**IF2211 – Strategi Algoritma**

**Tugas Kecil III**

**Travelling Salesman Problem Menggunakan Reduced Cost Matrix dan Complete Tour Weight**

oleh:

13515077 – Rionaldi Chandraseta



**TEKNIK INFORMATIKA**

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**2017**

# Deskripsi Persoalan

*Travelling Salesperson Problem* (TSP) adalah sebuah persoalan dimana diberikan beberapa kota dan jarak antar tiap kota-kota tersebut. Solusi yang dicari adalah cara untuk mendatangi setiap kota tepat sekali, kemudian kembali ke kota asal, dengan jarak tempuh minimal. Ada beberapa cara yang dapat digunakan untuk mencari solusi permasalahan ini. Pada kali ini, akan digunakan dua pendekatan *Branch and Bound*, yaitu *Reduced Cost Matrix* (matriks tereduksi) dan *Complete Tour Weight* (bobot tur lengkap).

Program yang dibuat harus dapat menyelesaikan persoalan TSP dengan menggunakan algoritma *Branch and Bound*. Nilai *bound* ditentukan melalui *reduced cost matrix* dan *complete tour weight*. Penyelesaian dengan *reduced cost matrix* memiliki masukan berupa graf berarah, sedangkan penyelesaian menggunakan *complete tour weight* memiliki masukan graf tidak berarah.

Masukan matriks dibaca dari file eksternal, program menghasilkan lintasan terpendek dan bobotnya, waktu pencarian solusi, jumlah simpul yang dibangkitkan, dan gambar graf yang menampilkan lintasan terpendek.

# Source Code Program (C++)

// File: Matrix.h

// Author: Rionaldi Chandraseta - 13515077

// Written on 2017-03-31

#ifndef MATRIX\_H

#define MATRIX\_H

#define INF -1

#include <fstream>

#include <iostream>

#include <string>

class Matrix {

public:

Matrix();

Matrix(std::string FileName);

Matrix(const Matrix& M);

~Matrix();

Matrix& operator=(const Matrix& M);

int getDist(int ori, int dest);

void setDist(int ori, int dest, int dist);

int getSize();

void write();

protected:

int\*\* Mem;

int MSize;

};

#endif

// File: Matrix.cpp

// Author: Rionaldi Chandraseta - 13515077

// Written on 2017-03-31

#include "Matrix.h"

Matrix::Matrix() {}

Matrix::Matrix(std::string FileName) {

// FileName must be validated first.

std::ifstream File(FileName);

std::string Str;

getline(File, Str);

int SpaceCount = 0;

int Idx = 0;

while (Idx < Str.length()) {

if (Str.at(Idx) == ' ') {

SpaceCount += 1;

}

Idx += 1;

}

MSize = SpaceCount + 1;

Mem = new int\* [MSize+1];

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

Mem[i] = new int [MSize+1];

}

// Read from beginning of file

File.clear();

File.seekg(0, std::ios::beg);

// INDEXING starts from 1

int row = 1;

while (!File.eof()) {

for (int col=1; col<=MSize; col++) {

File >> Mem[row][col];

}

row += 1;

}

File.close();

}

Matrix::Matrix(const Matrix& M) {

MSize = M.MSize;

Mem = new int\* [MSize+1];

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

Mem[i] = new int [MSize+1];

}

for (int row=1; row<=MSize; row++) {

for (int col=1; col<=MSize; col++) {

Mem[row][col] = M.Mem[row][col];

}

}

}

Matrix::~Matrix() {

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

delete [] Mem[i];

}

delete [] Mem;

}

Matrix& Matrix::operator=(const Matrix& M) {

MSize = M.MSize;

Mem = new int\* [MSize+1];

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

Mem[i] = new int [MSize+1];

}

for (int row=1; row<=MSize; row++) {

for (int col=1; col<=MSize; col++) {

Mem[row][col] = M.Mem[row][col];

}

}

return \*this;

}

int Matrix::getDist(int ori, int dest) {

return Mem[ori][dest];

}

void Matrix::setDist(int ori, int dest, int dist) {

Mem[ori][dest] = dist;

}

int Matrix::getSize() {

return MSize;

}

void Matrix::write() {

for (int row=1; row<=MSize; row++) {

for (int col=1; col<=MSize; col++) {

if (Mem[row][col] == -1) {

std::cout << " inf";

}

else{

if (Mem[row][col] < 10) {

std::cout << " ";

}

else {

std::cout << " ";

}

std::cout << Mem[row][col];

}

}

std::cout << std::endl;

}

}

// File: Path.h

// Author: Rionaldi Chandraseta - 13515077

// Written on 2017-03-31

#ifndef PATH\_H

#define PATH\_H

#include <vector>

class Path {

public:

Path();

Path(const Path& P);

void addNode(int nodeNum, double totalCost);

std::vector<int> getPath() const;

double getTotalCost() const;

private:

std::vector<int> TravelPath;

double TotalCost;

};

#endif

// File: Path.cpp

// Author: Rionaldi Chandraseta - 13515077

// Written on 2017-03-31

#include "Path.h"

Path::Path() {

TotalCost = 0;

}

Path::Path(const Path& P) {

for (int i=0; i<P.TravelPath.size(); i++) {

TravelPath.push\_back(P.TravelPath.at(i));

}

TotalCost = P.TotalCost;

}

void Path::addNode(int nodeNum, double totalCost) {

TravelPath.push\_back(nodeNum);

TotalCost = totalCost;

}

std::vector<int> Path::getPath() const {

return TravelPath;

}

double Path::getTotalCost() const {

return TotalCost;

}

// File: ComparePath.h

// Author: Rionaldi Chandraseta - 13515077

// Written on 2017-04-01

#ifndef COMPAREPATH\_H

#define COMPAREPATH\_H

#include "Path.h"

class ComparePath {

public:

bool operator() (Path& P1, Path& P2) {

if (P1.getTotalCost() == P2.getTotalCost()) {

return P1.getPath().size() < P2.getPath().size();

}

else {

return P1.getTotalCost() > P2.getTotalCost();

}

}

};

#endif

// File: RCMatrix.h

// Author: Rionaldi Chandraseta - 13515077

// Written on 2017-03-31

#include <queue>

#include "ComparePath.h"

