PENYELESAIAN PERSOALAN *TRAVELING SALESPERSON PROBLEM (TSP)* MENGGUNAKAN ALGORTIMA *BRANCH & BOUND*

Laporan Tugas Kecil 3

IF 2211 - Strategi Algoritma

Oleh:
AHMAD AIDIN
NIM: 13513020

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO & INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

April 2017

A. Persoalan

Pada tugas kecil ini, persoalan yang akan diselesaikan adalah persoalan *Traveling Salesperson Problem (TSP)* yaitu diberikan suatu graf yang yang memiliki bobot kemudian ditentukan rute kunjungan terpendek dari suatu simpul untuk mengunjungi seluruh lain. Graf yang diberikan adalah graf lengkap, yaitu graf yang satu simpul dengan seluruh simpul lain pasti terhubung dengan satu sisi.

Untuk menyelesaikan persoalan diatas digunakan algoritma *branch & bound* dengan dua pendekatan untuk menentukan nilai *bound*. Pendekatan pertama adalah dengan matriks bobot tereduksi. Pendekatan kedua adalah dengan bobot tur lengkap.

B. Source Code (JAVA)

CostMtx.java

```java
package com.stima;

import java.util.ArrayList;

public class CostMatrix {
    private int[][] mtx;
    private int cost;
    private int size;
    private static final int INFINITY = 999;

    public CostMatrix(int size){
        this.size = size;
        cost = 0;
        mtx = new int [size][size];
    }
    public CostMatrix(CostMatrix costMatrix){
        this.size = costMatrix.size;
        cost = costMatrix.cost;
        this.mtx = new int [size][size];
        for(int i=0; i<size; i++){
            for(int j=0; j<size; j++){
                this.mtx[i][j] = costMatrix.getMtx()[i][j];
            }
        }
    }
    public CostMatrix(int[][] mtx, int size, int cost){
        this.size = size;
        this.cost = cost;
        this.mtx = new int [size][size];
        for(int i=0; i<size; i++){
            for(int j=0; j<size; j++){
                this.mtx[i][j] = mtx[i][j];
            }
        }
    }

    public void addCost(int val){
        cost+=val;
    }

    public int[][] getMtx(){
        return mtx;
    }

    public int getSize(){
        return size;
    }

    public int getCost(){
        return cost;
    }

    public void change(int i, int j, int val){
        mtx[i][j] = val;
    }

    public void reduceCost(){
        //melakukan reduksi bobot pada matrix
        //mengupdate nilai cost
        for(int i=0; i<size; i++){
            int min =0;
            boolean found = false;
            int idxMin = 0;
            while(!found && idxMin <size){
                if(mtx[i][idxMin]!= INFINITY){
                    found = true;
                    min = mtx[i][idxMin];
                } else{
                    idxMin++;
                }
            }

            for(int j= idxMin; j<size; j++){
                if(mtx[i][j]<min){
                    min = mtx[i][j];
                }
            }

            cost+=min;
            if(min!=0) {
                for (int j = 0; j < size; j++) {
                    if (mtx[i][j] != INFINITY) {
                        mtx[i][j] -= min;
                    }
                }
            }
        }

        for(int i=0; i<size; i++) {
            int min =0;
            boolean found = false;
            int idxMin = 0;
            while(!found && idxMin <size){
                if(mtx[idxMin][i]!= INFINITY){
                    found = true;
                    min = mtx[idxMin][i];
                } else{
                    idxMin++;
                }
            }

            for(int j= idxMin; j<size; j++){
                if(mtx[j][i]<min){
                    min = mtx[j][i];
                }
            }

            cost += min;
            if(min!=0) {
                for (int j = 0; j < size; j++) {
                    if (mtx[j][i] != INFINITY) {
                        mtx[j][i] -= min;
                    }
                }
            }
        }
    }

    public void findFullTourCost(ArrayList<Integer> aliveNode, int nextNode){
        //mendapatkan bobot tur lengkap dari suatu matrix jika sudah melalui
```

```
//beberapa simpul aliveNode dan akan menuju nextNode
//mengupdate nilai cost
ArrayList<Integer> mandatoryNode = new ArrayList<>(aliveNode);
mandatoryNode.add(nextNode);
cost = 0;

for(int i=0; i<size; i++){
    int min1=0;
    int min2=0;
    if(mandatoryNode.size()<=1 || !mandatoryNode.contains(i)){
        boolean found = false;
        int idxMin1 = 0;

        if(mtx[i][0]==INFINITY){
            idxMin1++;
        }

        min1 = mtx[i][idxMin1];

        for(int j= idxMin1+1; j<size; j++){
            if(mtx[i][j]<min1){
                min1 = mtx[i][j];
                idxMin1=j;
            }
        }

        int idxMin2 = 0;
        while(!found && idxMin2 <size){
            if(mtx[i][idxMin2]!= INFINITY && idxMin2!=idxMin1){
                found = true;
                min2 = mtx[i][idxMin2];
            } else{
                idxMin2++;
            }
        }

        for(int j= idxMin2+1; j<size; j++){
            if(mtx[i][j]<min2 && j!=idxMin1){
                min2 = mtx[i][j];
            }
        }
    } else {
        int idx = mandatoryNode.lastIndexOf(i);
        if (idx == 0) {
```

```
                    min1 = mtx[i][mandatoryNode.get(idx+1)];
                    int idxMin = 0;
                    while(mtx[i][idxMin]==INFINITY ||
IdxMin==mandatoryNode.get(idx+1)){
                        idxMin++;
                    }

                    min2 = mtx[i][idxMin];
                    for(int j= idxMin+1; j<size; j++){
                        if(mtx[i][j]<min1 && idxMin!=mandatoryNode.get(idx+1)){
                            min2 = mtx[i][j];
                        }
                    }
                } else if (mandatoryNode.lastIndexOf(i) == mandatoryNode.size()-1)
                {
                    min1 = mtx[i][mandatoryNode.get(idx-1)];

                    int idxMin = 0;

                    while(mtx[i][idxMin]==INFINITY ||
idxMin==mandatoryNode.get(idx-1)){
                        idxMin++;
                    }

                    min2 = mtx[i][idxMin];

                    for(int j= idxMin+1; j<size; j++){
                        if(mtx[i][j]<min1 && idxMin!=mandatoryNode.get(idx-1)){
                            min2 = mtx[i][j];
                        }
                    }
                } else {
                    min1 = mtx[i][mandatoryNode.get(idx+1)];
                    min2 = mtx[i][mandatoryNode.get(idx-1)];
                }
            }
            cost+=min1;
            cost+=min2;
        }
        cost/=2;
    }

    public void printContent(){
        for(int i=0;i<size;i++){
            for(int j=0;j<size;j++){
                System.out.print(mtx[i][j]+" ");
            }
            System.out.println(" ");
        }
    }
}
```

