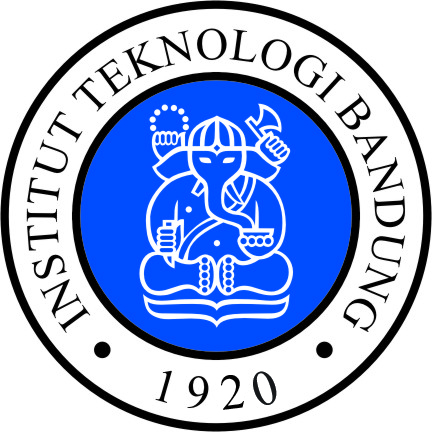
**Tugas Kecil 3**

**IF2211 Strategi Algoritma**

**Penyelesaian Persoalan *Travelling Salesman Problem* dengan Algoritma *Branch and Bound***

****

Disusun oleh :

Gianfranco Fertino Hwandiano – 13515118

1. Deskripsi Masalah

*Traveling Salesman Problem* (TSP) adalah suatu permasalahan dimana seorang sales harus melalui semua kota yang ditunjuk dengan jarak yang paling pendek dan setiap kota hanya boleh dilalui satu kali.

Masalah ini pertama kali dirumuskan sebagai masalah matematika pada tahun 1930 dan merupakan salah satu masalah yang paling intensif dalam mempelajari masalah optimasi, dan digunakan sebagai patokan bagi banyak metode optimasi dalam jumlah besar dengan cara yang tepat, dan metode yang mudah untuk diketahui, sehingga beberapa kasus dengan puluhan ribu kota dapat diselesaikan dengan baik. TSP memiliki beberapa aplikasi, seperti [perencanaan,](http://66.102.7.132/translate_c?hl=id&sl=en&u=http://en.wikipedia.org/wiki/Planning&prev=/search%3Fq%3DTSP%2B%28travelling%2Bsalesperson%2Bproblem%29%26hl%3Did%26sa%3DG&rurl=translate.google.co.id&usg=ALkJrhigdNJVtVsi5dhXYFuag7lpXNrVGQ) [logistik,](http://66.102.7.132/translate_c?hl=id&sl=en&u=http://en.wikipedia.org/wiki/Logistics&prev=/search%3Fq%3DTSP%2B%28travelling%2Bsalesperson%2Bproblem%29%26hl%3Did%26sa%3DG&rurl=translate.google.co.id&usg=ALkJrhjIGQQWy9f-xzTqxGzRzEF0wo4yEw" \o "Logistik) dan manufaktur. Dalam aplikasi ini, TSP merupakan konsep jarak perjalanan waktu atau biaya. Dalam banyak aplikasi, dapat muncul kendala seperti keterbatasan sumber daya atau waktu.

Buatlah program untuk menyelesaikan persoalan *Travelling Salesperson Problem* (TSP) dengan menggunakan Algoritma *Branch and Bound.* Nilai *bound* dihitung dengan *reduced cost matrix* dan dengan Bobot Tur Lengkap. Untuk penyelesaian dengan memanfaatkan *reduced cost matrix*, masukan berupa graf berarah. Sedangkan untuk penyelesaian dengan memanfaatkan Bobot Tur Lengkap, masukan berupa graf tidak berarah.

Masukan: matriks yang merepresentasikan bobot graf lengkap. Matriks dibaca dari berkas teks.

Keluaran: 1. Tur terpendek dan bobotnya

2. Waktu eksekusi

3. Jumlah simpul yang dibangkitkan

4. Gambar graf

5. Gambar tur terpendek (dinyatakan sebagai warna yang berbeda di dalam graf)

Untuk penggambaran graf, diperbolehkan menggunakan library yang tersedia di Internet, dan disebutkan sumber acuan untuk mendapatkan library tersebut.

Asumsi: tur selalu dimulai dari simpul 1.

1. *Source Code* Program dalam bahasa Java
2. Algoritma *Branch and Bound* dengan *reduced cost matrix (*menggunakan *library Jgraphx* untuk visualisasi *graph)*

import javax.swing.JFrame;

import com.mxgraph.swing.mxGraphComponent;

import com.mxgraph.view.mxGraph;

import com.mxgraph.layout.\*;

import java.awt.BorderLayout;

import java.io.\*;

import java.util.\*;

import java.util.Comparator;

import java.util.PriorityQueue;

/\* Class Node.