#include "Matrix.h"

#include "Path.h"

#ifndef RCMATRIX\_H

#define RCMATRIX\_H

class RCMatrix {

public:

RCMatrix();

RCMatrix(std::string FileName);

RCMatrix(const RCMatrix& R);

RCMatrix createDummy();

Matrix getMatrix();

int getBound();

int getNumCheckedNode();

int lowestRowDist(int row);

int lowestColDist(int col);

void travel(int orig, int dest);

void reduceMatrix();

void solve();

Path getShortestPath();

private:

Matrix Dist;

std::priority\_queue<Path, std::vector<Path>, ComparePath> CurrentTravelPath;

int Bound;

int NumCheckedNode;

};

#endif

// File: RCMatrix.cpp

// Author: Rionaldi Chandraseta - 13515077

// Written on 2017-04-01

#include "RCMatrix.h"

RCMatrix::RCMatrix() {}

RCMatrix::RCMatrix(std::string FileName) : Dist(FileName) {

Bound = 0;

NumCheckedNode = 1;

}

RCMatrix RCMatrix::createDummy() {

RCMatrix Dummy;

Dummy.Dist = Dist;

Dummy.Bound = Bound;

Dummy.NumCheckedNode = NumCheckedNode;

return Dummy;

}

RCMatrix::RCMatrix(const RCMatrix& R) {

Dist = R.Dist;

Bound = R.Bound;

NumCheckedNode = R.NumCheckedNode;

}

Matrix RCMatrix::getMatrix() {

return Dist;

}

int RCMatrix::getBound() {

return Bound;

}

int RCMatrix::getNumCheckedNode() {

return NumCheckedNode;

}

int RCMatrix::lowestRowDist(int row) {

int min = 0x3f3f3f3f;

int size = Dist.getSize();

for (int i=1; i<=size; i++) {

if ((Dist.getDist(row, i) < min) && (Dist.getDist(row, i) >= 0)) {

min = Dist.getDist(row, i);

}

}

if (min == 0x3f3f3f3f) {

min = 0;

}

return min;

}

int RCMatrix::lowestColDist(int col) {

int min = 0x3f3f3f3f;

int size = Dist.getSize();

for (int i=1; i<=size; i++) {

if ((Dist.getDist(i, col) < min) && (Dist.getDist(i, col) >= 0)) {

min = Dist.getDist(i, col);

}

}

if (min == 0x3f3f3f3f) {

min = 0;

}

return min;

}

void RCMatrix::travel(int orig, int dest) {

int size = Dist.getSize();

for (int col=1; col<=size; col++) {

Dist.setDist(orig, col, INF);

}

for (int row=1; row<=size; row++) {

Dist.setDist(row, dest, INF);

}

Dist.setDist(dest, 1, INF);

}

void RCMatrix::reduceMatrix() {

Bound = 0;

// Reduce every row

int size = Dist.getSize();

for (int row=1; row<=size; row++) {

int lowDist = lowestRowDist(row);

for (int col=1; col<=size; col++) {

if (Dist.getDist(row, col) >= 0) {

Dist.setDist(row, col, Dist.getDist(row, col) - lowDist);

}

}

Bound += lowDist;

}

// Reduce every column

for (int col=1; col<=size; col++) {

int lowDist = lowestColDist(col);

for (int row=1; row<=size; row++) {

if (Dist.getDist(row, col) >= 0) {

Dist.setDist(row, col, Dist.getDist(row, col) - lowDist);

}

}

Bound += lowDist;

}

}

void RCMatrix::solve() {

reduceMatrix();

Path PStart;

PStart.addNode(1, Bound);

CurrentTravelPath.push(PStart);

bool finished = false;

while (!finished) {

Path currentShortestPath = CurrentTravelPath.top();

std::vector<int> visited = currentShortestPath.getPath();

finished = (CurrentTravelPath.top().getPath().size() == Dist.getSize());

if (!finished) {

CurrentTravelPath.pop();

int lastVisited = visited.back();

RCMatrix AliveNode = createDummy();

int travelNode = 0;

while (travelNode < visited.size()) {

if (travelNode >= 1) {

AliveNode.travel(visited.at(travelNode-1), visited.at(travelNode));

if (travelNode < visited.size()-1) {

AliveNode.reduceMatrix();

}

}

travelNode += 1;

}

int size = Dist.getSize();

for (int dest=1; dest<=size; dest++) {

if (AliveNode.Dist.getDist(lastVisited, dest) >= 0) {

NumCheckedNode += 1;

Path currentCheckedPath(currentShortestPath);

RCMatrix PossibleNode(AliveNode);

PossibleNode.reduceMatrix();

double dist = (double) PossibleNode.Dist.getDist(lastVisited, dest);

double bound = (double) PossibleNode.getBound();

currentCheckedPath.addNode(dest, currentCheckedPath.getTotalCost()+dist+bound);

CurrentTravelPath.push(currentCheckedPath);

}

}

}

}

}

Path RCMatrix::getShortestPath() {

solve();

Path shortestPath(CurrentTravelPath.top());

shortestPath.addNode(1, shortestPath.getTotalCost());

return shortestPath;

}

// File: CTMatrix.h

// Author: Rionaldi Chandraseta - 13515077

// Written on 2017-03-31

#include <queue>

#include "ComparePath.h"

#include "Matrix.h"

#include "Path.h"

#ifndef CTMATRIX\_H

#define CTMATRIX\_H

class CTMatrix {

public:

CTMatrix(std::string FileName);

Matrix getMatrix();

int getNumCheckedNode();

double getInitialCost();

double getCost(int node);

double getCost(int orig, int dest);

double getNodesCost(std::vector<int> visited);

void solve();

Path getShortestPath();

private:

Matrix Dist;

std::priority\_queue<Path, std::vector<Path>, ComparePath> CurrentTravelPath;

int NumCheckedNode;

};

#endif

// File: CTMatrix.cpp

// Author: Rionaldi Chandraseta - 13515077

// Written on 2017-04-03

#include <algorithm>

#include "CTMatrix.h"

CTMatrix::CTMatrix(std::string FileName) : Dist(FileName) {

NumCheckedNode = 1;

}

Matrix CTMatrix::getMatrix() {

return Dist;

