Laporan Tugas

Penyelesaian Persoalan TSP dengan Algoritma *Branch and Bound*

Fadhil Imam Kurnia - 13515146

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

`

A. Permasalahan

Travelling Salesperson Problem (TSP) merupakan masalah klasik dalam dunia algoritma. Pada persoalan tersebut terdapat N buah kota serta jarak antara setiap kota satu sama lain. Kita harus menentukan perjalanan terpendek yang melalui setiap kota lainnya hanya sekali dan kembali lagi ke kota asal keberangkatan[1].

Telah ada beberapa metode pemecahan masalah yang dikerahkan untuk menyelesaikan persoalan TSP tersebut. Hingga saat ini belum ditemukan algoritma yang cukup efisien untuk menyelesaikan permasalahan ini (dalam orde polinomial). Dalam tugas persoalan TSP akan coba diselesaikan melalui algortima *Branch and Bound*. Algoritma tersebut memiliki proses yang hampir sama dengan BFS, namun pada algortima tersebut digunakan tidak digunakan *queue* biasa. Algoritma *Branch and Bound* akan lebih memilih simpul yang mendekati pada solusi, dan membunuh (*bound*) simpul lainnya jika dirasa tidak memungkinkan ditemukan solusi lainnya yang lebih baik.

Terdapat 2 pendekatan yang akan digunakan dalam algoritma *Branch and Bound* ini. Pendekatan pertama adalah dengan menggunakan *reduced cost matrix.* Pendekatan ini berusaha mencari solusi dengan menghitung total pengurang untuk mendapatkan matriks tereduksi, total pengurang tersebut akan digunakan untuk menyatakan biaya pada simpul hidup. Pendekatan kedua adalah dengan menggunakan bobot tur lengkap. Pendekatan ini berusaha menghitung biaya setiap simpul hidup dengan menghitung 2 sisi minumum pada setiap simpul, dengan catatan sisi yang sudah menjadi lintasan simpul hidup tersebut wajib diambil sebagai sisi minimum tadi.

B. Source Code

Untuk menyelesaikan TSP menggunakan Algoritma Branch and Bound digunakan bahasa Java, sehingga penyelesaian masalah tersebut menggunakan paradigma pemrograman berbasis objek. Dalam pengerjaan tugas ini digunakan *library* GraphStream[2] untuk menggambar graf dinamis serta graf yang merepresentasikan peta dari matriks ketetanggaan yang diberikan. Hasil *source code* yang telah dibuat akan dijabarkan pada bagian ini.

Pada kode yang telah dihasilkan terdapat kelas TSPSolver yang merupakan kelas utama, selain itu juga terdapat kelas ReducedCostMatrix untuk memudahkan penghitungan biaya saat menggunakan metode *Reduced Cost Matrix.* Terdapat juga kelas SimpleNode yang digunakan untuk menyimpan simpul-simpul aktif pada graf dinamis yang diciptakan saat proses pencarian solusi.

Secara garis besar penyelesaian menggunakan pendekatan *Reduced Cost Matrix* ataupun bobot tur lengkap memiliki langkah-langkah yang sama. Kedua pendekatan tersebut sama-sama menggunakan *Priority Queue* untuk mencari solusi. Perbedaan kedua pendekatan tersebut terdapat pada cara penentuan biaya untuk setiap simpul hidup. Pada file TSPSolver.java hal itu dapat diamati pada fungsi getNodeCostBT dan getReducedCostMatrix.