\*/

class Node {

/\* Constructor.

\*/

public Node(){

visited = new ArrayList<Integer>();

cost = -99;

matrix = new int[row][col];

}

/\* Constructor.

\* @param v array berisi path dari simpul awal ke simpul node ini.

\* @param b cost sesuai path yang telah dilalui.

\* @param temp matrix yang telah direduksi sesuai node ini.

\* @param r baris matrix.

\* @param c kolom matrix.

\*/

public Node(ArrayList<Integer> v, int b, int[][] temp, int r, int c){

row = r;

col = c;

Iterator<Integer> itr = v.iterator();

visited = new ArrayList<Integer>();

while (itr.hasNext()){

visited.add(itr.next());

}

cost = b;

matrix = new int[row][col];

for (int i = 0; i < row; i++)

for (int j = 0; j < col; j++)

matrix[i][j] = temp[i][j];

}

/\* Format Output kelas.

\*/

public String toString(){

String temp = new String("");

for (int i = 0; i < row; i++) {

for (int j = 0; j < col; j++)

temp += (Integer.toString(matrix[i][j]) + " ");

temp += "\n";

}

return "path : " + visited + "\ncost : " + cost + "\n" + temp;

}

/\* Path yang telah dilalui sejauh node ini dari simpul awal.

\*/

public ArrayList<Integer> visited;

/\* Cost sesuai path yang telah dilalui node ini.

\*/

public int cost;

/\* Matrix yang telah direduksi sesuai node ini.

\*/

public int matrix[][];

/\* banyaknya baris dan kolom

\*/

public int row, col;

}

/\* Class Priority Comparator.

\* Kelas yang menentukan pengaturan priority untuk priority queue.

\*/

class PriorityComparator implements Comparator<Object> {

public int compare(Object Q1, Object Q2){

Node a1 = (Node) Q1;

Node a2 = (Node) Q2;

return Float.compare(a1.cost,a2.cost);

}

}

/\* Class TspRCM

\*/

public class TspRCM extends JFrame{

/\* Constructor.

\* @param filename nama file tempat input berada.

\*/

TspRCM(String filename) throws IOException {

//set adjacency matrix

Scanner in = new Scanner(new File(filename));

row = in.nextInt();

col = in.nextInt();

matrixAwal = new int [row][col];

for (int i = 0; i < row; i++)

for (int j = 0; j < col; j++)

matrixAwal[i][j] = in.nextInt();

//set priority queue

Comparator<Object> comparator = new PriorityComparator();

queue = new PriorityQueue<Node>(10,comparator);

//set variable lain

path = new ArrayList<Integer>();

solusi = new Node();

}

/\* Prosedur yang mengatur visualisasi graf dengan menggunakan library

\*/

public void depictGraph() {

//setup gambar graph menggunakan API

mxGraph graph = new mxGraph();

Object parent = graph.getDefaultParent();

graph.getModel().beginUpdate();

try

{

Object v[] = new Object[row+1];

for (int i = 1; i <= row; i++)

v[i] = graph.insertVertex(parent, null, Integer.toString(i), 20, 20, 80, 30, "shape=ellipse;fontSize=18;");

Iterator<Integer> itr = solusi.visited.iterator();

int m = 1; // simpul awal

int n;

itr.next();

while (itr.hasNext()){

n = itr.next();

graph.insertEdge(parent, null, Integer.toString(matrixAwal[m-1][n-1]), v[m], v[n],"strokeColor=red;fontColor=red;rounded=1;fontSize=13;");

matrixAwal[m-1][n-1] = -99;

m = n;

}

for (int i = 0; i < row; i++)

for (int j = 0; j < col; j++)

if (matrixAwal[i][j] != -99)

graph.insertEdge(parent, null, Integer.toString(matrixAwal[i][j]), v[i + 1], v[j + 1], "rounded=1;fontSize=13;");

}

finally {

graph.getModel().endUpdate();

}

mxGraphComponent graphComponent = new mxGraphComponent(graph);

getContentPane().add(graphComponent, BorderLayout.CENTER);

mxCircleLayout layout2 = new mxCircleLayout(graph);

layout2.execute(parent);

mxParallelEdgeLayout layout1 = new mxParallelEdgeLayout(graph);

layout1.execute(parent);

}

/\* Getter.