TSP.java

```
package com.stima;

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Scanner;

public class TSP {
    CostMatrix costMatrix;
    ArrayList<Integer> aliveNode;
    int currentNode;
    private static final int INFINITY = 999;
    int risenNodeTree;

    public TSP(String filename) {
        ArrayList<Integer> content = new ArrayList<>();

        try {
            Scanner scanner = new Scanner(new File(filename));
            while (scanner.hasNextInt()) {
                content.add(scanner.nextInt());
            }
        } catch (IOException e){

        }

        int size = (int) Math.sqrt(content.size());
        int[][] mtxGraph = new int[size][size];

        for(int i=0; i<size; i++){
            for(int j=0; j<size; j++){
                int elmt = content.get(i*size+j);
                mtxGraph[i][j]=elmt;
            }
        }
        aliveNode = new ArrayList<>();
        currentNode = 0;
        risenNodeTree = 0;
        costMatrix = new CostMatrix(mtxGraph,size,0);
    }

    public TSP(CostMatrix costMatrix, int size, ArrayList<Integer> aliveNode, int
currentNode, int cost, int risenNodeTree){
        this.costMatrix = new CostMatrix(costMatrix);
        for (int i =0; i<aliveNode.size(); i++){
            this.aliveNode.add(aliveNode.get(i));
        }
        this.currentNode = currentNode;
        this.risenNodeTree = risenNodeTree;
    }

    public TSP( TSP tsp){
        this.costMatrix = new CostMatrix(tsp.costMatrix);
        aliveNode = new ArrayList<>();
        for (int i =0; i<tsp.aliveNode.size(); i++){
            this.aliveNode.add(tsp.aliveNode.get(i));
        }
        this.currentNode = tsp.currentNode;
        this.risenNodeTree= tsp.risenNodeTree;
    }

    public boolean isSolution(){
        boolean getSolution= true;
        int i=0;
        while(getSolution && i< costMatrix.getSize()) {
            if(!aliveNode.contains(i)){
                getSolution=false;
            }
            i++;
        }
        return getSolution;
    }

    public ArrayList<Integer> riseChild() {
        ArrayList<Integer> result = new ArrayList<>();
        for (int i = 0; i < costMatrix.getSize(); i++) {
            if (!aliveNode.contains(i)) {
                result.add(i);
            }
        }
        return result;
    }

    public void initReducedCostMtx(){
        aliveNode.add(currentNode);
        costMatrix.reduceCost();
        risenNodeTree = 1;
    }

    public void reducedCostMatrix(){
        if(isSolution()){

        } else {
            ArrayList<Integer> children = riseChild();
            CostMatrix[] childrenMtx= new CostMatrix[children.size()];

            int size = costMatrix.getSize();

            for(int idx=0; idx<children.size(); idx++){
                CostMatrix newCostMatrix = new CostMatrix(costMatrix);
                int addition =
costMatrix.getMtx()[currentNode][children.get(idx)];
                newCostMatrix.addCost(addition);
                for(int i=0; i<size; i++){
                    newCostMatrix.change(currentNode,i,INFINITY);
                }
                for(int i=0; i<size; i++){
                    newCostMatrix.change(i,children.get(idx),INFINITY);
                }
                newCostMatrix.change(children.get(idx),0, INFINITY);
                newCostMatrix.reduceCost();
                childrenMtx[idx] = newCostMatrix;
            }
            risenNodeTree+=children.size();
            if(childrenMtx.length!=0) {
                int minCost = childrenMtx[0].getCost();
                int idxMin = 0;
                for (int idx = 1; idx < children.size(); idx++) {
                    if(childrenMtx[idx].getCost()<minCost){
                        minCost = childrenMtx[idx].getCost();
                        idxMin = idx;
                    }
                }
                this.costMatrix = childrenMtx[idxMin];
                currentNode = children.get(idxMin);
                aliveNode.add(currentNode);
                reducedCostMatrix();
            }
        }
    }


    public void initFullTourCost(){
        costMatrix.findFullTourCost(aliveNode,currentNode);
        aliveNode.add(0);
        risenNodeTree = 1;
    }

    public void fullTourCost(){
        if(isSolution()){

        } else {
            ArrayList<Integer> children = riseChild();

            CostMatrix[] childrenMtx= new CostMatrix[children.size()];

            int size = costMatrix.getSize();

            for(int idx=0; idx<children.size(); idx++){
                CostMatrix newCostMatrix = new CostMatrix(costMatrix);
                newCostMatrix.findFullTourCost(aliveNode,children.get(idx));
                childrenMtx[idx] = newCostMatrix;
            }
            risenNodeTree+=children.size();
            if(childrenMtx.length!=0) {
                ArrayList<TSP> nextOpt = new ArrayList<>();

                int minCost = childrenMtx[0].getCost();
                int idxMin = 0;
                for (int idx = 1; idx < children.size(); idx++) {
                    if (childrenMtx[idx].getCost() < minCost) {
                        minCost = childrenMtx[idx].getCost();
                        idxMin = idx;
                    }
                }
```

```java
                for (int idx = 0; idx < children.size(); idx++) {
                    if(childrenMtx[idx].getCost()==minCost) {
                        TSP nextChild = new TSP(this);
                        nextChild.costMatrix = childrenMtx[idx];
                        nextChild.currentNode = children.get(idx);
                        nextChild.aliveNode.add(nextChild.currentNode);
                        nextChild.fullTourCost();
                        risenNodeTree = nextChild.risenNodeTree;
                        nextOpt.add(nextChild);
                    }
                }

                idxMin = 0;
                minCost = nextOpt.get(0).costMatrix.getCost();
                for (int i = 1; i<nextOpt.size(); i++){
                    if(nextOpt.get(i).costMatrix.getCost()==minCost){
                        idxMin = i;
                    }
                }

                this.costMatrix = nextOpt.get(idxMin).costMatrix;
                this.currentNode = nextOpt.get(idxMin).currentNode;
                this.aliveNode = nextOpt.get(idxMin).aliveNode;
                this.risenNodeTree = nextOpt.get(idxMin).risenNodeTree;
            }
        }
    }

    public void printContent(){
        for(int i = 0; i< costMatrix.getSize(); i++){
            for(int j = 0; j< costMatrix.getSize(); j++){
                System.out.print(costMatrix.getMtx()[i][j]+" ");
            }
            System.out.println(" ");
        }
    }

    public void printPath(){
        for(int i = 0; i< aliveNode.size(); i++){
         System.out.print((aliveNode.get(i)+1)+" ");
        }
    }
}
```