}

int CTMatrix::getNumCheckedNode() {

return NumCheckedNode;

}

double CTMatrix::getInitialCost() {

double totalCost = 0;

int size = Dist.getSize();

for (int row=1; row<=size; row++) {

double min = 0x3f3f3f3f;

double prevMin = 0x3f3f3f3f;

for (int col=1; col<=size; col++) {

if (Dist.getDist(row, col) >= 0) {

if (Dist.getDist(row, col) < min) {

prevMin = min;

min = Dist.getDist(row, col);

}

else if (Dist.getDist(row, col) < prevMin) {

prevMin = Dist.getDist(row, col);

}

}

}

totalCost += (min + prevMin);

}

return totalCost\*0.5;

}

double CTMatrix::getCost(int orig, int dest) {

double min = 0x3f3f3f3f;

int size = Dist.getSize();

for (int col=1; col<=size; col++) {

if ((col != dest) && (Dist.getDist(orig, col) >= 0) && (Dist.getDist(orig, col) < min)) {

min = Dist.getDist(orig, col);

}

}

double cost = min + Dist.getDist(orig, dest);

return cost;

}

double CTMatrix::getCost(int node) {

double min = 0x3f3f3f3f;

double prevMin = 0x3f3f3f3f;

int size = Dist.getSize();

for (int col=1; col<=size; col++) {

if ((col != node) && (Dist.getDist(node, col) >= 0)) {

if (Dist.getDist(node, col) < min) {

prevMin = min;

min = Dist.getDist(node, col);

}

else if (Dist.getDist(node, col) < prevMin) {

prevMin = Dist.getDist(node, col);

}

}

}

return (min + prevMin);

}

double CTMatrix::getNodesCost(std::vector<int> visited) {

double totalCost = 0;

if (visited.size()==1) {

totalCost = 2\*getInitialCost();

}

else {

int size = Dist.getSize();

int visSize = visited.size();

for (int node=1; node<=size; node++) {

std::ptrdiff\_t index = find(visited.begin(), visited.end(), node) - visited.begin();

// Not found in visited

if (find(visited.begin(), visited.end(), node) == visited.end()) {

totalCost += getCost(node);

}

else {

if (index == 0) {

if (visSize == Dist.getSize()+1) {

totalCost += Dist.getDist(visited.at(index), visited.at(visSize-2));

totalCost += Dist.getDist(visited.at(index), visited.at(index+1));

}

else {

totalCost += getCost(visited.at(index), visited.at(index+1));

}

}

else if (index == visSize-1) {

if (visSize == Dist.getSize()+1) {

totalCost += Dist.getDist(visited.at(index), visited.at(index-1));

totalCost += Dist.getDist(visited.at(index), visited.at(0));

}

else {

totalCost += getCost(visited.at(index), visited.at(index-1));

}

}

else {

totalCost += Dist.getDist(visited.at(index), visited.at(index-1));

totalCost += Dist.getDist(visited.at(index), visited.at(index+1));

}

}

}

}

return totalCost\*0.5;

}

void CTMatrix::solve() {

Path PStart;

double initialCost = getInitialCost();

PStart.addNode(1, initialCost);

CurrentTravelPath.push(PStart);

bool finished = false;

while (!finished) {

Path currentShortestPath = CurrentTravelPath.top();

std::vector<int> visited = currentShortestPath.getPath();

finished = CurrentTravelPath.top().getPath().size() == Dist.getSize();

if (!finished) {

CurrentTravelPath.pop();

int size = Dist.getSize();

for (int dest=1; dest<=size; dest++) {

if ((Dist.getDist(visited.back(), dest) >= 0) && (find(visited.begin(), visited.end(), dest) == visited.end())) {

NumCheckedNode += 1;

std::vector<int> checking = visited;

checking.push\_back(dest);

Path currentCheckedPath(currentShortestPath);

double newCost = getNodesCost(checking);

currentCheckedPath.addNode(dest, newCost);

CurrentTravelPath.push(currentCheckedPath);

}

}

}

}

}

Path CTMatrix::getShortestPath() {

solve();

Path shortestPath(CurrentTravelPath.top());

std::vector<int> finalPath = shortestPath.getPath();

finalPath.push\_back(1);

double finalCost = getNodesCost(finalPath);

shortestPath.addNode(1, finalCost);

return shortestPath;

}

// File: Main.cpp

// Author: Rionaldi Chandraseta - 13515077

// Written on 2017-03-31

#include <algorithm>

#include <chrono>

#include <fstream>

#include "Matrix.h"

#include "RCMatrix.h"

#include "CTMatrix.h"

int main() {

std::chrono::time\_point<std::chrono::system\_clock> start, end;

std::string FileName;

std::cout << "Enter the file name: ";

std::cin >> FileName;

std::string extLess = FileName.substr(0, FileName.size()-3);

std::cout << "Checking files..." << "\n";

Path shortestPath;

if ((FileName.at(0) == 'r') && (FileName.at(1) == 'c')) {

RCMatrix M(FileName);

std::cout << "Finished loading data" << "\n" << "\n";

std::cout << "== Travelling Salesman Problem with Reduced Cost Matrix ==" << "\n";

M.getMatrix().write();

std::cout << "\n";

start = std::chrono::system\_clock::now();

shortestPath = M.getShortestPath();

end = std::chrono::system\_clock::now();

std::cout << "Shortest path: ";

int numNode = shortestPath.getPath().size();

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

std::cout << shortestPath.getPath().at(i);

if (i<numNode-1) {

std::cout << " -> ";

}

}

std::cout << "\n";

std::cout << "Cost = " << shortestPath.getTotalCost() << "\n";

std::cout << "Checked " << M.getNumCheckedNode() << " nodes" << "\n";

std::chrono::duration<double> elapsed\_seconds = end-start;

std::cout << "Time taken: " << elapsed\_seconds.count()\*1000 << " ms\n";

// Output .gv

std::vector<int> path = shortestPath.getPath();

