Program 5:

Design and implement C/C++ program to obtain the Topological ordering of vertices in a given digraph.

Algorithm:

```
Algorithm topological_sort(a,n,T)
        //purpose :To obtain the sequence of jobs to be executed resut
          In topolocical order
        // Input:a-adjacency matrix of the given graph
        //n-the number of vertices in the graph
        //output:
        // T-indicates the jobs that are to be executed in the order

            For j<-0 to n-1 do
                  Sum-0
                  For  i<- 0to n-1 do
                     Sum<-sum+a[i][j]
              End for
                Top ← -1
                 For i<- 0 to n-1 do
                       If(indegree [i]=0)
                             Top <-top+1
                             S[top]<- i
                  End if
                  End for
       While top!= 1
            u<-s[top]
          top<-top-1
       Add u to solution vector T
           For each vertex v adjacent to u
                  Decrement indegree [v] by one
                       If(indegree [v]=0)
                        Top<-top+1
                        S[top]<-v
        End if
        End for
        End while
        Write T
        return
```

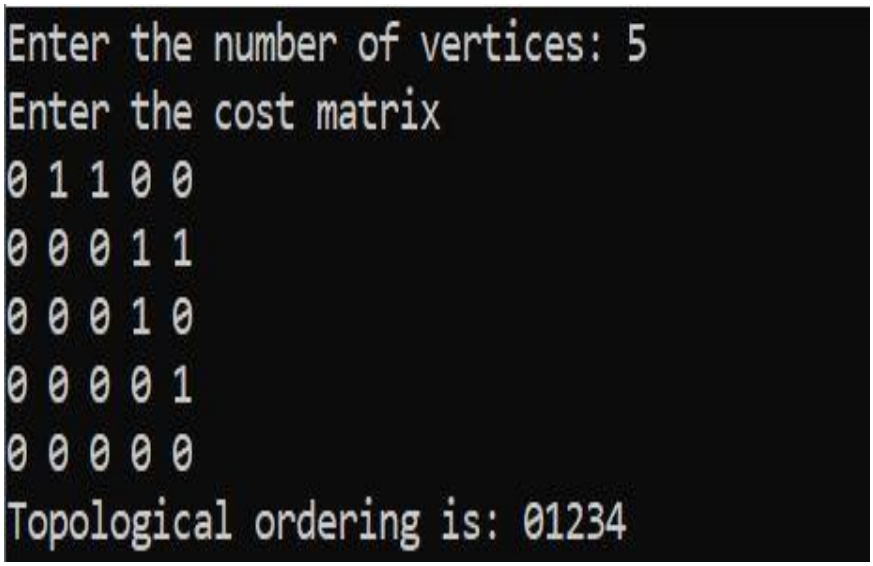
Code:

```
#include<stdio.h>
int cost[10][10],n,colsum[10];
```

```
void cal_colsum()
{
for(int j=0;j<n;j++)
{
colsum[j]=0;
for(int i=0;i<n;i++)
colsum[j]+=cost[i][j];
}
}
void source_removal()
{
int i,j,k,select[10]={0};
printf("Topological ordering is: ");
for(i=0;i<n;i++)
{
cal_colsum();
for(j=0;j<n;j++)
{
if(select[j]==0&&colsum[j]==0)
break;
}
printf("%d",j);
select[j]=1;
for(k=0;k<n;k++)
cost[j][k]=0;
}
}
void main()
{
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]);
source_removal();
}
```

Output:



Program 6:

Design and implement C/C++ program to solve 0/1 Knapsack Problem using Dynamic Programming method.

Algorithm:

```
Algorithm: 0/1Knapsack(S, W)
//Input: set S of items with benefit b_i and weight w_i; max. weight W
//Output: benefit of best subset with weight at most W
// Sk: Set of items numbered 1 to k.
//Define B[k,w] = best selection from Sk with weight exactly equal to w
{
    for w ← 0 to n-1 do
            B[w] ← 0
    for k ← 1 to n do
    {
        for w ← W downto w_k do
        {
            if B[w-w_k]+b_k > B[w] then
                B[w] ← B[w-w_k]+b_k
        }
    }
}
```

Code:

```
#include<stdio.h>

int n,m,p[10],w[10];

int max(int a, int b)

{

return(a>b?a:b);

}

void knapsack_DP()

{
```

```
int V[10][10],i,j;
for(i=0;i<=n;i++)
for(j=0;j<=m;j++)
if(i==0||j==0)
V[i][j]=0;
else
if(j<w[i])
V[i][j]=V[i-1][j];
else
V[i][j]=max(V[i-1][j],p[i]+V[i-1][j-w[i]]);
for(i=0;i<=n;i++)
{
for(j=0;j<=m;j++)
printf("%d ",V[i][j]);
printf("\n");
}
printf("Items included are: ");
while(n>0)
{
if(V[n][m]!=V[n-1][m])
{
printf("%d ",n);
m=m-w[n];
}
```

```
n--;
}
}
int main()
{
int i;
printf("Enter the number of items: ");
scanf("%d",&n);
printf("Enter the weights of n items: ");
for(i=1;i<=n;i++)
scanf("%d",&w[i]);
printf("Enter the prices of n items: ");
for(i=1;i<=n;i++)
scanf("%d",&p[i]);
printf("Enter the capacity of Knapsack: ");
scanf("%d",&m);
knapsack_DP();
}
```

Output:

```
Enter the number of items: 4
Enter the weights of n items: 7 3 4 5
Enter the prices of n items: 42 12 40 25
Enter the capacity of Knapsack: 10
0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 42 42 42 0
0 0 0 12 12 12 12 42 42 42 0
0 0 0 12 40 40 40 52 52 52 0
0 0 0 12 40 40 40 52 52 65 65
Items included are: 4 3
```