GraphDrawer.java

```java
package com.stima;

import javax.swing.JFrame;
import com.mxgraph.swing.mxGraphComponent;
import com.mxgraph.view.mxGraph;
import java.awt.*;
import java.util.ArrayList;

public class GraphDrawer extends JFrame
{
    public static final String REDUCED_COST = "reduceCostMatrix";
    public static final String FULL_TOUR_COST = "fullTourCost";
    private static final long serialVersionUID = -2707712944901661771L;
    private final int SIZE = 200;
    private final Point POS = new Point(500, 300);
    private Point[] pointsRoute;
    private ArrayList<Integer> routeResult;
    int[][] mtx;
    private String title;

    public int getNextNode(int node){
        int i = routeResult.indexOf(node);
        i++;
        i%=routeResult.size();
        return routeResult.get(i);
    }

    public GraphDrawer(String title, int[][] mtx){
        super(title);
        int size = mtx.length;
        this.title = title;
        this.mtx = new int[size][size];
        for(int i = 0; i<size; i++){
            for(int j = 0; j< size; j++) {
                this.mtx[i][j] = mtx[i][j];
            }
        }
    }

    public void draw(ArrayList<Integer> routeResult)
    {
        int size = mtx.length;
        pointsRoute = new Point[size];

        this.routeResult = new ArrayList<>(routeResult);

        for(int i =0; i<size; i++){
            int x = (int) (POS.x + SIZE * Math.cos(i * 2 * Math.PI / size));
            int y = (int) (POS.y + SIZE * Math.sin(i * 2 * Math.PI / size));
            Point p = new Point(x,y);
            pointsRoute[i] = p;
        }

        Object [] vertex= new Object[size];
        mxGraph graph = new mxGraph();
        Object parent = graph.getDefaultParent();

        graph.getModel().beginUpdate();
        try
        {   //mencetak semua vertex graph
            for(int i =0; i<size; i++) {
                int x = pointsRoute[i].x;
                int y = pointsRoute[i].y;
                String name = ""+(i+1);
                Object v = graph.insertVertex(parent, null, name, x, y, 30,
                        30,"rounded=true;strokeColor=red;fillColor=yellow");
                vertex[i]=v;
            }




            if(title.equals(FULL_TOUR_COST)) {//graf tidak berarah
                //mencetak semua edge graph
                for (int i = 0; i < size; i++) {
                    for (int j = i + 1; j < size; j++) {
                        String style;
                        if (getNextNode(i) == j || getNextNode(j) == i) {
                            style = "endArrow=none;strokeColor=blue";
                        } else {
                            style = "endArrow=none;";
                        }
                        graph.insertEdge(parent, null, null, vertex[i], vertex[j],
style);
                    }
                }
```

```
          } else { //graf berarah
             //mencetak semua edge graph
             for (int i = 0; i < size; i++) {
                for (int j = 0; j < size; j++) {
                   String style;
                   if(i!=j) {
                      if (getNextNode(i) == j) {
                         style = "strokeColor=blue";
                      } else {
                         style = "";
                      }
                      graph.insertEdge(parent, null, null, vertex[i],
vertex[j], style);
                   }
                }
             }
          }
       }

       finally {
          graph.getModel().endUpdate();
       }

       mxGraphComponent graphComponent = new mxGraphComponent(graph);
       getContentPane().add(graphComponent);
       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       setSize(400, 320);
       setVisible(true);
    }
}
```

