOfEdges.getText()),
                             Integer.parseInt(jTextFieldNumOfNodes.getText()),
                             Integer.parseInt(jTextFieldNumOfExperiments.getText())).start();
    } else {
        JOptionPane.showMessageDialog(this, "Please check your inputs", "Error",
                                       JOptionPane.ERROR_MESSAGE);
    }
    unfreeze();
}
private boolean isValidInputs() {
    boolean isValid = true;
    isValid &= isInRange(jTextFieldNumOfEdges, 1, 2000);
    isValid &= isInRange(jTextFieldNumOfNodes, 2, 500);
    isValid &= isInRange(jTextFieldNumOfExperiments, 1, 100);
    return isValid;
}
private boolean isInRange(JTextField field, int min, int max) {
    boolean isValid = true;
    try {
        int numOfNodes = Integer.parseInt(field.getText().trim());
        if (numOfNodes < min || numOfNodes > max) {
            field.setForeground(Color.red);
            isValid = false;
        }
    } catch (Exception e) {
        field.setForeground(Color.red);
        isValid = false;
    }
    if (isValid) {
        field.setForeground(Color.black);
    }
    return isValid;
}
private void jTextFieldNumOfExperimentsActionPerformed(java.awt.event.ActionEvent evt) {
    //TODO add your handling code here:
} //GEN-LAST:event_jTextFieldNumOfExperimentsActionPerformed

private void jTextFieldNumOfNodesActionPerformed(java.awt.event.ActionEvent evt) {
    //TODO add your handling code here:
} //GEN-LAST:event_jTextFieldNumOfNodesActionPerformed

private void jTextAreaOutputPropertyChange(java.beans.PropertyChangeEvent evt) {
    //TODO add your handling code here:
} //GEN-LAST:event_jTextAreaOutputPropertyChange

/**
 * @param args the command line arguments
 */
public static void main(String args[]) {
    /*
     * Set the Nimbus look and feel
     */
    //</editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
    */
    /* If Nimbus (introduced in Java SE 6) is not available, stay with the
     * default look and feel. For details see
     * http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
     */
    try {
        for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.
                      getInstalledLookAndFeels()) {
            if ("Nimbus".equals(info.getName())) {
                javax.swing.UIManager.setLookAndFeel(info.getClassName());
                break;
            }
        }
    } catch (ClassNotFoundException ex) {
        java.util.logging.Logger.getLogger(ApplicationFrame.class.getName()).log(java.
                      util.logging.Level.SEVERE, null, ex);
    } catch (InstantiationException ex) {
        java.util.logging.Logger.getLogger(ApplicationFrame.class.getName()).log(java.
                      util.logging.Level.SEVERE, null, ex);
    } catch (IllegalAccessException ex) {
        java.util.logging.Logger.getLogger(ApplicationFrame.class.getName()).log(java.
                      util.logging.Level.SEVERE, null, ex);
    } catch (javax.swing.UnsupportedLookAndFeelException ex) {
        java.util.logging.Logger.getLogger(ApplicationFrame.class.getName()).log(java.
                      util.logging.Level.SEVERE, null, ex);
    }
    //</editor-fold>
    */
    /* Create and display the form
     */
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            IGraphExperiment e = new NagamochiIbarkiExperiment();
        }
    });
} //GEN-FIRST:event_jTextAreaOutputPropertyChange
new ApplicationFrame(e).setVisible(true);
}
}

// Variables declaration - do not modify
private javax.swing.JButton jButtonCompute;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabelNumOfExperiments;
private javax.swing.JLabel jLabelNumOfNodes;
private javax.swing.JPanel jPanel1;
private javax.swing.JPanel jPanel2;
private javax.swing.JScrollPane jScrollPaneOutput;
private javax.swing.JSeparator jSeparator1;
private javax.swing.JSplitPane jSplitPane1;
private javax.swing JTextArea jTextAreaOutput;
private javax.swing.JTextField jTextFieldNumOfEdges;
private javax.swing.JTextField jTextFieldNumOfExperiments;
private javax.swing.JTextField jTextFieldNumOfNodes;

