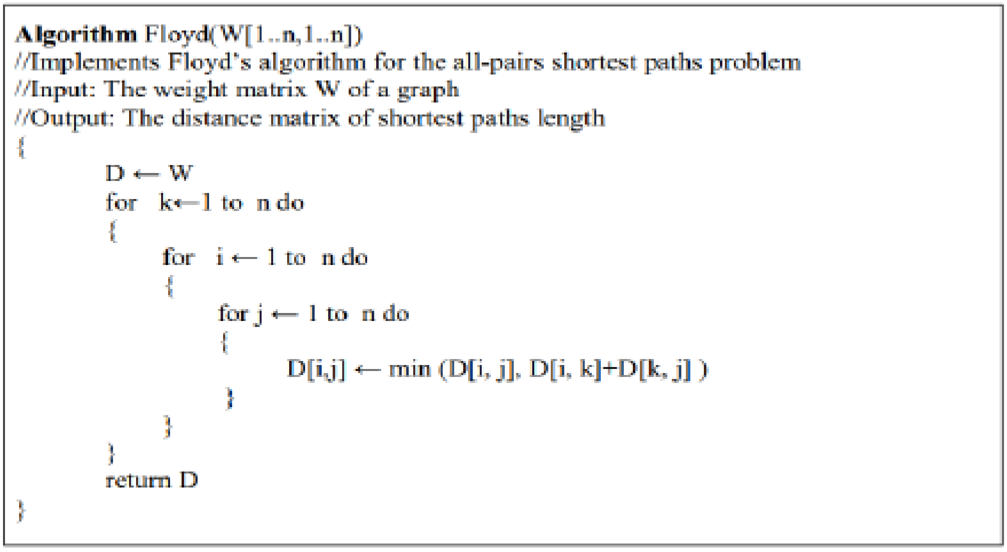
Program 3:

a. Design and implement C/C++ Program to solve All-Pairs Shortest Paths problem using Floyd’s algorithm.
b. Design and implement C/C++ Program to find the transitive closure using Warshal’s algorithm.

Program 3a

Algorithm:



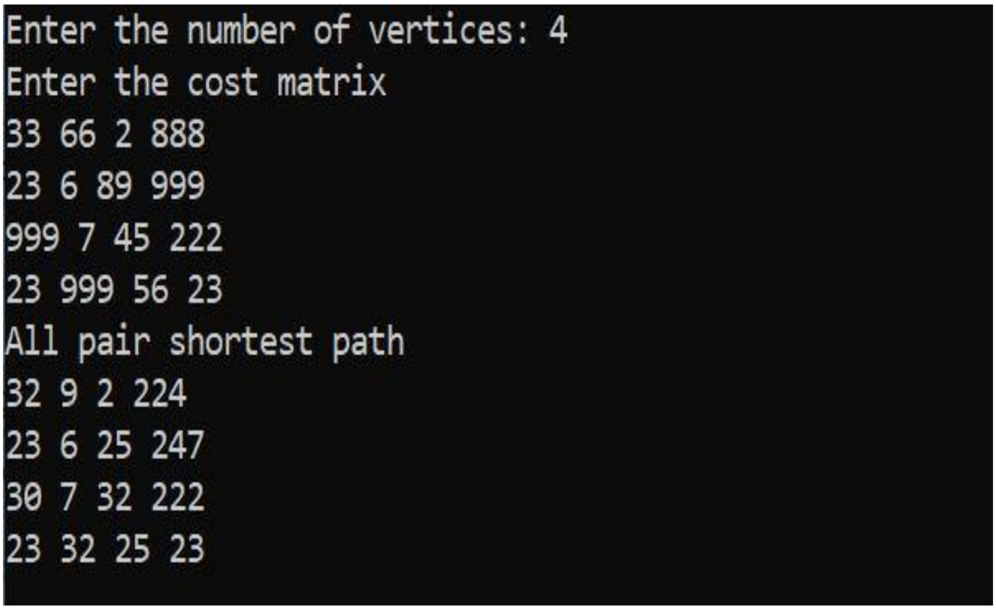
```
Algorithm Floyd(W[1..n,1..n])
//Implements Floyd's algorithm for the all-pairs shortest paths problem
//Input: The weight matrix W of a graph
//Output: The distance matrix of shortest paths length
{
        D ← W
        for  k←1 to  n do
        {
            for  i ← 1 to  n do
            {
                for j ← 1 to  n do
                {
                    D[i,j] ← min (D[i, j], D[i, k]+D[k, j] )
                }
            }
        }
        return D
}
```

