Appendix

Source Code:

```
package transportation;

import javax.swing.*;
import java.util.ArrayList;
import java.io.FileWriter;
import java.io.IOException;
import static java.lang.Math.min;
import java.io.BufferedReader;
import java.io.FileReader;
import java.awt.*;
import java.io.File;

public class Transportation
{

    private static ArrayList<JTextField> textFields1 = new ArrayList<>();
    private static ArrayList<JTextField> textFields2 = new ArrayList<>();

    static int supplyPoints;
    static int demandPoints;
    static int numberOfRows;
    static int numberOfColumns;

    static int[] supply = new int[40];
    static int[] demand = new int[40];
    static int[] rowPenalty = new int[40];
    static int[] colPenalty = new int[40];

    static int[][] cost = new int[40][40];
    static int[][] problemArray = new int[41][41];
    static int[][] solutionArray = new int[40][40];

    static String allMessage = "";

    public static void main(String[] args)
    {
        SwingUtilities.invokeLater(() -> createAndShowGUI());
    }

    private static void createAndShowGUI()
    {
        JFrame frame = new JFrame("Transportation Problem");
        JPanel mainPanel = new JPanel();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(1400, 750);

        // Set layout for the mainPanel
        mainPanel.setLayout(null); // For simplicity, using absolute
positioning
```

```
        // Add the mainPanel to the frame
        frame.add(mainPanel);
        frame.setVisible(true);

        ///start from here

        // Add a new label to the mainPanel
        JLabel newLabel = new JLabel("Number of Suppliers");
        newLabel.setBounds(30, 15, 200, 30); // Set the position and size
of the label
        mainPanel.add(newLabel);

        // Add a new label to the mainPanel
        JLabel newLabe2 = new JLabel("Number of Demands");
        newLabe2.setBounds(210, 15, 200, 30); // Set the position and
size of the label
        mainPanel.add(newLabe2);

        frame.setVisible(true);
        frame.setVisible(true);

        //end here
        // Call the addTextField method to add the JTextField dynamically
        addMultipleTextFields1(mainPanel);

        JButton retrieveButton = new JButton("Set Input Area");
        retrieveButton.setBounds(390, 20, 150, 18);
        mainPanel.add(retrieveButton);

        JButton solveButton = new JButton("Solve the Problem");
        solveButton.setBounds(550, 20, 150, 18);
        mainPanel.add(solveButton);

        retrieveButton.addActionListener(e ->
        {
            JTextField textFieldToGetValue1 = textFields1.get(0);
            JTextField textFieldToGetValue2 = textFields1.get(1);

            if (textFieldToGetValue1.getText().equals(""))
            {
                JOptionPane.showMessageDialog(null, "Enter a valid value
for supply points!");
            }
            else if (textFieldToGetValue2.getText().equals(""))
            {
                JOptionPane.showMessageDialog(null, "Enter a valid value
for demand points!");
            }
            else if (Integer.parseInt(textFieldToGetValue1.getText())>40
|| Integer.parseInt(textFieldToGetValue2.getText())>40)
            {
                JOptionPane.showMessageDialog(null, "Allowed problem size
is exceeded!");
            }
            else
            {
```

```
                supplyPoints =
Integer.parseInt(textFieldToGetValue1.getText());
                demandPoints =
Integer.parseInt(textFieldToGetValue2.getText());

                int listSize = textFields2.size();

                if (listSize!=0)
                {
                    for (int i=0; i<listSize;i++)
                    {
                        JTextField textFieldToRemove =
textFields2.get(i); // Remove the first textField from the list
                        mainPanel.remove(textFieldToRemove); // Remove
the textField from the panel
                    }
                    mainPanel.revalidate();
                    mainPanel.repaint();
                    textFields2.clear();

addMultipleTextFields2(mainPanel,(supplyPoints+1),(demandPoints+1), 30,
50);
                }
                else
                {

addMultipleTextFields2(mainPanel,(supplyPoints+1),(demandPoints+1), 30,
50);
                }
            }
        });

        solveButton.addActionListener(e ->
        {
            int listSize = textFields2.size();
            if (listSize==0)
            {
                JOptionPane.showMessageDialog(null, "Please create a
input table!");
            }
            else
            {
                int myCheck=0;
                for(int i=0;i<listSize;i++)
                {
                    JTextField textFieldToGetValue = textFields2.get(i);
                    if (textFieldToGetValue.getText().equals(""))
                    {
                        myCheck = 1;
                        break;
                    }
                }

                if (myCheck==1)
                {
```

```
                    JOptionPane.showMessageDialog(null, "Please fill
entire input table!");
                }
                else
                {
                    solveProblem();
                }
            }
        });
    }

    public static void addMultipleTextFields1(JPanel panel)
    {
        int x = 160;
        int y = 20;
        int textFieldWidth = 35;
        int textFieldHeight = 20;
        int horizontalSpacing = 180;

        for (int i = 0; i < 2; i++)
        {
            JTextField textField = new JTextField();
            textField.setBounds(x, y, textFieldWidth, textFieldHeight);
            textFields1.add(textField);
            panel.add(textField);
            x += horizontalSpacing; // Increment y-coordinate for the
next textField
        }
        panel.revalidate();
        panel.repaint();
    }

    public static void addMultipleTextFields2(JPanel panel, int count1,
int count2, int x, int y)
    {
        int xx = x;
        int yy = y;
        int textFieldWidth = 35;
        int textFieldHeight = 20;
        int verticalSpacing = 20;
        int horizontalSpacing = 40;


        for (int i = 1; i <= count1; i++) {
            for (int j = 1; j <= count2; j++){