\* Banyaknya Node yang telah dibangkitkan.

\*/

public int getCountNode(){

return countNode;

}

/\* Getter.

\* Path solusi akhir dari simpul awal ke semua simpul dan kembali lagi.

\*/

public ArrayList<Integer> getPath(){

for (int i = 0; i < solusi.visited.size(); i++)

solusi.visited.set(i,solusi.visited.get(i) + 1);

return solusi.visited;

}

/\* Prosedur yang melakukan copy isi dari matrix.

\*/

public void copyMatrix(int[][] src, int[][] dst){

for (int i = 0; i < row; i++)

for (int j = 0; j < col; j++)

dst[i][j] = src[i][j];

}

/\* Prosedur yang mencari nilai minimal pada row tertentu.

\*/

public int findMinRow(int[][] m, int row){

int i = 0;

while (i < col && m[row][i] == -99)

i++;

if (i == col) {

return -99;

}

else {

int min = m[row][i];

for (int j = i+1; j < col; j++)

if (m[row][j] != -99 && min > m[row][j])

min = m[row][j];

return min;

}

}

/\* Prosedur yang mencari nilai minimal pada col tertentu.

\*/

public int findMinCol(int[][] m, int col){

int i = 0;

while (i < row && m[i][col] == -99)

i++;

if (i == row) {

return -99;

}

else {

int min = m[i][col];

for (int j = i+1; j < row; j++)

if (m[j][col] != -99 && min > m[j][col])

min = m[j][col];

return min;

}

}

/\* Prosedur mereduksi baris tertentu.

\*/

public void reduceRow(int[][] m, int row, int subtractor) {

for (int i = 0; i < col; i++){

if (m[row][i] != -99)

m[row][i] -= subtractor;

}

}

/\* Prosedur mereduksi kolom tertentu.

\*/

public void reduceCol(int[][] m, int col, int subtractor) {

for (int i = 0; i < row; i++){

if (m[i][col] != -99)

m[i][col] -= subtractor;

}

}

/\* Prosedur yang mengassign nilai infinite pada baris tertentu.

\*/

public void makeRowInfinite(int[][] m, int row) {

for (int i = 0; i < col; i++)

m[row][i] = -99;

}

/\* Prosedur yang mengassign nilai infinite pada baris tertentu.

\*/

public void makeColInfinite(int[][] m, int col) {

for (int i = 0; i < row; i++)

m[i][col] = -99;

}

/\* Prosedur yang menghitung cost reduksi.

\* @param m matrix yang akan dihitung.

\* @param isRoot menentukan apakah node root atau bukan.

\*/

public int countCost(int[][] m, boolean isRoot) {

int temp;

int cost = 0;

if (!isRoot){

int s = path.get(path.size() - 1);

int r = path.get(path.size() - 2);

cost += m[r][s];

makeRowInfinite(m,r);

makeColInfinite(m,s);

m[s][0] = -99;

}

for (int i = 0; i < row; i++){

temp = findMinRow(m,i);

if (temp != 0 && temp != -99){

cost += temp;

reduceRow(m,i,temp);

}

}

for (int i = 0; i < col; i++){

temp = findMinCol(m,i);

if (temp != 0 && temp != -99){

cost += temp;

reduceCol(m,i,temp);

}

}

return cost;

}

/\* Prosedur yang menghitung cost total perjalanan.

\*/

public int countSolutionCost() {

int sum = 0;

Iterator<Integer> itr = solusi.visited.iterator();

int m = 1; // simpul awal

int n;

itr.next();

while (itr.hasNext()) {

n = itr.next();

sum += matrixAwal[m - 1][n - 1];

m = n;

}

return sum;

}

/\* Prosedur yang melakukan penyelesaian problem TSP

\* menggunakan algoritma Branch n bound versi reduksi matrix

\*/

public void solveBnB(){

int tempMatrix[][] = new int [row][col];

//Set root node

countNode++;

path.add(0);

copyMatrix(matrixAwal,tempMatrix);

int cost = countCost(tempMatrix, true);

queue.add(new Node(path,cost,tempMatrix,row,col));