// End of variables declaration

class OutputTextArea extends JTextArea implements Observer {
    int i = 1;
    @Override
    public void update(Observable o, Object o1) {
        if (o instanceof NagamochiIbarkiExperiment) {
            System.out.println("Experiment "+(i++)+" is runningggg");
            jTextAreaOutput.append(((NagamochiIbarkiExperiment) o).getCurrentResultMessage() + "\r\n");
        }
    }
}

2. ComputingThread.java

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package atnp2;

/**
 * @author emrah
 */
public class ComputingThread extends Thread{
    int e;
    int v;
    int ex;
    IGraphExperiment experiment;
    public ComputingThread(IGraphExperiment experiment, int edge, int vert, int exp) {
        e=edge;
        v=vert;
        ex=exp;
        this.experiment=experiment;
    }

    public void run(){
        experiment.setNumOfEdges(e);
        experiment.setNumOfVertices(v);
    }
}
3. CustomVertex.java

```java
package atnproject2;
import java.io.Serializable;

public class CustomVertex implements Cloneable, Serializable {

    private static final long serialVersionUID = -7876438071421049711L;

    private Long id;

    public CustomVertex(Long id) {
        super();
        this.id = id;
    }

    @Override
    public boolean equals(Object obj) {
        boolean result = false;
        if (this == obj) {
            result = true;
        } else if (obj == null || this.getClass() != obj.getClass()) {
            result = false;
        } else {
            CustomVertex objT = (CustomVertex) obj;
            result = this.id.equals(objT.id);
        }
        return result;
    }

    @Override
    public int hashCode() {
        return id.hashCode();
    }

    public Long getId() {
        return this.id;
    }

    @Override
    public String toString() {
        // return id.toString() + "˜" + (demand == null ? "null" : demand.toString());
        return id.toString();
    }
}
```

4. CustomVertexFactory.java

```java
package atnproject2;
import org.jgrapht.VertexFactory;

public class CustomVertexFactory implements VertexFactory {

    private static long id = 0;
```
public CustomVertexFactory() {
    super();
}

public static void reset() {
    id = 0;
}

@Override
public CustomVertex createVertex() {
    return new CustomVertex(new Long(++CustomVertexFactory.id));
}

5. DotFileCreater.java

/**
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package atnproject2;

import java.io.*;
import java.util.Iterator;
import java.util.Set;
import org.jgrapht.Graph;
import org.jgrapht.Graphs;
import org.jgrapht.graph.MultiGraph;
import org.jgrapht.graph.SimpleWeightedGraph;

/**
 * @author emrah
 */
public class DotFileCreater {

    // given a multi graph and a fileName, create an output file in dot language (see
    // Graphviz dot language)
    public static <V extends CustomVertex, E> File createDotFile(Graph<V, E> graph, String
            fileName) {
        File myFile = new File(fileName);
        try {
            PrintWriter outFile = new PrintWriter(new BufferedWriter(new FileWriter(myFile)));
            Set<E> edgeSet = graph.edgeSet();
            Iterator<E> itr = edgeSet.iterator();
            try {
                outFile.println("graph g {");
                outFile.println("node [fontsize="35", overlap=false, height=1.5, width=1.5];");
                while (itr.hasNext()) {
                    E edge = itr.next();
                    long id1 = graph.getEdgeSource(edge).getId();
                    long id2 = graph.getEdgeTarget(edge).getId();
                    if (graph instanceof SimpleWeightedGraph) {
                        outFile.println(id1 + " -- " + id2 + " [label=" + graph.getEdgeWeight
                                (edge) + "]");
                    } else if (graph instanceof MultiGraph) {
                        outFile.println(id1 + " -- " + id2 + ";");
                    }
                }
            }
            outFile.println("}");

