import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

// data structure to store graph edges

class Edge

{

// edge from source to dest having weight equal to weight

int source, dest, weight;

Edge(int source, int dest, int weight) {

this.source = source;

this.dest = dest;

this.weight = weight;

}

};

class Main {

// N is number of vertices in the graph

private static final int N = 4;

// define infinity as maximum value of the integer

private static final int INF = Integer.MAX\_VALUE;

// Function to run Bellman-Ford algorithm from given source

public static boolean BellmanFord(List<Edge> edges, int source)

{

// count number of edges present in the graph

int E = edges.size();

// cost[] stores shortest-path information

int[] cost = new int[N];

// initialize cost[]. Initially all vertices except

// source vertex have a weight of infinity

Arrays.fill(cost, INF);

cost[source] = 0;

int u, v, w, k = N;

// Relaxation step (run V-1 times)

while (--k > 0)

{

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

{

// edge from u to v having weight w

u = edges.get(j).source;

v = edges.get(j).dest;

w = edges.get(j).weight;

// if the cost to the destination u can be

// shortened by taking the edge u -> v

if (cost[u] != INF && cost[u] + w < cost[v])

{

// update cost to the new lower value

cost[v] = cost[u] + w;

}

}

}

// Run relaxation step once more for N'th time to

// check for negative-weight cycles

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

{

// edge from u to v having weight w

u = edges.get(i).source;

v = edges.get(i).dest;

w = edges.get(i).weight;

// if the cost to the destination u can be

// shortened by taking the edge u -> v

if (cost[u] != INF && cost[u] + w < cost[v]) {

return true;

}

}

return false;

}

public static void main(String[] args)

{

// given adjacency representation of matrix

int[][] adjMatrix =

{

{ 0, INF, -2, INF },

{ 4, 0, -3, INF },

{ INF, INF, 0, 2 },

{ INF, -1, INF, 0 }

};

// create a List to store graph edges

List<Edge> edges = new ArrayList<>();

for (int v = 0; v < N; v++) {

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

if (adjMatrix[v][u] != 0 && adjMatrix[v][u] != INF) {

edges.add(new Edge(v, u, adjMatrix[v][u]));

}

}

}

// run Bellman-Ford algorithm from each vertex as source

// and check for any Negative Weight Cycle

int i;

for (i = 0; i < N; i++) {

if (BellmanFord(edges, i)) {

System.out.print("Negative Weight Cycle Found!!");

break;

}

}

if (i == N) {

System.out.print("No Negative Weight Cycle Found");

}

}

}