TSPSolver.java

```
// File  : TSPSolver.h
// Name  : Fadhil Imam Kurnia - 13515146
// Main file for solving TSP

package main.java;

import org.graphstream.graph.Edge;
import org.graphstream.graph.Graph;
import org.graphstream.graph.Node;
import org.graphstream.graph.implementations.SingleGraph;

import java.applet.Applet;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.PriorityQueue;

public class TSPSolver extends Applet{

    private static int INF = Integer.MAX_VALUE;
    private static int REDUCED_COST_MATRIX_METHOD = 1;
    private static int BOBOT_TUR_LENGKAP_METHOD = 2;
    private static int CHOOSEN_METHOD = REDUCED_COST_MATRIX_METHOD;
    private static Integer[][] adjajency;
    private static int NUMBER_OF_NODE = 0;

    public static void main (String args[]) {
        new TSPSolver();
    }

    public TSPSolver() {
        System.out.println("Solving TSP using method no : "+CHOOSEN_METHOD);

        // Reading file
        MapReader reader = new MapReader("map2.json");
        adjajency = reader.getMap();
        NUMBER_OF_NODE = adjajency.length;

        // Displaying map to solve
        Graph graph = new SingleGraph("Map of Node");
        displayMap(graph);

        SimpleNode finalResult = null;
        if (CHOOSEN_METHOD == REDUCED_COST_MATRIX_METHOD){
            finalResult = initTSPWithReducedCostMatrix();
        } else if (CHOOSEN_METHOD == BOBOT_TUR_LENGKAP_METHOD){
            finalResult = initTSPWIthBobotTurLengkap();
        }

        // Drawing path
        if (finalResult != null)
            drawRoute(finalResult.getPath(), graph);

    }

    private void displayMap(Graph graph){
        graph.setStrict(false);
        graph.setAutoCreate(true);
        graph.display();

        boolean showArrow = (CHOOSEN_METHOD == REDUCED_COST_MATRIX_METHOD);

        for(int i = 0; i < NUMBER_OF_NODE; i++){
            for(int j = 0; j < NUMBER_OF_NODE; j++){
                if (adjajency[i][j] != INF){
                    graph.addEdge(Integer.toString(i)+Integer.toString(j),
                            Integer.toString(i), Integer.toString(j), showArrow);
                    try {
                        graph.getEdge(Integer.toString(i)+Integer.toString(j))
                                .setAttribute("ui.label",adjajency[i][j]);
                    } catch (Exception e) {

                    }
                }
            }
        }
```

```
        graph.getNode(Integer.toString(0)).setAttribute("ui.class","marked");

        for (Node node : graph) {
            node.addAttribute("ui.label", node.getId());
        }

        graph.addAttribute("ui.stylesheet", styleSheet);
    }



    private void drawRoute(ArrayList<Integer> path, Graph graph) {
        for (int i = 0; i < path.size()-1; i++){
            Edge cEdge = graph.getEdge(Integer.toString(path
            .get(i))+Integer.toString(path.get(i+1)));
            try {
                cEdge.setAttribute("ui.class","selected");
            } catch (Exception e) {
                cEdge = graph.getEdge(Integer.toString(path
                        .get(i+1))+Integer.toString(path.get(i)));
                cEdge.setAttribute("ui.class","selected");
            }
        }
        Edge cEdge = graph.getEdge(Integer.toString(path
                .get(path.size()-1))+Integer.toString(0));
        try {
            cEdge.setAttribute("ui.class","selected");
        } catch (Exception e) {
            cEdge = graph.getEdge(Integer.toString(0)+Integer.toString(path
                    .get(path.size()-1)));
            cEdge.setAttribute("ui.class","selected");
        }
    }



//    ---------------------------------------------------------------------------
//    BOBOT TUR LENGKAP
//    ---------------------------------------------------------------------------


    private SimpleNode initTSPWIthBobotTurLengkap() {
        // Preparing some variabel to solving the problem
        NodeComparator comparator = new NodeComparator();
        PriorityQueue<SimpleNode> lifeNode = new PriorityQueue<>(1,
                comparator);
        ArrayList<Integer> path = new ArrayList<>();
        int counter = 0;
        float finalCost = INF;
        boolean finish = false;
        boolean[] visited = new boolean[NUMBER_OF_NODE];
        for (int i = 0; i < NUMBER_OF_NODE; i++) {
            visited[i] = false;
        }

        Graph dGraph = new SingleGraph("Dynamic Graph");
        dGraph.addAttribute("ui.stylesheet", styleSheet);
        dGraph.setStrict(false);
        dGraph.setAutoCreate(true);
        dGraph.display();

        PriorityQueue<SimpleNode> solution = new PriorityQueue<>(1,comparator);

        // Preparing first node in graph
        path.add(0);
        float cost = getNodeCostBT(path);
        visited[0] = true;
        SimpleNode currentNode = new SimpleNode(counter,cost,path, visited);
        dGraph.addNode("0");
        dGraph.getNode("0").addAttribute("ui.label", "0 - ("+currentNode.getCost()
                +")");
        dGraph.getNode("0").setAttribute("ui.class","marked");

        // Start BnB
        long tStart = System.currentTimeMillis();
        lifeNode.add(currentNode);
        while (!lifeNode.isEmpty() && !finish) {
            currentNode = lifeNode.poll();

            if (currentNode.getCost() > finalCost) {
                finish = true;
            }

            for (int i = 0; i < NUMBER_OF_NODE && !finish; i++) {
```

```
                if                (isNextPathAvailable(currentNode.getPath(),i)                &&
!currentNode.getVisited()[i]){
                    counter++;
                    path = new ArrayList<>(currentNode.getPath().size()+1);
                    path.addAll(currentNode.getPath());
                    path.add(i);
                    visited = Arrays.copyOf(currentNode.getVisited(),
                            NUMBER_OF_NODE);
                    visited[i] = true;
                    SimpleNode childNode = new SimpleNode(
                            counter,
                            getNodeCostBT(path),
                            path,
                            visited);

                    lifeNode.add(childNode);

                    dGraph.addEdge(Integer.toString(currentNode.getId())
                            +Integer.toString(counter), Integer.toString
                            (currentNode.getId()), Integer.toString(counter));
                    Node cNode = dGraph.getNode(Integer.toString(counter));
                    cNode.setAttribute("ui.label", counter+" - ("+childNode.getCost
                                ()+")");

                    if (isSolutionNode(childNode)) {
                        cNode.setAttribute("ui.class","solution");
                        solution.add(childNode);
                        if (childNode.getCost() < finalCost)
                            finalCost = childNode.getCost();
                    }

                }
            }
        }
        long tEnd = System.currentTimeMillis();

        System.out.println("Number of solution : " + solution.size());

        System.out.println("One of the best solution :");
        SimpleNode finalresult = solution.poll();
        for (int j = 0; j < finalresult.getPath().size(); j++)
            System.out.print(finalresult.getPath().get(j)+1 + " ");
        System.out.print("1 ");
        System.out.println(" | Cost : >" + finalresult.getCost() + " | Real " +
                "Cost :" + calculateRealCost(finalresult) +
                " | Number of Node : "+counter);

        System.out.println("Elapsed time : " + (tEnd - tStart)/1000.0 +
                "seconds");

        dGraph.getNode(Integer.toString(finalresult.getId()))
                .setAttribute("ui" +
                        ".class",
                "final");

        return finalresult;

    }


    private boolean isNextPathAvailable(ArrayList<Integer> path, int nodeId) {
        return adjajency[path.get(path.size()-1)][nodeId] != INF;
    }

    private boolean isSolutionNode(SimpleNode node) {
        return node.getPath().size() == NUMBER_OF_NODE;
    }

    private float calculateRealCost(SimpleNode node){
        float cost = 0;
        ArrayList<Integer> path = node.getPath();

        for (int i = 0; i < path.size()-1; i++)
            cost += adjajency[path.get(i)][path.get(i+1)];
        cost += adjajency[path.get(path.size()-1)][0];

        return cost;
    }

    private float getNodeCostBT(ArrayList<Integer> path) {
        // Preparing
        float cost = 0;
        boolean[][] wajib = new boolean[NUMBER_OF_NODE][NUMBER_OF_NODE];
        for (int i = 0; i < NUMBER_OF_NODE; i++)
            for (int j = 0; j < NUMBER_OF_NODE; j++)
                wajib[i][j] = false;
```

```
        if (path.size() > 1) {
            System.out.print("GetCost untuk path :");
            for (int i = 0; i < path.size()-1; i++){
                wajib[path.get(i)][path.get(i+1)] = true;
                wajib[path.get(i+1)][path.get(i)] = true;
                System.out.print(path.get(i) + " ");
