**Algorithm Design**

**Project\_2**

**LIL103 /LingHui, Liu**

**10/15/2015**

**Pseudo-code:**

**Dijkstra’s algorithm:**

**void** find() {

// use point which is next to start point to initialize the map

**for** (**int** i = 0; i < matrix.length; i++) {

// if (matrix[startIndex][i] != maxDis)

distanceMap.put(i, matrix[startIndex][i]);

}

**while** (findedSet.size() != matrix.length) {

**int** currentMinIndex = currentMinIndex();

// use this point to update the distance with other point

**for** (**int** i = 0; i < matrix.length; i++) {

**if** (!findedSet.contains(i) && matrix[currentMinIndex][i] != ***maxDis***

&& matrix[currentMinIndex][i] + distanceMap.get(currentMinIndex) < distanceMap.get(i))

distanceMap.put(i, matrix[currentMinIndex][i] + distanceMap.get(currentMinIndex));

}

// put it into the found set

findedSet.add(currentMinIndex);

}

}

**Floyd’s algorithm:**

**int**[][] floyd(**int**[][] matrix){

**int** size=matrix.length;

//initialize dist and path

**for**(**int** i=0;i< size;i++){

**for**(**int** j=0;j< size;j++){

*path*[i][j]=0;

*dist*[i][j]=matrix[i][j]; //a[]

}

}

**for**(**int** k=0;k< size;k++)

{

**for**(**int** i=0;i< size;i++)

{

**for**(**int** j=0;j< size;j++)

{

**if**(*dist*[i][k]!=*INF*&&

*dist*[k][j]!=*INF*&&

*dist*[i][k]+*dist*[k][j]< *dist*[i][j])

{

*dist*[i][j]=*dist*[i][k]+*dist*[k][j];

*path*[i][j]=k;

}

}

}

}

**return** *dist*;

}

**source code :**

1. I create a Tool class to create different data structure for adjacent matrix (2 dimensions and one dimension) for random sparse matrix and complete matrix & half-complete matrix , and the one dimension matrix and linked list both are transferred from 2 dimensions matrix .

import java.util.Random;

import java.util.Scanner;

public class Tools {

public static void Create\_2dfull\_array()

{

//complete array means half of the matrix except the diagonal are full of the weight

int a ; // set the number of nodes

System.out.println("please input the number of the nodes: ");

Scanner sc = new Scanner(System.in);

a = sc.nextInt();

int[][] array = new int[a][a];

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

{

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

{

array[i][j] = 30 ;

}

}

int temp = 0 ;

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

{

for(int j = temp ;j < a ;j++)

{

if(i == j)

{

array[j][i] = 0;

}

else

{

Random rand = new Random();

int weight = rand.nextInt(9)+1;// set the weight smaller than 10 , easy to test

array[j][i] = weight ;

array[i][j] = array[j][i];

}

}

temp++;

}//print the matrix

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

{

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

{

System.out.print(array[j][i] +" ");

}

System.out.println();

}

}

// spare matrix means that in the half matrix , there is only the number of (length of matrix -1 ) nodes exist ,and every node must have at least one connection with other node.

public static int[][] Create\_2dsparse\_array()

{

int a ;

System.out.println("please input the number of the nodes");

Scanner sc = new Scanner(System.in);

a = sc.nextInt() ;

int[][] array = new int[a][a];

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

{

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

{

array[i][j] = 30 ;

}

}

int temp = 0 ;

for(int i = 0 ; i < a-1 ;i ++)

{

Random rand = new Random();

int weight = rand.nextInt(9)+1;

int place = rand.nextInt(a-i-1);

array[place+i+1][i] = weight ;

array[i][place+i+1] = weight ;

}// print the sparse matrix

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

{

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

{

System.out.print(array[i][j]+" ");

}

System.out.println();

}

return array;

}

//totally random matrix means the number of nodes smaller than complete matrix and bigger than sparse matrix , and one node must have at least one connection with other nodes

public static int[][] Create\_2d\_halfsparse\_array()

{

int a ;

int b ;

int INF=10000; // set INF as the block strength between two nodes

Random rand = new Random();

System.out.println("please input the number of the nodes");

Scanner sc = new Scanner(System.in);

a = sc.nextInt();

System.out.println("please input the number of the edges");

b = sc.nextInt();

while(b < a -1|| b > (a\*a-a)/2)

{

System.out.println("the number of edges isn't proper fot the array,pleas input again ,the range of the input number is: "+(a-1)+" - "+(a\*a-a)/2);

b = sc.nextInt();

}

int[][] array = new int[a][a];

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

{

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

{

array[i][j] = 10000 ;

}

}

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

{

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

{

if(i == j)

array[i][j] = 0;

}

}

int temp = 0 ;

for(int i = 0 ; i < a-1 ;i ++)

{

int weight = rand.nextInt(9)+1;

int place = rand.nextInt(a-i-1);

array[place+i+1][i] = weight ;

array[i][place+i+1] = weight ;

}

int total = 0;

do{

int x = rand.nextInt(a);

int y = rand.nextInt(a);

int weight = rand.nextInt(9)+1;

if(x != y)

{

if(array[x][y] == INF)

{

array[x][y] = weight;

array[y][x] = array[x][y];

total++;

}

else

continue;

}

else

{

array[x][y] = 0;

}

}while(total < b-a +1);

return array;

}

// one dimension array could be transfered by two dimension matrix by using formula array[(i-1)\*i/2+j] = matrix[i][j]

public static int[] Create\_1d\_array(int a, int b)

{

int INF=10000;// set INF as the block strength of two nodes

Random rand = new Random();

Scanner sc = new Scanner(System.in);

while(b < a -1|| b > (a\*a-a)/2)

{

System.out.println("the number of edges isn't proper fot the array,pleas input again ,the range of the input number is: "+(a-1)+" - "+(a\*a-a)/2);

b = sc.nextInt();

}

int[][] array = new int[a][a];

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

{

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

{

array[i][j] = 10000 ;

}

}

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

{

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

{

if(i == j)

array[i][j] = 0;

}

}

int temp = 0 ;

for(int i = 0 ; i < a-1 ;i ++)

{

int weight = rand.nextInt(9)+1;

int place = rand.nextInt(a-i-1);

array[place+i+1][i] = weight ;

array[i][place+i+1] = weight ;

}

int total = 0;

do{

int x = rand.nextInt(a);

int y = rand.nextInt(a);

int weight = rand.nextInt(9)+1;

if(x != y)

{

if(array[x][y] == INF)

{

array[x][y] = weight;

array[y][x] = array[x][y];

//System.out.println(x+" , "+y+" = "+array[x][y] );

total++;

}

else

continue;

}

else

{

array[x][y] = 0;

}

}while(total < b-a +1);

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

{

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

{

System.out.print(array[i][j]+" ");

}

System.out.println();

}

int[] array\_2 =new int[((a\*a)-a)/2];

temp = 0;

for(int i = 1 ;i < array.length ;i++)

{

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

{

array\_2[i\*(i-1)/2 + j] = array[i][j];

}

}

return array\_2 ;

}

**public** **static** **int**[] Create\_1d\_array() // for test case2

{

**int**[][] matrix= {{0, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 8 , 10000, 10000},

{10000, 0,17, 13 ,10000, 10000, 10000,17 ,12 , 10000, 10000,10000},

{10000,17 , 0, 10000, 10000, 10000, 10000, 10000, 10000, 7 ,10000, 9},

{10000,13 , 10000, 0, 10000, 10000,15, 10000, 10000,13 , 5 ,13},

{10000, 10000, 10000, 10000, 0,18 , 10000, 10000, 5 , 10000, 10000, 10000},

{10000, 10000, 10000, 10000,18 , 0, 10000,17, 10000, 10000, 10000, 10000},

{10000, 10000, 10000,15 , 10000, 10000, 0,17 , 10000, 10000, 8 ,11},

{10000,17 , 10000, 10000, 10000,17 ,17 , 0, 10000, 10000, 10000, 10000},

{10000,12 ,10000, 10000, 5 , 10000, 10000, 10000, 0, 10000,15 ,10000},

{8 , 10000, 7 ,13 ,10000, 10000, 10000, 10000, 10000, 0, 10000,14},

{10000, 10000, 10000, 5 , 10000, 10000, 8 , 10000,15, 10000, 0, 10000},

{10000, 10000, 9 ,13, 10000, 10000,11, 10000, 10000,14 , 10000, 0}

};

**int**[] array = **new** **int**[66];

**for**(**int** i = 1 ;i < 12 ;i++)

{

**for**(**int** j = 0; j<i ;j++)

{

array[i\*(i-1)/2 + j] = matrix[i][j];

}

}

**return** array ;

}

}

1. **Floyd’s algorithm**

**a)**

**the graph by an adjacency matrix using a two-dimensional array:**

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

public class FloydInGraph\_twoD{

private static int[][] dist ;

private static int[][] path ;

private static int INF=1000;

List< Integer> result=new ArrayList< Integer>();

public static void main(String args[])

{

int[][] matrix = Tools.Create\_2d\_halfsparse\_array(); // create 2dimension matrix

Scanner sc = new Scanner(System.in);

int size = matrix.length;

int begin ;

int end ;

long endtime = System.currentTimeMillis();

boolean choose = true ;

int start;

do{

System.out.println("do you want to find the specific path from one node to other node ?(true for yes ,false for no)");

choose = sc.nextBoolean();

if(choose == true)

{

System.out.println("please input the start node (from 0 to "+(matrix.length-1)+")");

start = sc.nextInt();

System.out.println("please input the end node (from 0 to "+(matrix.length-1)+")");

end = sc.nextInt();

FloydInGraph\_twoD graph=new FloydInGraph\_twoD(size);

int[][]D=floyd(matrix);

System.out.println("the weight is "+D[start][end]);

// graph.findCheapestPath(start,end,matrix);

List< Integer> list=graph.result;

System.out.println(start+" to "+end+",the shortest path is:");

System.out.print(start+"->");

pathway(start, end);

System.out.println(end);

}

else

{

System.out.println("EXIT!");

break;

}

}while(choose == true);

}

public static void pathway(int v,int u){ // print the path

if(path[v][u]!=0){

pathway(v,path[v][u]);

System.out.print(path[v][u]+"->");

pathway(path[v][u],u);

}

}

public static int[][] floyd(int[][] matrix){ // the key of the floyd algorithm

int size=matrix.length;