            return myFile;
        }
        catch (IOException e) {
            e.printStackTrace();
            return null;
        }
    }
}
7. IGraphExperiment.java

```java
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package atnproject2;

import java.util.Observable;

/**
 * @author emrah
 */
public abstract class IGraphExperiment extends Observable {

  protected int numOfExperiments;
  protected int numOfVertices;
  protected int numOfEdges;

  abstract void runSingleExperiment(int experimentId);
  abstract void runNExperiment(int N);

  void setNumOfExperiments(int N) {
    numOfExperiments = N;
  }

  public int getNumOfVertices() {
    return numOfVertices;
  }

  /**
   * @param numOfVertices the numOfVertices to set
   */
  public void setNumOfVertices(int numOfVertices) {
    this.numOfVertices = numOfVertices;
  }

  /**
   * @return the numOfEdges
   */
  public int getNumOfEdges() {
    return numOfEdges;
  }

  /**
   * @param numOfEdges the numOfEdges to set
   */
  public void setNumOfEdges(int numOfEdges) {
```

Appendix [6. IGraphExperiment.java] continued on next page...
7. NagamochiIbarkiExperiment.java

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package atnproject2;

import atnproject2.edgeConnectivity.NagamochiIbaraki;
import atnproject2.statistics.IncrementalStats;
import atnproject2.statistics.MedianStats;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Set;
import org.jgrapht.Graph;
import org.jgrapht.Graphs;
import org.jgrapht.EdgeWeightedGraph;
import org.jgrapht.graph.Multigraph;
import org.jgrapht.graph.SimpleWeightedGraph;

/**
 * @author emrah
 */

public class NagamochiIbarkiExperiment extends IGraphExperiment {

    private String currentResultMessage;
    private double edgeConnectivity;
    private double numOfCriticalEdges;

    @Override
    public void runSingleExperiment(int experimentId) {
        currentResultMessage = "Experiment" + (experimentId) + " is running...";
        setChanged();
        notifyObservers();
        Graph<CustomVertex, DefaultWeightedEdge> multigraph = new Multigraph<>
            (DefaultWeightedEdge.class);
        Graph<CustomVertex, DefaultWeightedEdge> simpleWeightedGraph = new
            SimpleWeightedGraph<>
            (DefaultWeightedEdge.class);
        Graph<CustomVertex, DefaultWeightedEdge> copyGraph = new SimpleWeightedGraph<>
            (DefaultWeightedEdge.class);
        RandGraphGenerator<CustomVertex, DefaultWeightedEdge> graphGenerator = new
            RandGraphGenerator<>();
        graphGenerator.generateRandomMultiGraph(multigraph, getNumOfVertices(),
            getNumOfEdges());
        DotFileCreator.createDotFile(multigraph, "multi_n" + getNumOfVertices() + ".dot");
        graphGenerator.multiGraphToSimpleWeightedGraph(multigraph, simpleWeightedGraph);
        DotFileCreator.createDotFile(simpleWeightedGraph, "simple_n" + getNumOfVertices() + ".dot");
        NagamochiIbaraki<CustomVertex, DefaultWeightedEdge> NIalgorithm = new
            NagamochiIbaraki<>();
        Graphs.addGraph(copyGraph, simpleWeightedGraph);
        edgeConnectivity = NIalgorithm.computeEdgeConnectivity(copyGraph);
        copyGraph = new SimpleWeightedGraph<>(DefaultWeightedEdge.class);
    }
}
Graphs.addGraph(copyGraph, simpleWeightedGraph);

Set<DefaultWeightedEdge> edgeSet = copyGraph.edgeSet();
List<DefaultWeightedEdge> criticalEdges = new ArrayList<>();
List<Double> edgeWeights = new ArrayList<>();
numOfCriticalEdges=0;
for (Iterator<DefaultWeightedEdge> it = edgeSet.iterator(); it.hasNext();)
{
    DefaultWeightedEdge edge = it.next();
    CustomVertex source = simpleWeightedGraph.getEdgeSource(edge);
    CustomVertex destination = simpleWeightedGraph.getEdgeTarget(edge);
    double weight = simpleWeightedGraph.getEdgeWeight(edge);
    if (weight > 1)
    {
        ((WeightedGraph)simpleWeightedGraph).setEdgeWeight(edge, weight - 1);
    }
    else
    {
        simpleWeightedGraph.removeEdge(edge);
    }
    copyGraph = new SimpleWeightedGraph<>(DefaultWeightedEdge.class);
    Graphs.addGraph(copyGraph, simpleWeightedGraph);
    if (Nlalgorithm.computeEdgeConnectivity(copyGraph) < edgeConnectivity)
    {
        criticalEdges.add(edge);
        edgeWeights.add(weight);
        numOfCriticalEdges += weight;
    }
    if (!simpleWeightedGraph.containsEdge(edge))
    {
        simpleWeightedGraph.addEdge(source, destination, edge);
    }
    else
    {
        ((WeightedGraph)simpleWeightedGraph).setEdgeWeight(edge, weight);
    }
}
for (int i = 0; i < criticalEdges.size(); i++)
{
    CustomVertexFactory.reset();
}