                if (i==count1 && j==count2)
                    continue;

                JTextField textField = new JTextField();
                textField.setBounds(xx, yy, textFieldWidth,
textFieldHeight);
                textFields2.add(textField);
                panel.add(textField);
                xx += horizontalSpacing;
            }
            yy += verticalSpacing;
```

```
            xx=x;
        }
        panel.revalidate();
        panel.repaint();
    }

    public static void solveProblem()
    {
        fillInitialArrays();

        allMessage = " ---------------------------------\n";
        allMessage+= "|     NORTHWEST CORNER METHOD      |\n";
        allMessage+= " ---------------------------------\n";
        allMessage+= "\nThe initial problem is\n";

        printTable();
        dummyCheck();
        printTable();
        northWestCornerMethod();

        fillInitialArrays();

        allMessage +="\n\n\n";
        allMessage += " ---------------------------------\n";
        allMessage += "|       LEAST COST METHOD          |\n";
        allMessage += " ---------------------------------\n";
        allMessage += "\nThe initial problem is\n\n";

        printTable();
        dummyCheck();
        printTable();
        leastCostMethod();

        fillInitialArrays();

        allMessage +="\n\n\n";
        allMessage+= " ---------------------------------\n";
        allMessage+= "|  VOGEL'S APPROXIMATION METHOD    |\n";
        allMessage+= " ---------------------------------\n";
        allMessage+= "\nThe initial problem is\n";

        printTable();
        dummyCheck();
        printTable();
        vogelsApproximationMethod();

        fillInitialArrays();

        allMessage +="\n\n\n";
        allMessage+= " ---------------------------------\n";
        allMessage+= "|  RUSSELL'S APPROXIMATION METHOD  |\n";
        allMessage+= " ---------------------------------\n";
        allMessage+= "\nThe initial problem is\n";

        printTable();
        dummyCheck();
```

```
        printTable();
        russellsApproximationMethod();

        /*fillInitialArrays();

        allMessage +="\n\n\n";
        allMessage+= " ----------------------------------\n";
        allMessage+= "|            MODI METHOD           |\n";
        allMessage+= " ----------------------------------\n";
        allMessage+= "\nThe initial problem is\n";

        printTable();
        dummyCheck();
        printTable();
        modiMethod();

        fillInitialArrays();

        allMessage +="\n\n\n";
        allMessage+= " ----------------------------------\n";
        allMessage+= "|      STEPPING STONE METHOD       |\n";
        allMessage+= " ----------------------------------\n";
        allMessage+= "\nThe initial problem is\n";

        printTable();
        dummyCheck();
        printTable();
        steppingStoneMethod();*/

        writeToFile();
        JOptionPane.showMessageDialog(null, "Problem solved!");

        String filePath = "solution.txt";
        try {
            File file = new File(filePath);
            if (!Desktop.isDesktopSupported()) {
                System.out.println("Desktop not supported");
                return;
            }

            Desktop desktop = Desktop.getDesktop();
            if (file.exists()) {
                desktop.open(file);
            } else {
                System.out.println("File doesn't exist");
            }
        } catch (IOException e) {
            System.out.println("Error opening the file: " +
e.getMessage());
        }
        //testerPrinter();
    }

    public static void fillInitialArrays()
    {
        numberOfRows = supplyPoints + 1;
```

```
        numberOfColumns = demandPoints + 1;

        int listSize = textFields2.size();

        int i=0;
        int j=0;

        for (i=0; i<=40; i++)
        {
            if(i!=40)
            {
                supply[i]=0;
                demand[i]=0;
            }


            for (j=0; j<=40; j++)
            {
                if(j!=40 && i!=40)
                {
                    cost[i][j]=0;
                    solutionArray[i][j]=0;
                }
                problemArray[i][j]=0;

            }
        }

        i=0;
        j=0;
        for (int k=0;k<listSize;k++)
        {
            JTextField textFieldToGetValue = textFields2.get(k);

            if ((k+1)%numberOfColumns==0)
            {
                supply[i] =
Integer.parseInt(textFieldToGetValue.getText());
                problemArray[i][j] = supply[i];
                i++;
                j=0;
            }
            else if((k+1)>(numberOfRows-1)*numberOfColumns)
            {
                demand[j] =
Integer.parseInt(textFieldToGetValue.getText());
                problemArray[i][j] = demand[j];
                j++;
            }
            else
            {
                cost[i][j] =
Integer.parseInt(textFieldToGetValue.getText());
                problemArray[i][j] = cost[i][j];
                j++;
            }
```

```
    }
}

public static void printTable()
{
    allMessage +="\n";
    for (int i=0; i<numberOfRows;i++)
    {
        for (int j=0;j<numberOfColumns;j++)
        {
            if (i!=numberOfRows-1 || j!=numberOfColumns-1)
            {
                allMessage += problemArray[i][j];
                if (solutionArray[i][j] != 0)
                {
                    allMessage += "(" + solutionArray[i][j] + ")";
                }
                allMessage += "\t\t";
            }
        }
        allMessage +="\n";
    }
    allMessage +="\n";
}

public static void dummyCheck()
{

    int totalSupply, totalDemand;

    totalSupply = 0;
    totalDemand = 0;

    for (int i=0; i<supplyPoints;i++)
    {
        totalSupply += supply[i];
    }

    for (int i=0; i<demandPoints;i++)
    {
        totalDemand += demand[i];
    }

    if (totalSupply>totalDemand)
    {
        numberOfColumns++;
        demand[numberOfColumns-2]=totalSupply-totalDemand;
    }
    else if (totalSupply<totalDemand)
    {
        numberOfRows++;
        supply[numberOfRows-2]=totalDemand-totalSupply;
    }

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