//Mulai eksekusi semua elemen queue

while (!queue.isEmpty()) {

Node temp = queue.poll();

path = temp.visited;

if (path.size() == row){ //Daun/solusi dari state tree dicapai

path.add(0); //ke simpul awal

if ((solusi.cost == -99) || (temp.cost < solusi.cost)){

solusi = new Node(path,temp.cost,temp.matrix,row,col);

//hapus semua node yang lower cost nya lebih besar

Iterator<Node> itr = queue.iterator();

while (itr.hasNext()){

Node temp2 = itr.next();

if (temp2.cost > solusi.cost)

itr.remove();

}

}

}

else {

//generate cabang

for (int i = 0; i < col; i++) {

if (!path.contains(i)){ //belum dikunjungi

countNode++;

copyMatrix(temp.matrix,tempMatrix);

path.add(i);

cost = countCost(tempMatrix, false) + temp.cost;

Node tes = new Node(path,cost,tempMatrix,row,col);

queue.add(tes);

path.remove(path.size() - 1);

}

}

}

}

}

public static void main(String arg[]) throws IOException {

TspRCM problem = new TspRCM("grafberarah.txt");

long startTime = System.currentTimeMillis();

problem.solveBnB();

long endTime = System.currentTimeMillis();

long duration = (endTime - startTime);

System.out.println("Solusi path : " +problem.getPath());

System.out.println("Cost path : " +problem.countSolutionCost());

System.out.println("Banyak node yang dibangitkan : " +problem.getCountNode());

System.out.println("Lama waktu proses : " + duration + " milliseconds");

problem.depictGraph();

problem.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

problem.setSize(400, 320);

problem.setVisible(true);

}

private int col;

private int row;

private int matrixAwal[][];

private Node solusi;

private PriorityQueue<Node> queue;

private ArrayList<Integer> path;

private int countNode = 0;

}

1. Algoritma *Branch and Bounrd* dengan Bobot Tur Lengkap

import javax.swing.JFrame;

import com.mxgraph.swing.mxGraphComponent;

import com.mxgraph.view.mxGraph;

import com.mxgraph.layout.\*;

import java.awt.BorderLayout;

import java.io.\*;

import java.util.\*;

import java.util.Comparator;

import java.util.PriorityQueue;

/\* Class Node.

\*/

class Node {

/\* Constructor.

\*/

public Node(){

visited = new ArrayList<Integer>();

bound = -99;

}

/\* Constructor.

\* @param v array berisi path dari simpul awal ke simpul node ini.

\* @param b cost sesuai path yang telah dilalui.

\*/

public Node(ArrayList<Integer> v, float b){

Iterator<Integer> itr = v.iterator();

visited = new ArrayList<Integer>();

while (itr.hasNext()){

visited.add(itr.next());

}

bound = b;

}

/\* Path yang telah dilalui sejauh node ini dari simpul awal.

\*/

public ArrayList<Integer> visited;

/\* Cost sesuai path yang telah dilalui node ini.

\*/

public float bound;

}

/\* Class Priority Comparator.

\* Kelas yang menentukan pengaturan priority untuk priority queue.

\*/

class PriorityComparator implements Comparator<Object> {

public int compare(Object Q1, Object Q2){

Node a1 = (Node) Q1;

Node a2 = (Node) Q2;

return Float.compare(a1.bound,a2.bound);

}

}

/\* Class Tsp

\*/

public class Tsp extends JFrame{

/\* Constructor.

\* @param filename nama file tempat input berada.

\*/

Tsp(String filename) throws IOException {

//set adjacency matrix

Scanner in = new Scanner(new File(filename));

row = in.nextInt();

col = in.nextInt();

matrix = new int [row][col];

visitedEdges = new boolean[row][col];

for (int i = 0; i < row; i++)

for (int j = 0; j < col; j++){

matrix[i][j] = in.nextInt();

visitedEdges[i][j] = false;

}

//set priority queue

Comparator<Object> comparator = new PriorityComparator();

queue = new PriorityQueue<Node>(10,comparator);

path = new ArrayList<Integer>();

solusi = new Node();

}

/\* Prosedur yang mengatur visualisasi graf dengan menggunakan library

\*/

public void depictGraph() {

//setup gambar graph menggunakan API

mxGraph graph = new mxGraph();