public String getCurrentResultMessage()
{
    return currentResultMessage;
}

@Override
public void runNExperiment(int numOfExperiments)
{
    IncrementalStats connStats = new IncrementalStats();
    MedianStats connStats2 = new MedianStats();
    IncrementalStats criticalEdgeStats = new IncrementalStats();
    MedianStats criticalEdgeStats2 = new MedianStats();
    for (int i = 1; i <= numOfExperiments; i++)
    {
        runSingleExperiment(i);
        connStats.add(edgeConnectivity);
        connStats2.add(edgeConnectivity);
        criticalEdgeStats.add(numOfCriticalEdges);
        criticalEdgeStats2.add(numOfCriticalEdges);
    }
    currentResultMessage = "\n";
    currentResultMessage += "Results of "+numOfExperiments+" experiments\n";
8. RandGraphGenerator.java

```java
public class RandGraphGenerator<V, E> {

    public void generateRandomMultiGraph(Graph<V, E> target, int numOfVertices, int numOfEdges) {
        GraphGenerator<V, E, V> generator = new RandomGraphGenerator<>((numOfVertices, numOfEdges);
        generator.generateGraph(target, new CustomVertexFactory(), new HashMap<String, V>(), new SimpleWeightedGraph);
    }

    public void multiGraphToSimpleWeightedGraph(Graph<V, E> source, Graph<V, E> target) {
        Graphs.addAllVertices(target, source.vertexSet());
        Set<E> edges = source.edgeSet();
        Iterator<E> itr = edges.iterator();
    }
}
```

Appendix [8. RandGraphGenerator.java] continued on next page...
```java
while (itr.hasNext()) {
    E edge = itr.next();
    E edge2 = target.addEdge(source.getEdgeSource(edge), source.getEdgeTarget(edge));
    if (edge2 == null) {
        E existingEdge = ((SimpleWeightedGraph<V, E>) target).getEdge(source.getEdgeSource(edge), source.getEdgeTarget(edge));
        ((SimpleWeightedGraph<V, E>) target).setEdgeWeight(existingEdge, ((SimpleWeightedGraph<V, E>) target).getEdgeWeight(existingEdge) + 1);
    }
}
```

### 9. IOrdering.java

```java
package atnproject2.edgeConnectivity;

import java.util.List;
import org.jgrapht.Graph;

/**
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

public interface IOrdering<V> {
    <E> List<V> sort(Graph<V, E> graph);
}
```

### 10. MAOrdering.java

```java
package atnproject2.edgeConnectivity;

import java.util.*;
import org.jgrapht.Graph;
import org.jgrapht.Graphs;
import org.jgrapht.util.FibonacciHeap;
import org.jgrapht.util.FibonacciHeapNode;

/**
 * @author emrah
 */

public class MAOrdering<V> implements IOrdering<V> {
    @Override
    public <E> List<V> sort(Graph<V, E> graph) {
        FibonacciHeap<V> heap = new FibonacciHeap<>();
        List<V> order = new ArrayList<>((graph.vertexSet()).size());
        Set<V> set = new LinkedHashSet<>();
        HashMap<V, FibonacciHeapNode> idToHeapNodeMap = new HashMap<>();
        Iterator<V> iter = graph.vertexSet().iterator();
        V initVertex = iter.next();
```
FibonacciHeapNode<V> initNode = new FibonacciHeapNode<>(initVertex);
heap.insert(initNode, -1);
idToHeapNodeMap.put(initVertex, initNode);