```
            for (int j=0;j<numberOfColumns;j++)
            {
                if (i==numberOfRows-1 && j==numberOfColumns-1)
                {
                    continue;
                }
                else if (i==numberOfRows-1)
                {
                    problemArray[i][j]=demand[j];
                }
                else if (j==numberOfColumns-1)
                {
                    problemArray[i][j]=supply[i];
                }
                else
                {
                    problemArray[i][j]=cost[i][j];
                }
            }
        }

        if (totalSupply>totalDemand)
        {
            allMessage +="Total supply is " + totalSupply + " unit and
total demand is " + totalDemand + " unit\n";
            allMessage +="Thus, total supply > total demand. We add dummy
demand point!\n";
        }
        else if (totalSupply < totalDemand)
        {
            allMessage +="Total supply is " + totalSupply + " unit and
total demand is " + totalDemand + " unit\n";
            allMessage +="Thus, total supply < total demand. We add dummy
supply point!\n";
        }
        else
        {
            allMessage +="Total supply is " + totalSupply + " unit and
total demand is " + totalDemand + " unit\n";
            allMessage +="Thus, total supply = total demand. No need for
dummy point.\n";
        }
    }

    public static void northWestCornerMethod()
    {
        int i=0;
        int j=0;

        while (i<(numberOfRows-1) && j<(numberOfColumns-1))
        {

            allMessage +="Supply point " + (i+1) + " has " + supply[i] +
" unit free capacity and demand point " + (j+1) + " has " + demand[j] + "
unit unsatisfied demand\n";
            int quantity = min(supply[i], demand[j]);
```

```
            allMessage += "Minimum is " + quantity + "\n";

            solutionArray[i][j] = quantity;
            problemArray[i][(numberOfColumns-1)] -= quantity;
            problemArray[(numberOfRows-1)][j] -= quantity;
            supply[i] -= quantity;
            demand[j] -= quantity;

            if (supply[i]==0 && demand[j]==0)
            {
                allMessage += "This exhausts the capacity of supply point
" + (i+1) + " and meets the complete demand of demand point " + (j+1) +
"\n";
                printTable();
                i++;
                j++;
                if (i<(numberOfRows-1) && j<(numberOfColumns-1))
                    allMessage +="We go diagonal!\n";
            }
            else if (supply[i]==0)
            {
                allMessage += "This exhausts the capacity of supply point
" + (i+1) + " and leaves " + (demand[j]) + " unit demand of demand point
" + (j+1) + "\n";
                printTable();
                i++;
                if (i<(numberOfRows-1) && j<(numberOfColumns-1))
                    allMessage +="We go vertically!\n";
            }
            else
            {
                allMessage += "This meets the complete demand of demand
point " + (j+1) + " and leaves " + (supply[i]) + " unit supply of supply
point " + (i+1) + "\n";
                printTable();
                j++;
                if (i<(numberOfRows-1) && j<(numberOfColumns-1))
                    allMessage +="We go horizontally!\n";
            }

        }

        allMessage +="Solution with Northwest Corner Method is ";
        sumProduct();
    }

    public static void sumProduct()
    {

        int totalCost = 0;

        for (int i=0; i<numberOfRows-1;i++)
        {
            for (int j=0; j<numberOfColumns-1;j++)
            {
                totalCost += problemArray[i][j] * solutionArray[i][j];
```

```
            }
        }

        allMessage += totalCost + "\n";
    }

    public static void writeToFile()
    {
        try
        {
            FileWriter myWriter = new FileWriter("solution.txt");
            myWriter.write(allMessage);
            myWriter.close();
        }
        catch (IOException e)
        {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }
    }

    public static void leastCostMethod()
    {

        int minI, minJ;
        minI = -2;
        minJ = -2;
        while(minI != -1 && minJ != -1)
        {
            int minCost = -1;
            int maxAllocation = 0;
            minI = -1;
            minJ = -1;

            for (int i=0;i<(numberOfRows-1);i++)
            {
                if (supply[i]==0)
                    continue;

                for (int j=0;j<(numberOfColumns-1);j++)
                {
                    if (demand[j]==0)
                        continue;

                    if (solutionArray[i][j]==0 && minCost==-1)
                    {
                        minCost = cost[i][j];
                        minI = i;
                        minJ = j;
                        maxAllocation = min(supply[i], demand[j]);
                    }
                    else if (solutionArray[i][j]==0 && cost[i][j] <
minCost)
                    {
                        minCost = cost[i][j];
                        minI = i;
```

