```
            printf("%d ->",ptr->info);
            q = ptr->firstEdge;
            while(q!=NULL)
            {
                    printf(" %d",q->destVertex->info);
                    q = q->nextEdge;
            }
            printf("\n");
            ptr = ptr->nextVertex;
        }
}/*End of display()*/
```

7.11 Transitive closure of a directed graph and Path Matrix

Transitive closure of a graph G is a graph G', where G' contains the same set of vertices as G and whenever there is a path from any vertex i to vertex j in G, there is an edge from i to j in G'.

The path matrix or reachability matrix of a graph G with n vertices is an n x n boolean matrix whose elements can be defined as-

$$P[i][j] = \begin{cases} 1 & \text{if there is a path from vertex i to vertex j} \\ 0 & \text{Otherwise.} \end{cases}$$

Thus the path matrix of a graph G is actually the adjacency matrix of its transitive closure G'. The path matrix of a graph is also known as the transitive closure matrix of the graph.

We will study two methods to compute the path matrix. The first method is by using powers of adjacency matrix and the second one is Warshall's algorithm. Here are some inferences that we can draw by looking at the path matrix-

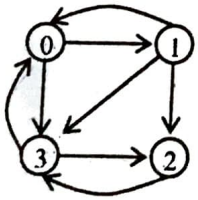
(i) If the element P[i][j] is equal to 1.
There is a path from vertex i to vertex j

(ii) If any main diagonal element i.e. any element P[i][i] in the path matrix is 1.
Graph contains a cycle

(iii) If all the elements in the path matrix are 1.
Graph is strongly connected

7:11.1 Computing Path matrix from powers of adjacency matrix

Let us take a graph and compute the path matrix for it from its adjacency matrix.



Adjacency Matrix A =

	0	1	2	3
0	0	1	0	1
1	1	0	1	1
2	0	0	0	1
3	1	0	1	0

Figure 7.31

Now we compute the matrix AM_2 by multiplying the adjacency matrix A with itself.

$AM_2 = A^2 =$

	0	1	2	3
0	2	0	2	1
1	1	1	1	2
2	1	0	1	0
3	0	1	0	2

In this matrix, value of $AM_2[i][j]$ will represent the number of paths of path length 2 from vertex i to vertex j. For example vertex 0 has two paths of path length 2 to vertex 2, and vertex 3 has two paths of path length 2 to itself, vertex 2 has one path of path length 2 to vertex 0, there is no path of path length 2 from vertex 2 to vertex

1. These paths may not be simple paths, i.e. all vertices in these paths need not be distinct. Now we compute the matrix AM_3 by multiplying the adjacency matrix A with AM_2.

$$AM_3 = AM_2*A = A^3 = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 1 & 2 & 1 & 4 \\ 3 & 1 & 3 & 3 \\ 0 & 1 & 0 & 2 \\ 3 & 0 & 3 & 1 \end{bmatrix} \end{array}$$

Here $AM_3[i][j]$ will represent the number of paths of path length 3 from vertex i to vertex j. For example, vertex 0 has 4 paths of path length 3 to vertex 3(paths 0-3-2-3, 0-3-0-3, 0-1-2-3, 0-1-0-3) and vertex 3 has no path of path length 3 to vertex 1. Similarly, we can find out the matrix AM_4.

$$AM_4 = AM_3*A = A^4 = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 6 & 1 & 6 & 4 \\ 4 & 3 & 4 & 7 \\ 3 & 0 & 3 & 1 \\ 1 & 3 & 1 & 6 \end{bmatrix} \end{array}$$

In general we can say that if AM_k is equal to A^k, then any element $AM_k[i][j]$ represents the number of paths of path length k from vertex i to vertex j.

Let us define a matrix X where

$X = AM_1 + AM_2 + \ldots\ldots\ldots\ldots + AM_n$

X[i][j] denotes the number of paths, of path length n or less than n, from vertex i to vertex j. Here n is the total number of vertices in the graph.

For the graph in figure 7.31, the value of X will be-

$$X = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 9 & 4 & 9 & 10 \\ 9 & 5 & 9 & 13 \\ 4 & 1 & 4 & 4 \\ 5 & 4 & 5 & 9 \end{bmatrix} \end{array}$$

From definition of path matrix we know that P[i][j]=1 if there is a path from i to j, and this path can have length n or less than n. Now in the matrix X, if we replace all nonzero entries by 1 then we will get the path matrix or reachability matrix.