while (iter.hasNext()) {
    V vertex = iter.next();
    FibonacciHeapNode<V> node = new FibonacciHeapNode<>(vertex);
    heap.insert(node, 0);
idToHeapNodeMap.put(vertex, node);
}

while (!heap.isEmpty()) {
    V min = heap.removeMin().getData();
idToHeapNodeMap.remove(min);
    if (!set.add(min)) {
        System.err.println(min + " is already added");
        System.exit(-1);
    }
    order.add(min);
    List<V> neighbors = Graphs.neighborListOf(graph, min);
    for (Iterator<V> it = neighbors.iterator(); it.hasNext();)
    {
        V neighbor = it.next();
        FibonacciHeapNode<V> neighborHeapnode = idToHeapNodeMap.get(neighbor);
        if (idToHeapNodeMap.containsKey(neighbor)) {
            heap.decreaseKey(neighborHeapnode, neighborHeapnode.getKey() - graph.getEdgeWeight(graph.getEdge(min, neighbor))); //System.out.println(neighbor.getId()+" key is decreased to "+neighborHeapnode.getKey());
        }
    }
    //System.out.println(set);
    return order;
}

11. NagamochiIbaraki.java

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package atnproject2.edgeConnectivity;

import atnproject2.CustomVertexFactory;
import java.util.Iterator;
import java.util.List;
import org.jgrapht.Graph;
import org.jgrapht.Graphs;
import org.jgrapht.VertexFactory;
import org.jgrapht.graph.SimpleWeightedGraph;

/**
 * @author emrah
 */
public class NagamochiIbaraki\langle V, E \rangle 
{
    public double computeEdgeConnectivity(Graph\langle V, E \rangle graph) {
        if (graph.vertexSet().size() == 2) {
            V[] vert = (V[]) graph.vertexSet().toArray();
            return graph.getEdgeWeight(graph.getEdge(vert[0], vert[1]));
        }
        MAOrdering\langle V \rangle ordering = new MAOrdering<>();
        List\langle V \rangle list = ordering.sort(graph);
        int lastIndex = list.size() - 1;
        // lambda(x, y) is equal to the degree of the last vertex in the MAOrdering
        List\langle V \rangle neighbors = Graphs.neighborListOf(graph, list.get(lastIndex));
        double lambda = 0;
        for (Iterator\langle V \rangle it = neighbors.iterator(); it.hasNext();)
        {
            V neighbor = it.next();
            lambda += graph.getEdgeWeight(graph.getEdge(list.get(lastIndex), neighbor));
        }
        contractGraph(graph, list.get(lastIndex), list.get(lastIndex - 1));
        return min(lambda, computeEdgeConnectivity(graph));
    }

    private void contractGraph(Graph\langle V, E \rangle graph, V v1, V v2) {
        VertexFactory\langle V \rangle factory = new CustomVertexFactory();
        V addedVertex = factory.createVertex();
        graph.addVertex(addedVertex);
        transferEdges(graph, v1, addedVertex);
        transferEdges(graph, v2, addedVertex);
    }

    private void transferEdges(Graph\langle V, E \rangle graph, V v1, V addedVertex) {
        List\langle V \rangle neighborList = Graphs.neighborListOf(graph, v1);
        for (Iterator\langle V \rangle it = neighborList.iterator(); it.hasNext();)
        {
            V neighbor = it.next();
            double edgeWeight = graph.getEdgeWeight(graph.getEdge(v1, neighbor));
            if (!neighbor.equals(addedVertex)) {
                if (graph.addEdge(neighbor, addedVertex) == null) {
                    E existingEdge = ((SimpleWeightedGraph\langle V, E \rangle) graph).getEdge(neighbor, addedVertex);
                    ((SimpleWeightedGraph\langle V, E \rangle) graph).setEdgeWeight(existingEdge, ((
                        SimpleWeightedGraph\langle V, E \rangle) graph).getEdgeWeight(existingEdge) +
                        edgeWeight);
                } else {
                    ((SimpleWeightedGraph\langle V, E \rangle) graph).setEdgeWeight(graph.getEdge(neighbor,
                        addedVertex), edgeWeight);
                }
            }
            graph.removeVertex(v1);
        }
    }

    private double min(double a, double b) {
        return (a < b) ? a : b;
    }
}

12. IncrementalFreq.java

/*

Appendix [12. IncrementalFreq.java] continued on next page...
package atnproject2.statistics;

import java.io.PrintStream;