//initialize dist and path

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

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

path[i][j]=0;

dist[i][j]=matrix[i][j]; //a[]

}

}

for(int k=0;k< size;k++) // keep update the matrix as

{

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

{

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

{

if(dist[i][k]!=INF&&

dist[k][j]!=INF&&

dist[i][k]+dist[k][j]< dist[i][j])

{

dist[i][j]=dist[i][k]+dist[k][j];

path[i][j]=k;

}

}

}

}

return dist;

}

public FloydInGraph\_twoD(int size){

this.path=new int[size][size];

this.dist=new int[size][size];

}

}

**b)**

**the graph by an adjacency list using a linked list**

import java.io.IOException;

import java.util.Random;

import java.util.Scanner;

public class floyd\_linkedlist {

private static int INF = 10000;

private static char[] vexs ;

private static EData[] edges;

private static int[][]path;

// the top node in the linked table for linked list

private class ENode {

int ivex; // the position for this node ' edge

int weight; // weight for this edge

ENode nextEdge; // the indicator for next edge

}

// the top node for the linked list

private class VNode {

char data; // top node's info

ENode firstEdge; // the edge to the first node

};

private int mEdgNum; // the number of the edge

private VNode[] mVexs; // the set of node set

public floyd\_linkedlist(char[] vexs, EData[] edges) {

// initialize the node and the edge

int vlen = vexs.length;

int elen = edges.length;

// initialize the top node

mVexs = new VNode[vlen];

for (int i = 0; i < mVexs.length; i++) {

mVexs[i] = new VNode();

mVexs[i].data = vexs[i];

mVexs[i].firstEdge = null;

}

// initialize the top edge

mEdgNum = elen;

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

// read the start node and the end node of the edge

char c1 = edges[i].start;

char c2 = edges[i].end;

int weight = edges[i].weight;

int p1 = getPosition(c1);

int p2 = getPosition(c2);

// initialize the edge no.1

ENode node1 = new ENode();

node1.ivex = p2;

node1.weight = weight;

// make node1 link to the end of the p1 linked list

if(mVexs[p1].firstEdge == null)

mVexs[p1].firstEdge = node1;

else

linkLast(mVexs[p1].firstEdge, node1);

//initialize the node2

ENode node2 = new ENode();

node2.ivex = p1;

node2.weight = weight;

// make node2 linking to the end of the p2 linked list

if(mVexs[p2].firstEdge == null)

mVexs[p2].firstEdge = node2;

else

linkLast(mVexs[p2].firstEdge, node2);

}

}

//make node linking to the end of the list

private void linkLast(ENode list, ENode node) {

ENode p = list;

while(p.nextEdge!=null)

p = p.nextEdge;

p.nextEdge = node;

}

private int getPosition(char ch) {

for(int i=0; i<mVexs.length; i++)

if(mVexs[i].data==ch)

return i;

return -1;

}

public static void pathway(int v, int u) {

if (path[v][u] != 0) {

pathway(v, path[v][u]);

System.out.print(path[v][u] + "->");

pathway(path[v][u], u);

}

}

//get the weight of edge from start to end , if start and end is block , then return MAX

private int getWeight(int start, int end) {

if (start==end)

return 0;

ENode node = mVexs[start].firstEdge;

while (node!=null) {

if (end==node.ivex)

return node.weight;

node = node.nextEdge;

}

return INF;

}

//Floyd algorithm to account the shortest path of every nodes in the graph.

//path[i][j] =k means the shortest path from i to j will pass the node k

//dist[i][j] =sum means the shortest path' strength betwwen i and j is sum

public void floyd(int[][] path, int[][] dist,int start,int end) {

for (int i = 0; i < mVexs.length; i++) {

for (int j = 0; j < mVexs.length; j++) {

dist[i][j] = getWeight(i, j);

path[i][j] = 0;

}

}

// calculate the shortest distance between i and j

for (int k = 0; k < mVexs.length; k++) {

for (int i = 0; i < mVexs.length; i++) {

for (int j = 0; j < mVexs.length; j++) {

// if the path pass the node k is shorter than original nodes, then update the dist and the path

int tmp = (dist[i][k]==INF || dist[k][j]==INF) ? INF : (dist[i][k] + dist[k][j]);

if (dist[i][j] > tmp) {

dist[i][j] = tmp;

path[i][j] = k;

}

}

}

}

// print the answer of the shortest path

System.out.printf("floyd: \n");

System.out.println("thr shortest path from "+start+" to "+end +" is "+dist[start][end]);

}

// the structure of the edges

private static class EData {

char start;

char end;

int weight;

public EData(char start, char end, int weight) {

this.start = start;

this.end = end;

this.weight = weight;

}

};

public static void main(String[] args) {

// char[] vexs = {'A', 'B', 'C', 'D', 'E', 'F', 'G'};

// EData[] edges = {

// // start end weight

// new EData('A', 'B', 12),

// new EData('A', 'F', 16),

// new EData('A', 'G', 14),

// new EData('B', 'C', 10),

// new EData('B', 'F', 7),

// new EData('C', 'D', 3),

// new EData('C', 'E', 5),

// new EData('C', 'F', 6),

// new EData('D', 'E', 4),

// new EData('E', 'F', 2),

// new EData('E', 'G', 8),

// new EData('F', 'G', 9),

// };

input\_list();

floyd\_linkedlist pG;

pG = new floyd\_linkedlist(vexs, edges);

int[] prev = new int[pG.mVexs.length];

int[] dist = new int[pG.mVexs.length];

path = new int[pG.mVexs.length][pG.mVexs.length];

int[][] floy = new int[pG.mVexs.length][pG.mVexs.length];

int Start = 2 ;

int End = 8 ;

boolean choose =true;

Scanner sx = new Scanner(System.in);

do{

System.out.println("do u want to find the specific way from one node to another ?");

choose = sx.nextBoolean();

if(choose)

{

System.out.println("please input the start node : ");

Start = sx.nextInt();

System.out.println("please input the end node : ");

End = sx.nextInt();

pG.floyd(path, floy,Start,End);

System.out.print(Start+"->");

pathway(Start,End);

System.out.println(End);

}

else

{

System.out.println("Exit!");

break;

}

}while(choose);

}

public static void input\_list()

{

Random rand = new Random();

Scanner sc = new Scanner(System.in);

int temp = 0;

System.out.println("please input the number of nodes: ");

int node = sc.nextInt();

System.out.println("please input the number of edges: range from "+(node-1)+" to "+(node\*node/2-node));

int edge = sc.nextInt();

vexs = new char[node];

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

{

vexs[i] = (char) (i+48);

}

edges = new EData[edge];

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

{

int x = rand.nextInt(node);

int y = rand.nextInt(node);

if(x > y)

{

temp = x ;

x = y;

y = temp ;

}

char m = (char) (x+48);

char n = (char) (y+48);

int weight = rand.nextInt(19)+1;

edges[i] = new EData(m,n,weight);

}

}

}

**c) the graph by an adjacency matrix using a one-dimensional array for storing only the elements of the lower triangle in the adjacency matrix.**

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

public class Floyd\_array\_shortest {

private static int[] array;

private static int[] array1;

private static int[][] path;

private static int INF = 1000;

private List<Integer> result = new ArrayList<Integer>();

public static void main(String args[]) {

int a;

int b;

int start;

int end;

System.out.println("please input the number of the nodes");

Scanner sc = new Scanner(System.in);

a = sc.nextInt();

System.out.println("please input the number of the edges");

b = sc.nextInt();

array = Tools.Create\_1d\_array(a, b);

boolean choose = true;

do {

System.out.println(

"do you want to find the specific path from one node to other node ?(true for yes ,false for no)");

choose = sc.nextBoolean();

if (choose == true) {

System.out.println("please input the start node (from 0 to " + (a - 1) + ")");

start = sc.nextInt();

System.out.println("please input the end node (from 0 to " + (a - 1) + ")");

end = sc.nextInt();

floyd(a);

int temp2 = 0;

int v=start;

int u=end;

if(v < u)

{

temp2 = u ;

u=v;

v=temp2;

}

int temp = (v - 1) \* v / 2 + u;

System.out.println("the weight is " + array[temp]);

// graph.findCheapestPath(start,end,matrix);

System.out.println(start + " to " + end + ",the shortest path is:");

System.out.print(start + "->");

pathway(start, end);

System.out.println(end);

} else {

System.out.println("EXIT!");

break;

}

} while (choose == true);

}

public static void pathway(int v, int u) {

if (path[v][u] != 0) {

pathway(v, path[v][u]);

System.out.print(path[v][u] + "->");

pathway(path[v][u], u);

}

}

// if the start node bigger than end node , we need to exchange the value of start and end

public static int getWeight(int v, int u) {

if (v == u) {

return 0;

} else if (v < u) {

int temp = v;

v = u;

u = temp;

}

return array[(v - 1) \* v / 2 + u];

}

//set the exchange of start and end value

public static void setWeight(int v, int u, int value) {

if (v < u) {

int temp = v;

v = u;

u = temp;

}

array[(v - 1) \* v / 2 + u] = value;

}

public static void floyd(int vertex) {

path = new int[vertex][vertex];

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

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

path[i][j] = 0;

}

}

for (int k = 0; k < vertex; k++) { // following the order from the first to next node

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

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

if (getWeight(i, k) + getWeight(k, j) < getWeight(i, j)) {

setWeight(i, j, getWeight(i, k) + getWeight(k, j));

path[i][j] = k;

}

}

}

}

}

}

1. **Dijkstra’s algorithm**

**a) the graph by an adjacency matrix using a two-dimensional array**

import java.util.Arrays;

import java.util.HashMap;

import java.util.HashSet;

import java.util.Iterator;

import java.util.Map.Entry;

import java.util.Scanner;

import java.util.Set;

public class DJIGraph\_twoD {

private static final int minDis = 0;

private static final int maxDis = 10000;

// graph's adjacent matrix

int[][] matrix;

int startIndex;

// use for store the distance from start point to current point

HashMap<Integer, Integer> distanceMap = new HashMap<Integer, Integer>();

// use for store the shortest point set which already been found

Set<Integer> findedSet = new HashSet<Integer>();

public DJIGraph\_twoD(int[][] matrix, int start) {

this.matrix = matrix;

this.startIndex = start;

// findedSet.add(startIndex);