Object parent = graph.getDefaultParent();

graph.getModel().beginUpdate();

try

{

Object v[] = new Object[row+1];

for (int i = 1; i <= row; i++)

v[i] = graph.insertVertex(parent, null, Integer.toString(i), 20, 20, 80, 30, "shape=ellipse;fontSize=18;");

Iterator<Integer> itr = solusi.visited.iterator();

int m = 1; // simpul awal

int n;

itr.next();

while (itr.hasNext()){

n = itr.next();

graph.insertEdge(parent, null, Integer.toString(matrix[m-1][n-1]), v[m], v[n], "strokeColor=red;fontColor=red;rounded=1;endArrow=none;fontSize=13;");

matrix[m-1][n-1] = -99;

m = n;

}

for (int i = 0; i < row; i++)

for (int j = i+1; j < col; j++)

if (matrix[i][j] != -99 && matrix[j][i] != -99)

graph.insertEdge(parent, null, Integer.toString(matrix[i][j]), v[i + 1], v[j + 1], "rounded=1;endArrow=none;fontSize=13;");

}

finally{

graph.getModel().endUpdate();

}

mxGraphComponent graphComponent = new mxGraphComponent(graph);

getContentPane().add(graphComponent, BorderLayout.CENTER);

mxCircleLayout layout2 = new mxCircleLayout(graph);

layout2.execute(parent);

}

/\* Getter.

\* Banyaknya Node yang telah dibangkitkan.

\*/

public int getCountNode(){

return countNode;

}

/\* Getter.

\* Path solusi akhir dari simpul awal ke semua simpul dan kembali lagi.

\*/

public ArrayList<Integer> getPath(){

for (int i = 0; i < solusi.visited.size(); i++)

solusi.visited.set(i,solusi.visited.get(i) + 1);

return solusi.visited;

}

/\* Prosedur yang menghitung lower bound

\*/

public float countBound(){

int count;

float bnd = 0;

PriorityQueue<Integer> temp = new PriorityQueue<Integer>();

for (int i = 0; i < row; i++){

count = 0;

for (int j = 0; j < col; j++){

if (matrix[i][j] != -99){

if (visitedEdges[i][j]){

bnd += matrix[i][j];

count++;

}

else{

temp.offer(matrix[i][j]);

}

}

}

switch (count) {

case 0: bnd += temp.poll();

bnd += temp.poll();

break;

case 1: bnd += temp.poll();

break;

default: break;

}

temp.clear();

}

return bnd/2;

}

/\* Prosedur yang mengatur isi dari visitedEdges sesuai path saat tertentu.

\*/

public void checkVisitedEdges(){

Iterator<Integer> itr = path.iterator();

//reset visited edges

for (int i = 0; i < row; i++)

for (int j = 0; j < col; j++){

visitedEdges[i][j] = false;

}

//store visited edges berdasarkan path

int m = 0; // simpul awal

int n;

itr.next();

while (itr.hasNext()){

n = itr.next();

visitedEdges[m][n] = true;

visitedEdges[n][m] = true;

m = n;

}

}

/\* Prosedur yang menghitung cost total perjalanan.

\*/

public int countCost(){

int sum = 0;

Iterator<Integer> itr = solusi.visited.iterator();

int m = 1; // simpul awal

int n;

itr.next();

while (itr.hasNext()){

n = itr.next();

sum += matrix[m-1][n-1];

m = n;

}

return sum;

}

/\* Prosedur yang melakukan penyelesaian problem TSP

\* menggunakan algoritma Branch n bound versi bobot tur lengkap

\*/

public void solveBnB(){

countNode++;

path.add(0); //simpul awal

float bound = countBound();

queue.add(new Node(path,bound));

while (!queue.isEmpty()){

Node temp = queue.poll();

path = temp.visited;

checkVisitedEdges();

if (path.size() == row){ //Daun/solusi dari state tree dicapai

visitedEdges[path.get(path.size() - 1)][0] = true;

visitedEdges[0][path.get(path.size() - 1)] = true;

path.add(0); //ke simpul awal

bound = countBound();

if ((solusi.bound == -99) || (bound < solusi.bound)){

solusi = new Node(path,bound);

//hapus semua node yang lower bound nya lebih besar

Iterator<Node> itr = queue.iterator();

while (itr.hasNext()){

Node temp2 = itr.next();

if (temp2.bound > solusi.bound)

itr.remove();