//XXX This implementation is very restricted, to be made more flexible
// using hashtables.
/**
 * A class that can collect frequency information on integer input.
 * right now it can handle only unsigned input. It simply ignores negative
 * numbers.
 */
public class IncrementalFreq implements Cloneable {

    // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
    // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
    // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =

    /** The number of items inserted. */
    private int n;

    /** freq[i] holds the frequency of i. primitive implementation, to be changed */
    private int[] freq = null;

    /**
     * The capacity, if larger than 0. Added values larger than or equal to
     * this one will be ignored.
     */
    private final int N;

    // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
    // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
    // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =

    /**
     * @param maxvalue Values in the input set larger than this one will be ignored.
     * However, if it is negative, no values are ignored.
     */
    public IncrementalFreq(int maxvalue) {
        N = maxvalue + 1;
        reset();
    }

    // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
/** Calls <code>this(-1)</code>, that is, no values will be ignored. 
* @see #IncrementalFreq(int) */

```java
public IncrementalFreq() {
    this(-1);
}
```

// Reset the state of the object. After calling this, all public methods
* behave the same as they did after constructing the object. */

```java
public void reset() {
    if ( freq==null || N==0 ) freq = new int[0];
    else for(int i=0; i<freq.length; ++i) freq[i]=0;
    n = 0;
}
```

// methods

/**
 * Adds item <code>i</code> to the input set.
 * It calls <code>add(i,1)</code>.
 * @see #add(int,int)
 */

```java
public final void add( int i ) { add(i,1); }
```

/**
 * Adds item <code>i</code> to the input set <code>k</code> times.
 * That is, it increments counter <code>i</code> by <code>k</code>.
 * If, however, <code>i</code> is negative, or larger than the maximum defined
 * at construction time (if a maximum was set at all) the operation is ignored.
 */

```java
public void add( int i, int k ) {
    if( N>0 && i>=N ) return;
    if( i<0 || k<=0 ) return;
    // Increase number of items by k.
    n+=k;
    // If index i is out of bounds for the current array of counters,
    // increase the size of the array to i+1.
    if( i>=freq.length )
    {
        int tmp[] = new int[i+1];
        System.arraycopy(freq, 0, tmp, 0, freq.length);
        freq=tmp;
    }
    // Finally, increase counter i by k.
    freq[i]+=k;
}
```
/** Returns number of processed data items.  
* This is the number of items over which the class holds statistics.  
*/  
public int getN() { return n; }

//

/**  
* Returns the number of occurrences of the given integer.  
*/  
public int getFreq(int i) {
    if( i>=0 && i<freq.length ) return freq[i];
    else return 0;
}

//

/**  
* Performs an element-by-element vector subtraction of the  
* frequency vectors. If <code>strict</code> is true, it  
* throws an IllegalArgumentException if <code>this</code> is  
* not strictly larger than <code>other</code> (element by element)  
* (Note that both frequency vectors are positive.)  
* Otherwise just sets those elements in <code>this</code> to zero  
* that are smaller than those of <code>other</code>.  
* @param other The instance of IncrementalFreq to subtract  
* @param strict See above explanation  
*/  
public void remove(IncrementalFreq other, boolean strict) {
    // check if other has non-zero elements in non-overlapping part
    if( strict && other.freq.length>freq.length )
    {
        for( int i=other.freq.length-1; i>=freq.length; --i )
        {
            if (other.freq[i]!=0)  
                throw new IllegalArgumentException();
        }
    }

    final int minLength = Math.min(other.freq.length, freq.length);
    for ( int i=minLength-1; i>=0; i-- )
    {
        if ( strict && freq[i] < other.freq[i] )
            throw new IllegalArgumentException();
        final int remove = Math.min(other.freq[i], freq[i]);
        n -= remove;
        freq[i] -= remove;
    }