Matrix G(FileName);

path.pop\_back();

std::string fullPath = "../usr/graph\_" + extLess + ".gv";

std::cout << "GraphViz file generated to " << fullPath << "\n";

std::ofstream oFile;

oFile.open(fullPath.c\_str());

oFile << "digraph {\n";

oFile << "\tgraph [layout=circo, overlap=scale, splines=true, mindist=2.5]\n";

oFile << "\tnode [shape=circle]\n";

oFile << "\t1 [peripheries=2]\n";

for (int row=1; row<=G.getSize(); row++) {

std::ptrdiff\_t index = find(path.begin(), path.end(), row) - path.begin();

for (int col=1; col<=G.getSize(); col++) {

if (row != col) {

oFile << "\t" << row << " -> " << col << " [label=" << G.getDist(row, col);

if (index == path.size()-1) {

if (path.at(0) == col) {

oFile << ", color=red, penwidth=3.0, fontcolor=red]\n";

} else {

oFile << "]\n";

}

} else {

if (path.at(index+1) == col) {

oFile << ", color=red, penwidth=3.0, fontcolor=red]\n";

} else {

oFile << "]\n";

}

}

}

}

}

oFile << "}";

oFile.close();

}

else if ((FileName.at(0) == 'c') && (FileName.at(1) == 't')) {

CTMatrix M(FileName);

std::cout << "Finished loading data" << "\n";

std::cout << "== Travelling Salesman Problem with Complete Tour ==" << "\n";

M.getMatrix().write();

std::cout << "\n";

start = std::chrono::system\_clock::now();

shortestPath = M.getShortestPath();

end = std::chrono::system\_clock::now();

std::cout << "Shortest path: ";

int numNode = shortestPath.getPath().size();

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

std::cout << shortestPath.getPath().at(i) << " ";

if (i<numNode-1) {

std::cout << " -> ";

}

}

std::cout << "\n";

std::cout << "Cost = " << shortestPath.getTotalCost() << "\n";

std::cout << "Checked " << M.getNumCheckedNode() << " nodes" << "\n";

std::chrono::duration<double> elapsed\_seconds = end-start;

std::cout << "Time taken: " << elapsed\_seconds.count()\*1000 << " ms\n";

// Output .gv

std::vector<int> path = shortestPath.getPath();

Matrix G(FileName);

path.pop\_back();

std::string fullPath = "../usr/graph\_" + extLess + ".gv";

std::cout << "GraphViz file generated to " << fullPath << "\n";

std::ofstream oFile;

oFile.open(fullPath);

oFile << "graph {\n";

oFile << "\tgraph [layout=circo, overlap=scale, splines=true, mindist=2.5]\n";

oFile << "\tnode [shape=circle]\n";

oFile << "\t1 [peripheries=2]\n";

for (int row=1; row<=G.getSize(); row++) {

std::ptrdiff\_t index = find(path.begin(), path.end(), row) - path.begin();

for (int col=1; col<row; col++) {

if (row != col) {

oFile << "\t" << row << " -- " << col << " [label=" << G.getDist(row, col);

if (index == path.size()-1) {

if ((col == path.at(0)) || (col == path.at(index-1))) {

oFile << ", color=red, penwidth=3.0, fontcolor=red]\n";

} else {

oFile << "]\n";

}

} else {

if ((path.at(index+1) == col) || (path.at(index-1) == col)) {

oFile << ", color=red, penwidth=3.0, fontcolor=red]\n";

} else {

oFile << "]\n";

}

}

}

}

}

oFile << "}";

oFile.close();

}

else {

std::cout << "File not found!" << "\n";

std::cout << "Exiting program..." << "\n";

}

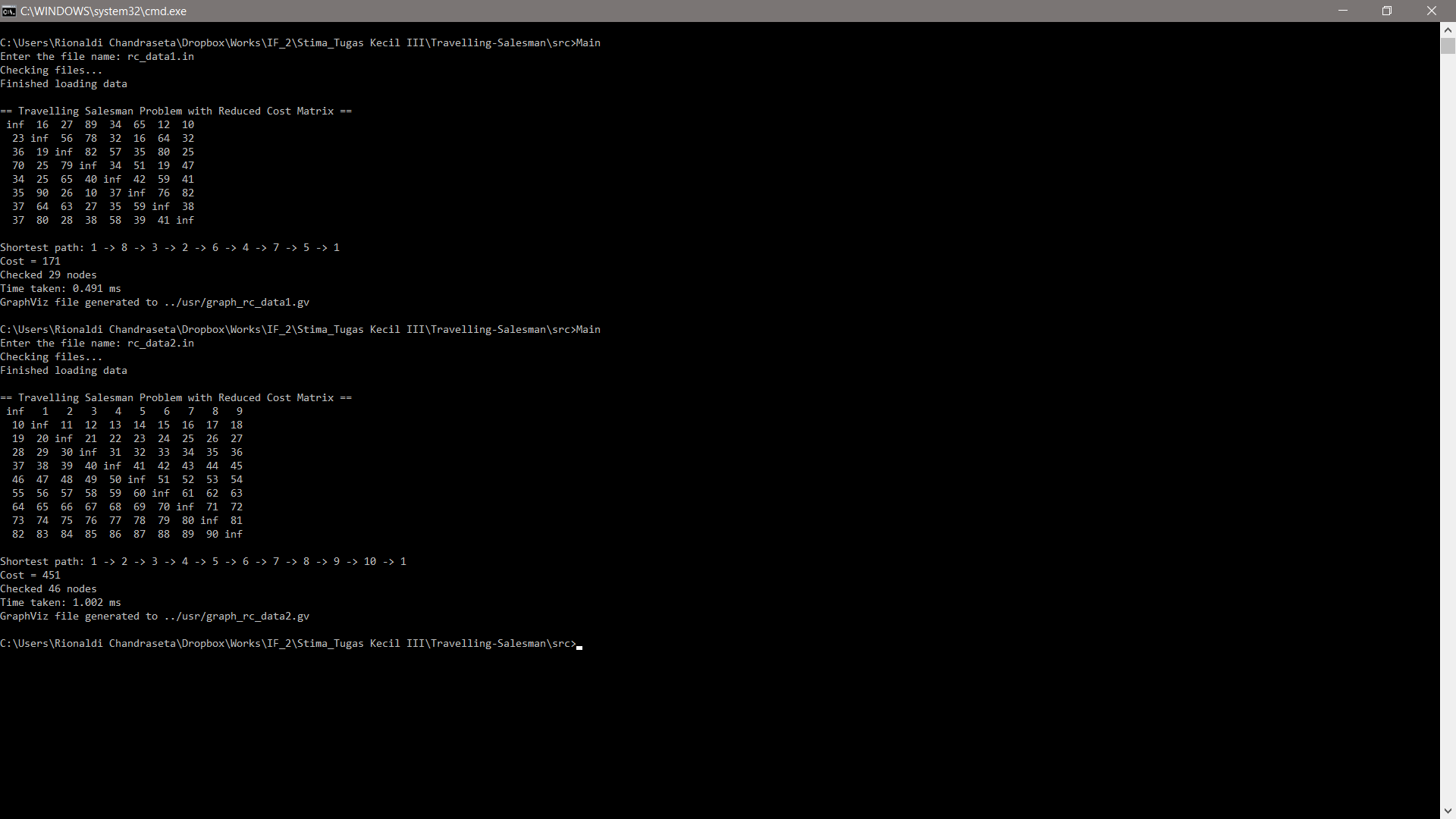
return 0;