            }
            System.out.print(path.get(path.size()-1));
            System.out.println();
        }

        for (int i = 0; i < NUMBER_OF_NODE; i++) {
            // Get 2 minimum edges from available edge
            int min1, min2;
            if (adjajency[i][0] < adjajency[i][1]) {
                min1 = adjajency[i][0];
                min2 = adjajency[i][1];
            } else {
                min1 = adjajency[i][1];
                min2 = adjajency[i][0];
            }
            ArrayList<Integer> candidate = new ArrayList<>(2);
            if (wajib[i][0])
                candidate.add(adjajency[i][0]);
            if (wajib[i][1])
                candidate.add(adjajency[i][1]);
            for (int j = 2; j < NUMBER_OF_NODE; j++) {
                if (wajib[i][j])
                    candidate.add(adjajency[i][j]);
                if (adjajency[i][j] < min1) {
                    if (min2 > min1)
                        min2 = min1;
                    min1 = adjajency[i][j];
                } else if (adjajency[i][j] < min2)
                    min2 = adjajency[i][j];
            }

            if (candidate.size() == 1) {
                if (candidate.get(0) != min1)
                    min2 = candidate.remove(0);
            } else if (candidate.size() == 2) {
                min1 = candidate.remove(0);
                min2 = candidate.remove(0);
            }

            System.out.print(" ("+min1 +" + "+ min2+") ");
            cost += min1 + min2;
        }

        System.out.println(" = "+cost);

        return cost/2;
    }


//    ---------------------------------------------------------------------------
//    REDUCED COST MATRIX
//    ---------------------------------------------------------------------------

    private SimpleNode initTSPWithReducedCostMatrix() {
        // Preparing some variabels to solving the problem
        NodeComparator comparator = new NodeComparator();
        PriorityQueue<SimpleNode> lifeNode = new PriorityQueue<>(1,
                comparator);
        ArrayList<Integer> path = new ArrayList<>();
        int counter = 0;
        float finalCost = INF;
        boolean finish = false;
        boolean[] visited = new boolean[NUMBER_OF_NODE];
        for (int i = 0; i < NUMBER_OF_NODE; i++) {
            visited[i] = false;
        }

        Graph dGraph = new SingleGraph("Dynamic Graph");
        dGraph.addAttribute("ui.stylesheet", styleSheet);
        dGraph.setStrict(false);
        dGraph.setAutoCreate(true);
        dGraph.display();

        PriorityQueue<SimpleNode> solution = new PriorityQueue<>(1,comparator);

        // Preparing first node in graph
        path.add(0);
        visited[0] = true;
        ReducedCostMatrix firstRCM = getReducedCostMatrix(adjajency,null,path);
        float cost = firstRCM.getCost();
        adjajency = firstRCM.getMatrix();
```

```
        SimpleNode currentNode = new SimpleNode(counter,cost,path, visited);
        currentNode.setMatrix(adjajency);
        dGraph.addNode("0");
        dGraph.getNode("0").addAttribute("ui.label", "0 - ("+currentNode.getCost()
                +")");
        dGraph.getNode("0").setAttribute("ui.class","marked");

        // Start BnB
        long tStart = System.currentTimeMillis();
        lifeNode.add(currentNode);
        while (!lifeNode.isEmpty() && !finish) {
            currentNode = lifeNode.poll();

            if (currentNode.getCost() > finalCost) {
                finish = true;
            }

            for (int i = 0; i < NUMBER_OF_NODE && !finish; i++) {

                if(isNextPathAvailable(currentNode.getPath(),i)&&!currentNode.getVisited()[i]){
                    counter++;
                    path = new ArrayList<>(currentNode.getPath().size()+1);
                    path.addAll(currentNode.getPath());
                    path.add(i);
                    visited = Arrays.copyOf(currentNode.getVisited(),
                            NUMBER_OF_NODE);
                    visited[i] = true;

                    ReducedCostMatrix childRCM = getReducedCostMatrix(currentNode
                            .getMatrix(),currentNode.getCost(),path);
                    SimpleNode childNode = new SimpleNode(
                            counter,
                            childRCM.getCost(),
                            path,
                            visited);
                    childNode.setMatrix(childRCM.getMatrix());

                    lifeNode.add(childNode);

                    dGraph.addEdge(Integer.toString(currentNode.getId())
                            +Integer.toString(counter), Integer.toString
                            (currentNode.getId()), Integer.toString(counter));
                    Node cNode = dGraph.getNode(Integer.toString(counter));
                    cNode.setAttribute("ui.label", counter+" - ("+childNode.getCost
                            ()+")");



                    if (isSolutionNode(childNode)) {
                        cNode.setAttribute("ui.class","solution");
                        solution.add(childNode);
                        if (childNode.getCost() < finalCost)
                            finalCost = childNode.getCost();
                    }

                }
            }

        }

        long tEnd = System.currentTimeMillis();

        System.out.println("Number of solution : " + solution.size());

        System.out.println("One of the best solution :");
        SimpleNode finalresult = solution.poll();
        for (int j = 0; j < finalresult.getPath().size(); j++)
            System.out.print(finalresult.getPath().get(j) + " ");
        System.out.println(" | Cost : " + finalresult.getCost() + " | " +
                "Number of Node : "+counter);

        System.out.println("Elapsed time : " + (tEnd - tStart)/1000.0 +
                " seconds");

        dGraph.getNode(Integer.toString(finalresult.getId()))
                .setAttribute("ui" +
                                ".class",
                        "final");

        return finalresult;
    }

    private ReducedCostMatrix getReducedCostMatrix(final Integer[][] prevMatrix,
                                                   Float pCost,
                                                   ArrayList<Integer> path){

        // menyalin reduced cost matrix sebelumnya ke matrix lokal
```

```
        Integer[][] matrix = new Integer[NUMBER_OF_NODE][NUMBER_OF_NODE];
        for (int i = 0; i < NUMBER_OF_NODE; i++){
            System.arraycopy(prevMatrix[i], 0, matrix[i], 0, NUMBER_OF_NODE);
        }

        int row = 0;
        int col = 0;

        // membuat infinite 2 path terakhir
        if(path.size() > 1) {
            row = path.get(path.size()-2);
            col = path.get(path.size()-1);
            for (int i = 0; i < NUMBER_OF_NODE; i++){
                matrix[row][i] = INF;
            }
            for (int i = 0; i < NUMBER_OF_NODE; i++){
                matrix[i][col] = INF;
            }
            matrix[col][0] = INF;
        }

        float cost = (pCost == null)? 0 : pCost;
        // menghitung cost dari reduksi baris
        for (int i = 0; i < NUMBER_OF_NODE; i++){
            int min = INF;
            for (int j = 0; j < NUMBER_OF_NODE; j++){
                if (matrix[i][j] < min)
                    min = matrix[i][j];
            }
            if (min != INF && min != 0){
                for (int j = 0; j < NUMBER_OF_NODE; j++){
                    if (matrix[i][j] != INF)
                        matrix[i][j]-= min;
                }
                cost += min;
            }
        }
        // menghitung cost dari reduksi kolom
        for (int i = 0; i < NUMBER_OF_NODE; i++){
            int min = INF;
            for (int j = 0; j < NUMBER_OF_NODE; j++){
                if (matrix[j][i] < min)
                    min = matrix[j][i];
            }
            if (min != INF && min != 0){
                for (int j = 0; j < NUMBER_OF_NODE; j++){
                    if (matrix[j][i] != INF)
                        matrix[j][i] -= min;
                }
                cost += min;
            }
        }

        if (path.size() > 1)
            cost += adjajency[row][col];

        ReducedCostMatrix reducedCostMatrix = new ReducedCostMatrix();
        reducedCostMatrix.setMatrix(matrix);
        reducedCostMatrix.setCost(cost);


        // Printing some data
        System.out.print("Menghasilkan matriks untuk path:");
        for (Integer cPath : path) {
            System.out.print(cPath + " ");
        }
        System.out.println();
        for(int i = 0; i < NUMBER_OF_NODE; i++){
            for(int j = 0; j < NUMBER_OF_NODE; j++){
                if (matrix[i][j] != INF)
                    System.out.print(matrix[i][j]+"\t");
                else
                    System.out.print("∞\t");
            }
            System.out.println();
        }
        System.out.println("cost:" + cost);
        System.out.println();
        System.out.println();

        return reducedCostMatrix;
    }

}
```