Code:

```
#include<stdio.h>
int min(int a, int b)
{
return(a<b?a:b);
}
void floyd(int D[][10],int n)
{
for(int k=1;k<=n;k++)
for(int i=1;i<=n;i++)
for(int j=1;j<=n;j++)
D[i][j]=min(D[i][j],D[i][k]+D[k][j]);
}
```

```
int main()
{
int n, cost[10][10];
printf("Enter the number of vertices: ");
scanf("%d",&n);
printf("Enter the cost matrix \n");
for(int i=1;i<=n;i++)
for(int j=1;j<=n;j++)
scanf("%d",&cost[i][j]);
floyd(cost,n);
printf("All pair shortest path \n");
for(int i=1;i<=n;i++)
{
for(int j=1;j<=n;j++)
printf("%d ",cost[i][j]);
printf("\n");
}
}
```

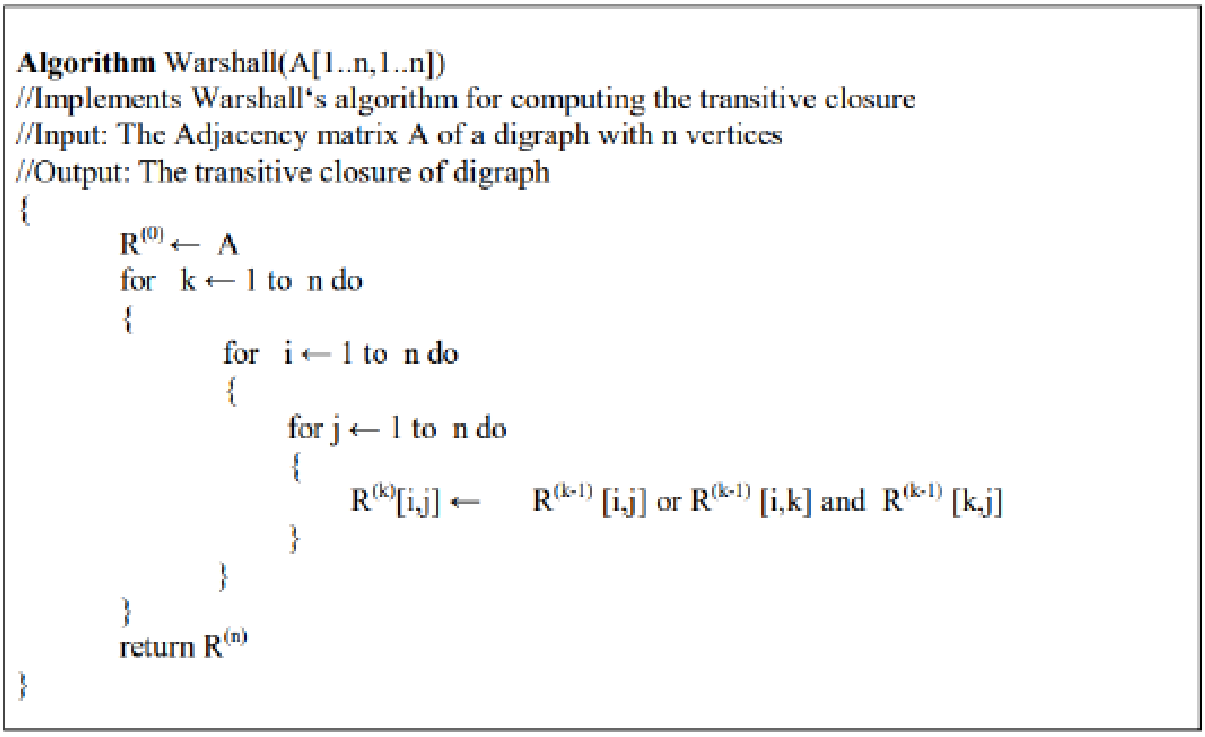
Output:



```
Enter the number of vertices: 4
Enter the cost matrix
33 66 2 888
23 6 89 999
999 7 45 222
23 999 56 23
All pair shortest path
32 9 2 224
23 6 25 247
30 7 32 222
23 32 25 23
```

