**Task 5: Dynamic Programming – Warshall’s & Floyd’s algorithm**

**a.**

**AIM**

To compute the transitive closure of a given directed graph using Warshall's algorithm

**ALGORITHM**

ALGORITHM Warshall(A[1..n, 1..n])

//Implements Warshall’s algorithm for computing the transitive closure

//Input: The adjacency matrix A of a digraph with n vertices

//Output: The transitive closure of the digraph

← A



for k ← 1 to n do

for i ← 1 to n do

for j ← to n do

R(k)[i, j] ← R(k-1)[i, j] or (R(k-1)[i, k] and R(k-1)[k, j])

return R(n)

**PROGRAM**

#include<stdio.h>

const int MAX = 100;

void WarshallTransitiveClosure(int graph[MAX][MAX], int numVert);

int main(void)

{

int i, j, numVert;

int graph[MAX][MAX];

printf("Warshall's Transitive Closure\n");

printf("Enter the number of vertices : ");

scanf("%d",&numVert);

printf("Enter the adjacency matrix :-\n");

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

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

scanf("%d",&graph[i][j]);

WarshallTransitiveClosure(graph, numVert);

printf("\nThe transitive closure for the given graph is :-\n");

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

{

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

{

printf("%d\t",graph[i][j]);

}

printf("\n");

}

return 0;

}

void WarshallTransitiveClosure(int graph[MAX][MAX], int numVert)

{

int i,j,k;

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

{

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

{

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

{

if (graph[i][j] || (graph[i][k] && graph[k][j]))

graph[i][j] = 1;

}

}

}

}

**TESTCASE 1**

Warshall's Transitive Closure

Enter the number of vertices : 4

Enter the adjacency matrix :-

0 1 0 0 0 0 0 1 0 0 0 0 1 0 1 0

The transitive closure for the given graph is :-

1 1 1 1

1 1 1 1

0 0 0 0

1 1 1 1

**TEST CASE 2**

Warshall's Transitive Closure

Enter the number of vertices : 4

Enter the adjacency matrix :-

1 1 0 1 0 1 1 0 0 0 1 1 0 0 0 1

The transitive closure for the given graph is :-

1 1 1 1

0 1 1 1

0 0 1 1

0 0 0 1

**RESULT**

Thus computing the transitive closure of a given directed graph using Warshall's algorithm was executed successfully

**b.**

**AIM**

To implement the all pairs shortest path problem using Floyd’s algorithm

**ALGORITHM**

ALGORITHM Floyd(W[1..n, 1..n])

//Implements Floyd’s algorithm for the all-pairs shortest-paths problem

//Input: The weight matrix W of a graph with no negative-length cycle

//Output: The distance matrix of the shortest paths’ lengths

D ←W //is not necessary if W can be overwritten

for k←1 to n do

for i ←1 to n do

for j ←1 to n do

D[i, j ]←min{D[i, j ], D[i, k]+ D[k, j]}

return D

**PROGRAM**

#include<stdio.h>

#define V 4

#define INF 99999

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

void floydWarshell(int graph[][V])

{

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

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

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

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

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

{

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

{

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

{

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

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

}

}

}

printSolution(dist);

}

void printSolution(int dist[][V])

{

printf("Following matrix shows the shortest distances"

" between every pair of vertices \n");

int i, j;

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

{

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

{

if (dist[i][j] == INF)

printf("%7s", "INF");

else

printf("%7d", dist[i][j]);

}

printf("\n");

}

}

int main()

{

int graph[V][V] = { { 0, 5, INF, 10 },

{ INF, 0, 3, INF },

{ INF, INF, 0, 1 },

{ INF, INF, INF, 0 }

};

// Print the solution

floydWarshell(graph);

return 0;

}

**OUTPUT**

**Following matrix shows the shortest distances between every pair of vertices**

0 5 8 9

INF 0 3 4

INF INF 0 1

INF INF INF 0

**RESULT**

Thus implementing the all pairs shortest path problem using Floyd’s algorithm was executed successfully