}

public void find() {

// use point which is next to start point to initialize the map

for (int i = 0; i < matrix.length; i++) {

// if (matrix[startIndex][i] != maxDis)

distanceMap.put(i, matrix[startIndex][i]);

}

while (findedSet.size() != matrix.length) {

int currentMinIndex = currentMinIndex();

// use this point to update the distance with other point

for (int i = 0; i < matrix.length; i++) {

if (!findedSet.contains(i) && matrix[currentMinIndex][i] != maxDis

&& matrix[currentMinIndex][i] + distanceMap.get(currentMinIndex) < distanceMap.get(i))

distanceMap.put(i, matrix[currentMinIndex][i] + distanceMap.get(currentMinIndex));

}

// put it into the finded set

findedSet.add(currentMinIndex);

}

}

// print the distance from the start point to other point .

public void printDistance() {

Iterator<Entry<Integer, Integer>> it = distanceMap.entrySet().iterator();

int min = 10000;

int minIndex = -1;

while (it.hasNext()) {

Entry<Integer, Integer> entry = it.next();

System.out.println(startIndex + "---->" + entry.getKey() + ":" + entry.getValue());

}

}

// return the shortest distance point which is not the finded set

private int currentMinIndex() {

Iterator<Entry<Integer, Integer>> it = distanceMap.entrySet().iterator();

int min = 10000;

int minIndex = -1;

while (it.hasNext()) {

Entry<Integer, Integer> entry = it.next();

if (!findedSet.contains(entry.getKey()) && entry.getValue() < min) {

min = entry.getValue();

minIndex = entry.getKey();

}

}

return minIndex;

}

public static void path(int[] P, int x) {

if (P[x] != 0) {

path(P, P[x]);

System.out.print(P[x] + "->");

}

}

public static int reslove(int[][] adjMat, int s, int t, boolean flag) {

// whether the parameter is right

if (s < 0 || t < 0 || s >= adjMat.length || t >= adjMat.length) {

System.out.println("Wrong! The node isn't in the graph");

return 10000;

}

// to note if the node already finished the scan

boolean[] isVisited = new boolean[adjMat.length];

// to store the shortest distance from s to every node

int[] d = new int[adjMat.length];

int[] P = new int[adjMat.length];

// initialize the data

for (int i = 0; i < adjMat.length; i++) {

isVisited[i] = false;

d[i] = adjMat[s][i];

P[i] = 0;

}

d[s] = 0; // obviously the distance from s to s is 0

isVisited[s] = true; // mark s as an node which already been visted

// Queue.isEmpty() the number of nodes which haven't been scanned

int unVisitedNum = adjMat.length;

// the index of the Min. node which stored in the subpath

int index = s;

System.out.println("Path : ");

while (unVisitedNum > 0) {

int min = 10000;

for (int i = 0; i < adjMat.length; i++) { // get the index of the

// Min element in i line

if (!isVisited[i] && min > d[i]) {

min = d[i];

index = i;

}

}

isVisited[index] = true;

for (int i = 0; i < adjMat.length; i++) {

if (!isVisited[i] && d[index] + adjMat[index][i] < d[i]) {

d[i] = d[index] + adjMat[index][i];

P[i] = index;

}

}

unVisitedNum--;

}

System.out.println(Arrays.toString(P));

System.out.println("the shortest path of "+s+" to "+t+" is "+d[t]);

System.out.println("the shortest path is : ");

if (flag) {

System.out.print(s + "->");

path(P, t);

System.out.println(t);

}

return d[t];

}

public static void main(String[] args) {

int[][] inputMatrix = Tools.Create\_2d\_halfsparse\_array();

long starttime = System.currentTimeMillis();

for (int i = 0; i < inputMatrix.length; i++) {

DJIGraph\_twoD path = new DJIGraph\_twoD(inputMatrix, i);

path.find();

path.printDistance();

}

long endtime = System.currentTimeMillis();

System.out.println("using time is "+(endtime - starttime)+" ms");

boolean choose = true;

int start;

int end;

Scanner sc = new Scanner(System.in);

System.out.println("please input the node which u want to find :");

start = sc.nextInt();

System.out.println("please input the node which u want to find as end :");

end = sc.nextInt();

reslove(inputMatrix, start, end, true);

}

}

**b) the graph by an adjacency list using a linked list**

import java.io.IOException;

import java.util.Random;

import java.util.Scanner;

public class Dji\_linkedlist {

private static int INF = 10000;

private static char[] vexs;

private static EData[] edges;

private static int[] P;

// the top node in the linked table for linked list

private class ENode {

int ivex; // the position for this node ' edge

int weight; // weight for this edge

ENode nextEdge; // the indicator for next edge

}

// the top node for the linked list

private class VNode {

char data; // top node's info

ENode firstEdge; // the edge to the first node

};

private int mEdgNum; // the number of the edge

private VNode[] mVexs; // the set of node set

public Dji\_linkedlist(char[] vexs, EData[] edges) {

// initialize the node and the edge

int vlen = vexs.length;

int elen = edges.length;

// initialize the top node

mVexs = new VNode[vlen];

for (int i = 0; i < mVexs.length; i++) {

mVexs[i] = new VNode();

mVexs[i].data = vexs[i];

mVexs[i].firstEdge = null;

}

// initialize the top edge

mEdgNum = elen;

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

// read the start node and the end node of the edge

char c1 = edges[i].start;

char c2 = edges[i].end;

int weight = edges[i].weight;

// initialize the edge no.1

int p1 = getPosition(c1);

int p2 = getPosition(c2);

ENode node1 = new ENode();

node1.ivex = p2;

node1.weight = weight;

// make node1 link to the end of the p1 linked list

if (mVexs[p1].firstEdge == null)

mVexs[p1].firstEdge = node1;

else

linkLast(mVexs[p1].firstEdge, node1);

//initialize the node2

ENode node2 = new ENode();

node2.ivex = p1;

node2.weight = weight;

// make node2 linking to the end of the p2 linked list

if (mVexs[p2].firstEdge == null)

mVexs[p2].firstEdge = node2;

else

linkLast(mVexs[p2].firstEdge, node2);

}

}

private void linkLast(ENode list, ENode node) {

ENode p = list;

while (p.nextEdge != null)

p = p.nextEdge;

p.nextEdge = node;

}

private int getPosition(char ch) {

for (int i = 0; i < mVexs.length; i++)

if (mVexs[i].data == ch)

return i;

return -1;

}

//get edge's weight from start to end ,if block, then return max

private int getWeight(int start, int end) {

if (start == end)

return 0;

ENode node = mVexs[start].firstEdge;

while (node != null) {

if (end == node.ivex)

return node.weight;

node = node.nextEdge;

}

return INF;

}

public static void path(int x) {

if (P[x] != 0) {

path(P[x]);

System.out.print(P[x] + "->");

}

}

//dijkstra's shortest path means the shortest path form node vs to another nodes

//prev[i]' value is the node before node i in all of the node from node vs to node i ' shortest path

//dist[i] means the length from node vs to node i

public void dijkstra(int vs, int[] dist, int end) {

// flag[i] = true means the shortest path from vs to i already success

boolean[] flag = new boolean[mVexs.length];

// initialize

for (int i = 0; i < mVexs.length; i++) {

flag[i] = false; // have not gotten the shortest path of the i node

P[i] = 0; //node i' former node is 0

dist[i] = getWeight(vs, i); //the shortest path for node i is the weight from node vs to node i

}

// initialize the node vs itself

flag[vs] = true;

dist[vs] = 0;

//scan (mVexs.length-1)times to find the shortest path for every node

int k = 0;

for (int i = 1; i < mVexs.length; i++) {

// find the current shortest path

//find the nearest node from vs from un-found the shortest path 'snode

int min = INF;

for (int j = 0; j < mVexs.length; j++) {

if (flag[j] == false && dist[j] < min) {

min = dist[j];

k = j;

}

}

// mark the node k as the shortest path which already got

flag[k] = true;

// revise current and the shortest path and the former node

// when get the shortest path which pass the node k , update the shortest path and the former node of the shortest path's node which have not been found

for (int j = 0; j < mVexs.length; j++) {

int tmp = getWeight(k, j);

tmp = (tmp == INF ? INF : (min + tmp)); // 防止溢出

if (flag[j] == false && (tmp < dist[j])) {

dist[j] = tmp;

P[j] = k;

}

}

}

// print the answer of the shortest path under dijkstra alogrighm

System.out.printf("dijkstra(%c): \n", mVexs[vs].data);

System.out.printf(" shortest path of(%c, %c) is %d\n", mVexs[vs].data, mVexs[end].data, dist[end]);

}

// structure of the edge

private static class EData {

char start;

char end;

int weight;

public EData(char start, char end, int weight) {

this.start = start;

this.end = end;

this.weight = weight;

}

};

public static void main(String[] args) {

// char[] vexs = {'A', 'B', 'C', 'D', 'E', 'F', 'G'};

// EData[] edges = {

// // start end weight

// new EData('A', 'B', 12),

// new EData('A', 'F', 16),

// new EData('A', 'G', 14),

// new EData('B', 'C', 10),

// new EData('B', 'F', 7),

// new EData('C', 'D', 3),

// new EData('C', 'E', 5),

// new EData('C', 'F', 6),

// new EData('D', 'E', 4),

// new EData('E', 'F', 2),

// new EData('E', 'G', 8),

// new EData('F', 'G', 9),

// };

input\_list();

Dji\_linkedlist pG;

pG = new Dji\_linkedlist(vexs, edges);

P= new int[pG.mVexs.length];

int[] dist = new int[pG.mVexs.length];

boolean choose = true;

int Start = 2;

int End = 8;

Scanner sx = new Scanner(System.in);

do {

System.out.println("do u want to find the specific way from one node to another ?");

choose = sx.nextBoolean();

if (choose) {

System.out.println("please input the start node : ");

Start = sx.nextInt();

System.out.println("please input the end node : ");

End = sx.nextInt();

pG.dijkstra(Start, dist, End);

System.out.print(Start + "->");

path(End);

System.out.println(End);

} else {

System.out.println("Exit!");

break;

}

} while (choose);

}

public static void input\_list() {

Random rand = new Random();

Scanner sc = new Scanner(System.in);

int temp = 0;

System.out.println("please input the number of nodes: ");

int node = sc.nextInt();

System.out.println(

"please input the number of edges: range from " + (node - 1) + " to " + (node \* node / 2 - node));

int edge = sc.nextInt();

vexs = new char[node];