$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

This graph is strongly connected since all the entries are equal to 1.

```
/*P7.4 Program to find out the path matrix by powers of adjacency matrix*/
#include<stdio.h>
#define MAX 100
void display(int matrix[MAX][MAX]);
void pow_matrix(int p,int adjp[MAX][MAX]);
void multiply(int mat1[MAX][MAX],int mat2[MAX][MAX],int mat3[MAX][MAX]);
void create_graph();
int adj[MAX][MAX];
int n;

main()
{
        int adjp[MAX][MAX];
        int x[MAX][MAX],path[MAX][MAX],i,j,p;
        create_graph();
```

```
        printf("The adjacency matrix is :\n");
        display(adj);
        /*Initialize all elements of matrix x to zero*/
        for(i=0; i<n; i++)
                for(j=0; j<n; j++)
                   x[i][j] = 0;
        /*All the powers of adj will be added to matrix x */
        for(p=1; p<=n; p++)
        {
                pow_matrix(p,adjp);
                printf("Adjacency matrix raised to power %d is - \n",p);
                display(adjp);
                for(i=0; i<n; i++)
                  for(j=0; j<n; j++)
                        x[i][j] = x[i][j]+adjp[i][j];
        }
        printf("The matrix x is :\n");
        display(x);
        for(i=0; i<n; i++)
                for(j=0; j<n; j++)
                        if(x[i][j]==0)
                                path[i][j] = 0;
                        else
                                path[i][j] = 1;

        printf("The path matrix is :\n");
        display(path);
}/*End of main()*/
void create_graph()
{
        int i,max_edges,origin,destin;
        printf("Enter number of vertices : ");
        scanf("%d",&n);
        max_edges = n*(n-1);
        for(i=1; i<=max_edges; i++)
        {
                printf("Enter edge %d( -1 -1 ) to quit : ",i);
                scanf("%d %d",&origin,&destin);
                if((origin==-1) && (destin==-1))
                        break;
                if(origin>=n || destin>=n || origin<0 || destin<0)
                {
                        printf("Invalid edge!\n");
                        i--;
                }
                else
                        adj[origin][destin] = 1;
        }/*End of for*/
}/*End of create_graph()*/
/*This function computes the pth power of matrix adj and stores result in adjp*/
void pow_matrix(int p,int adjp[MAX][MAX])
{
        int i,j,k,tmp[MAX][MAX];
        /*Initially adjp is equal to adj*/
        for(i=0; i<n; i++)
          for(j=0; j<n; j++)
           adjp[i][j] = adj[i][j];
        for(k=1; k<p; k++)
        {
                /*Multiply adjp with adj and store result in tmp*/
                multiply(adjp,adj,tmp);
                for(i=0; i<n; i++)
                  for(j=0; j<n; j++)
                     adjp[i][j] = tmp[i][j]; /*New adjp is equal to tmp*/
```

```
        }
}/*End of pow_matrix()*/
/*This function multiplies mat1 and mat2 and stores the result in mat3*/
void multiply(int mat1[MAX][MAX],int mat2[MAX][MAX],int mat3[MAX][MAX])
{
        int i,j,k;
        for(i=0; i<n; i++)
                for(j=0; j<n; j++)
                {
                        mat3[i][j] = 0;
                        for(k=0; k<n; k++)
                                mat3[i][j] = mat3[i][j] + mat1[i][k] * mat2[k][j];
                }
}/*End of multiply()*/
void display(int matrix[MAX][MAX])
{
        int i,j;
        for(i=0; i<n; i++)
        {
                for(j=0; j<n; j++)
                        printf("%4d",matrix[i][j]);
                printf("\n");
        }
        printf("\n");
}/*End of display()*/
```

7.11.2 Warshall's Algorithm

We have seen that we can find the path matrix P of a given graph G by using powers of adjacency matrix. Warshall gave an efficient technique for finding path matrix of a graph known as Warshall's algorithm. Let us take a graph G of n vertices 0, 1, 2,.....n-1. We will define Boolean matrices P_{-1}, P_0, P_1, ... P_{n-1} where $P_k[i][j]$ is defined as-

$$P_k[i][j] = \begin{cases} 1 & \text{If there is a simple path from vertex i to vertex j which does not use any intermediate vertex greater than k, i.e. all intermediate vertices belong to the set } \{0,1,.....k\} \\ 0 & \text{Otherwise} \end{cases}$$

$P_{-1}[i][j] = 1$ If there is a simple path from vertex i to vertex j, which does not use any intermediate vertex.

$P_0[i][j] = 1$ If there is a simple path from vertex i to vertex j, which does not use any other intermediate vertex except possibly vertex 0.

$P_1[i][j] = 1$ If there is a simple path from vertex i to vertex j, which does not use any other intermediate vertex except possibly 0,1.

$P_2[i][j] = 1$ If there is a simple path from vertex i to vertex j which does not use any other intermediate vertices except possibly vertices 0,1,2.

....

....

$P_k[i][j] = 1$ If there is a simple path from vertex i to vertex j which does not use any other intermediate vertices except possibly 0,1,.....k

....

