Q-1: Given a directed graph represented by an adjacency matrix, find the transitive closure of the graph using Warshall's algorithm.

Sample test case:

|  |
| --- |
| Input:  1 1 0 1  0 1 1 0  0 0 1 1  0 0 0 1  Output:  Transitive Closure:  1 1 1 1  0 1 1 1  0 0 1 1  0 0 0 1 |

Solution:

#include<stdio.h>

// Number of vertices in the graph

#define V 4

// A function to print the solution matrix

void printSolution(int reach[][V]);

// Prints transitive closure of graph[][]

// using Floyd Warshall algorithm

void transitiveClosure(int graph[][V])

{

/\* reach[][] will be the output matrix

// that will finally have the

shortest distances between

every pair of vertices \*/

int reach[V][V], i, j, k;

/\* Initialize the solution matrix same

as input graph matrix. Or

we can say the initial values of

shortest distances are based

on shortest paths considering

no intermediate vertex. \*/

for (i = 0; i < V; i++)

for (j = 0; j < V; j++)

reach[i][j] = graph[i][j];

/\* Add all vertices one by one to the

set of intermediate vertices.

---> Before start of a iteration,

we have reachability values for

all pairs of vertices such that

the reachability values

consider only the vertices in

set {0, 1, 2, .. k-1} as

intermediate vertices.

----> After the end of a iteration,

vertex no. k is added to the

set of intermediate vertices

and the set becomes {0, 1, .. k} \*/

for (k = 0; k < V; k++)

{

// Pick all vertices as source one by one

for (i = 0; i < V; i++)

{

// Pick all vertices as destination for the above picked source

for (j = 0; j < V; j++)

{

// If vertex k is on a path from i to j,

// then make sure that the value of reach[i][j] is 1

reach[i][j] = reach[i][j] ||

(reach[i][k] && reach[k][j]);

}

}

}

// Print the shortest distance matrix

printSolution(reach);

}

/\* A utility function to print solution \*/

void printSolution(int reach[][V])

{

printf ("Transitive Closure: \n");

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

{

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

{

/\* because "i==j means same vertex"

and we can reach same vertex

from same vertex. So, we print 1....

and we have not considered this in

Floyd Warshall Algo. so we need to

make this true by ourself

while printing transitive closure.\*/

if(i == j)

printf("1 ");

else

printf ("%d ", reach[i][j]);

}

printf("\n");

}

}

int main()

{

int graph[V][V] = { {1, 1, 0, 1},

{0, 1, 1, 0},

{0, 0, 1, 1},

{0, 0, 0, 1}

};

// Print the solution

transitiveClosure(graph);

return 0;

}

Q-2: Given a weighted directed graph represented by an adjacency matrix, find the shortest distances between all pairs of vertices using Warshall's algorithm.

Sample test case:

|  |
| --- |
| Input: matrix = {{0,1,43},{1,0,6},{-1,-1,0}}  Output: {{0,1,7},{1,0,6},{-1,-1,0}}  ::We can reach 2 from 0 as 0->1->2 and the cost will be 1+6=7 which is less than 43. |

Solution:

#include<bits/stdc++.h>

using namespace std;

class Solution {

public:

// Function to find the shortest distance between all pairs of vertices

void shortest\_distance(vector<vector<int>>&matrix){

int n=matrix.size();

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

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

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

// If there is a path from vertex i to vertex k and from vertex k to vertex j

if(matrix[i][k]!=-1 && matrix[k][j]!=-1){

if(matrix[i][j]==-1){

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

}

else{

matrix[i][j]=min(matrix[i][j], matrix[i][k]+matrix[k][j]);

}

}

}

}

}

}

};

int main(){

int n;

cin >> n;

//Initialize a matrix with -1 representing no direct edge between vertices

vector<vector<int>>matrix(n, vector<int>(n, -1));

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

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

cin >> matrix[i][j];

}

}

Solution obj;

obj.shortest\_distance(matrix);

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

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

cout << matrix[i][j] << " ";

}

cout << "\n";

}

return 0;

}

Q-3: Given a weighted directed graph represented by an adjacency matrix,

determine if the graph contains any negative cycles using Warshall's algorithm.

Sample test case:

|  |
| --- |
| Input:  Adjacency Matrix:  {0 , 1 , INF , INF},  {INF , 0 , -1 , INF},  {INF , INF , 0 , -1},  {-1 , INF , INF , 0}  Output:  Yes |

Solution:

#include<bits/stdc++.h>

using namespace std;

// Number of vertices in the graph

#define V 4

/\* Define Infinite as a large enough value. This value will be used for vertices not connected to each other \*/

#define INF 99999

// A function to print the solution matrix

void printSolution(int dist[][V]);

// Returns true if graph has negative weight cycle

// else false.

bool negCyclefloydWarshall(int graph[][V])

{

/\* dist[][] will be the output matrix that will

finally have the shortest distances between every pair of vertices \*/

int dist[V][V], i, j, k;

/\* Initialize the solution matrix same as input

graph matrix. Or we can say the initial values of shortest distances are based on shortest

paths considering no intermediate vertex. \*/

for (i = 0; i < V; i++)

for (j = 0; j < V; j++)

dist[i][j] = graph[i][j];

/\* Add all vertices one by one to the set of

intermediate vertices.

---> Before start of a iteration, we have shortest

distances between all pairs of vertices such

that the shortest distances consider only the

vertices in set {0, 1, 2, .. k-1} as intermediate

vertices.

----> After the end of a iteration, vertex no. k is

added to the set of intermediate vertices and

the set becomes {0, 1, 2, .. k} \*/

for (k = 0; k < V; k++)

{

// Pick all vertices as source one by one

for (i = 0; i < V; i++)

{

// Pick all vertices as destination for the

// above picked source

for (j = 0; j < V; j++)

{

// If vertex k is on the shortest path from

// i to j, then update the value of dist[i][j]

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

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

}

}

}

// If distance of any vertex from itself becomes negative, then there is a negative weight cycle.

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

if (dist[i][i] < 0)

return true;

}

return false;

}

int main()

{

/\* Let us create the following weighted graph

1

(0)----------->(1)

/|\ |

| |

-1 | |-1

| \|/

(3)<-----------(2)

-1

\*/

int graph[V][V] = { {0 , 1 , INF , INF},

{INF , 0 , -1 , INF},

{INF , INF , 0 , -1},

{-1 , INF , INF , 0}};

if (negCyclefloydWarshall(graph))

cout << "Yes";

else

cout << "No";

return 0;

}