}

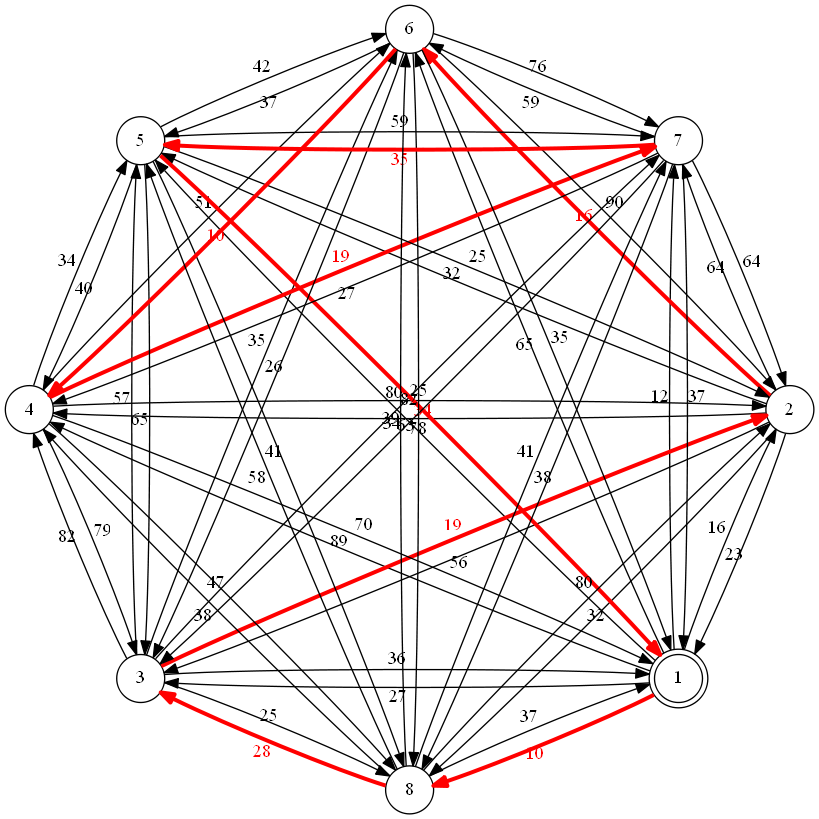
# Output

File rc\_data1.in dan rc\_data2.in berisi data uji penyelesaian TSP dengan *reduced cost matrix*. File ct\_data1.in dan ct\_data2.in berisi data uji penyelesaian TSP dengan Bobot Tur Lengkap. Penggambaran graf menggunakan GraphViz dengan *circo engine*. Ide menggunakan GraphViz karena telah menggunakannya di tugas IF2210 – Pemrograman Berorientasi Objek, *circo engine* karena rekomendasi dari teman yang menggunakannya.