....

$P_{n-1}[i][j] = 1$ If there is a simple path from vertex i to vertex j which does not use any other intermediate vertices except possibly 0, 1, ...n-1.

Here P_{-1} represents the adjacency matrix and P_{n-1} represents the path matrix, let us see why.

$P_{-1}[i][j]=1$, If there is a simple path from vertex i to vertex j which does not use any vertex. The only way to go from i to j without using any vertex is to go directly from i to j. Hence $P_{-1}[i][j]=1$ means that there is an edge from i to j. So P_{-1} will be the adjacency matrix.

$P_{n-1}[i][j]=1$, If there is a simple path from i to j which does not use any vertices except 0, 1,...n-1. There are total n vertices means this path can use all n vertices; hence from the definition of path matrix we observe that P_{n-1} is the path matrix.

We know that P_{-1} is equal to the adjacency matrix, which we can easily find out from the graph. We have to find matrices $P_0, P_1, \ldots\ldots\ldots P_{n-1}$. If we know how to find matrix P_k from matrix P_{k-1} then we can find all these matrices. So now let us see how to find the value of $P_k[i][j]$ by looking at the matrix P_{k-1}.

If $P_{k-1}[i][j]$ is 1, $P_k[i][j]$ will also be 1, let us see why.

$P_{k-1}[i][j]=1$, implies that there is a simple path(say P1) from i to j which does not use any vertices except possibly 0, 1, ...k-1 or we can say that this path does not use any vertices numbered higher than k-1. So it is obvious that this path does not use any vertices numbered higher than k also or we can say that this path does not use any other vertices except possibly 0, 1, ... k. So $P_k[i][j]$ will be equal to 1.

Path P1 from i to j uses intermediate vertices from the set {0,1, ...k-1}
$P_{k-1}[i][j]=1$

Path P1 from i to j uses intermediate vertices from the set {0,1, ...k}
$P_k[i][j]=1$

Now we will consider the case when $P_{k-1}[i][j] = 0$. In this case $P_k[i][j]$ can be 0 or it can be 1. Let us find out the condition in which $P_k[i][j]$ will be 1 when $P_{k-1}[i][j]$ is 0.

$P_{k-1}[i][j]$ is 0, means there is no path from i to j using intermediate vertices 0,1, ...,k-1.
$P_k[i][j]$ is 1, means there is a path from i to j using intermediate vertices 0,1,.....k

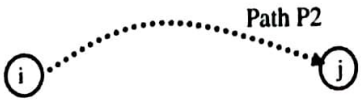
This means that when we use only vertices 0,1,.....,k-1 we have no path from i to j but when we use vertices 0,1,.....,k we get a path(say P2) from i to j. This path P2 will definitely pass through vertex k, so we can break it into two subpaths-

(i) path P2a from i to k using vertices 0,1,.....k-1.
(ii) path P2b from k to j using vertices 0,1,.....k-1.

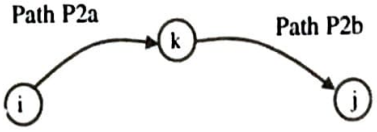
Since we have taken out k and the path P2 is simple(k can't be repeated), the paths P2a and P2b will have intermediate vertices from the set {0,1,2.....k-1}.

From path P2a we can write that $P_{k-1}[i][k]=1$
From path P2b we can write that $P_{k-1}[k][j]=1$

No path from i to j using intermediate vertices from the set {0,1,.....,k-1}
$P_{k-1}[i][j]=0$



Path P2 with intermediate vertices from the set {0,1,.....,k}
$P_k[i][j]=1$



Path P2 broken into two paths.
Paths P2a and paths P2b with intermediate vertices from the set {0,1,.....,k-1}
$P_{k-1}[i][k]=1$ and $P_{k-1}[k][j]=1$

For existence of path P2, the paths P2a and P2b should exist, i.e. for $P_k[i][j]$ to be 1 when $P_{k-1}[i][j]$ is zero, the values of $P_{k-1}[i][k]$ and $P_{k-1}[k][j]$ should be 1.

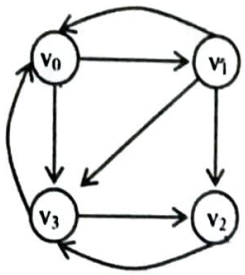
We can conclude that if $P_{k-1}[i][j]=0$ then $P_k[i][j]$ can be equal to 1 only
if $P_{k-1}[i][k]=1$ and $P_{k-1}[k][j]=1$.