SimpleNode.java

```
// File  : SimpleNode.java
// Name  : Fadhil Imam Kurnia - 13515146
// Class file for life node

package main.java;

import java.util.ArrayList;
import java.util.Arrays;

public class SimpleNode {
    int id;
    float cost;
    Integer[][] matrix = null;
    ArrayList<Integer> path = null;
    boolean[] visited = null;

    public SimpleNode() {
        this.id = -1;
        this.cost = -1;
    }

    public SimpleNode(int id, float cost, ArrayList<Integer> path, boolean[]
            visited) {
        this.id = id;
        this.cost = cost;
        this.path = path;
        this.visited = visited;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public float getCost() {
        return cost;
    }

    public void setCost(float cost) {
        this.cost = cost;
    }

    public Integer[][] getMatrix() {
        return matrix;
    }

    public void setMatrix(Integer[][] matrix) {
        this.matrix = new Integer[matrix.length][matrix.length];
        for (int i = 0; i < matrix.length; i++)
            this.matrix[i] = Arrays.copyOf(matrix[i], matrix.length);
    }

    public ArrayList<Integer> getPath() {
        return path;
    }

    public void setPath(ArrayList<Integer> path) {
        this.path = path;
    }

    public boolean[] getVisited() {
        return visited;
    }

    public void setVisited(boolean[] visited) {
        this.visited = visited;
    }
}
```