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

vexs[i] = (char) (i + 48);

}

edges = new EData[edge];

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

int x = rand.nextInt(node);

int y = rand.nextInt(node);

if (x > y) {

temp = x;

x = y;

y = temp;

}

char m = (char) (x + 48);

char n = (char) (y + 48);

int weight = rand.nextInt(19) + 1;

edges[i] = new EData(m, n, weight);

}

}

}

**c) the graph by an adjacency matrix using a one-dimensional array for storing only the elements of the lower triangle in the adjacency matrix.**

import java.util.Arrays;

import java.util.HashMap;

import java.util.HashSet;

import java.util.Iterator;

import java.util.Scanner;

import java.util.Set;

import java.util.Map.Entry;

public class Dji\_array\_shortest {

private static final int minDis = 0;

private static final int maxDis = 10000;

// graph's adjacent matrix

private static int[] matrix;

int startIndex;

// use for store the distance from start point to current point

HashMap<Integer, Integer> distanceMap = new HashMap<Integer, Integer>();

// use for store the shortest point set which already been found

Set<Integer> findedSet = new HashSet<Integer>();

public Dji\_array\_shortest(int[] matrix, int start) {

this.matrix = matrix;

this.startIndex = start;

// findedSet.add(startIndex);

}

public void find(int u) {

// use point which is next to start point to initialize the map

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

// if (matrix[startIndex][i] != maxDis)

distanceMap.put(i, getWeight(startIndex,i));

}

while (findedSet.size() != u) {

int currentMinIndex = currentMinIndex();

// use this point to update the distance with other point

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

if (!findedSet.contains(i) && getWeight(currentMinIndex,i) != maxDis

&& getWeight(currentMinIndex,i) + distanceMap.get(currentMinIndex) < distanceMap.get(i))

distanceMap.put(i, getWeight(currentMinIndex,i) + distanceMap.get(currentMinIndex));

}

// put it into the finded set

findedSet.add(currentMinIndex);

}

}

// print the distance from the start point to other point .

public void printDistance() {

Iterator<Entry<Integer, Integer>> it = distanceMap.entrySet().iterator();

int min = 10000;

int minIndex = -1;

while (it.hasNext()) {

Entry<Integer, Integer> entry = it.next();

System.out.println(startIndex + "---->" + entry.getKey() + ":" + entry.getValue());

}

}

// return the shortest distance point which is not the finded set

private int currentMinIndex() {

Iterator<Entry<Integer, Integer>> it = distanceMap.entrySet().iterator();

int min = 10000;

int minIndex = -1;

while (it.hasNext()) {

Entry<Integer, Integer> entry = it.next();

if (!findedSet.contains(entry.getKey()) && entry.getValue() < min) {

min = entry.getValue();

minIndex = entry.getKey();

}

}

return minIndex;

}

public static void path(int[] P, int x) {

if (P[x] != 0) {

path(P, P[x]);

System.out.print(P[x] + "->");

}

}

public static int reslove(int[] matrix, int s, int t, boolean flag,int u) {

// whether the parameter is right

if (s < 0 || t < 0 || s >= u || t >= u) {

System.out.println("Wrong! The node isn't in the graph");

return 10000;

}

// to note if the node already finished the scan

boolean[] isVisited = new boolean[u];

// to store the shortest distance from s to every node

int[] d = new int[u];

int[] P = new int[u];

// initialize the data

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

isVisited[i] = false;

d[i] = setWeight(s,i);

P[i] = 0;

}

d[s] = 0; // obviously the distance from s to s is 0

isVisited[s] = true; // mark s as an node which already been visted

// Queue.isEmpty() the number of nodes which haven't been scanned

int unVisitedNum = u;

// the index of the Min. node which stored in the subpath

int index = s;

System.out.println("Path : ");

while (unVisitedNum > 0) {

int min = 10000;

for (int i = 0; i < u; i++) { // get the index of the

// Min element in i line

if (!isVisited[i] && min > d[i]) {

min = d[i];

index = i;

}

}

isVisited[index] = true;

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

if (!isVisited[i] && d[index] + getWeight(index,i) < d[i]) {

d[i] = d[index] + setWeight(index,i);

P[i] = index;

}

}

unVisitedNum--;

}

System.out.println(Arrays.toString(P));

System.out.println("the shortest path of "+s+" to "+t+" is "+d[t]);

System.out.println("the shortest path is : ");

if (flag) {

System.out.print(s + "->");

path(P, t);

System.out.println(t);

}

return d[t];

}

public static void main(String[] args) {

int a = 0;

int b = 0;

Scanner sc = new Scanner(System.in);

System.out.println("Please input the number of the nodes : ");

a = sc.nextInt();

System.out.println("Please input the number of the edges : ");

b = sc.nextInt();

matrix = Tools.Create\_1d\_array(a, b);

long starttime = System.currentTimeMillis();

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

Dji\_array\_shortest path = new Dji\_array\_shortest(matrix, i);

path.find(a);

path.printDistance();

}

long endtime = System.currentTimeMillis();

System.out.println("using time is "+(endtime - starttime)+" ms");

boolean choose = true;

int start;

int end;

do{

System.out.println(

"do you want to find the specific path from one node to other node ?(true for yes ,false for no)");

choose = sc.nextBoolean();

if(choose = true)

{

System.out.println("please input the node which u want to find :");

start = sc.nextInt();

System.out.println("please input the node which u want to find as end :");

end = sc.nextInt();

reslove(matrix, start, end, true,a);

}

else

{

System.out.println("Exit!");

break;

}

}while(choose=true);

}

public static int getWeight(int v, int u) {

if (v == u) {

return 0;

} else if (v < u) {

int temp = v;

v = u;

u = temp;

}

return matrix[(v - 1) \* v / 2 + u];

}

public static int setWeight(int v, int u) {

if (v < u) {

int temp = v;

v = u;

u = temp;

}

int value = matrix[(v - 1) \* v / 2 + u];

return value;

}

}

**the storage differences**

for complete graph,

the linked list could storage the most amount of data ,

the two dimensional matrix could storage less,

and the one dimensional matrix could storage the least amount of the data causing it is the weaken style of the two dimensional matrix.

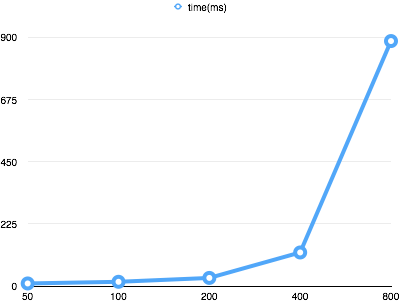
And for sparse graph ,

The situation is same as the complete graph, but linked list has the most efficient way to store the data .

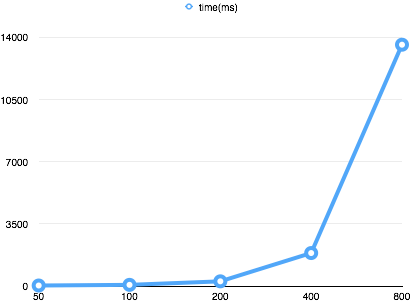
**E. performance differences by different data structures**

For two dimensional matrix: (sparse graph)

The Floyd algorithm:

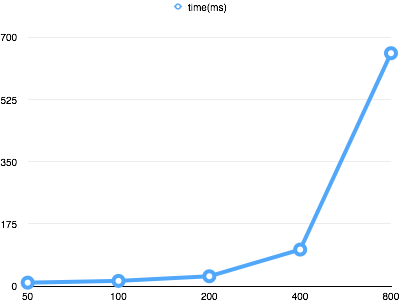


Dijkstra’s algorithm

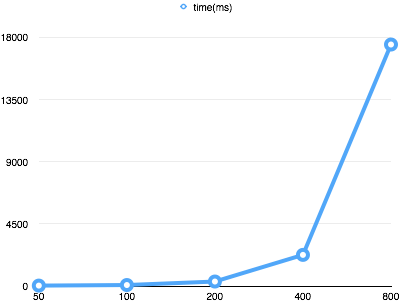
****

For two dimensional matrix: (complete graph)

Floyd algorithm

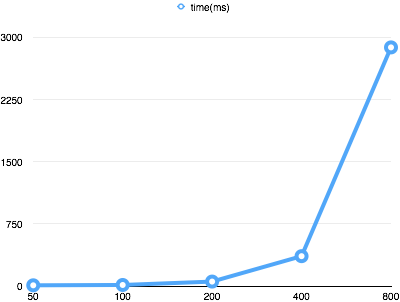


Dijkstra’s algorithm

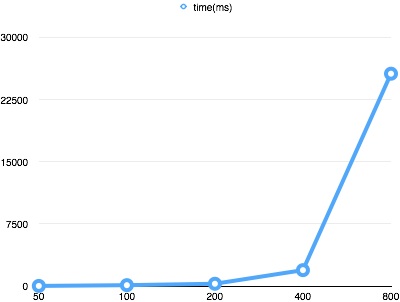


For one dimensional matrix: (sparse graph)

Floyd algorithm

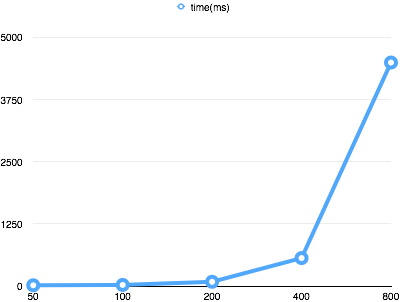


Dijkstra’s algorithm

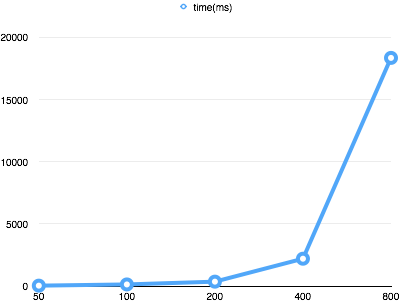


For one dimensional matrix: (complete graph)