So we have two situations when $P_k[i][j]$ can be 1

1. $P_{k-1}[i][j] = 1$ or
2. $P_{k-1}[i][k] = 1$ and $P_{k-1}[k][j] = 1$

To find any element $P_k[i][j]$ we will proceed as –
First see $P_{k-1}[i][j]$, If it is equal to 1, then $P_k[i][j]=1$, done
If $P_{k-1}[i][j]=0$, then see $P_{k-1}[i][k]$ and $P_{k-1}[k][j]$, if both are 1 then $P_k[i][j]=1$, done
Otherwise $P_k[i][j] = 0$

Let us take the same graph as in figure 7.31 and find out the values of P_{-1}, P_0, P_1, P_2, P_3



The first matrix P_{-1} is the adjacency matrix.

$$P_{-1} = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

Now we have to find the matrix P_0
To find any element $P_0[i][j]$ we will proceed as-
First see $P_{-1}[i][j]$, If it is equal to 1, then $P_0[i][j]=1$
Otherwise If $P_{-1}[i][j]=0$, then see $P_{-1}[i][0]$ and $P_{-1}[0][j]$, if both are 1 then $P_0[i][j]=1$
Otherwise $P_0[i][j]=0$

Calculation of some elements of matrix P_0-

* Find $P_0[2][3]$
  $P_{-1}[2][3] = 1$ so $P_0[2][3]=1$
* Find $P_0[3][1]$
  $P_{-1}[3][1] = 0$, so look at $P_{-1}[3][0]$ and $P_{-1}[0][1]$, both are 1 hence $P_0[3][1] = 1$
* Find $P_0[2][1]$
  $P_{-1}[2][1] = 0$, so look at $P_{-1}[2][0]$ and $P_{-1}[0][1]$, one of them is 0, hence $P_0[2][1] = 0$
* Find $P_0[2][2]$
  $P_{-1}[2][2] = 0$, so look at $P_{-1}[2][0]$ and $P_{-1}[0][2]$, both are 0, hence $P_0[2][2] = 0$

It is clear that if an entry is 1 in matrix P_{-1}, then it will also be 1 in P_0. So we can just copy all the 1's and see if the zero entries of P_{-1} can be changed to 1 in P_0. Changing of a zero entry to 1 implies that we get a path if we use 0 as the intermediate vertex.

$$P_0 = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

Now we have to find matrix P_1.

* Find $P_1[0][2]$
  $P_0[0][2] = 0$, so look at $P_0[0][1]$ and $P_0[1][2]$, both are 1, hence $P_1[0][2]=1$
* Find $P_1[2][1]$
  $P_0[2][1] = 0$, so look at $P_0[2][1]$ and $P_0[1][1]$, one of them is zero, hence $P_1[2][1]=0$

$$P_1 = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

* Find $P_2[2][1]$

$P_1[2][1] = 0$, so look at $P_1[2][2]$ and $P_1[2][1]$, both are zero, hence $P_2[2][1]=0$

$$P_2 = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

And

$$P_3 = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

Here P_{-1} is the adjacency matrix and P_3 is the path matrix of the graph. In the program all the calculation can be done in place using a single two dimensional array P.

```
/*P7.5 Program to find path matrix by Warshall's algorithm*/
#include<stdio.h>
#define MAX 100
void display(int matrix[MAX][MAX], int n);
int adj[MAX][MAX];
int n;
void create_graph();
main()
{
        int i,j,k;
        int P[MAX][MAX];
        create_graph();
        printf("The adjacency matrix is :\n");
        display(adj,n);
        for(i=0; i<n; i++)
           for(j=0; j<n; j++)
                 P[i][j] = adj[i][j];
        for(k=0; k<n; k++)
        {
                for(i=0; i<n; i++)
                  for(j=0; j<n; j++)
                      P[i][j] = (P[i][j] || (P[i][k] && P[k][j]));
                printf("P%d is :\n",k);
                display(P,n);
        }
        printf("P%d is the path matrix of the given graph\n",k-1);
}/*End of main() */
void display(int matrix[MAX][MAX],int n)
{
        int i,j;
        for(i=0; i<n; i++)
        {
                for(j=0; j<n; j++)
                        printf("%3d",matrix[i][j]);
                printf("\n");
        }
```

```
}/*End of display()*/
void create_graph()
{
        int i,max_edges,origin,destin;
        printf("Enter number of vertices : ");
        scanf("%d",&n);
        max_edges = n*(n-1);
        for(i=1; i<=max_edges; i++)
        {
                printf("Enter edge %d( -1 -1 ) to quit : ",i);
                scanf("%d %d",&origin,&destin);
                if((origin==-1) && (destin==-1))
                        break;
                if(origin>=n || destin>=n || origin<0 || destin<0)
                {
                        printf("Invalid edge!\n");
                        i--;
                }
                else
                        adj[origin][destin] = 1;
        }/*End of for*/
}/*End of create_graph()*/
```