**Date-**

**Assignment No. :**

**Problem Statement:**

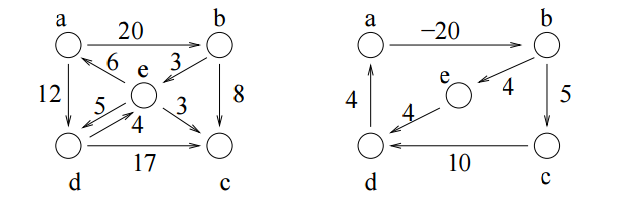
Program in C to find the path matrix of a graph using Warshall’s algorithm.

**Theory:**

This is a classical algorithm by which we can determine whether there is a path from any vertex vi to another vertex vj 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+A2+A3+…..+An 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 specification:**

1. I : The incidence matrix of dimension (n x n) of the given graph
2. vs : The source vertex to start the search from

**Output specification:**

1. A two dimensional array I[1..n][1..n] whose starting index is 1 and ending index is n, size of the array being (n x n).
2. A stack to store the intermediate vertices, say S.

**Steps:**

/\*1 is the adjacency matrix of a graph G, and vs ¡s the source vertex from

which traversal would start \*/

1. Repeat step 2 For(all v E V)
2. Status[v]=unvisited //initially all node is made unvisited

[End of For loop]

1. Set Status[vs]=visited
2. Set U=vs
3. Push(S, vs) //Push is a function to push an element into any stack S

[Starting Do-While loop]

found = FALSE

1. Repeat through step 9 to step 21 For(all y E V)
2. If(status[v]=unvisited AND v is adjacent to u)
3. Print u, v
4. Push(S, v) // insert an element into stack S
5. status[v] =visited
6. u=v
7. found = TRUE
8. Break

[End of If structure]

1. If (found=FALSE)
2. u=POP(S) // delete an element from stack S

[End of If structure]

[End of For loop]

1. Repeat through step 7 to step 20 while(Q is not empty OR v is visited)

[End of Do-While loop]

1. End

**Source Code:**

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

struct node //Creation of node

{

int data;

struct node \*next;

} \*h=NULL;

struct node \*getnode(int data) //Dynamic allocation

{

struct node \*temp;

temp= (struct node \*) malloc (sizeof (struct node));

temp->data=data;

temp->next=NULL;

return temp;

}

void push(int data) //Definition of the push function

{

struct node \*t,\*x;

x=getnode (data);

if (h==NULL)

{

h=x;

}

else

{

x->next=h;

h=x;

}

}

int pop() //Definition of the pop function

{

int u;

if (h==NULL)

printf ( “UNDERFLOW”);

else

{

u=h—>data;

h=h—>next;

}

return u;

}

int status(int s[30],int n) //Definition of the status function

{

int i;

for (i=1; i<=n; i++)

{

if(s[i]==0)

return 1;

}

return 0;

}

void dfs(int l[10][10],int n,int vs) //Function for the searching algorithm

{

int i, u, s[30] , found;

for (i=1; i<=n; i++)

s [i]=0;

s [vs]=1;

u=vs;

push(vs);

do

{

do

{

found=0;

for (i=1; i<=n; i++)

{

if(s[i]==0&&l[u][i]==1)

{

printf(”\n%d %d”,u,i);

push(i);

s[i] =1;

found=1;

u=i;

break;

}

}

if (found==0)

{

u=pop();

}

}while(h!=NULL) ;

for ( i=1; i<=n; i++)

{

if(s[i]==0)

{

s[i]=1;

push(i);

u=i;

break;

}

}

for (i=1 ; i<=n; i++)

{

if(l[u][i]==1&&s[u]!=2)

{

printf(”\n%d to %d”,u,i);

s[u]=2;

break;

}

}

}while (status (s, n));

}

void show(int l[10][10],int n)

{

int i,j;

printf(”\n”) ;

for (i=1; i<=n; i++)

{

for(j=1; j<=n;j++)

printf(” %d “,l[i][j]);

printf(”\n”);

}

}

int main()

{

int choice;

int l[10][10],n,i,j,vs;

printf(”Enter order of the adjacency matrix : “);

scanf (“%d”, &n);

for (i=1; i<=n; i++)

for(j=1; j<=n; j++)

{

printf(”Enter weight between %d & %d”,i,j);

scanf(”%d”,&l[i][j]);

}

printf(”\nAdjacency matrix is . . .\n”);

show (l, n) ;

printf(“Enter source vertex :”);

scanf(“%d”,&vs);

dfs(l,n,vs);

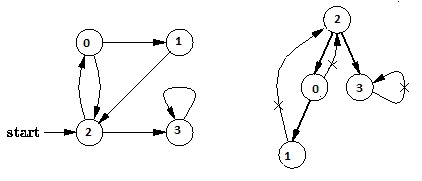
getch();

return 0;

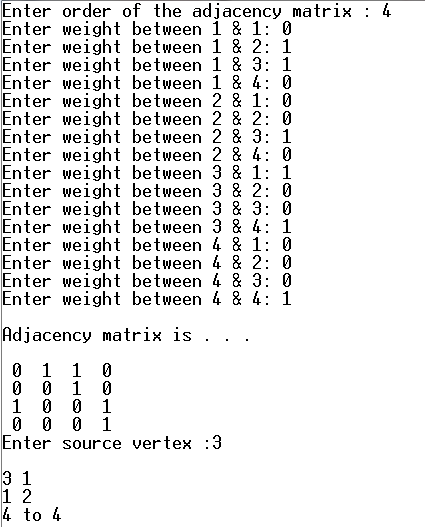
}

**Input & Output:**

Input graph:



Output of program:



**Discussion:**

1. Setting a nodes ( with Stack ) label takes O( 1 ) time.
2. Each Nodes Is labeled twice:
   1. Once as Unexplored.
   2. Once as Visited.
3. Each Edge is labeled twice:
   1. Once as Unexplored.
   2. Once as Discovery or BACK.
4. Because the adjacency list of each nodes is scanned only when the nodes is Pop, each adjacency list is scanned at most once. Total time spent in scanning adjaceny list is O ( E ) [ in worst case ]. As initializations, takes O( V ) times, then total running time of DFS is O( V + E ).