Pada *source code* tersebut, *priority queue* digunakan untuk menyimpan objek SimpleNode. Biaya dan *path* yang ada pada setiap simpul digunakan untuk mengatur prioritas simpul tersebut dalam *queue.* Simpul yang memiliki biaya paling rendah dan *path* yang hampir selesai akan memiliki prioritas paling tinggi. Pencarian solusi akan berhenti saat biaya simpul-simpul yang tersisa dalam *queue* lebih tinggi dibandingkan dengan biaya minimum solusi yang sudah ditemukan. *Source code* selengkapnya dapat dilihat pada https://gitlab.com/fadhilimamk/BnB-TSP/tree/master.

C. Hasil Eksekusi Program

Hasil akhir yang dapat diperoleh dari program diantaranya adalah lintasan terpendek yang didapat beserta solusinya, waktu eksekusi program dalam satuan detik, jumlah simpul yang dibangkitkan untuk mencari solusi, gambar graf dinamis saat proses pencarian, serta gambar tur terpendek pada peta. Simpul awal akan ditandai dengan warna merah, dan simpul solusi ditandai dengan warna biru. Jalur pada peta akan diwarnai dengan warna merah agar dapat dibedakan dengan jalur lainnya.

1. Kasus Uji 1 menggunakan pendeketan *reduced cost matrix*

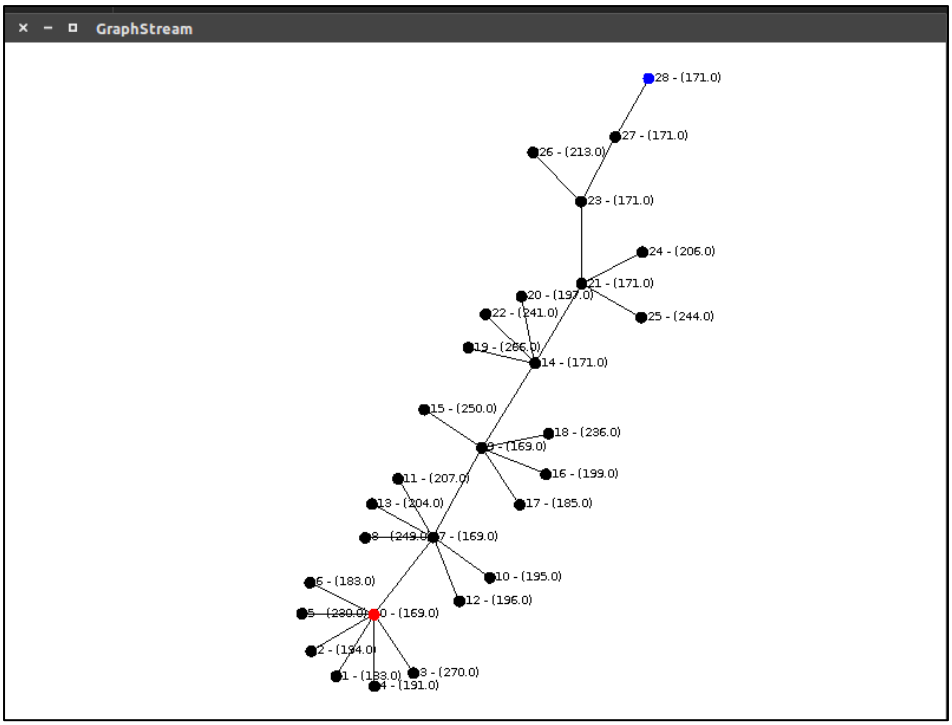
$$\begin{bmatrix} \infty & 16 & 27 & 89 & 34 & 65 & 12 & 10 \\ 23 & \infty & 56 & 78 & 32 & 16 & 64 & 32 \\ 36 & 19 & \infty & 82 & 57 & 35 & 80 & 25 \\ 70 & 25 & 79 & \infty & 34 & 51 & 19 & 47 \\ 34 & 25 & 65 & 40 & \infty & 42 & 59 & 41 \\ 35 & 90 & 26 & 10 & 37 & \infty & 76 & 82 \\ 37 & 64 & 63 & 27 & 35 & 59 & \infty & 38 \\ 37 & 80 & 28 & 38 & 58 & 39 & 41 & \infty \end{bmatrix}$$

a. Hasil eksekusi

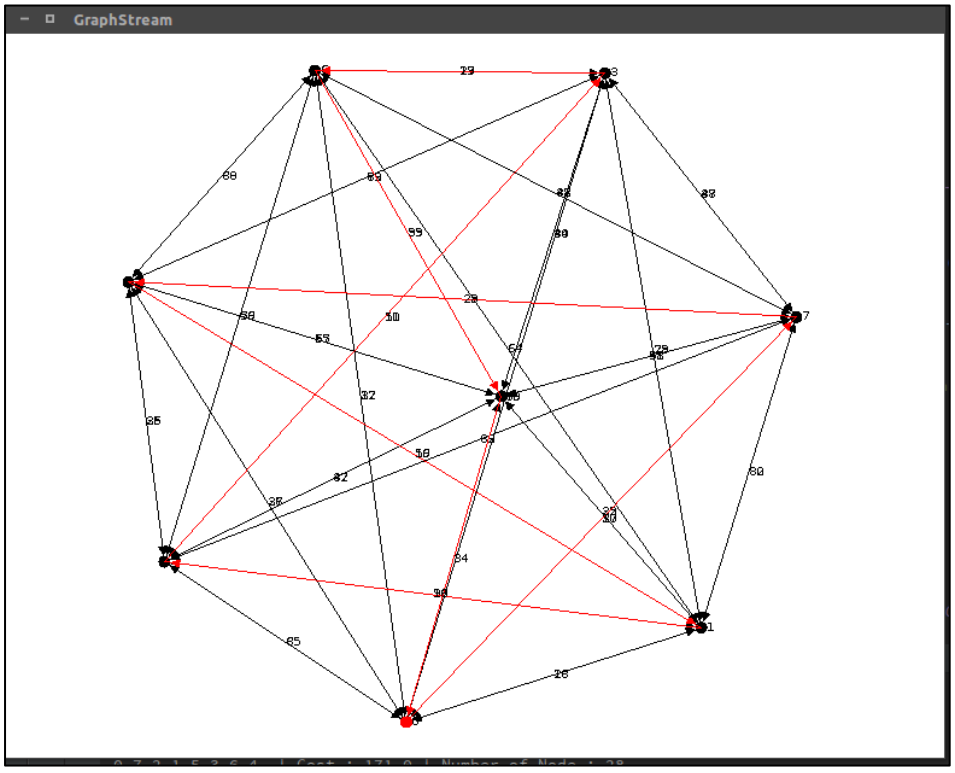
```
/usr/lib/jvm/java-1.7.0-openjdk-amd64/bin/java ...
Solving TSP using method no : 1
One of the best solution :
0 7 2 1 5 3 6 4  | Cost : 171.0 | Number of Node : 28
Elapsed time : 0.002 seconds
```