```
                        minJ = j;
                        maxAllocation = min(supply[i], demand[j]);
                    }
                    else if(solutionArray[i][j]==0 && cost[i][j]==minCost
&& maxAllocation < min(supply[i], demand[j]))
                    {
                        minCost = cost[i][j];
                        minI = i;
                        minJ = j;
                        maxAllocation = min(supply[i], demand[j]);
                    }
                }
            }

            if (minI == -1 || minJ == -1)
            {
                break;
            }

            int quantity = min(supply[minI], demand[minJ]);
            allMessage += "Min cost is " + minCost + " at supply point "
+ (minI+1) + " and demand point " + (minJ+1) + " with maximum allocation
amount " + quantity + "\n";
            solutionArray[minI][minJ] = quantity;
            supply[minI] -= quantity;
            demand[minJ] -= quantity;
            problemArray[minI][numberOfColumns-1] -= quantity;
            problemArray[numberOfRows-1][minJ] -= quantity;
            if (supply[minI]==0 && demand[minJ]==0)
            {
                allMessage += "This exhausts the capacity of supply point
" + (minI+1) + " and meets the complete demand of demand point " +
(minJ+1) + "\n";
            }
            else if (supply[minI]==0)
            {
                allMessage += "This exhausts the capacity of supply point
" + (minI+1) + " and leaves " + (demand[minJ]) + " unit demand of demand
point " + (minJ+1) + "\n";
            }
            else
            {
                allMessage += "This meets the complete demand of demand
point " + (minJ+1) + " and leaves " + (supply[minI]) + " unit supply of
supply point " + (minI+1) + "\n";
            }
            printTable();
        }

        allMessage +="Solution with Least Cost Method is ";
        sumProduct();
    }

    public static void vogelsApproximationMethod()
    {
```

```
int myCheck=1;
int rowMin1, rowMin2, colMin1, colMin2;
int maxPenalty, maxAllocation, minCost, selectedI, selectedJ;
int maxAllocation2, minCost2, selectedI2, selectedJ2;

while (myCheck!=0)
{
    maxPenalty =-1;
    maxAllocation = -1;
    minCost=-1;
    selectedI=-1;
    selectedJ=-1;

    for (int i=0;i<numberOfRows-1;i++)
        rowPenalty[i]=-1;

    for (int i=0; i<numberOfColumns-1;i++)
        colPenalty[i]=-1;

    for (int i=0; i<numberOfRows-1;i++)
    {
        if (supply[i]==0)
            continue;

        rowMin1=-1;
        rowMin2=-1;

        for (int j=0; j<numberOfColumns-1;j++)
        {
            if (demand[j]==0)
                continue;

            if (rowMin1==-1)
            {
                rowMin1=cost[i][j];
            }
            else if (cost[i][j]<=rowMin1)
            {
                rowMin2=rowMin1;
                rowMin1=cost[i][j];
            }
            else if(rowMin2==-1)
            {
                rowMin2=cost[i][j];
            }
            else if (cost[i][j]<rowMin2)
            {
                rowMin2=cost[i][j];
            }
        }

        if (rowMin2==-1)
        {
            rowPenalty[i]=rowMin1;
        }
        else
```

```
        {
            rowPenalty[i]=rowMin2-rowMin1;
        }
    }

    for (int j=0; j<numberOfColumns-1;j++)
    {
        if (demand[j]==0)
            continue;

        colMin1=-1;
        colMin2=-1;

        for (int i=0; i<numberOfRows-1;i++)
        {
            if (supply[i]==0)
                continue;

            if (colMin1==-1)
            {
                colMin1=cost[i][j];
            }
            else if (cost[i][j]<=colMin1)
            {
                colMin2=colMin1;
                colMin1=cost[i][j];
            }
            else if(colMin2==-1)
            {
                colMin2=cost[i][j];
            }
            else if (cost[i][j]<colMin2)
            {
                colMin2=cost[i][j];
            }
        }

        if (colMin2==-1)
        {
            colPenalty[j]=colMin1;
        }
        else
        {
            colPenalty[j]=colMin2-colMin1;
        }
    }

    for (int i=0; i<numberOfRows-1;i++)
    {
        if (supply[i]==0)
            continue;

        if(maxPenalty==-1 || rowPenalty[i]>maxPenalty)
        {
            for (int j=0;j<numberOfColumns-1;j++)
            {
```

```
                        if (demand[j] == 0)
                            continue;

                        if ((minCost==-1) || (rowPenalty[i]>maxPenalty)
|| (cost[i][j]<minCost) || (minCost==cost[i][j] &&
maxAllocation<min(supply[i], demand[j])))
                        {
                            maxPenalty = rowPenalty[i];
                            maxAllocation = min(supply[i], demand[j]);
                            minCost = cost[i][j];
                            selectedI=i;
                            selectedJ=j;
                        }
                    }

                }
                else if (maxPenalty == rowPenalty[i])
                {
                    minCost2=-1;
                    maxAllocation2=-1;
                    selectedI2=-1;
                    selectedJ2=-1;
                    for (int j=0;j<numberOfColumns-1;j++)
                    {
                        if (demand[j] == 0)
                            continue;
                        if ((minCost2==-1) || (cost[i][j]<minCost2) ||
(minCost2==cost[i][j] && maxAllocation2<min(supply[i], demand[j])))
                        {
                            maxAllocation2 = min(supply[i], demand[j]);
                            minCost2 = cost[i][j];
                            selectedI2=i;
                            selectedJ2=j;
                        }

