Date:

ASSIGNMENT NO.:

PROBLEM STATEMENT:-

Programming in C to find the path matrix using Warshall's algorithm.

THEORY:-

This is a classical algorithm by which we can determine whether there is a path from any vertex v_i to another vertex v_j either directly or through one or more intermediate vertices. In other words, we can test the reachability of all the pairs of vertices in a graph. The path matrix can be computed from the adjacency matrix A by $P = A+A^2+A^3+\ldots+A^n$ where n= no. of vertices.This method is computationally not efficient at all. To compute the path matrix from a given graph, another elegant method is Warshall's algorithm. This algorithm treats the entries in the adjacency matrix as bit entries & performs AND (ʌ) & OR (v) Boolean operations on them. The heart of the algorithm is a trio of loops, which operates very much like the loops in the classic algorithms for matrix multiplication.

Example :

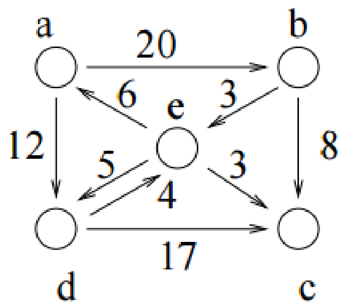


Fig 1: Without negative cost cycle

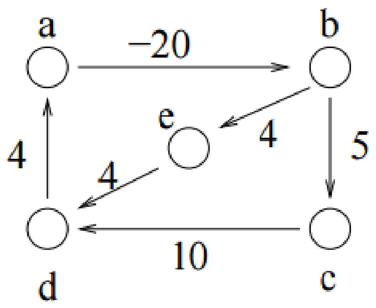


fig 2: With negative cost cycle

ALGORITHM:-

Input:- A graph **G** whose pointer to its adjacency matrix is **GPTR** & vertices are labeled as 1,2,….,N; N being the number of vertices in the graph.
Output:- The path matrix **a**.
Data structure:- Matrix representation of graph **G** with pointer as **GPTR**.

Algorithm for main() function:

Step 1: Input "Enter number of vertices"
Step 2: Read n
Step 3: Repeat through step 4 to step 8 for (i=0 to n) do
Step 4: Repeat through step 5 to step 7 for (j= 0 to n) do

Step 5: Print the existence of path between vertices
Step 6: Read a[i][j]
Step 7: Next j
    [End of inner for loop]
Step 8: Next i
    [End of outer for loop]
Step 9: Call the method display (n,a)
Step 10: Repeat through step 11 to step 16 for (k=0 to n) do
Step 11: Repeat through step 12 to step 15 for (i=0 to n) do
Step 12: Repeat through step 13 to step 14 for (j=0 to n) do
Step 13: Set a[i][j]=a[i][j] OR (a[i][k] AND a[k][j])
Step 14: Next j
    [End of for loop]
Step 15: Next i
    [End of for loop]
Step 16: Next k
    [End of outer for loop]
Step 17: Call the method display(n,a)
Step 18: Stop

Algorithm for the method display():

Step 1: Repeat through step 2 to step for (i=0 to n) do
Step 2: Repeat through step 3 to step for (j=0 to n) do
Step 3: Print a[i][j]
Step 4: Next j
    [End of inner for loop]
Step 5: Next i
    [End of outer for loop]
Step 6: Stop

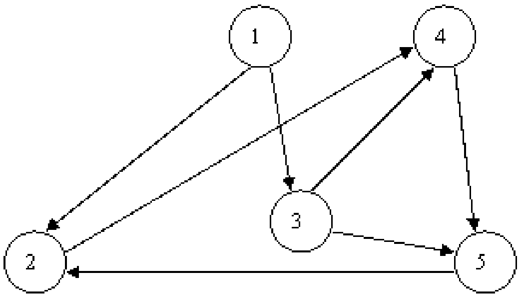
SOURCE CODE:-

```
#include<stdio.h>
void display(int n,int a[20][20]);    //prototype declaration
void main()
{
int a[20][20],i,j,k,n;  //variable declaration
printf("\nEnter the total number of vertices:\t");
scanf("%d",&n);
printf("\nThe existence of path between every pair of  vertices:\n");
printf("1->There is a path between vertices\n0->There is no path between vertices\n");
//loop for taking inputs of the matrix from the user
for(i=0;i<=n;i++)
```

```
{
for(j=0;j<=n;j++)
{
printf("\nEnter the existence of path between vertices  %d & %d:\t",i,j);
scanf("%d",&a[i][j]);
}
}
printf("\n\nThe adjacency matrix is:\n");
display(n,a);  //calling method display
//loop for finding the minimum distance
for(k=0;k<=n;k++)
{
for(i=0;i<=n;i++)
{
for(j=0;j<=n;j++)
{
a[i][j]=a[i][j] || (a[i][k] && a[k][j]);
}
}
}
printf("\n\nThe minimum distance between every pair of vertices:\n");
display(n,a);
}
void display(int n,int a[20][20])
{    //method to display the matrix
int i,j;
for(i=0;i<=n;i++)
{
for(j=0;j<=n;j++)
{
printf(" %d ",a[i][j]);
}
printf("\n");
}
}
```

INPUT-OUTPUT:-

The given graph is:



Enter the total no. of vertices: 5
The existence of path between every pair of vertices:
1->There is a path between vertices
0->There is no path between vertices

Enter the existence of path between vertices 1&1: 0
Enter the existence of path between vertices 1&2: 1
Enter the existence of path between vertices 1&3: 1
Enter the existence of path between vertices 1&4: 0
Enter the existence of path between vertices 1&5: 0
Enter the existence of path between vertices 2&1: 0
Enter the existence of path between vertices 2&2: 0
Enter the existence of path between vertices 2&3: 0
Enter the existence of path between vertices 2&4: 1
Enter the existence of path between vertices 2&5: 0
Enter the existence of path between vertices 3&1: 0
Enter the existence of path between vertices 3&2: 0
Enter the existence of path between vertices 3&3: 0
Enter the existence of path between vertices 3&4: 1
Enter the existence of path between vertices 3&5: 1
Enter the existence of path between vertices 4&1: 0
Enter the existence of path between vertices 4&2: 0
Enter the existence of path between vertices 4&3: 0
Enter the existence of path between vertices 4&4: 0
Enter the existence of path between vertices 4&5: 1

The adjacency matrix:

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

The minimum distance between every pair of vertices:

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

DISCUSSION:

A) It is one of the most commonly used shortest path algorithm. A shortest path between two vertices is a path, which has the least no. of edges among several paths in between two vertices.

B) It is an iterative process. The first iteration consists of finding the existence of path from one vertex to another vertex either directly or indirectly via any intermediate vertex or pivot vertex say v_i. The second iteration finds the existence of path from one vertex to another vertex with v_1 & v_2 or both as pivot & so on.

C) It is the most efficient method to compute the shortest path between every pair of vertices. It requires N_3 comparisons & has an order of complexity $O(N_3)$.
Floyd & Dijkstra are two other methods employed to determine the shortest path between vertices.