Main.java

```
package com.stima;

public class Main {
   public static final String REDUCED_COST = "reduceCostMatrix";
   public static final String FULL_TOUR_COST = "fullTourCost";

   public static void main(String[] args) {
      TSP tspReducedCostMtx1 = new TSP("input1.txt");
      TSP tspReducedCostMtx2 = new TSP("input2.txt");
      TSP tspFullTour1= new TSP("input3.txt");
      TSP tspFullTour2 = new TSP("input4.txt");
      long startTime;
```

```
        long endTime;
        long duration;

        //Reduced Cost Matrix 1
        GraphDrawer drawer1 = new
GraphDrawer(REDUCED_COST,tspReducedCostMtx1.costMatrix.getMtx());
        startTime = System.nanoTime();
        tspReducedCostMtx1.initReducedCostMtx();
        tspReducedCostMtx1.reducedCostMatrix();
        endTime = System.nanoTime();
        duration = (endTime - startTime)/1000000;

        System.out.print("rute terpendek:");
        tspReducedCostMtx1.printPath();
        System.out.println();
        System.out.println("bobot: " +
tspReducedCostMtx1.costMatrix.getCost());
        System.out.println("jumlah simpul yang dibangkintkan: " +
tspReducedCostMtx1.risenNodeTree);
        System.out.println("Excecution time: " + duration +" ms");
        drawer1.draw(tspReducedCostMtx1.aliveNode);

        System.out.println();
        System.out.println();

        //Reduced Cost Matrix 2
        GraphDrawer drawer2 = new
GraphDrawer(REDUCED_COST,tspReducedCostMtx2.costMatrix.getMtx());
        startTime = System.nanoTime();
        tspReducedCostMtx2.initReducedCostMtx();
        tspReducedCostMtx2.reducedCostMatrix();
        endTime = System.nanoTime();
        duration = (endTime - startTime)/1000000;

        System.out.print("rute terpendek :");
        tspReducedCostMtx2.printPath();
        System.out.println();
        System.out.println("bobot: " +
tspReducedCostMtx2.costMatrix.getCost());
        System.out.println("jumlah simpul yang dibangkintkan: " +
tspReducedCostMtx2.risenNodeTree);
        System.out.println("Excecution time: " + duration +" ms");
        drawer2.draw(tspReducedCostMtx2.aliveNode);
```

```java
        System.out.println();
        System.out.println();

        //Full Tour Cost 1
        GraphDrawer drawer3 = new
GraphDrawer(FULL_TOUR_COST,tspFullTour1.costMatrix.getMtx());
        startTime = System.nanoTime();
        tspFullTour1.initFullTourCost();
        tspFullTour1.fullTourCost();
        endTime = System.nanoTime();
        duration = (endTime - startTime)/1000000;

        System.out.print("rute terpendek :");
        tspFullTour1.printPath();
        System.out.println();
        System.out.println("bobot: " + tspFullTour1.costMatrix.getCost());
        System.out.println("jumlah simpul yang dibangkintkan: " +
tspFullTour1.risenNodeTree);
        System.out.println("Excecution time: " + duration +" ms");
        drawer3.draw(tspFullTour1.aliveNode);

        System.out.println();
        System.out.println();

        //Full Tour Cost 2
        GraphDrawer drawer4 = new
GraphDrawer(FULL_TOUR_COST,tspFullTour2.costMatrix.getMtx());
        startTime = System.nanoTime();
        tspFullTour2.initFullTourCost();
        tspFullTour2.fullTourCost();
        endTime = System.nanoTime();
        duration = (endTime - startTime)/1000000;

        System.out.print("rute terpendek :");
        tspFullTour2.printPath();
        System.out.println();
        System.out.println("bobot: " + tspFullTour2.costMatrix.getCost());
        System.out.println("jumlah simpul yang dibangkintkan: " +
tspFullTour2.risenNodeTree);
        System.out.println("Excecution time: " + duration +" ms");
        drawer4.draw(tspFullTour2.aliveNode);
    }
}
```