## rc\_data1.in

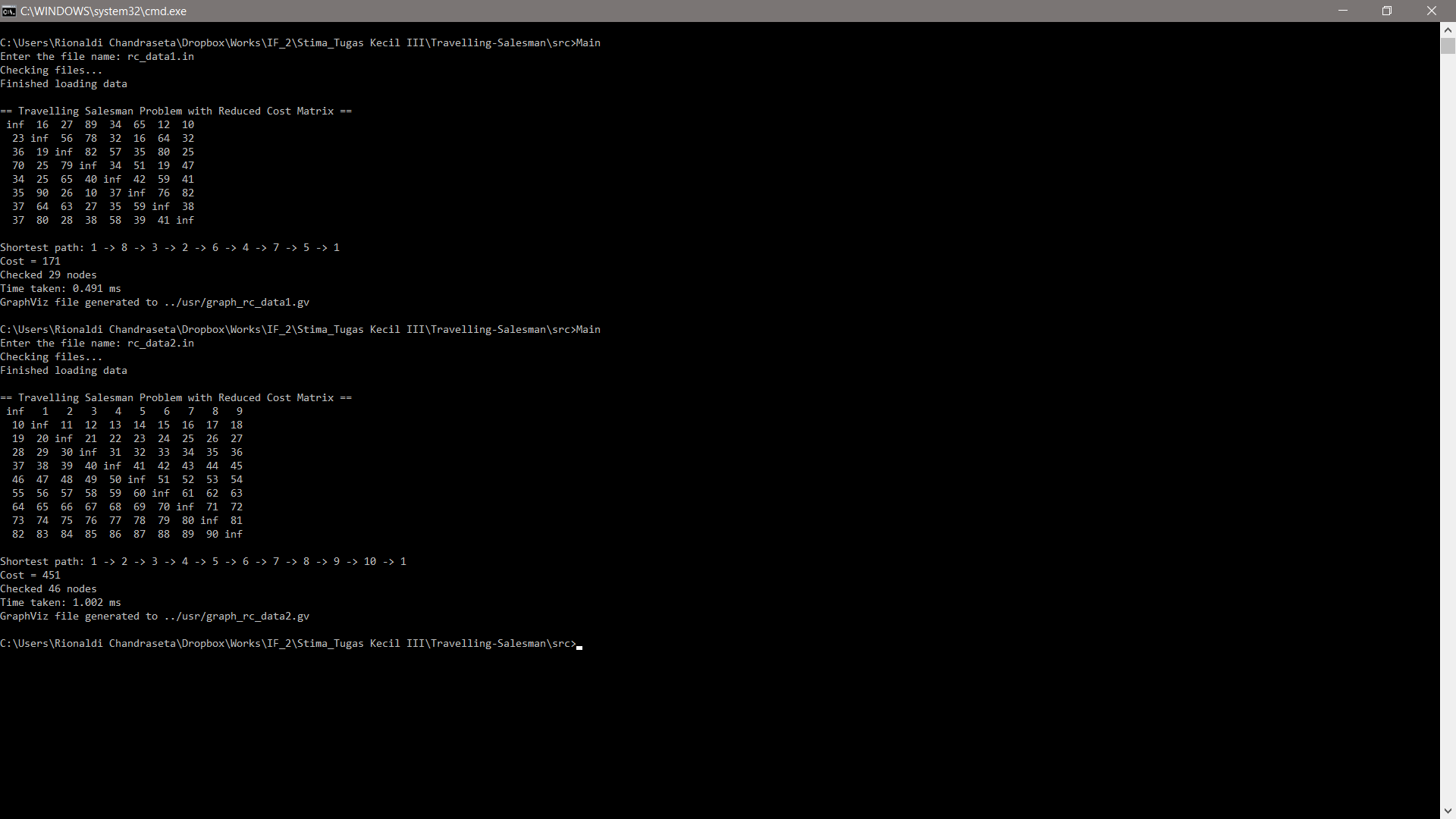


Screenshot 1 Hasil ouput rc\_data1.in

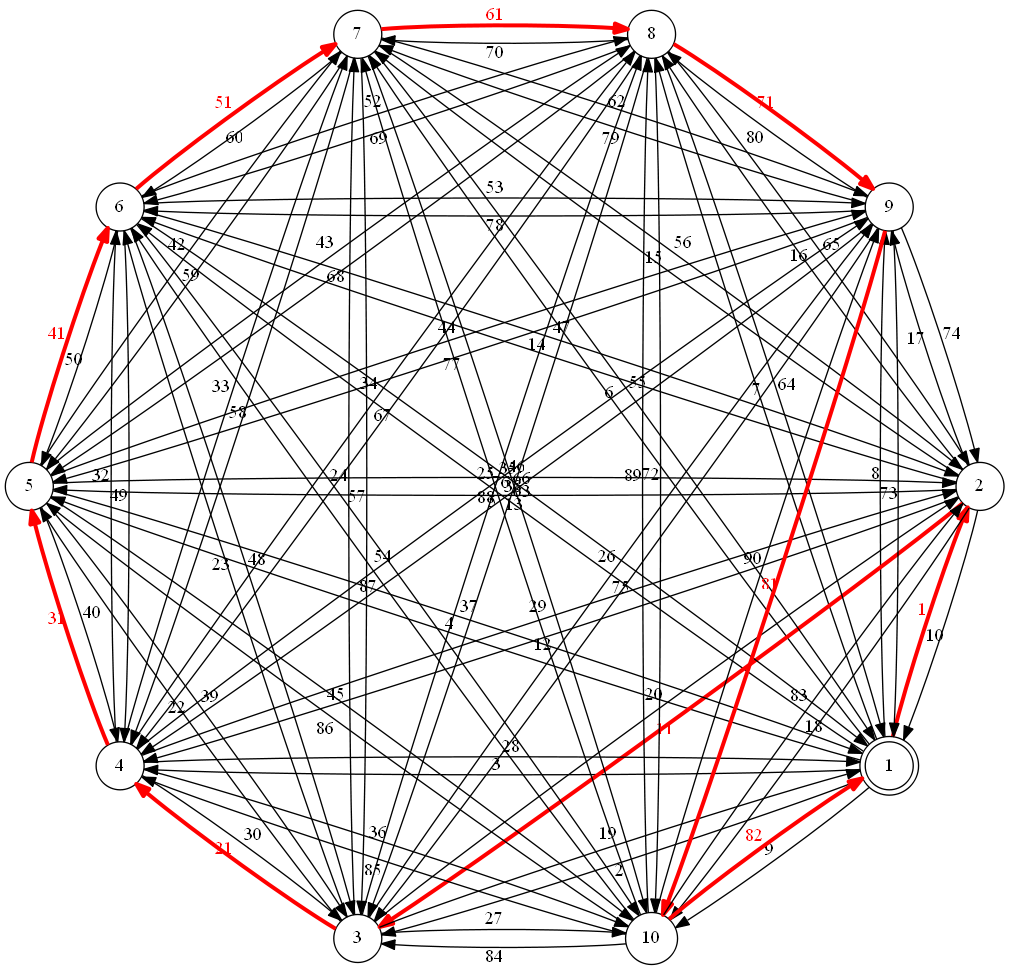


Graph 1 Lintasan tersingkat dari rc\_data1.in

## rc\_data2.in

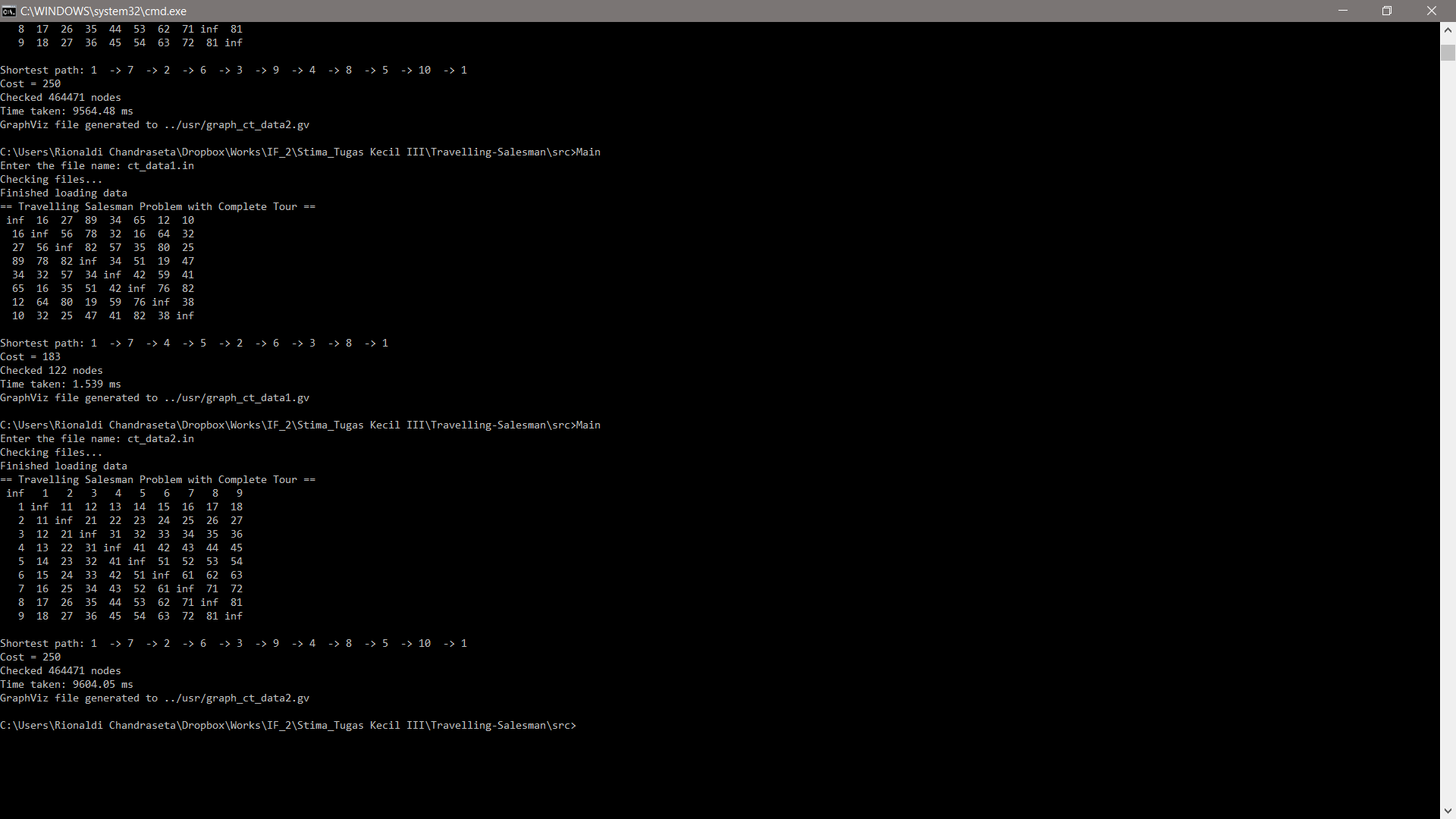


