/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Online Java Compiler.

Code, Compile, Run and Debug java program online.

Write your code in this editor and press "Run" button to execute it.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

// A class to store a graph edge

class Edge

{

int src, dest, weight, paisa;

Edge(int src, int dest, int weight, int paisa)

{

this.src = src;

this.dest = dest;

this.weight = weight;

this.paisa = paisa;

}

}

// A class to store adjacency list nodes

class Node

{

int value, weight, paisa;

Node(int value, int weight, int paisa)

{

this.value = value;

this.weight = weight;

this.paisa = paisa;

}

@Override

public String toString() {

return this.value + " (" + this.weight + " " + this.paisa + ")";

}

}

// A class to represent a graph object

class Graph

{

static String[] person = {"Gautam", "Kumar", "Mahato", "John", "Maxx", "David"};

// A list of lists to represent an adjacency list

List<List<Node>> adjList = new ArrayList<>();

// Constructor to construct a graph

public Graph(List<Edge> edges)

{

// find the maximum numbered vertex

int n = 0;

for (Edge e: edges) {

n = Integer.max(n, Integer.max(e.src, e.dest));

}

// allocate memory for the adjacency list

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

adjList.add(i, new ArrayList<>());

}

// add edges to the directed graph

for (Edge e: edges)

{

// allocate new node in adjacency list from src to dest

adjList.get(e.src).add(new Node(e.dest, e.weight, e.paisa));

// uncomment below line for undirected graph

// allocate new node in adjacency list from dest to src

// adjList.get(e.dest).add(new Node(e.src, e.weight));

}

}

// Function to print adjacency list representation of a graph

public static void printGraph(Graph graph)

{

int src = 0;

int n = graph.adjList.size();

while (src < n)

{

// print current vertex and all its neighboring vertices

for (Node edge: graph.adjList.get(src)) {

System.out.print("value: "+ edge.value + " ");

System.out.print("weight: "+ edge.weight +" ");

System.out.print("paisa: "+ edge.paisa +" ");

System.out.print("name: "+ person[src] +" ");

//System.out.println("edge"+ edge.weight);

}

System.out.println();

src++;

}

System.out.println(graph.adjList.get(2));

System.out.println(graph.adjList.get(2).get(0).weight + " " + graph.adjList.get(2).get(0).paisa);

}

// add weight to a particular node

public static void addWeight(Graph graph, int src, int dest, int money)

{

graph.adjList.get(src).get(dest).weight = graph.adjList.get(src).get(dest).weight + money;

System.out.print(person[src] + " to add money " + person[dest] + ": ");

System.out.println(graph.adjList.get(src).get(dest).weight);

setPaisaValue(graph.adjList.get(dest), src, money);

}

// add equal weight to all nodes

public static void addEqualWeight(Graph graph, int src, int money)

{

System.out.println(graph.adjList.get(src));

for (Node node: graph.adjList.get(src)) {

node.weight = node.weight + money/5;

int dest = node.value;

// System.out.println("Data " + graph.adjList.get(dest));

// System.out.println("dest " + graph.adjList.get(dest).get(src));

setPaisaValue(graph.adjList.get(dest), src, money/5);

}

// System.out.println(src+"--> "+graph.adjList.get(src));

// System.out.println("0--> "+graph.adjList.get(0));

// System.out.println("2--> "+graph.adjList.get(2));

// System.out.println("3--> "+graph.adjList.get(3));

// System.out.println("4--> "+graph.adjList.get(4));

}

//set paisa

public static void setPaisaValue(List<Node> list, int src, int moneyLend){

for(Node node: list){

if(node.value == src){

node.paisa = node.paisa + moneyLend;

}

}

}

//crete List of Edges in the Graph accoriding to the number of users

public static List<Edge> creteEdges(int n){

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

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

if(i != j){

edgeList.add(new Edge(i,j,0,0));

}

}

}

return edgeList;

}

// find the total money

public static void findTotal(Graph graph)

{

int src = 0;

int n = graph.adjList.size();

int totalWeight = 0;

int totalPaisa = 0;

while (src < n)

{

totalWeight = 0;

totalPaisa = 0;

for (Node edge: graph.adjList.get(src)) {

totalWeight = totalWeight + edge.weight;

totalPaisa = totalPaisa + edge.paisa;

}

System.out.println("totalWeight " + totalWeight + "," + "totalPaisa "+ totalPaisa);

src++;

}

}

public static void display(Graph graph){

System.out.println("0--> "+graph.adjList.get(0));

System.out.println("1--> "+graph.adjList.get(1));

System.out.println("2--> "+graph.adjList.get(2));

System.out.println("3--> "+graph.adjList.get(3));

System.out.println("4--> "+graph.adjList.get(4));

}

}

public class Main

{

public static void main (String[] args)

{

// Input: List of edges in a weighted digraph (as per the above diagram)

// tuple `(x, y, w)` represents an edge from `x` to `y` having weight `w`

// List<Edge> edges = Arrays.asList(

// new Edge(0, 1, 6), new Edge(1, 2, 7), new Edge(2, 0, 5),

// new Edge(2, 1, 4), new Edge(3, 2, 10), new Edge(4, 5, 1),

// new Edge(5, 4, 3));

List<Edge> edges = Graph.creteEdges(5);

// construct a graph from the given list of edges

Graph graph = new Graph(edges);

// print adjacency list representation of the graph

Graph.printGraph(graph);

Graph.addWeight(graph, 2, 0, 15);

// add equal weight

Graph.addEqualWeight(graph, 1, 100);

Graph.addWeight(graph, 2, 0, 150);

Graph.addEqualWeight(graph, 1, 100);

Graph.addWeight(graph, 3, 1, 55);

Graph.addEqualWeight(graph, 4, 100);

Graph.display(graph);

Graph.findTotal(graph);

}

}