C. Input dan Output

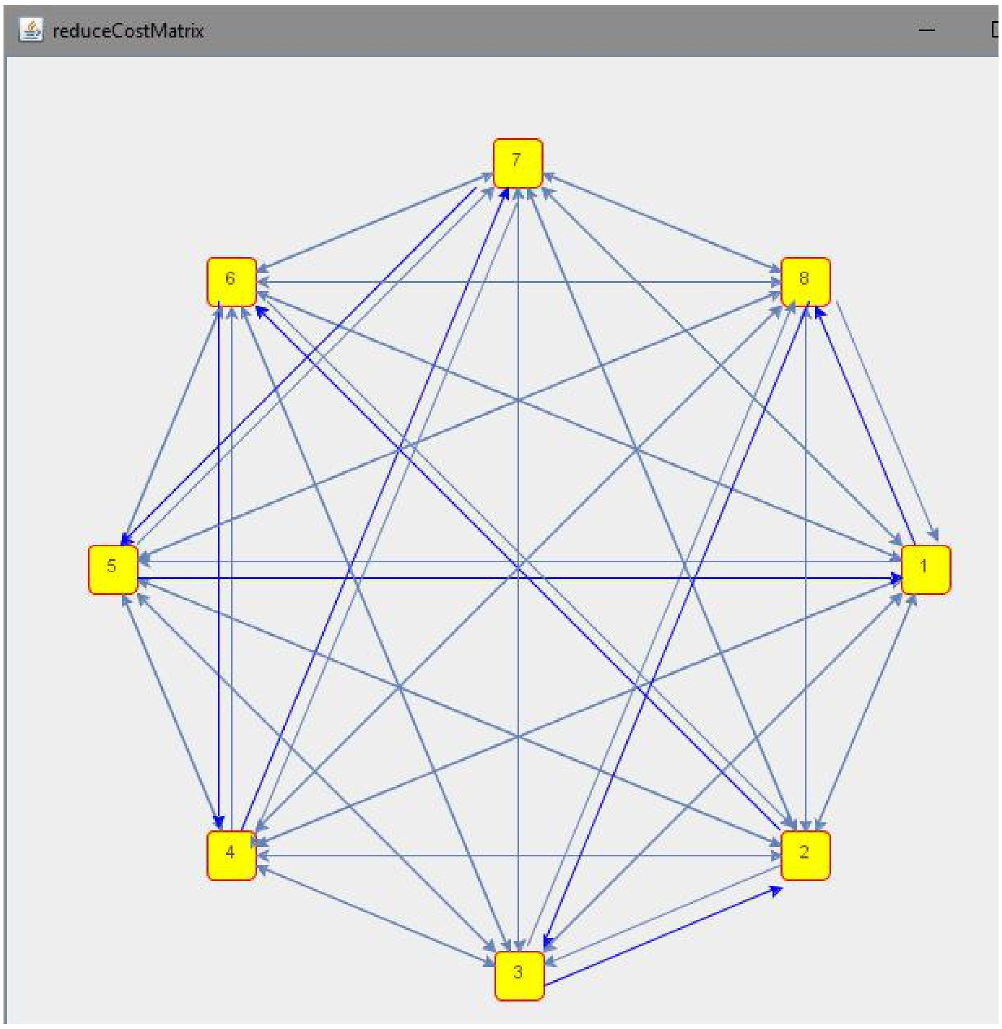
Data input berupa matrix graf berbobot. Simbol ∞ digantikan dengan angka 999.
Sisi yang berwarna biru pada gambar graf output adalah rute terpendek yang diperoleh