Screenshot 2 Hasil output rc\_data2.in

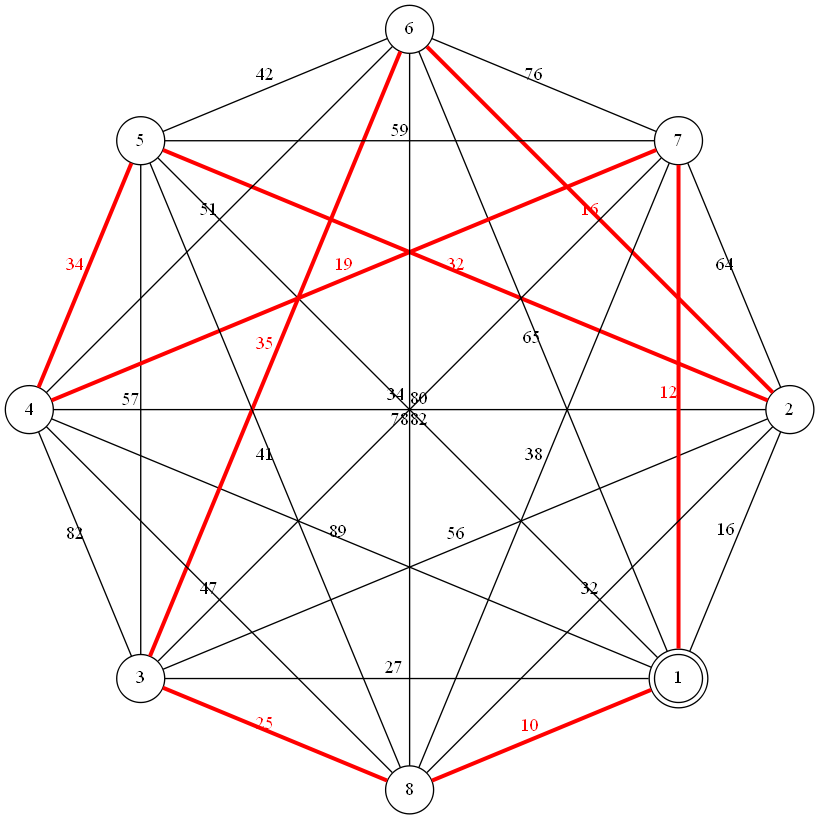


Graph 2 Lintasan tersingkat dari rc\_data2.in

## ct\_data1.in

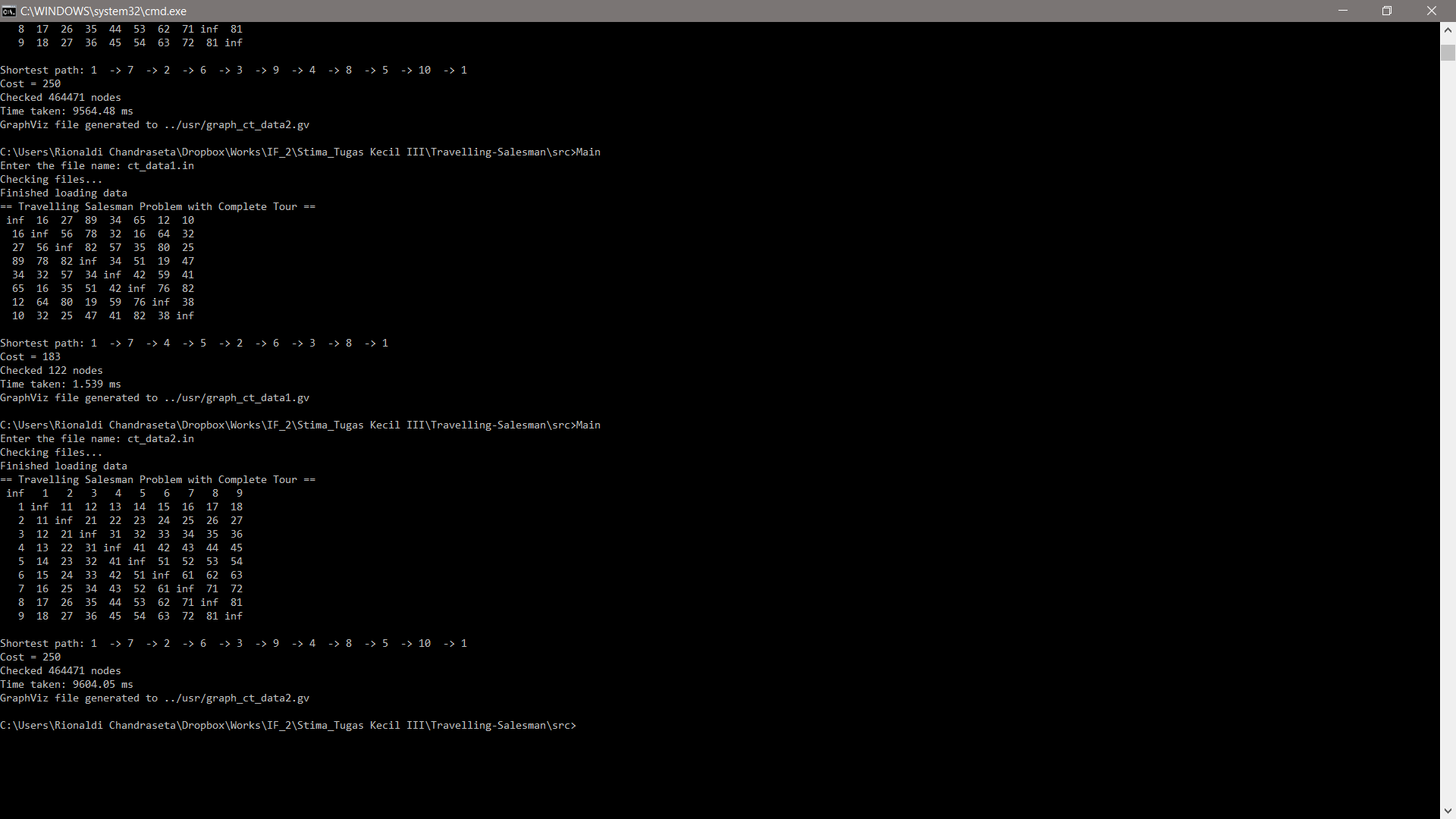


