Assignment Number

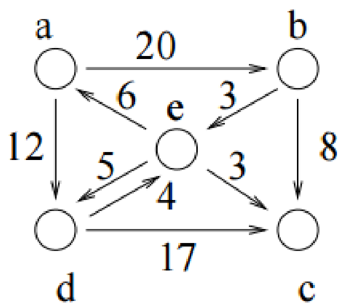
Problem Statement

Program in C to find the path matrix of a graph 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 $P = A+A^2+A^3+.....+A^n$ 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 (∧) & 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 :



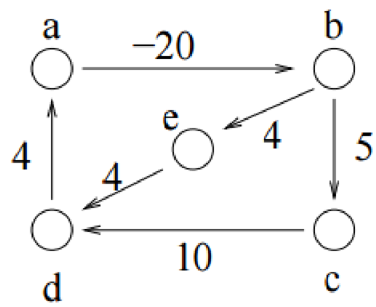


Fig 1: Without negative cost cycle **Fig 2: With negative cost cycle**

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**.

Steps :

Algorithm_Main()

Step 1 : Print "Enter number of vertices"

Step 2 : Input n

Step 3 : Repeat through step 4.a to step 4.b for (i = 0 to n)

- a) Repeat through step a.I to step a.III for (j = 0 to n)
  - I. Print the existence of path between vertices
  - II. Read a[i][j]
  - III. Next j

    [End of inner for loop]
- b) Next i

  [End of outer for loop]

Step 4 : Call Display (n,a)

Step 5 : Repeat through step 5.a to step 5.b for (k = 0 to n)

- a) Repeat through step a.I to step a.II for (i = 0 to n)
  - I. Repeat through step I.i to step I.ii for (j = 0 to n)
    - i. Set a[i][j] = a[i][j] v (a[i][k] ^ a[k][j])
    - ii. Next j

    [End of for loop]
  - II. Next i

  [End of for loop]
- b) Next k

[End of outer for loop]

Step 6 : Call Display(n,a)

Step 7 : Stop

Algorithm_Display()

Step 1 : Repeat through step 1.a to step 1.b for (i = 0 to n)
  a) Repeat through step a.I to step a.II for (j = 0 to n)
    I. Print a[i][j]
    II. Next j
    [End of inner for loop]
  b) Next i
  [End of outer for loop]
Step 2 : Stop

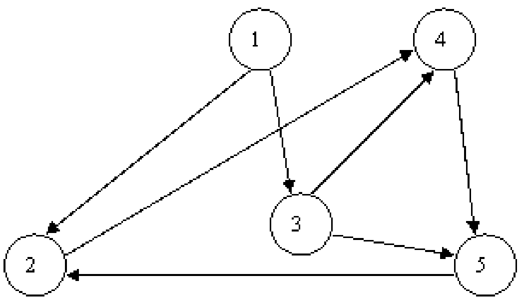
Source Code

```
#include <stdio.h>

void display(int n, int a[20][20]); // prototype declaration

int main() {
   int a[20][20], i, j, k, n; // variable declaration
   printf("\nEnter the total number of vertices : ");
   scanf("%d", &n);
   printf("\nThe existence of path between every pair of  vertices : ");
   printf("\n1 : There is a path between vertices\n0 : There is no path
between vertices");
   // 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 : ",
              i+1, j+1);
         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);
    return 0;
}

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 and Output

The given graph is :



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

The adjacency matrix is :
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

1. 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.
2. 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.
3. 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 $O(N_3)$.
4. Floyd & Dijkstra are two other methods employed to determine the shortest path between vertices.