Gambar 1 Hasil eksekusi kasus uji 1 *reduced cost matrix*

b. Gambar Graf Dinamis dan Rute pada Peta



Gambar 2 Pohon dinamis kasus uji 1 *reduced cost matrix*



Gambar 3 Hasil peta dengan rute terpendek kasus uji 1 *reduced cost matrix*

2. Kasus Uji 2 menggunakan pendeketan *reduced cost matrix*

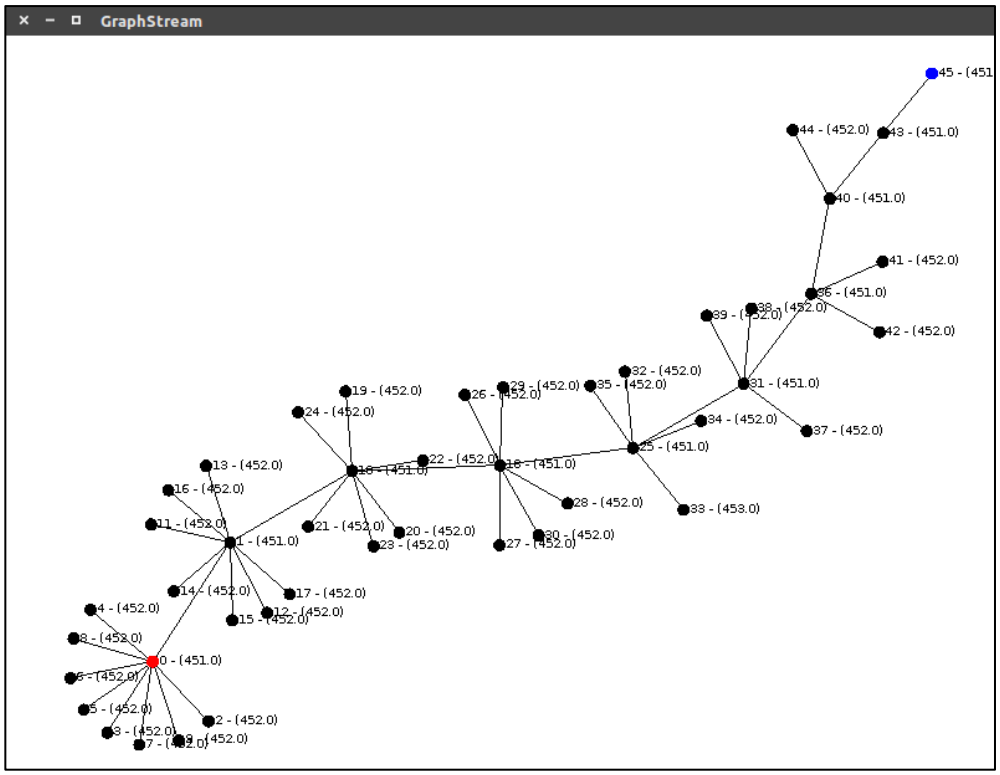
$$\begin{bmatrix} \infty & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 10 & \infty & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 \\ 19 & 20 & \infty & 21 & 22 & 23 & 24 & 25 & 26 & 27 \\ 28 & 29 & 30 & \infty & 31 & 32 & 33 & 34 & 35 & 36 \\ 37 & 38 & 39 & 40 & \infty & 41 & 42 & 43 & 44 & 45 \\ 46 & 47 & 48 & 49 & 50 & \infty & 51 & 52 & 53 & 54 \\ 55 & 56 & 57 & 58 & 59 & 60 & \infty & 61 & 62 & 63 \\ 64 & 65 & 66 & 67 & 68 & 69 & 70 & \infty & 71 & 72 \\ 73 & 74 & 75 & 76 & 77 & 78 & 79 & 80 & \infty & 81 \\ 82 & 83 & 84 & 85 & 86 & 87 & 88 & 89 & 90 & \infty \end{bmatrix}$$

a. Hasil eksekusi

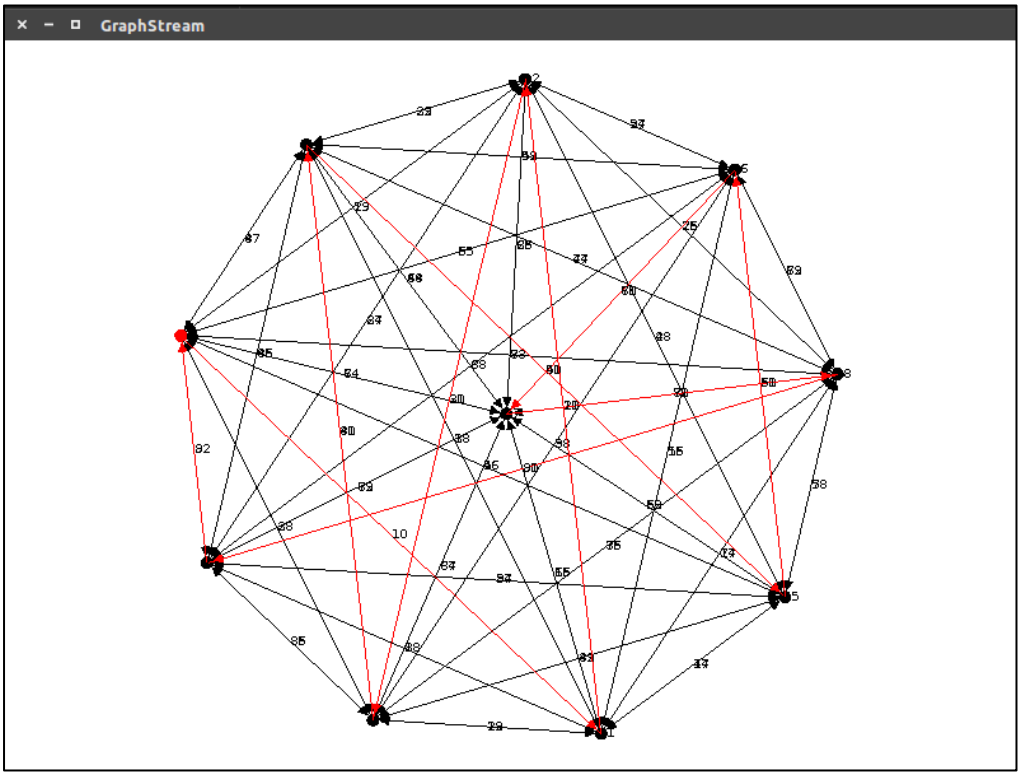
```
/usr/lib/jvm/java-1.7.0-openjdk-amd64/bin/java ...
Solving TSP using method no : 1
One of the best solution :
0 1 2 3 4 5 6 7 8 9  | Cost : 451.0 | Number of Node : 45
Elapsed time : 0.005 seconds
```