Program 3b

Algorithm:

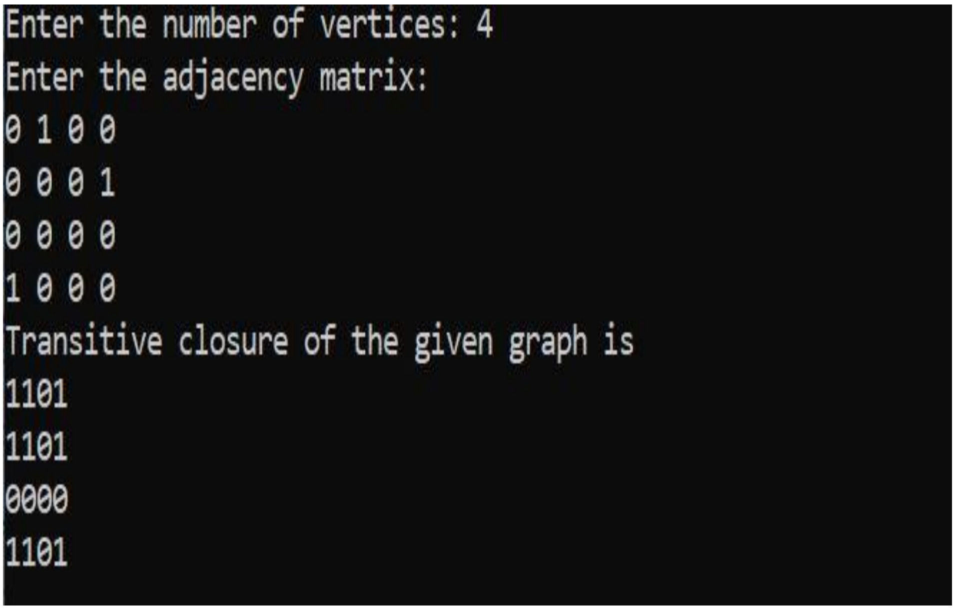


```
Algorithm Warshall(A[1..n,1..n])
//Implements Warshall's algorithm for computing the transitive closure
//Input: The Adjacency matrix A of a digraph with n vertices
//Output: The transitive closure of digraph
{
        $R^{(0)}$ ← A
        for  k ← 1 to  n do
        {
                for  i ← 1 to  n do
                {
                        for j ← 1 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 $R^{(n)}$
}
```

Code:

```
#include<stdio.h>
void warshal(int A[][10], int n)
{
for(int k=1;k<=n;k++)
for(int i=1;i<=n;i++)
for(int j=1;j<=n;j++)
A[i][j]=A[i][j] || (A[i][k]&&A[k][j]);
}
void main()
{
int n,adj[10][10];
printf("Enter the number of vertices: ");
scanf("%d",&n);
printf("Enter the adjacency matrix: \n");
for(int i=1;i<=n;i++)
for(int j=1;j<=n;j++)
scanf("%d",&adj[i][j]);
warshal(adj,n);
printf("Transitive closure of the given graph is \n");
for(int i=1;i<=n;i++)
{
for(int j=1;j<=n;j++)
printf("%d",adj[i][j]);
printf("\n");
}
}
```

Output:

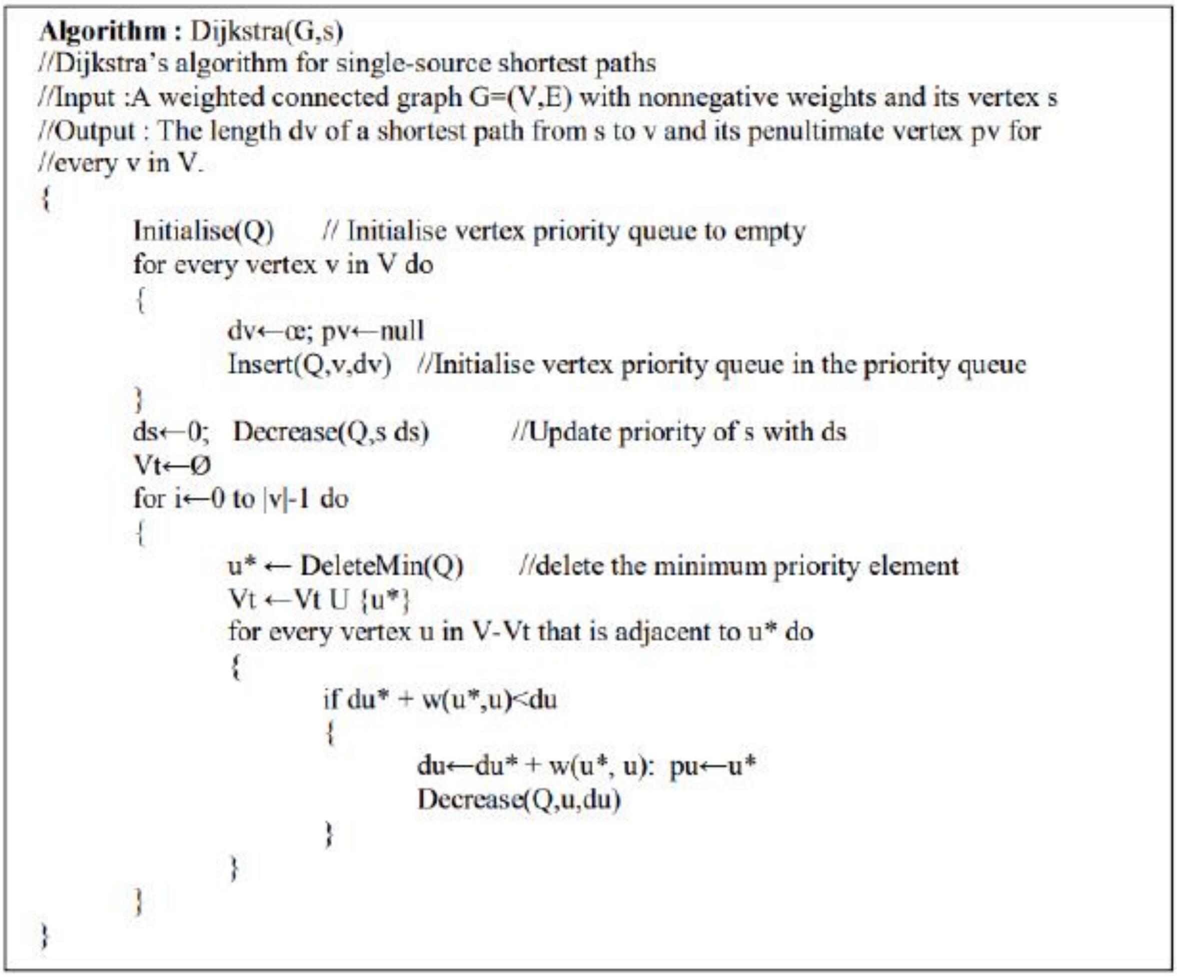


```
Enter the number of vertices: 4
Enter the adjacency matrix:
0 1 0 0
0 0 0 1
0 0 0 0
1 0 0 0
Transitive closure of the given graph is
1101
1101
0000
1101
```

Program 4:

Design and implement C/C++ program to find shortest path from a given vertex in a weighted connected graph to other vertices using Dijkstra's algorithm.

