Al-Imam Mohammad Ibn Saud Islamic University
Collage of Computer and Information Sciences
Department of Computer Sciences

جامعة الإمام محمد بن سعود الإسلامية
كلية علوم الحاسب والمعلومات
قسم علوم الحاسب

CS 215: Design and Analysis of Algorithms	**1437-1438 H, Second semester**
Course Project	**Due to: Sunday (Apr 9 , 2019)**

Bridge To Nowhere

Done by

Budur Alkhalawi

Manar Aldossari

Sarah Alturki

Supervised by

T.Ghada Alosaimi

Pseudocode and Time Complexity

```
1. Algorithm connectedCitiesTable(N,A,B)
2. INPUT : N number of cities , A is array of north Cities, B is array south Cities.
3. OUTPUT : cct(Dynamic table ) contain the highest number of possible bridges.
4.          let cct[1..N+1,1..N+1]be new table
5.          for i=1 to N+1 do
6.                      cct[i,1]=0
7.          end for
8.          for j=1 to N+1 do
9.                      cct[1,j]=0
10.         end for
11.         for i=2 to N+1 do
12.           for j=2 to N+1 do
13.                     if(A[i]=B[j])then
14.                                 cct[i,j]=1+cct[i-1,j-1]
15.                      else
16.                                 cct[i,j] =max(cct[i-1,j],cct[i,j-1])
17.                     end if
18.           end for
19.         end for
20.         return cct
21. end algorithm
```

```
1. Algorithm ConnectedCitiesSolution(A, cct [][],i ,j)
2. INPUT : cct(Dynamic table ) contain the highest number of possible bridges, A is array of north
Cities.
3. OUTPUT : s connected Cities in the optimal solution.
4.    s=nil
5.     while (i ≠ 0 and j ≠ 0)
6.          if (cct [i,j]= cct [i-1,j])then
7.                       i=i-1
8.          else if (cct [i,j]= cct [i,j-1])then
9.                       j=j-1
10.         else
11.                     s=A[i]+s
12.                     i=i-1
13.                     j=j-1
14.                    end if
15.     end while
16.     return s
17. end algorithm
```

Analysis of connectedCitiesTable Algorithm:

* Initialization O(1).
* First loop takes O(n+1).
* Second loop takes O(n+1).
* The nested loop takes O(n^2).
* The running time of algorithm is : O(n^2).

Analysis of ConnectedCitiesSolution Algorithm:

* Initialization O(1).
* The loop takes O(2n).
* The running time of algorithm is : O(n).

Implementation

```java
//Algorthim project
package bridge.to.nowhere;

//_____________________________________________________________________________
//graph imports
import com.mxgraph.layout.mxCircleLayout;
import com.mxgraph.model.mxGraphModel;
import com.mxgraph.swing.mxGraphComponent;
import com.mxgraph.util.mxConstants;
import com.mxgraph.util.mxUtils;
import org.jgrapht.ext.JGraphXAdapter;
import org.jgrapht.graph.DefaultEdge;
import org.jgrapht.graph.SimpleGraph;
//GUI imports
import javax.swing.*;
import java.awt.*;
import java.util.Collection;
//read from file imports
import java.util.Scanner;
import java.io.File;

//_____________________________________________________________________________
public class BridgeToNowhere {

    static Scanner input = new Scanner(System.in);

    // instance variables
    static Scanner readFile;
    static int northCities[];
    static int southCities[];
    static int numberOfCities;
    //_____________________________________________________

    //method for north and south cities Initialization
    public static int[] citiesInitialization(boolean isSouth) {
        int cities[] = new int[numberOfCities + 1];
        cities[0] = 0;
        //south cities
        if (isSouth) {
            for (int i = 1; i <= numberOfCities; i++) {
                cities[i] = i;
            }
            //north cities
        } else {
            for (int i = 1; i <= numberOfCities; i++) {
                cities[i] = readFile.nextInt();
            }
        }
        return cities;
    }
    //_____________________________________________________
```

```java
    //method for connected Cities Array gneration to know the highest number of possible bridges
    public static int[][] connectedCitiesArray() {
        int lcs[][] = new int[numberOfCities + 1][numberOfCities + 1];

        for (int i = 0; i <= numberOfCities; i++) {
            lcs[0][i] = 0;
        }
        for (int i = 0; i <= numberOfCities; i++) {
            lcs[i][0] = 0;
        }

        for (int i = 1; i <= numberOfCities; i++) {
            for (int j = 1; j <= numberOfCities; j++) {
                if (northCities[i] == southCities[j]) {
                    lcs[i][j] = 1 + lcs[i - 1][j - 1];
                } else {
                    lcs[i][j] = Math.max(lcs[i - 1][j], lcs[i][j - 1]);
                }
            }
        }
        return lcs;
    }

    //_____________________________________________________
//    public static String connectedCitiesString(String connected , int CCArray[][] , int i ,int j){
//        if(CCArray[i][j] == 0)
//            return connected;
//
//        if(CCArray[i][j] == CCArray[i-1][j])
//            return connectedCitiesString(connected , CCArray , i-1,j);
//
//        if(CCArray[i][j] == CCArray[i][j-1])
//            return connectedCitiesString(connected , CCArray , i,j-1);
//
//        connected = northhCities[i]+" " +connected ;
//        return connectedCitiesString(connected , CCArray , i-1,j-1);
//
//    }
    public static String connectedCitiesString(String connected, int CCArray[][], int j, int i) {
        String s = "";

        while (i != 0 && j != 0) {
            if (CCArray[i][j] == CCArray[i - 1][j]) {
                i -= 1;
            } else if (CCArray[i][j] == CCArray[i][j - 1]) {
                j -= 1;
            } else {
                s = northCities[i] + " " + s;
                i -= 1;
                j -= 1;
            }

        }
        return s;
    }

    //_____________________________________________________
```