                    }

                    if (maxAllocation2>maxAllocation)
                    {
                        maxAllocation = maxAllocation2;
                        minCost = minCost2;
                        selectedI = selectedI2;
                        selectedJ = selectedJ2;
                    }

                }
            }

            for (int j=0; j<numberOfColumns-1;j++)
            {
                if (demand[j]==0)
                    continue;

                if(colPenalty[j]>maxPenalty)
                {
                    for (int i=0;i<numberOfRows-1;i++)
```

```
                    {
                        if (supply[i] == 0)
                            continue;

                        if ((colPenalty[j]>maxPenalty) ||
(cost[i][j]<minCost) || (minCost==cost[i][j] &&
maxAllocation<min(supply[i], demand[j])))
                        {
                            maxPenalty = colPenalty[j];
                            maxAllocation = min(supply[i], demand[j]);
                            minCost = cost[i][j];
                            selectedI=i;
                            selectedJ=j;
                        }
                    }

                }
                else if (maxPenalty == colPenalty[j])
                {
                    minCost2=-1;
                    maxAllocation2=-1;
                    selectedI2=-1;
                    selectedJ2=-1;
                    for (int i=0;i<numberOfRows-1;i++)
                    {
                        if (supply[i] == 0)
                            continue;

                        if ((minCost2==-1) || (cost[i][j]<minCost2) ||
(minCost2==cost[i][j] && maxAllocation2<min(supply[i], demand[j])))
                        {
                            maxAllocation2 = min(supply[i], demand[j]);
                            minCost2 = cost[i][j];
                            selectedI2=i;
                            selectedJ2=j;
                        }

                    }

                    if (maxAllocation2>maxAllocation)
                    {
                        maxAllocation = maxAllocation2;
                        minCost = minCost2;
                        selectedI = selectedI2;
                        selectedJ = selectedJ2;
                    }
                }
            }

            int quantity = min(supply[selectedI], demand[selectedJ]);
            allMessage+="Max penalty is " + maxPenalty;

            if (rowPenalty[selectedI]==maxPenalty)
                allMessage+=" in row " + (selectedI+1) + "\n";
            else
                allMessage+=" in column " + (selectedJ+1) + "\n";
```

```
            allMessage += "Min cost is " + minCost + " at supply point "
+ (selectedI+1) + " and demand point " + (selectedJ+1) + " with maximum
allocation amount " + quantity + "\n";
            solutionArray[selectedI][selectedJ] = quantity;
            supply[selectedI] -= quantity;
            demand[selectedJ] -= quantity;
            problemArray[selectedI][numberOfColumns-1] -= quantity;
            problemArray[numberOfRows-1][selectedJ] -= quantity;

            if (supply[selectedI]==0 && demand[selectedJ]==0)
            {
                allMessage += "This exhausts the capacity of supply point
" + (selectedI+1) + " and meets the complete demand of demand point " +
(selectedJ+1) + "\n";
            }
            else if (supply[selectedI]==0)
            {
                allMessage += "This exhausts the capacity of supply point
" + (selectedI+1) + " and leaves " + (demand[selectedJ]) + " unit demand
of demand point " + (selectedJ+1) + "\n";
            }
            else
            {
                allMessage += "This meets the complete demand of demand
point " + (selectedJ+1) + " and leaves " + (supply[selectedI]) + " unit
supply of supply point " + (selectedI+1) + "\n";
            }
            printTable2();

            myCheck = 0;
            for (int i=0;i<numberOfRows-1;i++)
            {
                if (supply[i]>0)
                {
                    myCheck=1;
                    break;
                }
            }
        }
        allMessage +="Solution with Vogel's Approximation Method is ";
        sumProduct();

    }

    public static void printTable2()
    {
        allMessage +="\n";
        for (int i=0; i<numberOfRows;i++)
        {
            for (int j=0;j<numberOfColumns;j++)
            {
                if (i!=numberOfRows-1 || j!=numberOfColumns-1)
                {
                    allMessage += problemArray[i][j];
                    if (solutionArray[i][j] != 0)
```

```
                {
                    allMessage += "(" + solutionArray[i][j] + ")";
                }
                allMessage += "\t\t";
            }
        }
        if (i<numberOfRows-1)
            allMessage +=rowPenalty[i]+ "\n";
        else
            allMessage +="\n";
    }

    for (int j=0;j<numberOfColumns-1;j++)
    {
        allMessage += colPenalty[j];
        allMessage += "\t\t";
    }
    allMessage +="\n\n";
}

public static void russellsApproximationMethod()
{
    int myCheck=0;
    int minCost;
    int selectedI, selectedJ;
    selectedI=0;
    selectedJ=0;

    while(myCheck!=-1)
    {

        for (int i=0;i<numberOfRows-1;i++)
            rowPenalty[i]=-1;

        for (int i=0; i<numberOfColumns-1;i++)
            colPenalty[i]=-1;

        for (int i=0;i<numberOfRows-1;i++)
        {
            if (supply[i]==0)
                continue;

            for (int j=0; j<numberOfColumns-1;j++)
            {
                if (demand[j]==0)
                    continue;

                if (rowPenalty[i]<cost[i][j])
                    rowPenalty[i] = cost[i][j];