    //

    /**  
    * Prints current frequency information. Prints a separate line for  
    * all values from 0 to the capacity of the internal representation using the  
    * format  
    * &lt;pre&gt;  
    * value occurrences  
    * &lt;/pre&gt;  
    * That is, numbers with zero occurrences will also be printed.  
    */

/**
   * public void printAll( PrintStream out ) {
   *   for(int i=0; i<freq.length; ++i)
   *   {   out.println("+freq[i]");
   * }
   *
   //   
   // Prints current frequency information. Prints a separate line for
   // all values that have a number of occurrences different from zero using the
   // format
   // <pre>value occurrences</pre>
   /**
   * public void print( PrintStream out ) {
   *   for(int i=0; i<freq.length; ++i)
   *   {   if(freq[i]!=0) out.println("+freq[i]");
   * }
   *
   //   
   public String toString() {
   
   StringBuilder result=new StringBuilder(""");
   for(int i=0; i<freq.length; ++i)
   {   if (freq[i] != 0)
       result.append(" ( ").append(i).append(" , ").append(freq[i]).append(" ) ");
   }
   return result.toString();
   }
   *
   //   
   /** An alternative method to convert the object to String */
   
   public String toArithmeticExpression() {
   
   StringBuilder result=new StringBuilder(""");
   for(int i=freq.length-1; i>=0; i--)
   {   if (freq[i] != 0)
       result.append(freq[i]).append(" * ").append(i).append(" + ");
   }

   if (result.length()==0)
   return " (empty) ";
   else
   return result.substring(0, result.length()-1);
   }
   //
*/
public Object clone() throws CloneNotSupportedException {
    IncrementalFreq result = (IncrementalFreq) super.clone();
    if (freq != null) result.freq = freq.clone();
    return result;
}

/**
 * Tests equality between two IncrementalFreq instances.
 * Two objects are equal if both hold the same set of numbers that have
 * occurred non-zero times and the number of occurrences is also equal for
 * these numbers.
 */
public boolean equals(Object obj) {
    if (! (obj instanceof IncrementalFreq)) return false;
    IncrementalFreq other = (IncrementalFreq) obj;
    final int minlength = Math.min(other.freq.length, freq.length);
    for (int i=minlength-1; i>=0; i--)
        if (freq[i] != other.freq[i])
            return false;
    if (freq.length > minlength) other = this;
    for (int i=minlength; i<other.freq.length; i++)
        if (other.freq[i] != 0)
            return false;
    return true;
}

/**
 * Hashcode (consistent with {@link #equals}). Probably you will never want to
 * use this, but since we have {@link #equals}, we must implement it.
 */
public int hashCode() {
    int sum = 0;
    for (int i=0; i<freq.length; ++i) sum += freq[i]*i;
    return sum;
}

/*
public static void main(String[] pars) {
    IncrementalFreq ifq = new IncrementalFreq(Integer.parseInt(pars[0]));
    for (int i=1; i<pars.length; ++i)
    {
        ifq.add(Integer.parseInt(pars[i]));
    }
    ifq.print(System.out);
    System.out.println(ifq);
} */
13. IncrementalStats.java

/*
 * Copyright (c) 2003–2005 The BISON Project
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU Lesser General Public License version 2 as
 * published by the Free Software Foundation.
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU Lesser General Public License for more details.
 * You should have received a copy of the GNU Lesser General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

package atnproject2.statistics;

/**
 * A class that can keep track of some statistics like variance, average, min,
 * max incrementally. That is, when adding a new data item, it updates the
 * statistics.
 */
public class IncrementalStats {

    // __________________________ fields __________________________

    private double min;
    private double max;
    private double sum;
    private double sqsum;
    private int n;
    private int countmin;
    private int countmax;

    // __________________________ initialization __________________________

    /** Calls {@link #reset}. */
    public IncrementalStats() { reset(); }

    /** Resets the statistics to reflect the zero elements set.
     * Min and max are set to positive and negative infinity, respectively.
     */
    public void reset() {

}
countmin = 0;
countmax = 0;
min = Double.POSITIVE_INFINITY;
max = Double.NEGATIVE_INFINITY;
sum = 0.0;
sqrsum = 0.0;
n = 0;
}