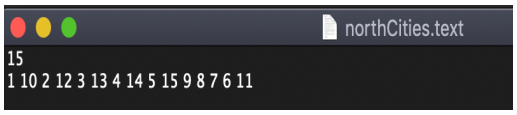
```java
    public static void visualize(int numberOfBridge, String connectedCitiesString, SimpleGraph graph) {
        JFrame frame = new JFrame();
        frame.setLayout(new BorderLayout());
        //panel1 info
        JPanel panel1 = new JPanel();
        JLabel la = new JLabel("The highest number of possible bridges : ");
        panel1.add(la);
        JTextField textField = new JTextField(numberOfBridge + "");
        panel1.add(textField);
        textField.disable();
        frame.add(panel1, BorderLayout.NORTH);

        //panel2
        JPanel panel2 = new JPanel();
        JLabel la2 = new JLabel("Connected Cities : ");
        panel2.add(la2);
        JTextField t22 = new JTextField(connectedCitiesString);
        panel2.add(t22);
        t22.disable();
        panel1.add(panel2, BorderLayout.AFTER_LAST_LINE);

        //panel3 Graph visualization
        JPanel p3 = new JPanel();

        JGraphXAdapter jgxAdapter = new JGraphXAdapter<String, DefaultEdge>(graph);
        mxGraphComponent graphComponent = new mxGraphComponent(jgxAdapter);
        mxGraphModel graphModel = (mxGraphModel) graphComponent.getGraph().getModel();
        Collection<Object> cells = graphModel.getCells().values();
        // This part to remove arrow from edge
        mxUtils.setCellStyles(graphComponent.getGraph().getModel(),
                cells.toArray(), mxConstants.STYLE_ENDARROW, mxConstants.NONE);
        frame.getContentPane().add(graphComponent);
        mxCircleLayout layout = new mxCircleLayout(jgxAdapter);
        layout.execute(jgxAdapter.getDefaultParent());

        graphComponent.setSize(750, 750);

        JScrollPane Scrl = new JScrollPane(graphComponent,
                ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS,
                ScrollPaneConstants.HORIZONTAL_SCROLLBAR_ALWAYS);

        frame.getContentPane().add(Scrl, BorderLayout.CENTER);

        //panel4
        JPanel p4 = new JPanel();
        JTextField t1 = new JTextField("              Northh Cities          ");
        p4.add(t1, BorderLayout.EAST);

        JTextField t2 = new JTextField("             South Cities             ");
        p4.add(t2, BorderLayout.WEST);
        t1.disable();
        t2.disable();
        frame.add(p4, BorderLayout.SOUTH);
        frame.setVisible(true);
        frame.setTitle("Bridge To Nowhere");
        frame.setSize(1300, 1000);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
```

```
169
170     //_____________________________________________________________
171     public static SimpleGraph graph(String connectedCitiesString) {
172         SimpleGraph<String, DefaultEdge> graph = new SimpleGraph<String, DefaultEdge>(DefaultEdge.class);
173
174         //south Vertex
175         for (int i = 1; i <= numberOfCities; i++) {
176             graph.addVertex("s" + southCities[i]);
177         }
178         //north Vertex
179         for (int i = numberOfCities; i >= 1; i--) {
180             graph.addVertex("n" + northCities[i]);
181         }
182
183         //edges
184         while (connectedCitiesString.length() != 0) {
185
186             int i = connectedCitiesString.indexOf(" ");
187             if (i <= 0) {
188                 break;
189             }
190             int index = Integer.parseInt(connectedCitiesString.substring(0, i));
191             graph.addEdge("n" + index, "s" + index);
192             connectedCitiesString = connectedCitiesString.substring(i + 1, connectedCitiesString.length());
193
194         }
195         return graph;
196     }
197
198     //_____________________________________________________________
199     //main metho
200     public static void main(String[] args) {
201         try {
202             readFile = new Scanner(new File("northCities.text"));
203
204         } catch (Exception e) {
205             System.out.println("Sorry the file northCities.text is not found");
206         }
207
208         //reads number of cities from the file
209         numberOfCities = readFile.nextInt();
210         northCities = citiesInitialization(false);
211         southCities = citiesInitialization(true);
212
213         int[][] CCArray = connectedCitiesArray();
214         String connectedCitiesString = connectedCitiesString("", CCArray, numberOfCities, numberOfCities);
215
216         SimpleGraph<String, DefaultEdge> graph = graph(connectedCitiesString);
217
218         int numberOfBridge = CCArray[numberOfCities][numberOfCities];
219         visualize(numberOfBridge, connectedCitiesString, graph);
220
221     }
222
223 }
```