            }
        }

        for (int j=0;j<numberOfColumns-1;j++)
        {
            if (demand[j]==0)
```

```
                    continue;

                for (int i=0; i<numberOfRows-1;i++)
                {
                    if (supply[i]==0)
                        continue;

                    if (colPenalty[j]<cost[i][j])
                        colPenalty[j] = cost[i][j];
                }
            }

            minCost = 1;

            for (int i=0;i<numberOfRows-1;i++)
            {
                if (supply[i]==0)
                    continue;

                for (int j=0; j<numberOfColumns-1;j++)
                {
                    if (demand[j]==0)
                        continue;

                    allMessage+="Delta(" + (i+1) + "," + (j+1) + ")=c(" +
(i+1) + "," + (j+1) + ")-(U(" + (i+1) + ")+V(" + (j+1) + ")=" +
cost[i][j] + "-(" + rowPenalty[i] + "+" + colPenalty[j] + ")=" +
(cost[i][j]-(rowPenalty[i] + colPenalty[j])) + "\n";

                    if (minCost>(cost[i][j]-(rowPenalty[i] +
colPenalty[j])))
                    {
                        minCost = (cost[i][j]-(rowPenalty[i] +
colPenalty[j]));
                        selectedI = i;
                        selectedJ = j;
                    }
                }
            }

            int quantity = min(supply[selectedI], demand[selectedJ]);
            allMessage+="Min delta is " + minCost + " at supply point " +
(selectedI+1) + " and demand point " + (selectedJ+1) + " with maximum
allocation amount " + quantity + "\n";

            solutionArray[selectedI][selectedJ] = quantity;
            supply[selectedI] -= quantity;
            demand[selectedJ] -= quantity;
            problemArray[selectedI][numberOfColumns-1] -= quantity;
            problemArray[numberOfRows-1][selectedJ] -= quantity;

            if (supply[selectedI]==0 && demand[selectedJ]==0)
            {
                allMessage += "This exhausts the capacity of supply point
" + (selectedI+1) + " and meets the complete demand of demand point " +
(selectedJ+1) + "\n";
```

```
            }
            else if (supply[selectedI]==0)
            {
                allMessage += "This exhausts the capacity of supply point
" + (selectedI+1) + " and leaves " + (demand[selectedJ]) + " unit demand
of demand point " + (selectedJ+1) + "\n";
            }
            else
            {
                allMessage += "This meets the complete demand of demand
point " + (selectedJ+1) + " and leaves " + (supply[selectedI]) + " unit
supply of supply point " + (selectedI+1) + "\n";
            }
            printTable2();
            myCheck=-1;

            for (int i=0;i<numberOfRows-1;i++)
            {
                if (supply[i]>0)
                {
                    myCheck=0;
                    break;
                }
            }
        }

        allMessage +="Solution with Russell's Approximation Method is ";
        sumProduct();
    }

    public static void modiMethod()
    {
        int myControl=1;
        int myControl2=1;
        int rowOrCol=-1;
        int maxNumAllocation=0;
        int currentAllocation=0;
        int relatedRowOrCol=-1;
        int i1,j1,i2,j2,i3,j3,i4,j4;
        int mindij;
        int minAllocation=0;
        i1=-1;
        j1=-1;
        i2=-1;
        j2=-1;
        i3=-1;
        j3=-1;
        i4=-1;
        j4=-1;

        allMessage+="First, we find an initial basic feasible solution by
using Vogel's Approximation Method \n";
        vogelsApproximationMethod();
        allMessage+="\n";

        while (myControl==1)
```

```
        {
            maxNumAllocation=0;

            for (int i=0;i<numberOfRows-1;i++)
            {
                currentAllocation=0;

                for (int j=0; j<numberOfColumns-1;j++)
                {
                    if (solutionArray[i][j]>0)
                    {
                        currentAllocation++;
                    }
                }

                if (currentAllocation>maxNumAllocation)
                {
                    maxNumAllocation = currentAllocation;
                    rowOrCol = 1;
                    relatedRowOrCol=i;
                }
            }

            for (int j=0; j<numberOfColumns-1;j++)
            {
                currentAllocation=0;

                for (int i=0;i<numberOfRows-1;i++)
                {
                    if (solutionArray[i][j]>0)
                    {
                        currentAllocation++;
                    }
                }

                if (currentAllocation>maxNumAllocation)
                {
                    maxNumAllocation = currentAllocation;
                    rowOrCol = 2;
                    relatedRowOrCol=j;
                }
            }

            allMessage+="Maximum number of allocation is ";
            if(rowOrCol==1)
                allMessage+="in row ";
            else
                allMessage+="in column";
            allMessage+=" " + (relatedRowOrCol+1) + " thus, substituting
";
            if(rowOrCol==1)
                allMessage+="u(" + (relatedRowOrCol+1) + ")=0, we
get:\n";
            else
                allMessage+="v(" + (relatedRowOrCol+1) + ")=0, we
get:\n";
```

```
                for (int i=0;i<numberOfRows-1;i++)
                    supply[i]=0;

                for(int j=0;j<numberOfColumns-1;j++)
                    demand[j]=0;

                if (rowOrCol==1)
                {
                    rowPenalty[relatedRowOrCol]=0;
                    supply[relatedRowOrCol]=1;
                }
                else
                {
                    colPenalty[relatedRowOrCol]=0;
                    demand[relatedRowOrCol]=1;
                }

                myControl2=1;
                while (myControl2==1)
                {
                    for (int i=0;i<numberOfRows-1;i++)
                    {
                        if (supply[i]==0)
                            continue;