Floyd algorithm

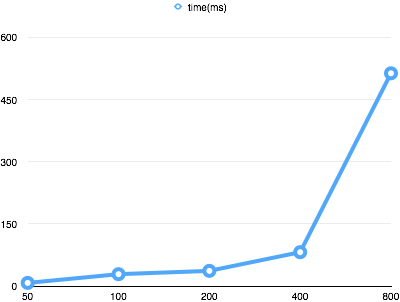


Dijkstra’s algorithm

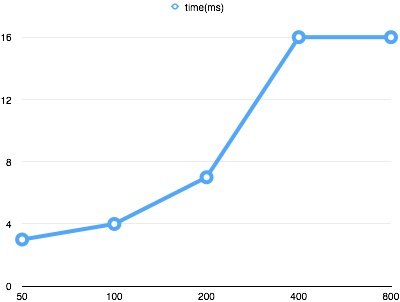


For linked list: (sparse graph)

The floyd algorithm:

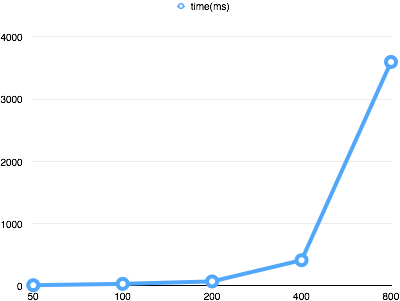


Dijkstra’s algorithm

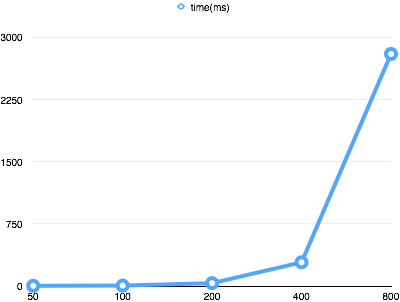


For linked list: (complete graph)

Floyd algorithm



Dijkstra’s algorithm



F. Test Case No.1 (I use the ‘0’ as the first node to replace the ‘1’ as the first node which is showed in the case )

Take an example :

Path 1:

From vertex 8 to vertex 6: Path: v8 -> v2 -> v3 -> v5 -> v10 -> v6 Weight: 54

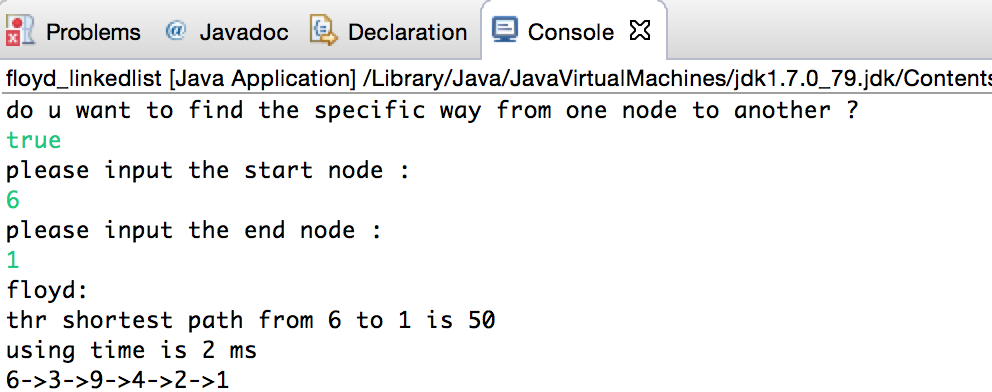
Equals with

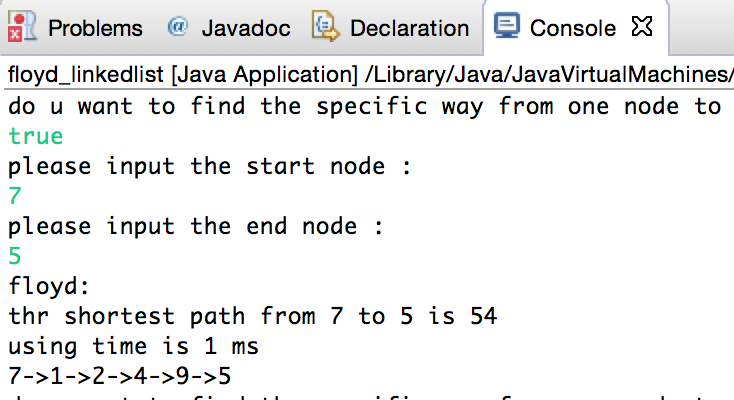
From vertex 7 to vertex5:

Path:v7->v1->v2->v4->v9->v5

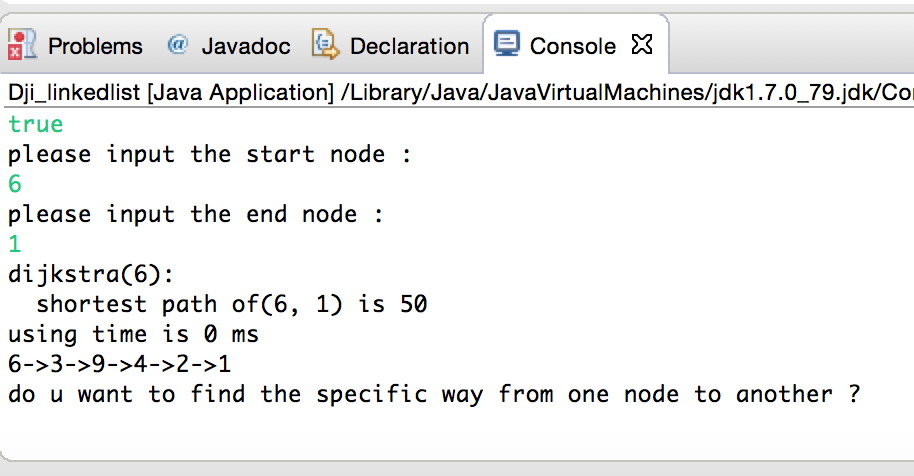
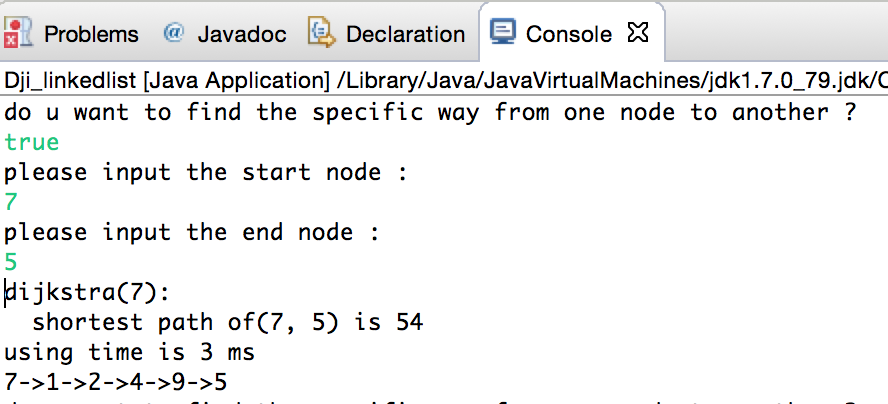
Weight:54

Floyd algorithm linked list

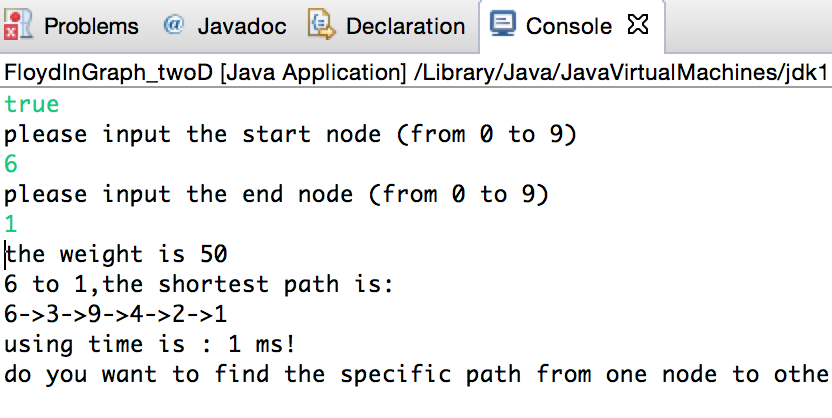
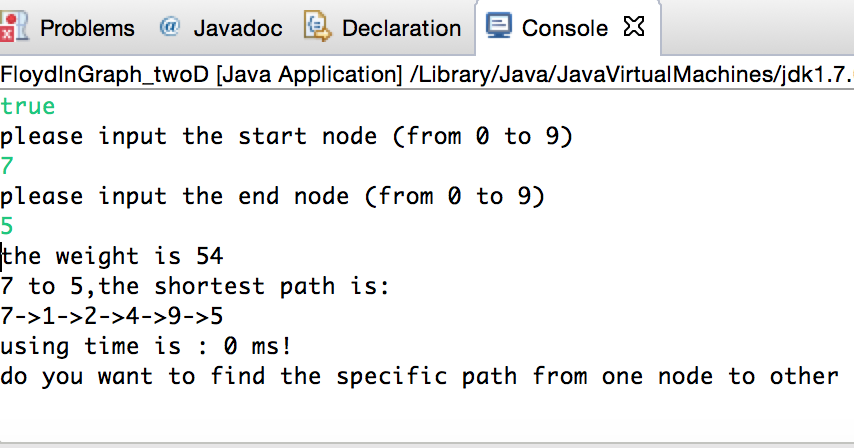




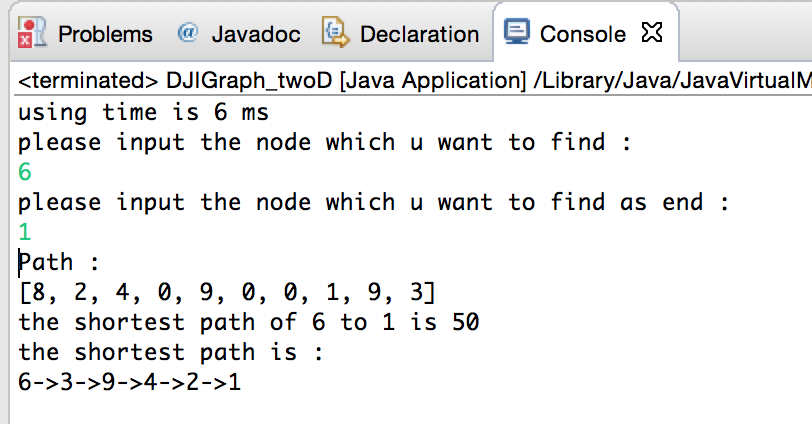
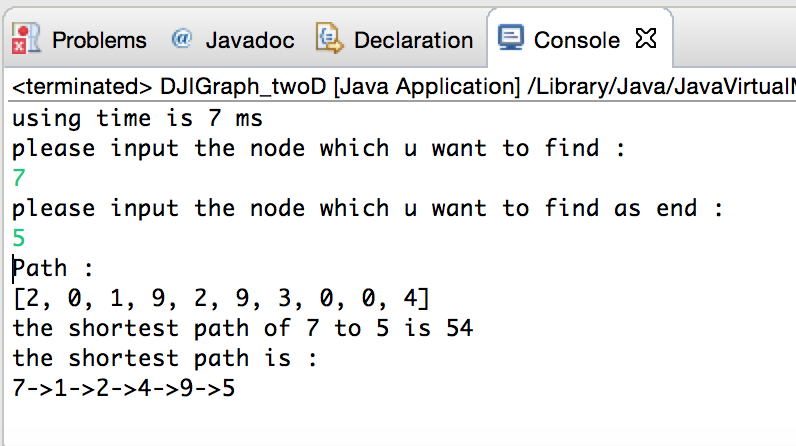
Dijkstra’s algorithm linked list