Screenshot 3 Hasil output ct\_data1.in

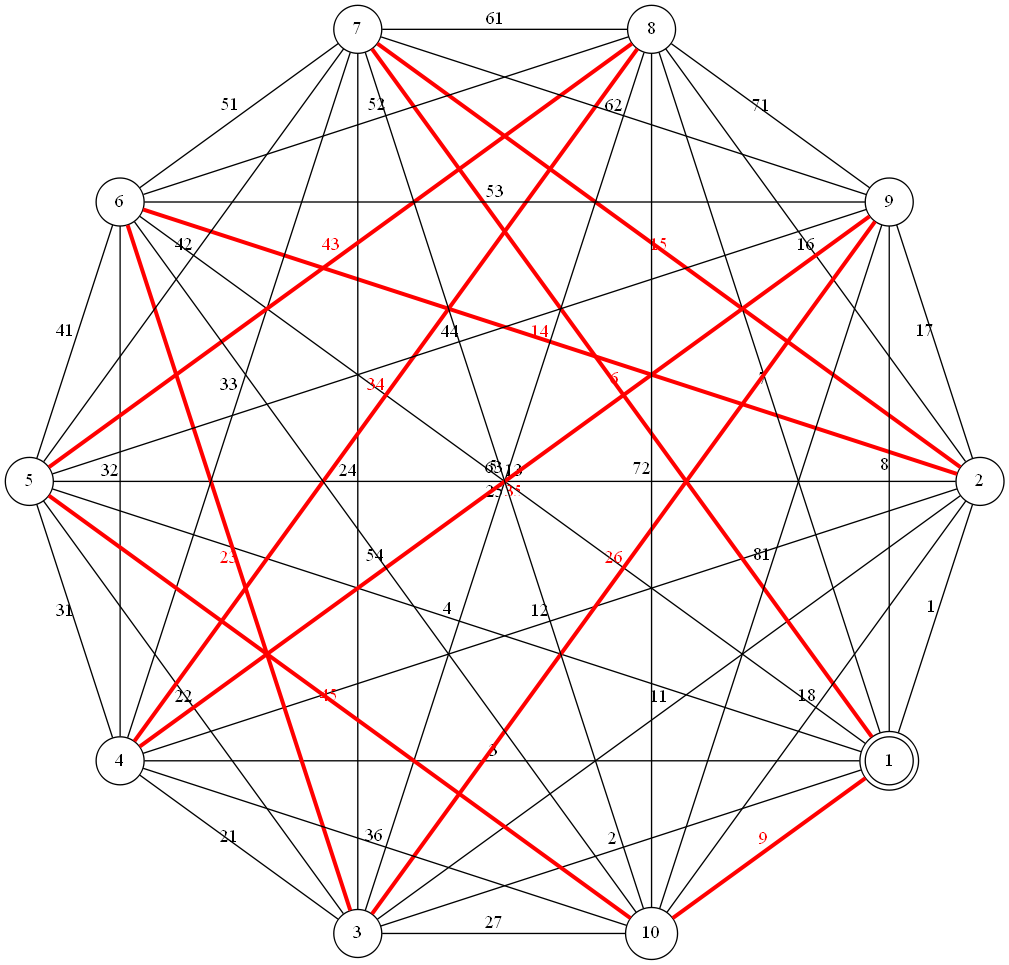


Graph 3 Lintasan tersingkat dari ct\_data1.in

## ct\_data2.in



Screenshot 4 Hasil output ct\_data2.in



Graph 4 Lintasan tersingkat dari ct\_data2.in