TSP B&B dengan Reduced Cost Matrix

Input:

```
999 16 27 89 34 65 12 10
23 999 56 78 32 16 64 32
36 19 999 82 57 35 80 25
70 25 79 999 34 51 19 47
34 25 65 40 999 42 59 41
35 90 26 10 37 999 76 82
37 64 63 27 35 59 999 38
37 80 28 38 58 39 41 999
```

Output:

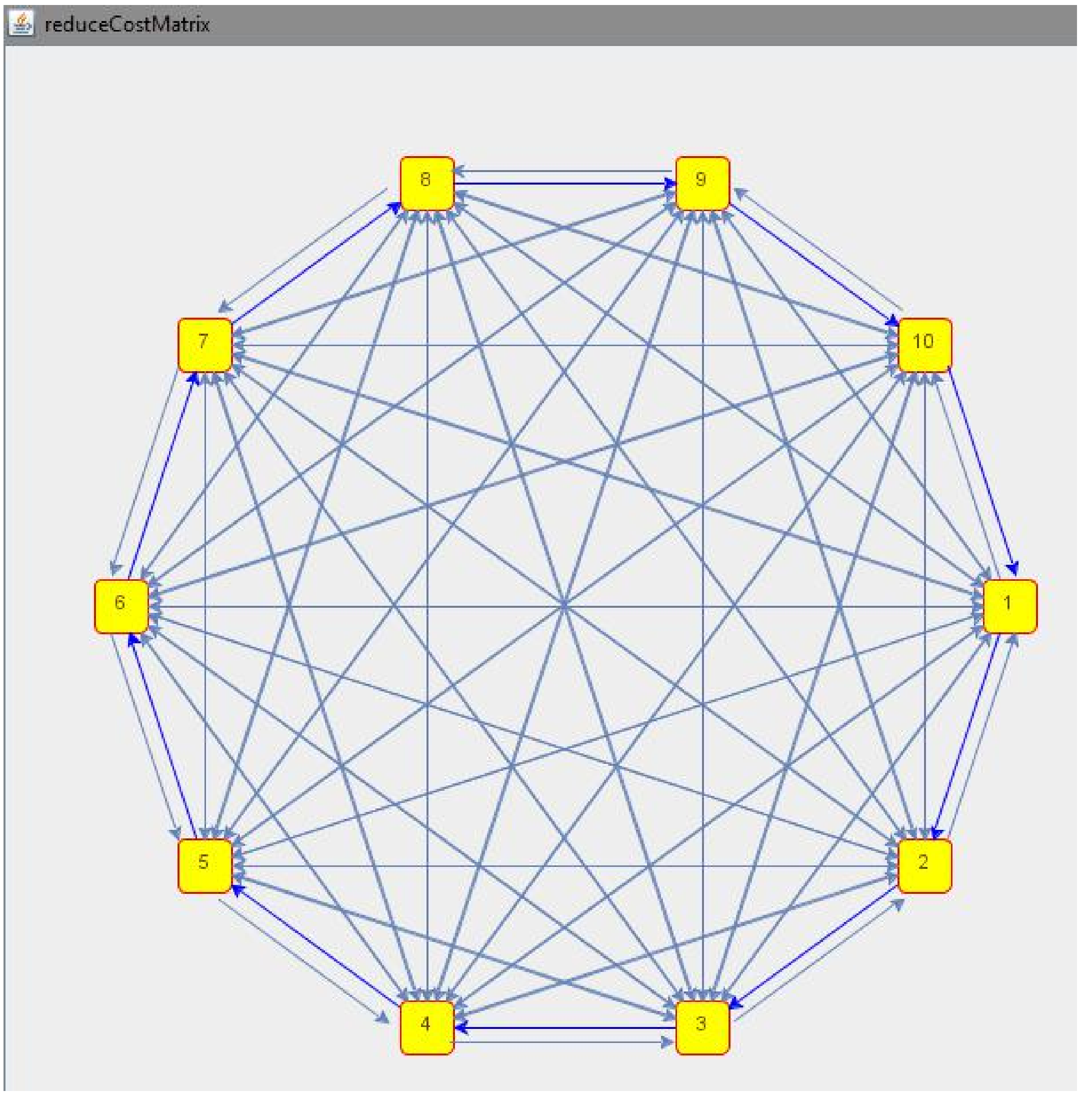


```
rute terpendek:1 8 3 2 6 4 7 5
bobot: 171
jumlah simpul yang dibangkintkan: 29
Excecution time: 4 ms
```

Input:

```
999  1   2  3  4   5  6   7  8  9
10 999 11 12 13 14 15 16 17 18
19 20 999 21 22 23 24 25 26 27
28 29 30 999 31 32 33 34 35 36
37 38 39 40 999 41 42 43 44 45
46 47 48 49 50 999 51 52 53 54
55 56 57 58 59 60 999 61 62 63
64 65 66 67 68 68 70 999 71 72
73 74 75 76 77 78 79 80 999 81
82 83 84 85 86 87 88 89 90 999
```

Output:



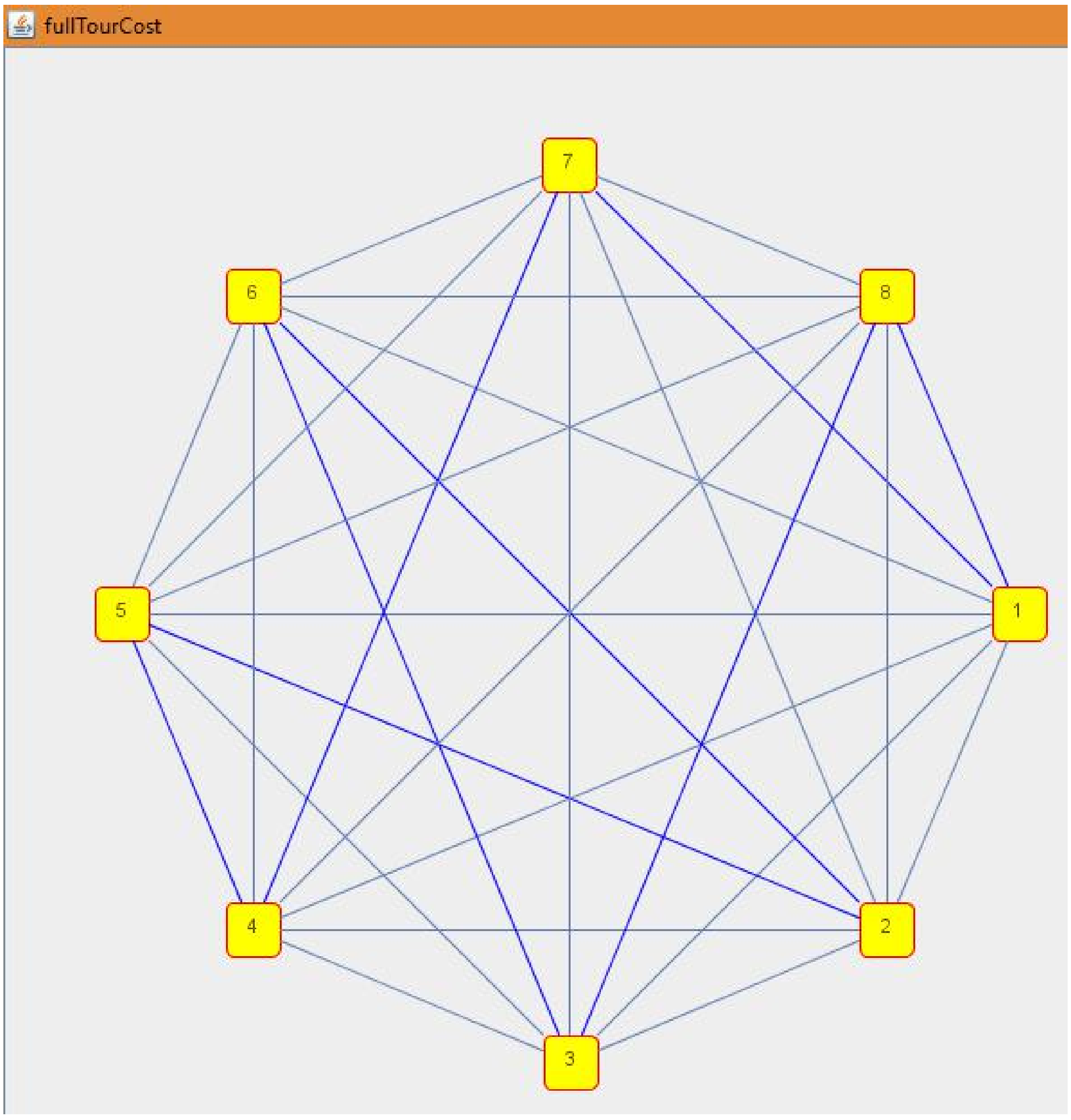
```
rute terpendek :1 2 3 4 5 6 7 8 9 10
bobot: 451
jumlah simpul yang dibangkintkan: 46
Excecution time: 5 ms
```

TSP B&B dengan Bobot Tour Lengkap

Input:

```
999 16 27 89 34 65 12 10
16 999 56 78 32 16 64 32
27 56 999 82 57 35 80 25
89 78 82 999 34 51 19 47
34 32 57 34 999 42 59 41
65 16 35 51 42 999 76 82
12 64 80 19 59 76 999 38
10 32 25 47 41 82 38 999
```

Output:

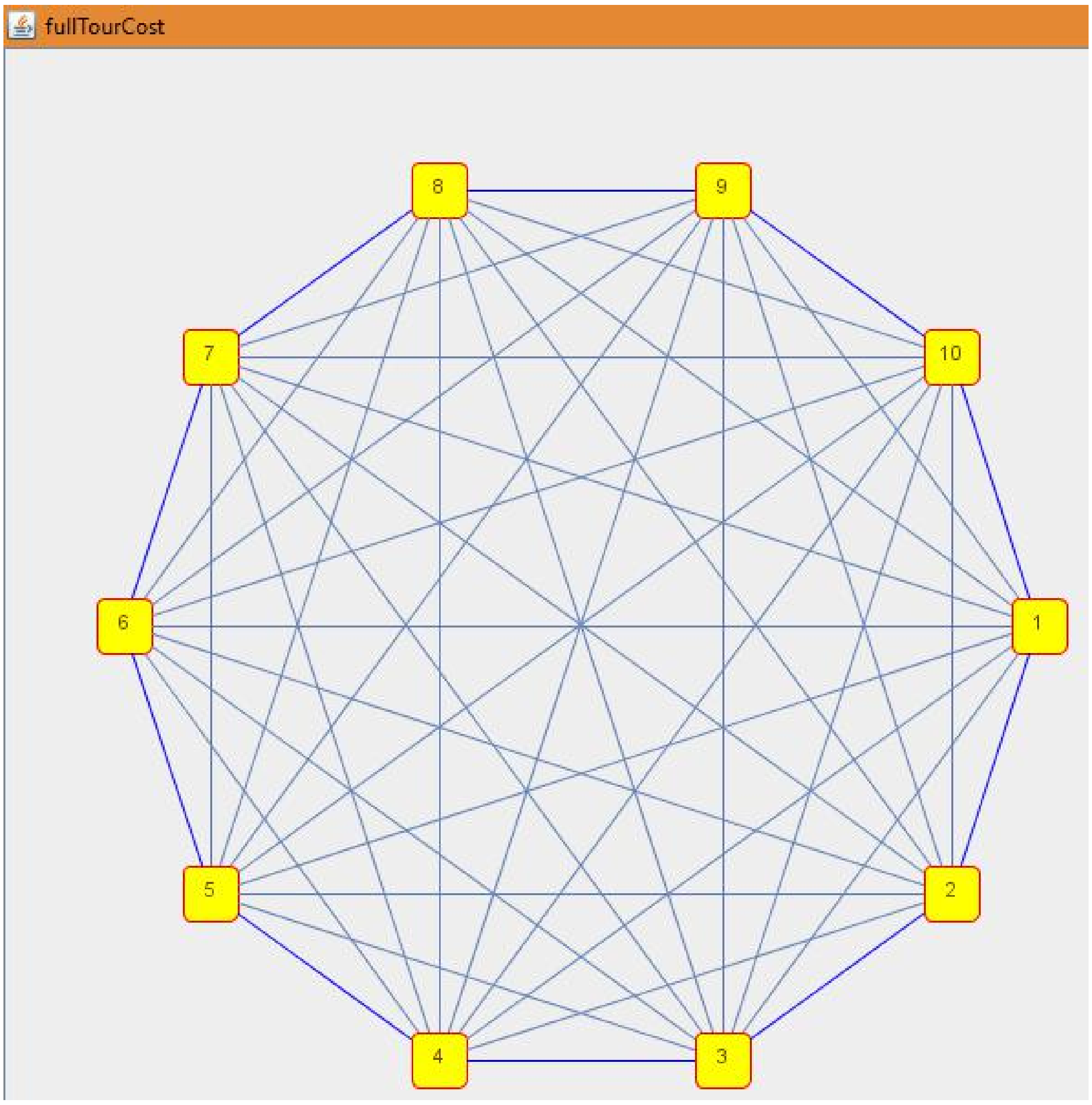


```
rute terpendek :1 8 3 6 2 5 4 7
bobot: 185
jumlah simpul yang dibangkintkan: 29
Excecution time: 3 ms
```

Input:

```
999 1  2   3  4   5  6  7   8  9
1 999 11 12 13 14 15 16 17 18
2 11 999 21 22 23 24 25 26 27
3 12 21 999 31 32 33 34 35 36
4 13 22 31 999 41 42 43 44 45
5 14 23 32 41 999 51 52 53 54
6 15 24 33 42 51 999 61 62 63
7 16 25 34 43 52 61 999 71 72
8 17 26 35 44 53 62 71 999 81
9 18 27 36 45 54 63 72 81 999
```

Output:



```
rute terpendek :1 2 3 4 5 6 7 8 9 10
bobot: 406
jumlah simpul yang dibangkintkan: 46
Excecution time: 30 ms
```

Poin	Ya	Tidak
1. Program berhasil dikompilasi	√	
2. Program berhasil *running*	√	
3. Program dapat membaca input file berisi matriks bobot dan mendapatkan tur terpendek beserta bobotnya	√	
4. Program dapat mengeluarkan gambar graf dari matriks masukan	√	
5. Program dapat mengeluarkan gambar tur terpendek	√	

Library:
Kode sumber: https://github.com/jgraph/jgraphx
User manual: https://jgraph.github.io/mxgraph/docs/manual_javavis.html#1.7.2