                        for (int j=0; j<numberOfColumns-1;j++)
                        {
                            if (demand[j]==1)
                                continue;

                            if(solutionArray[i][j]>0 && demand[j]==0)
                            {
                                colPenalty[j]=cost[i][j]-rowPenalty[i];
                                demand[j]=1;
                                allMessage +="v(" + (j+1) + ") = c(" + (i+1)
+ "," + (j+1) + ") - u(" + (i+1) + ")--> " + cost[i][j] + " - " +
rowPenalty[i] + " = " + colPenalty[j] + "\n";
                            }
                        }
                    }

                    for (int j=0;j<numberOfColumns-1;j++)
                    {
                        if (demand[j]==0)
                            continue;

                        for (int i=0; i<numberOfRows-1;i++)
                        {
                            if (supply[i]==1)
                                continue;

                            if(solutionArray[i][j]>0)
                            {
                                rowPenalty[i]=cost[i][j]-colPenalty[j];
                                supply[i]=1;
```

```
                            allMessage +="u(" + (i+1) + ") = c(" + (i+1)
+ "," + (j+1) + ") - v(" + (j+1) + ")--> " + cost[i][j] + " - " +
colPenalty[j] + " = " + rowPenalty[i] + "\n";
                        }
                    }
                }

                myControl2=0;
                for (int i=0; i<numberOfRows-1;i++)
                {
                    if(supply[i] == 0)
                    {
                        myControl2=1;
                        break;
                    }
                }

                if (myControl2==0)
                {
                    for (int j=0;j<numberOfColumns-1;j++)
                    {
                        if(demand[j]==0)
                        {
                            myControl2=1;
                            break;
                        }
                    }
                }
            }

            printTable2();

            myControl=0;
            mindij=0;
            for (int i=0;i<numberOfRows-1;i++)
            {
                for (int j=0;j<numberOfColumns-1;j++)
                {
                    if (solutionArray[i][j]==0)
                    {
                        allMessage+="d(" + (i+1) + "," + (j+1) + ") = c("
+ (i+1) + "," + (j+1) + ") - (u(" + (i+1) + ") + v(" + (j+1) + ")) = " +
cost[i][j] + " - (" + rowPenalty[i] + " + " + colPenalty[j] + ") = " +
(cost[i][j] - rowPenalty[i] - colPenalty[j]) + "\n";
                    }

                    if (cost[i][j] - rowPenalty[i] - colPenalty[j]<0)
                    {
                        myControl=1;
                        if (mindij>cost[i][j] - rowPenalty[i] -
colPenalty[j])
                        {
                            mindij = cost[i][j] - rowPenalty[i] -
colPenalty[j];
                            i1=i;
                            j1=j;
```

```
                        }
                    }
                }
            }

            if (myControl==0)
            {
                allMessage+="All d(i,j) values are non-negative. The
solution is optimal.\n";
                printTable();
                continue;
            }

            allMessage+="The minimum negative value from all d(i,j)
(opportunity cost) = d(" + (i1+1) + "," + (j1+1) + ") = " + mindij +
"\n";

            for (int i=0; i<numberOfRows-1;i++)
            {
                if(solutionArray[i][j1]!=0)
                {
                    for(int j=0; j<numberOfColumns-1;j++)
                    {
                        if(solutionArray[i1][j]!=0 &&
solutionArray[i][j]!=0)
                        {
                            i2=i1;
                            i3=i;
                            i4=i;
                            j2=j;
                            j3=j1;
                            j4=j;
                            minAllocation = min(solutionArray[i2][j2],
solutionArray[i3][j3]);
                        }
                    }
                }
            }

            allMessage+="The closed loop is S" + (i1+1) + "D" + (j1+1) +
" - S" + (i2+1) + "D" + (j2+1) + " - S" + (i4+1) + "D" + (j4+1) + " - S"
+ (i3+1) + "D" + (j3+1) + "\n";
            allMessage+="We can allocate " + minAllocation + " unit from
S" + (i2+1) + "D" + (j2+1) + " and S" + (i3+1) + "D" + (j3+1) + " to S" +
+ (i1+1) + "D" + (j1+1) + " and S" + (i4+1) + "D" + (j4+1) + "\n";
            solutionArray[i1][j1] += minAllocation;
            solutionArray[i2][j2] -= minAllocation;
            solutionArray[i3][j3] -= minAllocation;
            solutionArray[i4][j4] += minAllocation;
            allMessage+="New solution is: \n";
            printTable();
            allMessage+="We continue with the next iteration!\n";
        }

        allMessage +="Optimal solution with MODI Method is ";
        sumProduct();
```

```
    }

    public static void steppingStoneMethod()
    {
        int myControl=1;
        int minCost=0;
        int maxAllocation = 0;
        int totalCost = 0;

        int i1=-1;
        int i2=-1;
        int i3=-1;
        int i4=-1;

        int j1=-1;
        int j2=-1;
        int j3=-1;
        int j4=-1;

        int fi1=-1;
        int fi2=-1;
        int fi3=-1;
        int fi4=-1;

        int fj1=-1;
        int fj2=-1;
        int fj3=-1;
        int fj4=-1;

        allMessage+="First, we find an initial basic feasible solution by
using Vogel's Approximation Method \n";
        vogelsApproximationMethod();
        allMessage+="\n";

        while (myControl==1)
        {
            myControl = 0;
            minCost = 0;
            allMessage += "Create closed loop for unoccupied cells, we
get:\n";

            for (int i=0;i<numberOfRows-1;i++)
            {
                for (int j=0; j<numberOfColumns-1;j++)
                {
                    if (solutionArray[i][j]==0)
                    {
                        i1=i;
                        j1=j;