Algorithm:



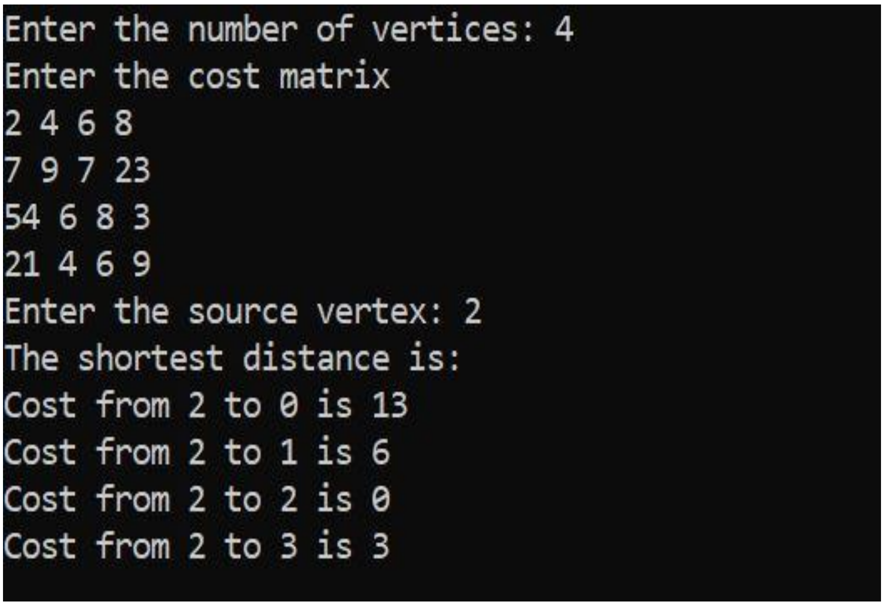
```
Algorithm : Dijkstra(G,s)
//Dijkstra's algorithm for single-source shortest paths
//Input :A weighted connected graph G=(V,E) with nonnegative weights and its vertex s
//Output : The length dv of a shortest path from s to v and its penultimate vertex pv for
//every v in V.
{
        Initialise(Q)     // Initialise vertex priority queue to empty
        for every vertex v in V do
        {
                dv←œ; pv←null
                Insert(Q,v,dv)   //Initialise vertex priority queue in the priority queue
        }
        ds←0;  Decrease(Q,s ds)        //Update priority of s with ds
        Vt←Ø
        for i←0 to |v|-1 do
        {
                u* ← DeleteMin(Q)      //delete the minimum priority element
                Vt ←Vt U {u*}
                for every vertex u in V-Vt that is adjacent to u* do
                {
                        if du* + w(u*,u)<du
                        {
                                du←du* + w(u*, u):  pu←u*
                                Decrease(Q,u,du)
                        }
                }
        }
}
```

Code:

```
#include<stdio.h>

int cost[10][10],n,dist[10];

int minm(int m, int n)

{

return((m<n)?m:n);
}
void dijkstra(int source)
{
int s[10]={0};
int min, w=0;
for(int i=0;i<n;i++)
dist[i]=cost[source][i];
dist[source]=0;
s[source]=1;
for(int i=0;i<n-1;i++)
{
min=999;
for(int j=0;j<n;j++)
{
if((s[j]==0)&&(min>dist[j]))
{
min=dist[j];
w=j;
}
}
s[w]=1;
for(int v=0;v<n;v++)
{
if(s[v]==0&&cost[w][v]!=999)
{
dist[v]=minm(dist[v],dist[w]+cost[w][v]);
}
}
}
}
int main()
```

```
{
int source;
printf("Enter the number of vertices: ");
scanf("%d",&n);
printf("Enter the cost matrix \n");
for(int i=0;i<n;i++)
for(int j=0;j<n;j++)
scanf("%d",&cost[i][j]);
printf("Enter the source vertex: ");
scanf("%d",&source);
dijkstra(source);
printf("The shortest distance is: \n");
for(int i=0;i<n;i++)
printf("Cost from %d to %d is %d\n",source,i,dist[i]);
}
```

Output:



```
Enter the number of vertices: 4
Enter the cost matrix
2 4 6 8
7 9 7 23
54 6 8 3
21 4 6 9
Enter the source vertex: 2
The shortest distance is:
Cost from 2 to 0 is 13
Cost from 2 to 1 is 6
Cost from 2 to 2 is 0
Cost from 2 to 3 is 3
```