Gambar 4 Hasil eksekusi kasus uji 2 *reduced cost matrix*

b. Gambar Graf Dinamis dan Rute pada Peta



Gambar 5 Pohon dinamis kasus uji 2 *reduced cost matrix*



Gambar 6 Hasil peta dengan rute terpendek kasus uji 2 *reduced cost matrix*

3. Kasus Uji 1 menggunakan pendeketan bobot tur lengkap

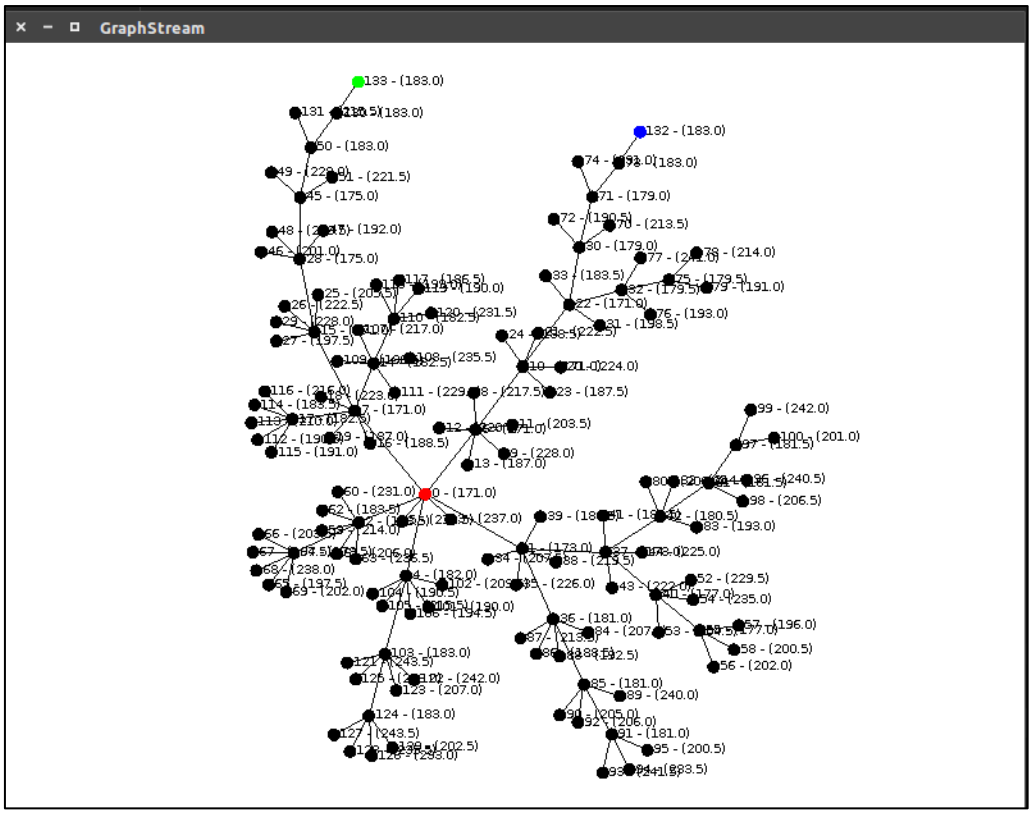
$$\begin{bmatrix} \infty & 16 & 27 & 89 & 34 & 65 & 12 & 10 \\ 16 & \infty & 56 & 78 & 32 & 16 & 64 & 32 \\ 27 & 56 & \infty & 82 & 57 & 35 & 80 & 25 \\ 89 & 78 & 82 & \infty & 34 & 51 & 19 & 47 \\ 34 & 32 & 57 & 34 & \infty & 42 & 59 & 41 \\ 65 & 16 & 35 & 51 & 42 & \infty & 76 & 82 \\ 12 & 64 & 80 & 19 & 59 & 76 & \infty & 38 \\ 10 & 32 & 25 & 47 & 41 & 82 & 38 & \infty \end{bmatrix}$$

a. Hasil eksekusi

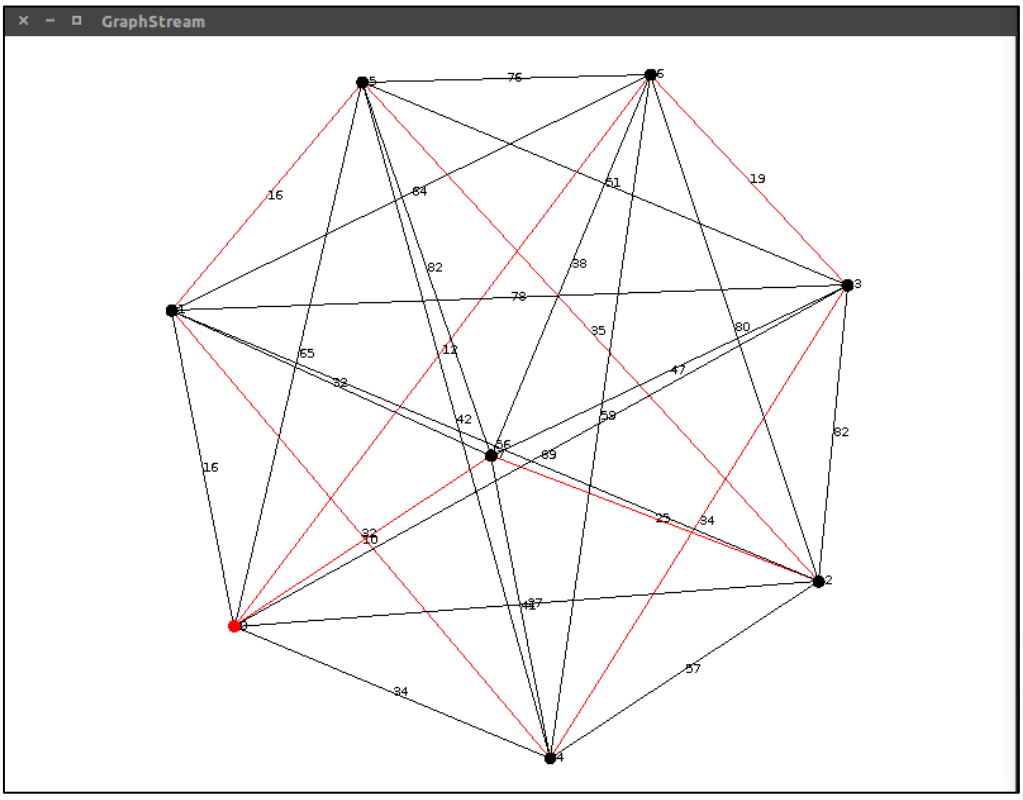
```
/usr/lib/jvm/java-1.7.0-openjdk-amd64/bin/java ...
2
Number of solution : 2
One of the best solution :
1 7 4 5 2 6 3 8 1  | Cost : >183.0 | Real Cost :183.0 | Number of Node : 133
Elapsed time : 0.026seconds
```