}

}

visitedEdges[path.get(path.size() - 1)][0] = false;

visitedEdges[0][path.get(path.size() - 1)] = false;

}

else {

//generate cabang

for (int i = 0; i < col; i++) {

if (!path.contains(i)){ //belum dikunjungi

countNode++;

visitedEdges[path.get(path.size() - 1)][i] = true;

visitedEdges[i][path.get(path.size() - 1)] = true;

path.add(i);

bound = countBound();

queue.add(new Node(path,bound));

path.remove(path.size() - 1);

visitedEdges[path.get(path.size() - 1)][i] = false;

visitedEdges[i][path.get(path.size() - 1)] = false;

}

}

}

}

}

public static void main(String arg[]) throws IOException {

Tsp problem = new Tsp("graftidakberarah.txt");

long startTime = System.currentTimeMillis();

problem.solveBnB();

long endTime = System.currentTimeMillis();

long duration = (endTime - startTime);

System.out.println("Solusi path : " +problem.getPath());

System.out.println("Cost path : " +problem.countCost());

System.out.println("Banyak node yang dibangitkan : " +problem.getCountNode());

System.out.println("Lama waktu proses : " + duration + " milliseconds");

problem.depictGraph();

problem.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

problem.setSize(400, 320);

problem.setVisible(true);

}

private int col;

private int row;

private int matrix[][];

private Node solusi;

private PriorityQueue<Node> queue;

private ArrayList<Integer> path;

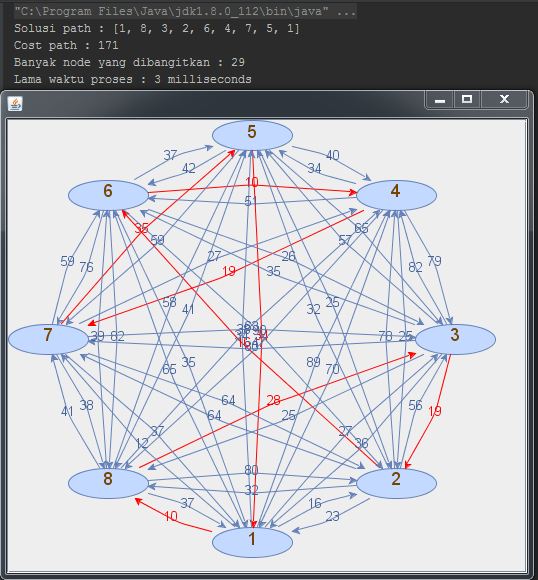
private boolean visitedEdges[][];

private int countNode = 0;

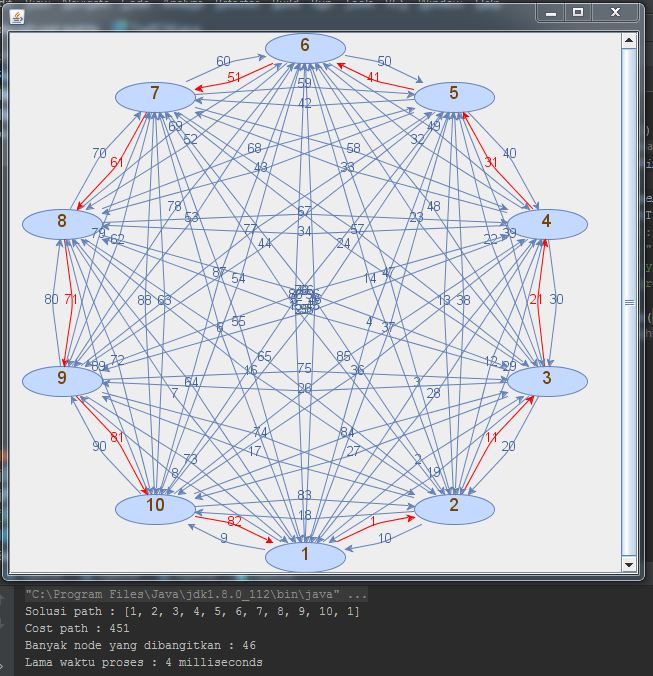
}

1. Contoh *Input* dan *Output*
2. *Branch and Bound* dengan *Reduced Cost Matrix*









1. *Branch and Bound* dengan Bobot Tur Lengkap