Floyd algorithm two dimensional array

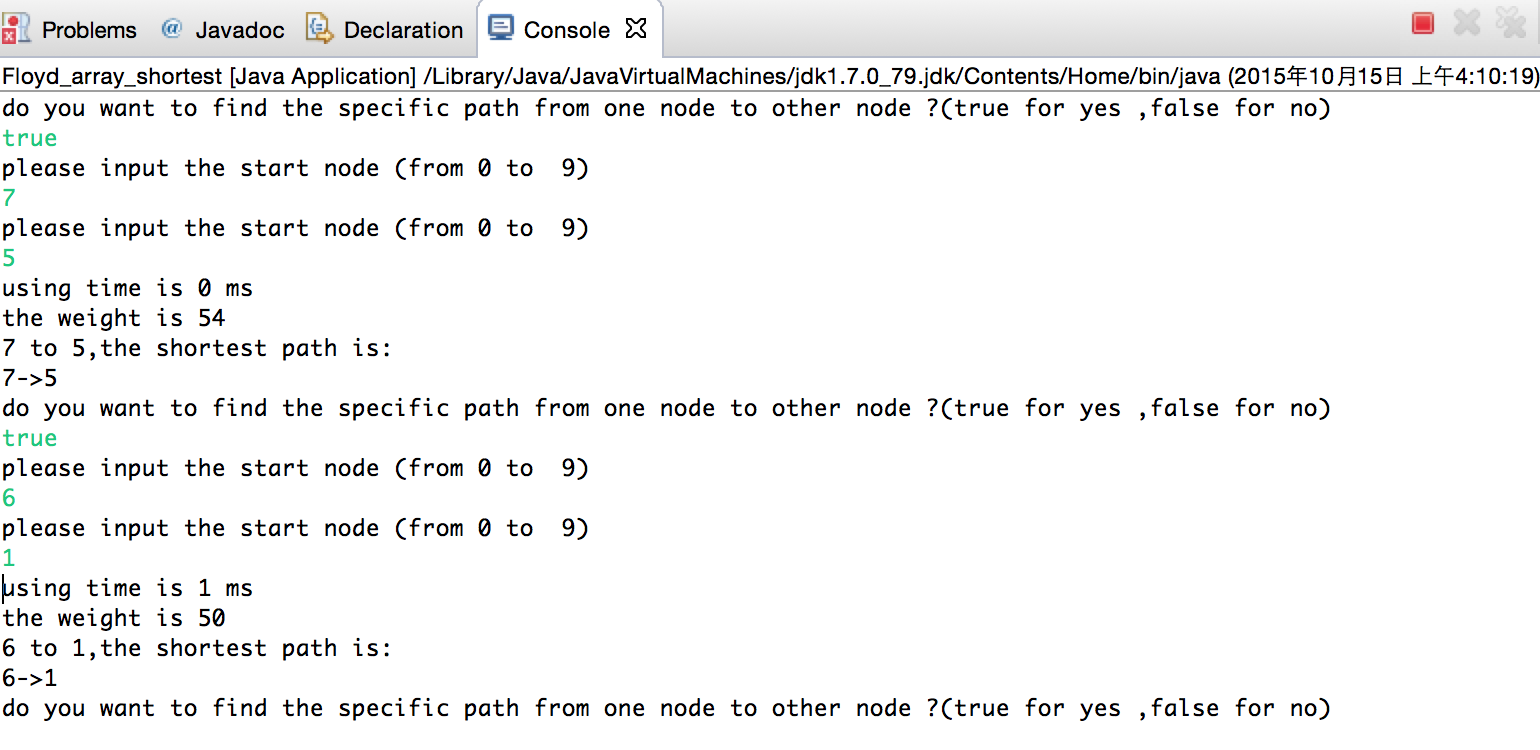


Dijkstra’s algorithm two dimensional array



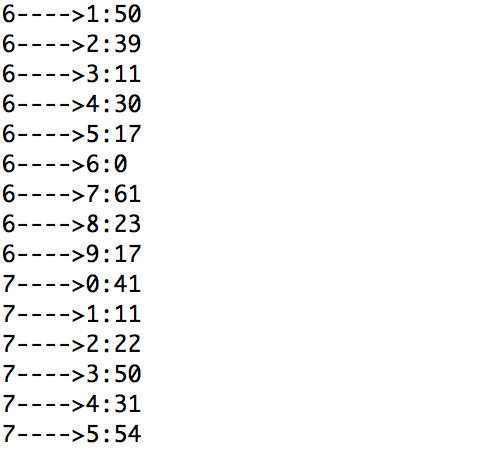
Floyd algorithm one dimensional array

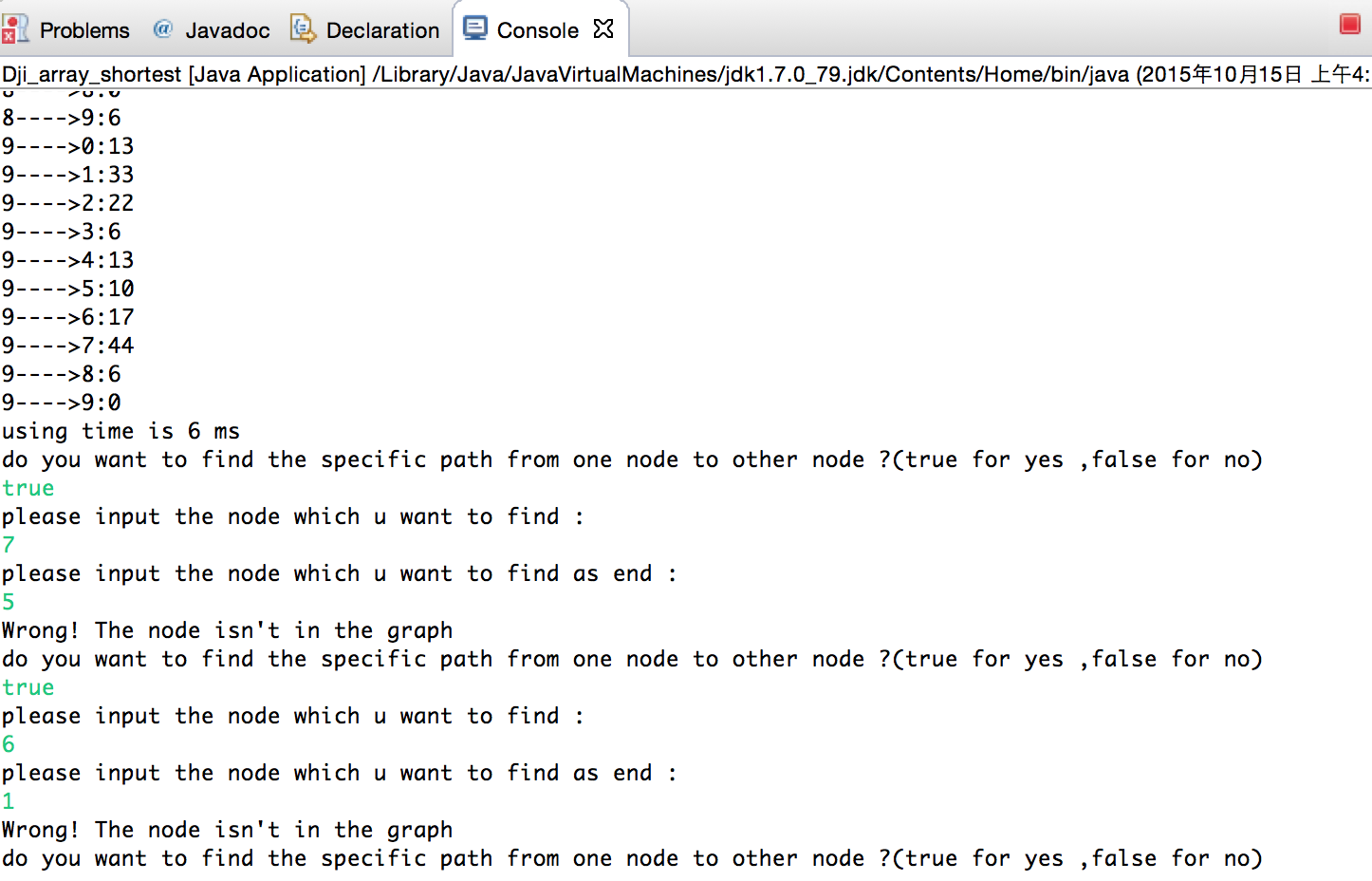
(there is some problem existing ,when I give the program the specific array , the path of the answer can’t output correctly , but this function could be used when array made by program itself )