Gambar 7 Hasil eksekusi kasus uji 1 bobot tur lengkap

b. Gambar Graf Dinamis dan Rute pada Peta



Gambar 8 Pohon dinamis kasus uji 1 bobot tur lengkap



Gambar 9 Hasil peta dengan rute terpendek kasus uji 1 bobot tur lengkap

4. Kasus Uji 2 menggunakan pendeketan bobot tur lengkap

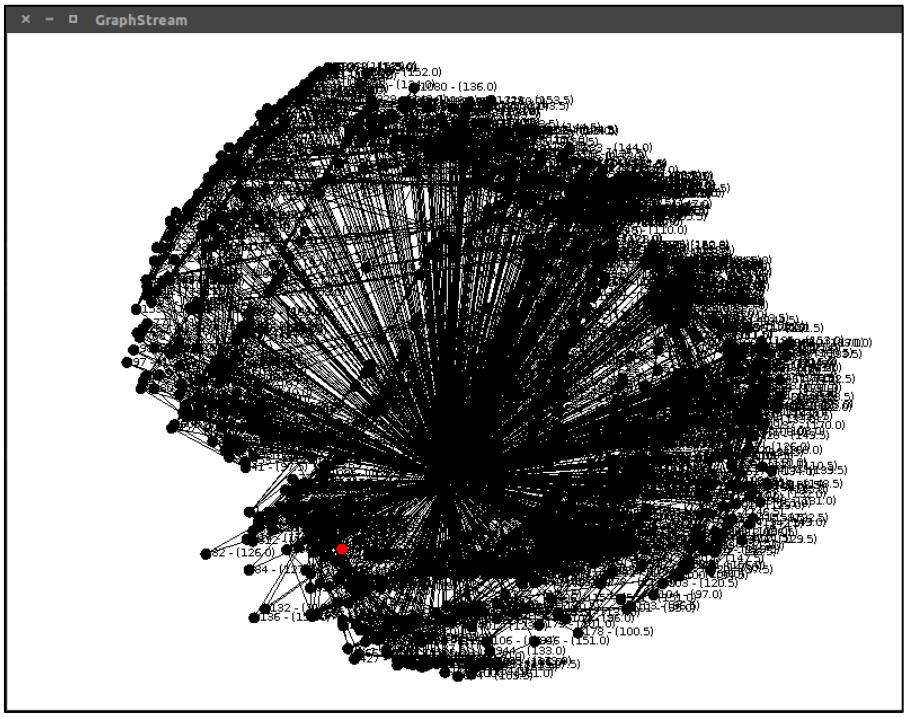
$$\begin{bmatrix} \infty & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & \infty & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 \\ 2 & 11 & \infty & 21 & 22 & 23 & 24 & 25 & 26 & 27 \\ 3 & 12 & 21 & \infty & 31 & 32 & 33 & 34 & 35 & 36 \\ 4 & 13 & 22 & 31 & \infty & 41 & 42 & 43 & 44 & 45 \\ 5 & 14 & 23 & 32 & 41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

a. Hasil eksekusi

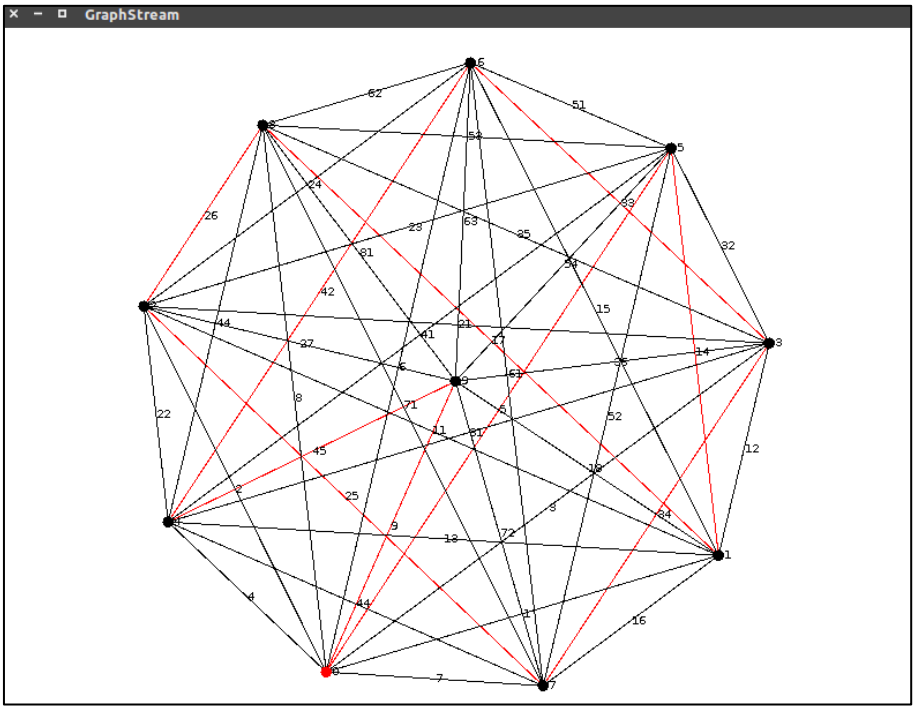
```
/usr/lib/jvm/java-1.7.0-openjdk-amd64/bin/java ...
2
Number of solution : 73732
One of the best solution :
1 6 2 9 3 8 4 7 5 10 1  | Cost : >246.0 | Real Cost :250.0 | Number of Node : 469010
Elapsed time : 2.888seconds
```

Gambar 10 Hasil eksekusi kasus uji 2 bobot tur lengkap

b. Gambar Graf Dinamis dan Rute pada Peta



Gambar 11 Pohon dinamis kasus uji 2 bobot tur lengkap



Gambar 12 Hasil peta dengan rute terpendek kasus uji 2 bobot tur lengkap

Daftar Pustaka dan Referensi

[1] Rinaldi Munir, Diktat Kuliah IF2251 Strategi Algoritmik, STEI, 2006.
[2] GraphStream : http://graphstream-project.org/