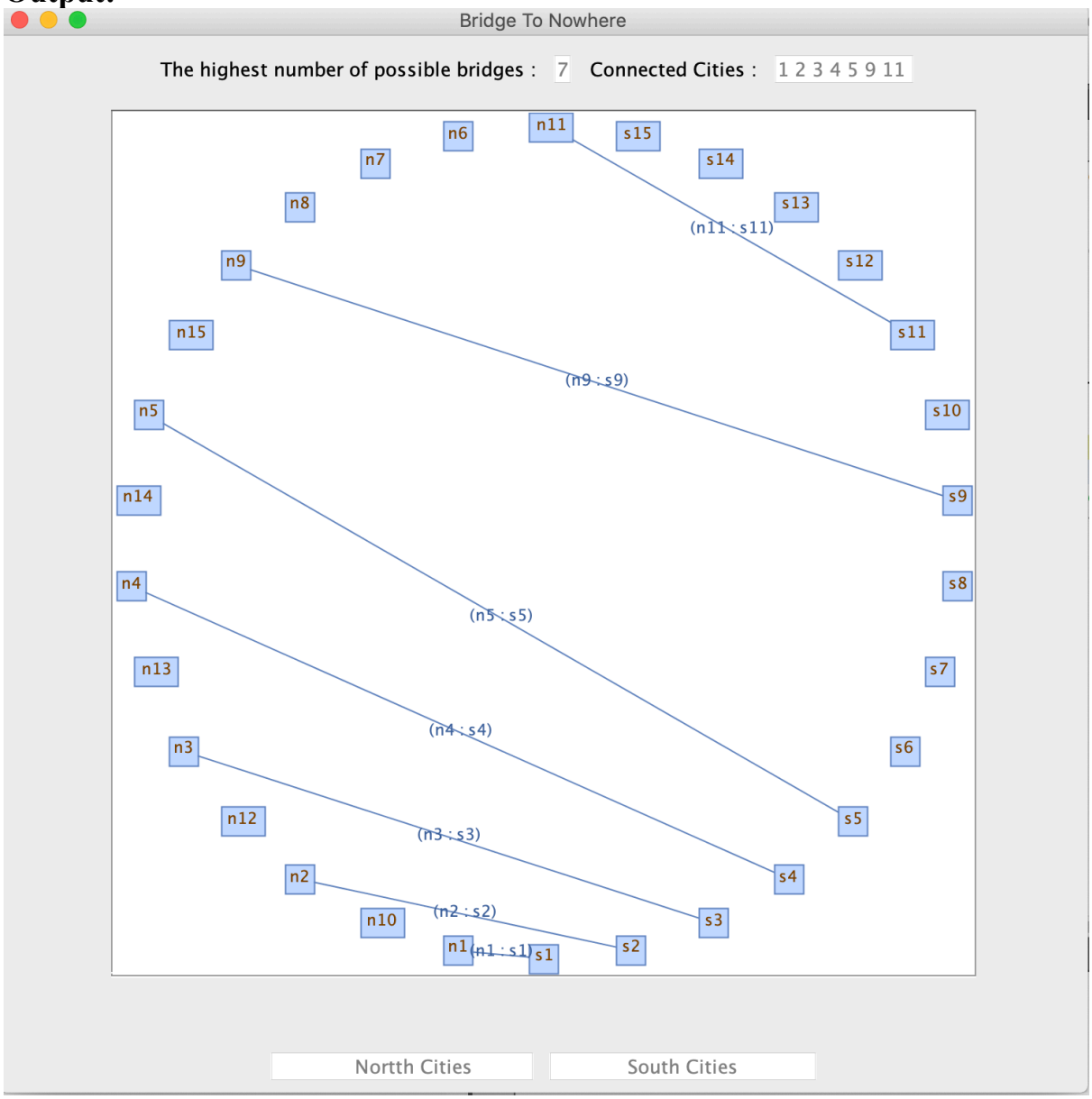
Testing Results

First Testing Results:

Input:



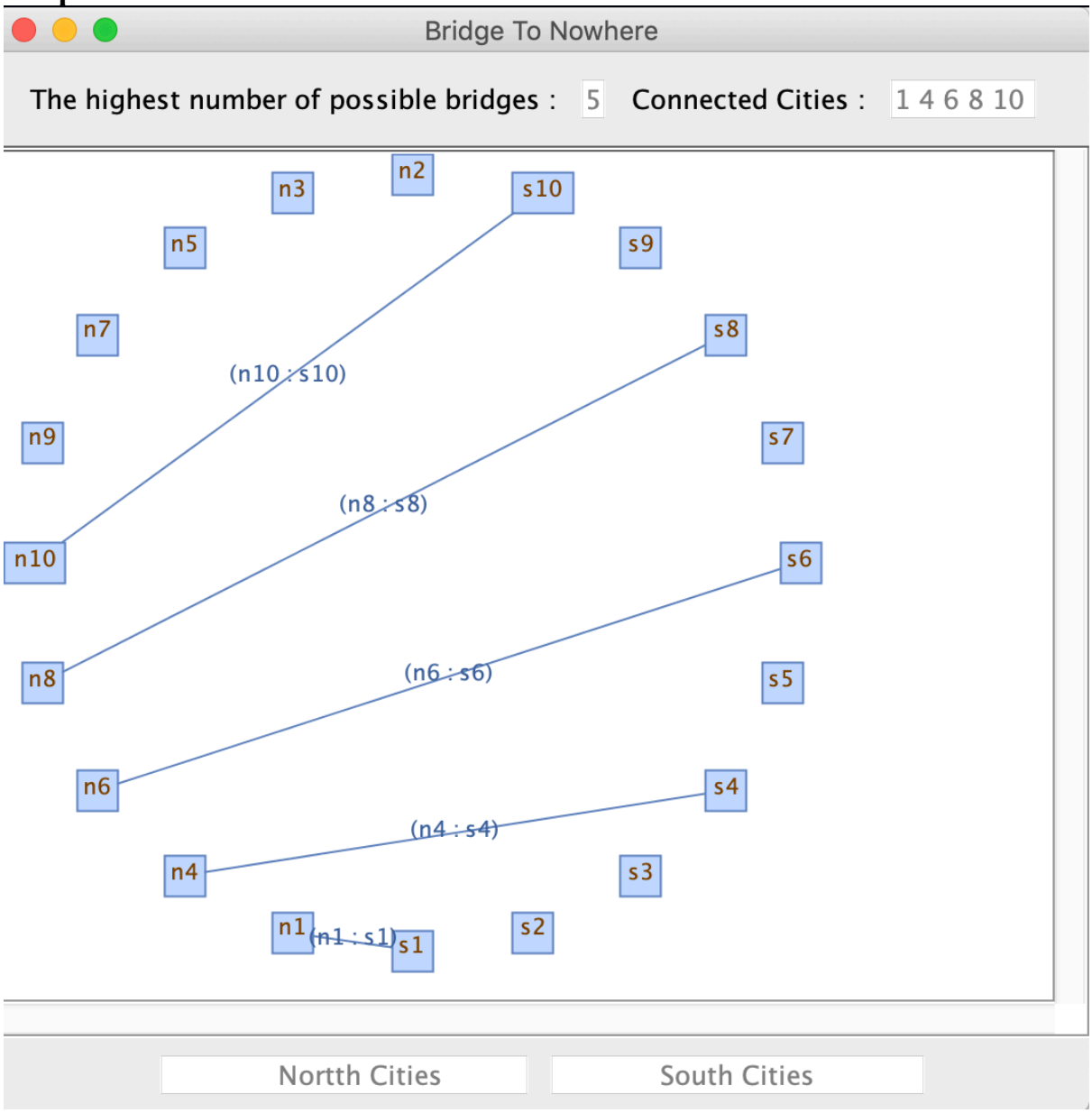
Output:



Second Testing Results:

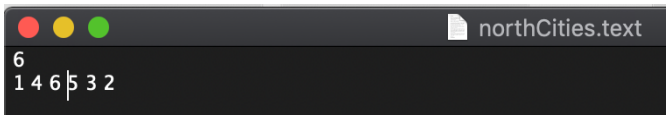
Input:

Output:

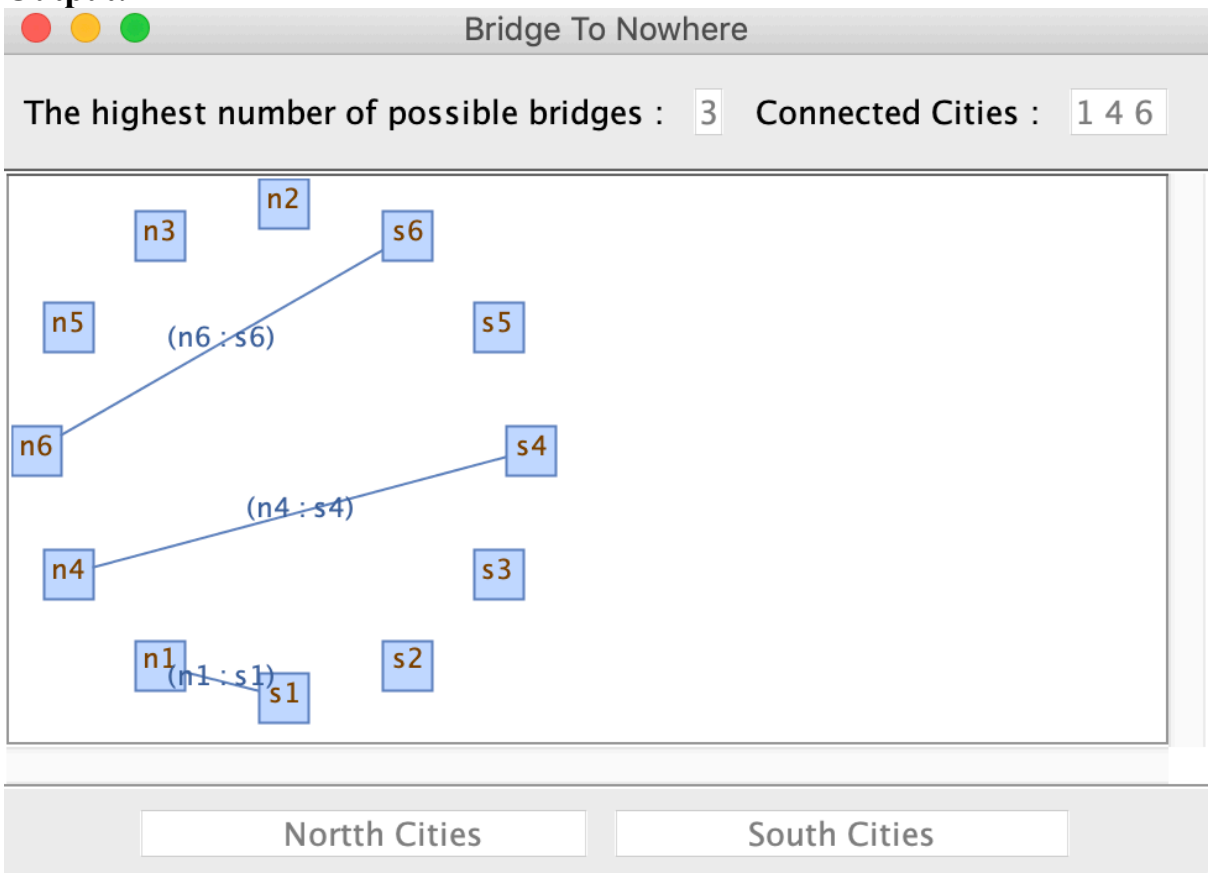


Third Testing Results:

Input:



Output:



conclusion

In our project we found the maximum optimal solution for connoting the two side of the country without crossing the bridges using the longest common subsequence Algorithm.

In this project we learn how to reuse another algorithms in our solution, and how to visualize our solution by using GUI and JGraphT Library .

Applications of longest common subsequence Algorithm :

1- finding the longest common substring of two strings.
2- compare DNA of two different organisms to find how similar they are.
3- Finding the minimum crossing routes by finding the largest number of common routes between stations in the design process.

References :

1-introduction to algorithms thomas h. cormen charles e. leiserson Ronald l. rivest Clifford stein 3rd edition

2- JGraphT library (1.3.0 jar-0.8.3 jar) https://jgrapht.org/

3-Swing(GUI widget toolkit).