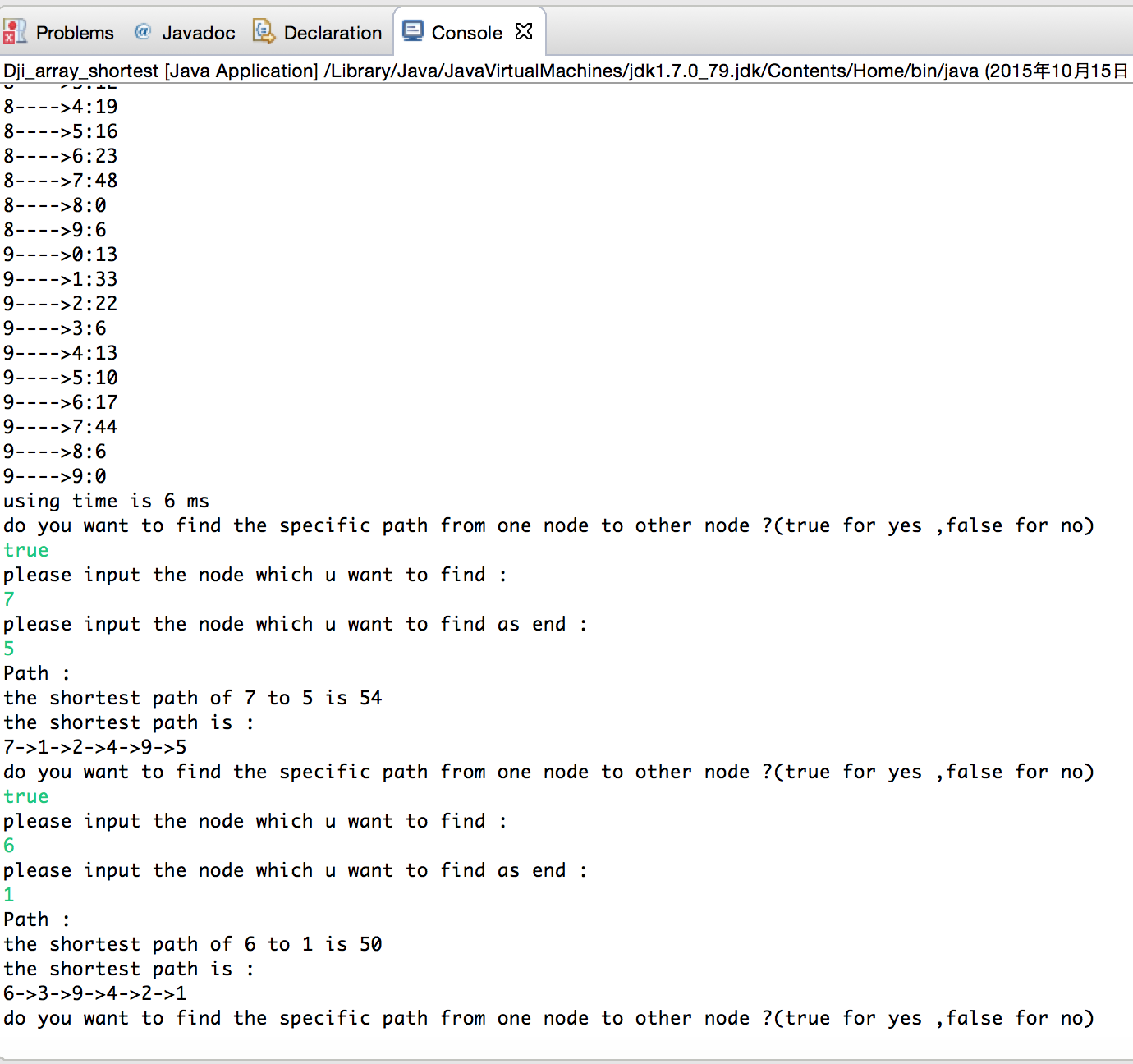


Dijkstra’s algorithm one dimensional array

There still have some problems existing in this part , when I give the program a specific array , it can’t show as normal random array which make my result wrong, but actually , the path and total weight already have been found .

The result of the right result show as below :



but fortunately , I found the reason and fixed this question finally .LOL

Case2 :

The matrix show as below:(I use 10000 to show block between two nodes)

10000 10000 10000 10000 10000 10000 10000 10000 10000 8 10000 10000

10000 10000 17 13 10000 10000 10000 17 12 10000 10000 10000

10000 17 10000 10000 10000 10000 10000 10000 10000 7 10000 9

10000 13 10000 10000 10000 10000 15 10000 10000 13 5 13

10000 10000 10000 10000 10000 18 10000 10000 5 10000 10000 10000

10000 10000 10000 10000 18 10000 10000 17 10000 10000 10000 10000

10000 10000 10000 15 10000 10000 10000 17 10000 10000 8 11

10000 17 10000 10000 10000 17 17 10000 10000 10000 10000 10000

10000 12 10000 10000 5 10000 10000 10000 10000 10000 15 10000

8 10000 7 13 10000 10000 10000 10000 10000 10000 10000 14

10000 10000 10000 5 10000 10000 8 10000 15 10000 10000 10000

10000 10000 9 13 10000 10000 11 10000 10000 14 10000 10000

so I will build a array to store this matrix ,

**int**[][] matrix= {{10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 8 , 10000, 10000},

{10000, 10000,17, 13 ,10000, 10000, 10000,17 ,12 , 10000, 10000,10000},

{10000,17 , 10000, 10000, 10000, 10000, 10000, 10000, 10000, 7 ,10000, 9},

{10000,13 , 10000, 10000, 10000, 10000,15, 10000, 10000,13 , 5 ,13},

{10000, 10000, 10000, 10000, 10000,18 , 10000, 10000, 5 , 10000, 10000, 10000},

{10000, 10000, 10000, 10000,18 , 10000, 10000,17, 10000, 10000, 10000, 10000},

{10000, 10000, 10000,15 , 10000, 10000, 10000,17 , 10000, 10000, 8 ,11},

{10000,17 , 10000, 10000, 10000,17 ,17 , 10000, 10000, 10000, 10000, 10000},

{10000,12 ,10000, 10000, 5 , 10000, 10000, 10000, 10000, 10000,15 ,10000},

{8 , 10000, 7 ,13 ,10000, 10000, 10000, 10000, 10000, 10000, 10000,14},

{10000, 10000, 10000, 5 , 10000, 10000, 8 , 10000,15, 10000, 10000, 10000},

{10000, 10000, 9 ,13, 10000, 10000,11, 10000, 10000,14 , 10000, 10000}

};

Floyd algorithm linked list

(this time , I use character ‘A’ ‘B’’C’’D’ … to represent the node 1,2,3,4,5)

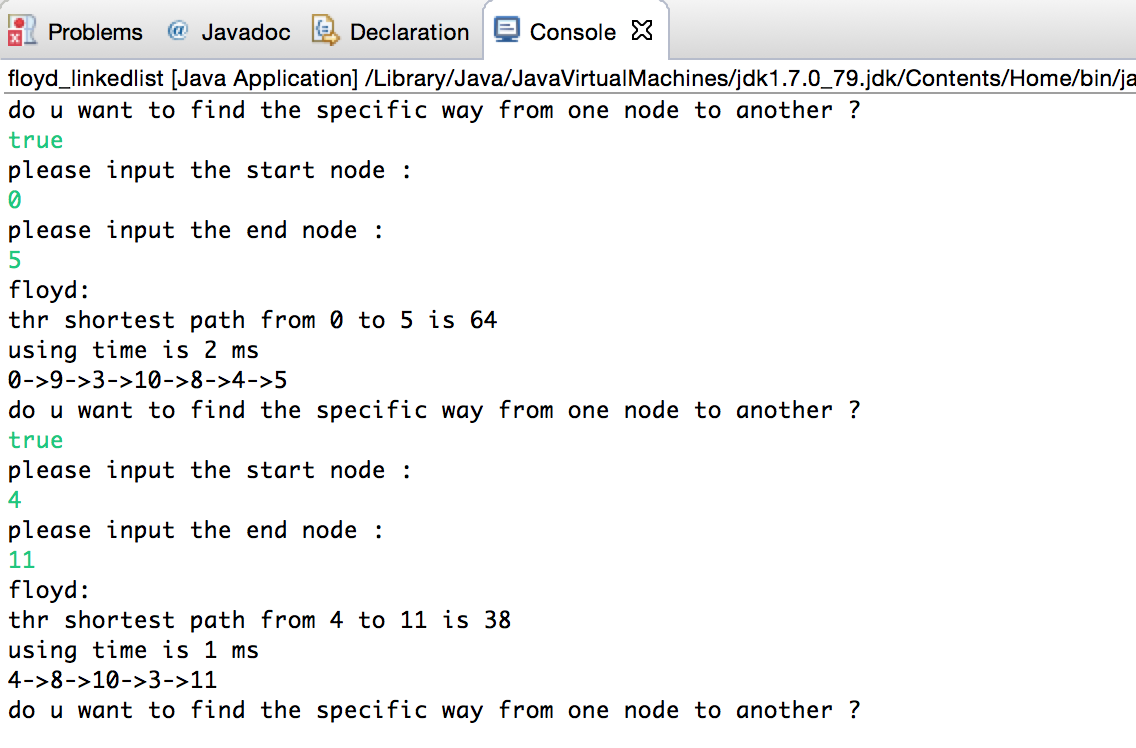
And same as the case one , all of the serial number of node will minus 1 , which means in case

From vertex 1 to vertex 6: Path: v1 -> v10 -> v4 -> v11 -> v9 -> v5 -> v6 Weight: 64

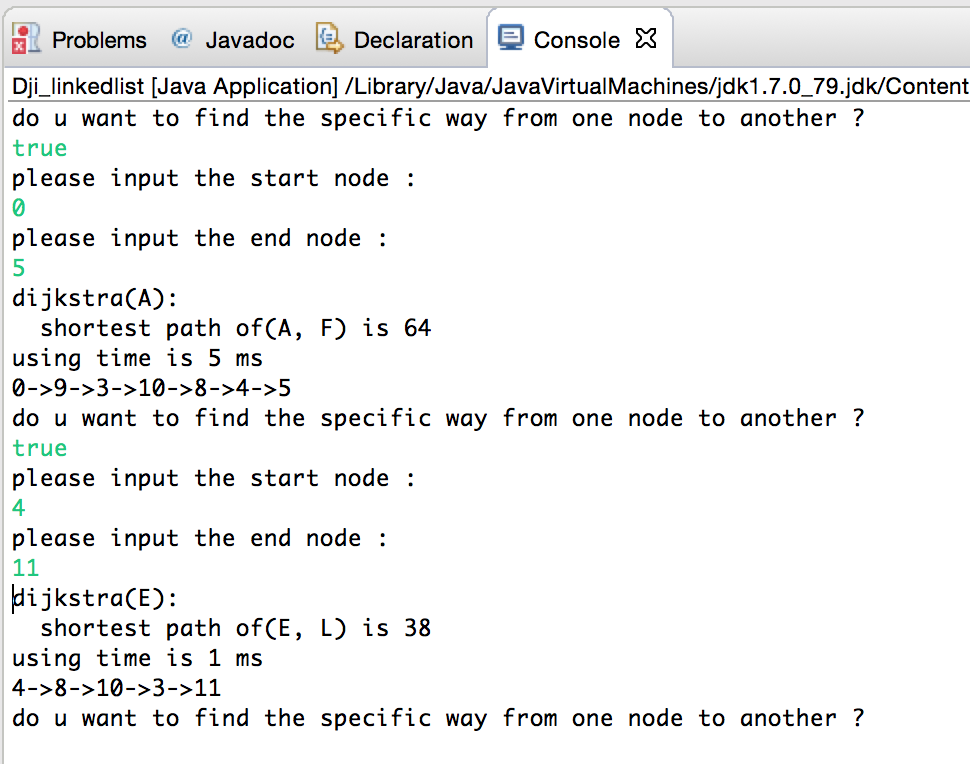
Equals

From vertex 0 to vertex 5 :