                        for (int ii=0; ii<numberOfRows-1;ii++)
                        {
                            if(solutionArray[ii][j1]!=0)
                            {
                                for(int jj=0; jj<numberOfColumns-1;jj++)
                                {
```

```
                                    if(solutionArray[i1][jj]!=0 &&
solutionArray[ii][jj]!=0)
                                    {
                                        i2=i1;
                                        i3=ii;
                                        i4=ii;

                                        j2=jj;
                                        j3=j1;
                                        j4=jj;

                                        totalCost = cost[i1][j1] -
cost[i2][j2] - cost[i3][j3] + cost[i4][j4];

                                        allMessage+="S" + (i1+1) + "D" +
(j1+1) + " - S" + (i2+1) + "D" + (j2+1) + " - S" + (i4+1) + "D" + (j4+1)
+ " - S" + (i3+1) + "D" + (j3+1) + " with total cost " + totalCost +
"\n";

                                        if (totalCost<minCost)
                                        {
                                            myControl=1;
                                            fi1=i1;
                                            fi2=i2;
                                            fi3=i3;
                                            fi4=i4;
                                            fj1=j1;
                                            fj2=j2;
                                            fj3=j3;
                                            fj4=j4;
                                            minCost = totalCost;
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }

            if (myControl==0)
            {
                allMessage+="All cost values are non-negative. The
solution is optimal.\n";
                printTable();
                continue;
            }

            maxAllocation = min(solutionArray[fi2][fj2],
solutionArray[fi3][fj3]);

            allMessage+="We have negative cost, thus the solution is not
optimal.\n";
            allMessage+="The min cost closed loop is S" + (fi1+1) + "D" +
(fj1+1) + " - S" + (fi2+1) + "D" + (fj2+1) + " - S" + (fi4+1) + "D" +
(fj4+1) + " - S" + (fi3+1) + "D" + (fj3+1) + "\n";
```

```
            allMessage+="We can allocate " + maxAllocation + " unit from
S" + (fi2+1) + "D" + (fj2+1) + " and S" + (fi3+1) + "D" + (fj3+1) + " to
S" + + (fi1+1) + "D" + (fj1+1) + " and S" + (fi4+1) + "D" + (fj4+1) +
"\n";
            solutionArray[fi1][fj1] += maxAllocation;
            solutionArray[fi2][fj2] -= maxAllocation;
            solutionArray[fi3][fj3] -= maxAllocation;
            solutionArray[fi4][fj4] += maxAllocation;
            allMessage+="New solution is: \n";
            printTable();
            allMessage+="We continue with the next iteration!\n";

        }
        allMessage +="Optimal solution with Stepping Stone Method is ";
        sumProduct();
    }



    public static void testerPrinter()
    {
        String s1="";
        String s2="";
        String s3="";
        String s4="";
        String s5="";
        String s6="";
        String s7="";
        String s8="";
        String s9="";

        s1 = "Number of supply points is " + supplyPoints;
        s2 = "Number of demand points is " + demandPoints;
        s3 = "Number of rows " + numberOfRows;
        s4 = "Number of columns " + numberOfColumns;

        s5 = "Supply array [";
        for (int i=0; i<numberOfRows-1;i++)
        {
            s5 += supply[i] + " ";
        }
        s5 += "]";

        s6 = "Demand array [";
        for (int i=0; i<numberOfColumns-1;i++)
        {
            s6 += demand[i] + " ";
        }
        s6 += "]";

        s7 = "Cost array [";
        for (int i=0; i<numberOfRows-1;i++)
        {
            for (int j=0; j<numberOfColumns-1;j++)
            {
                s7 += cost[i][j] + " ";
```

```
            }
            s7 += "\n";
        }
        s7 += "]";

        s8 = "Problem array [";
        s9 = "Solution array [";
        for (int i=0; i<numberOfRows;i++)
        {
            for (int j=0; j<numberOfColumns;j++)
            {
                s8 += problemArray[i][j] + " ";
                s9 += solutionArray[i][j] + " ";
            }
            s8 += "\n";
            s9 += "\n";
        }
        s8 += "]";
        s9 += "]";

        JOptionPane.showMessageDialog(null, s1);
        JOptionPane.showMessageDialog(null, s2);
        JOptionPane.showMessageDialog(null, s3);
        JOptionPane.showMessageDialog(null, s4);
        JOptionPane.showMessageDialog(null, s5);
        JOptionPane.showMessageDialog(null, s6);
        JOptionPane.showMessageDialog(null, s7);
        JOptionPane.showMessageDialog(null, s8);
        JOptionPane.showMessageDialog(null, s9);
    }
}
```