/** Updates the statistics according to this element. It calls
 * <code>add(item,1)</code>. 
 * @see #add(double,int) */
public final void add(double item) { add(item,1); }

/** Updates the statistics assuming element <code>item</code> is added
 * <code>k</code> times. */
public void add(double item, int k) {
    if (item < min) {
        min = item;
        countmin = 0;
    }
    if (item == min) countmin += k;
    if (item > max) {
        max = item;
        countmax = 0;
    }
    if (item == max) countmax += k;
    n += k;
    if (k == 1) {
        sum += item;
        sqrsum += item * item;
    } else {
        sum += item * k;
        sqrsum += item * item * k;
    }
}

/** The number of data items processed so far */
public int getN() { return n; }

/** The maximum of the data items */
public double getMax() { return max; }
/** The minimum of the data items */
public double getMin() { return min; }

/** Returns the number of data items whose value equals the maximum. */
public int getMaxCount() { return countmax; }

/** Returns the number of data items whose value equals the minimum. */
public int getMinCount() { return countmin; }

/** The sum of the data items */
public double getSum() { return sum; }

/** The sum of the squares of the data items */
public double getSqrSum() { return sqrsum; }

/** The average of the data items */
public double getAverage() { return sum/n; }

/** The empirical variance of the data items. Guaranteed to be larger or equal to 0.0. If due to rounding errors the value becomes negative, it returns 0.0. */
public double getVar() {
    double var=
        (((double)n) / (n-1)) * (sqrsum/n - getAverage()*getAverage());
    return (var>=0.0?var:0.0);
    // XXX note that we have very little possibility to increase numeric
    // stability if this class is "greedy", ie, if it has no memory
    // In a more precise implementation we could delay the calculation of
    // statistics and store the data in some intelligent structure
}

/** the empirical standard deviation of the data items */
public double getStd() { return Math.sqrt(getVar()); }

/** Prints the following quantities separated by spaces in a single line
 in this order.
 Minimum, maximum, number of items, average, variance, number of minimal
 items, number of maximal items. */
publUc String toString() {
    return min+"+max+"+n+"+sum/n+"+getVar()+"+
    countmin+"+countmax;
}
14. MedianStats.java

/*
 * Copyright (c) 2003–2005 The BISON Project
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU Lesser General Public License version 2 as
 * published by the Free Software Foundation.
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU Lesser General Public License for more details.
 * You should have received a copy of the GNU Lesser General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

package atnproject2.statistics;
import java.util.ArrayList;
import java.util.Collections;

/**
 * This class adds the ability to retrieve the median element to the
 * {@link IncrementalStats} class. Note that this class actually stores all
 * the elements, so (unlike in its superclass) storage requirements depend
 * on the number of items processed.
 * @author giampa
 */
public class MedianStats extends IncrementalStats
{

/** Structure to store each entry. */
private final ArrayList<Double> data=new ArrayList<Double>();

/** Calls {@link #reset}. */
public MedianStats()
{
    reset();
}

/**
 * Retrieves the median in the current data collection.
 * @return The current median value.
 */
public double getMedian()
{
    double result;

    if (data.isEmpty())
        throw new IllegalArgumentException("Data vector is empty!");

    // Sort the arraylist
    Collections.sort(data);

    // Get the median
    return Collections.min(data);
}
if (data.size() % 2 != 0) { // odd number
    result = data.get(data.size() / 2);
} else { // even number:
    double a = data.get(data.size() / 2);
    double b = data.get(data.size() / 2 - 1);
    result = (a + b) / 2;
}
return result;

public void add(double item, int k)
{
    for (int i = 0; i < k; ++i) {
        super.add(item, 1);
        data.add(new Double(item));
    }
}

public void reset()
{
    super.reset();
    if (data != null)
        data.clear();
}

public static void main(String[] args)
{
    MedianStats s = new MedianStats();
    for (int i = 0; i < args.length; i++)
        s.add(Double.parseDouble(args[i]));
    System.out.println("Average : "+s.getAverage());
    System.out.println("Median : "+s.getMedian());
}