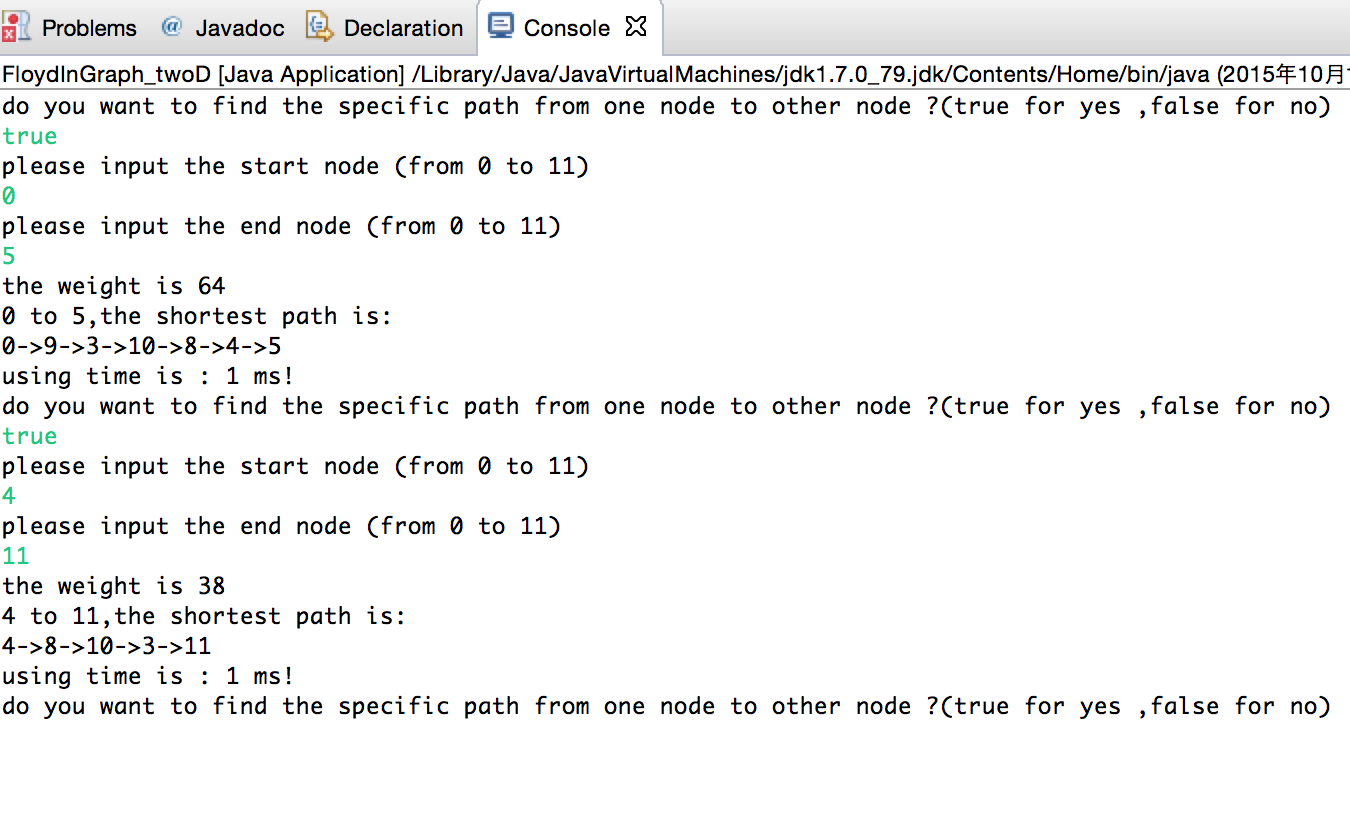
Path : v0->v9->v3->v10->v8->v4->v5

Weight :64

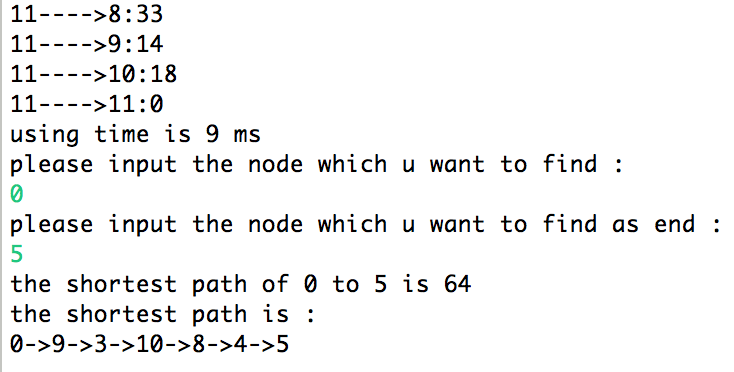
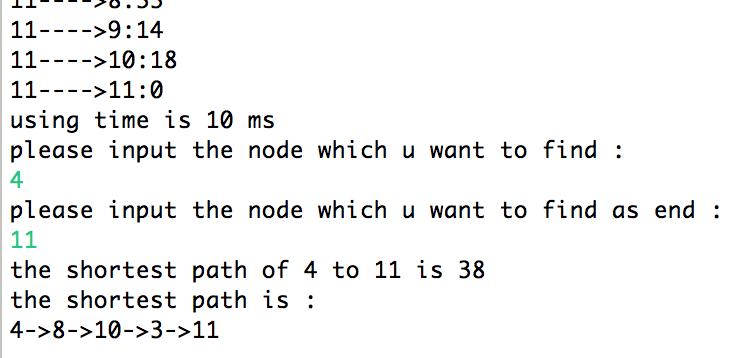
Dijkstra’s algorithm in linked list



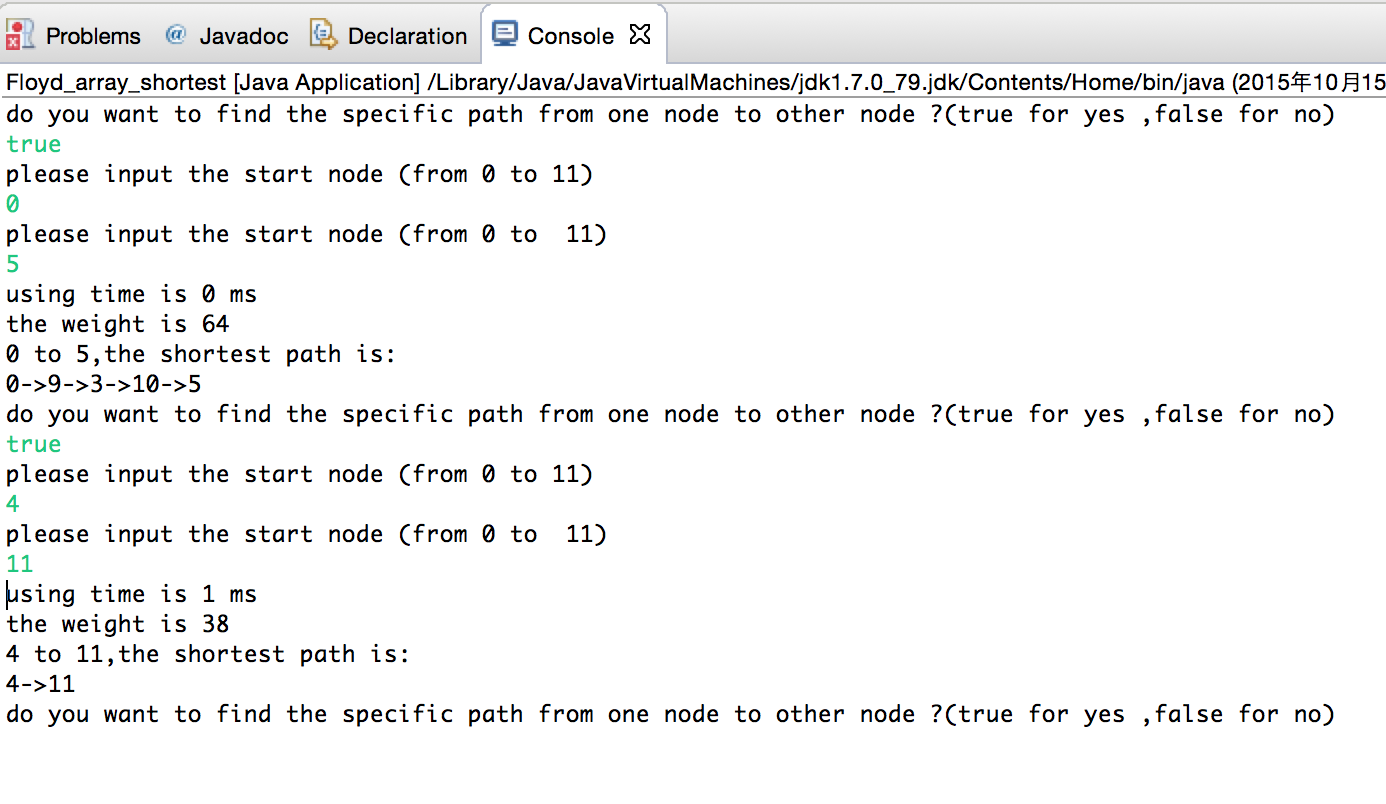
Floyd algorithm in two dimensional array



Dijkstra’s algorithm in linked list



Floyd algorithm in one dimensional array



Dijkstra’s algorithm in one dimensional array

