EXPERIMENT-1

OBJECTIVE

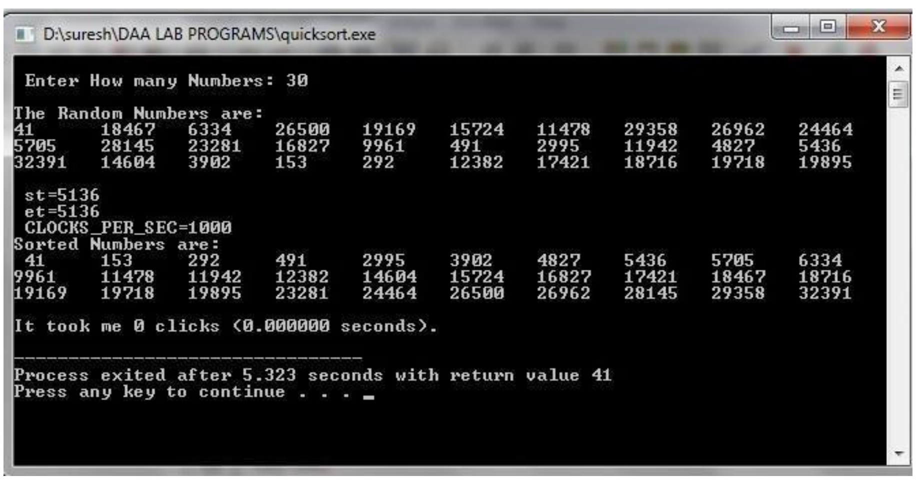
Sort a given set of elements using the Quicksort method and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

PROGRAM

```
#include <stdio.h>

#include <time.h>

voidExch(int *p, int *q)

{ int temp = *p; *p = *q; *q = temp; }

voidQuickSort(int a[], int low, int high)

{

 int i, j, key, k; if(low>=high)

ret urn;

key=lo w;

i=low+1;

j=high;

while(i<=j)

 { while ( a[i] <= a[key] ) i=i+1;

while ( a[j] > a[key] ) j=j -1;

if(i<J)

Exch(&a[i], &a[j]);

}

 Exch(&a[j], &a[key]);

QuickSort(a, low, j1);
```

```
QuickSort(a, j+1, high);

} void main()

{ int n, a[1000],k;

clock_tst,et; double ts; clrscr();

 printf("\n Enter How many Numbers: ");

scanf("%d", &n);

printf("\nThe Random Numbers are:\n");

 for(k=1; k<=n; k++){ a[k]=rand(); printf("%d\t",a[k ]);

 }

st=clock(); QuickSort(a, 1, n);

et=clock();

ts=(double)(et-st)/CLOCKS _PER_SEC;

printf("\nSorted Numbers are: \n ");

for(k=1; k<=n; k++) printf("%d\t", a[k]);

printf("\nThe time taken is %e",ts);

}
```

OUTPUT



```
D:\suresh\DAA LAB PROGRAMS\quicksort.exe

 Enter How many Numbers: 30

The Random Numbers are:
41      18467   6334    26500   19169   15724   11478   29358   26962   24464
5705    28145   23281   16827   9961    491     2995    11942   4827    5436
32391   14604   3902    153     292     12382   17421   18716   19718   19895

 st=5136
 et=5136
 CLOCKS_PER_SEC=1000
Sorted Numbers are:
 41     153     292     491     2995    3902    4827    5436    5705    6334
9961    11478   11942   12382   14604   15724   16827   17421   18467   18716
19169   19718   19895   23281   24464   26500   26962   28145   29358   32391

It took me 0 clicks (0.000000 seconds).

--------------------------------
Process exited after 5.323 seconds with return value 41
Press any key to continue . . .
```

EXPERIMENT-2

OBJECTIVE

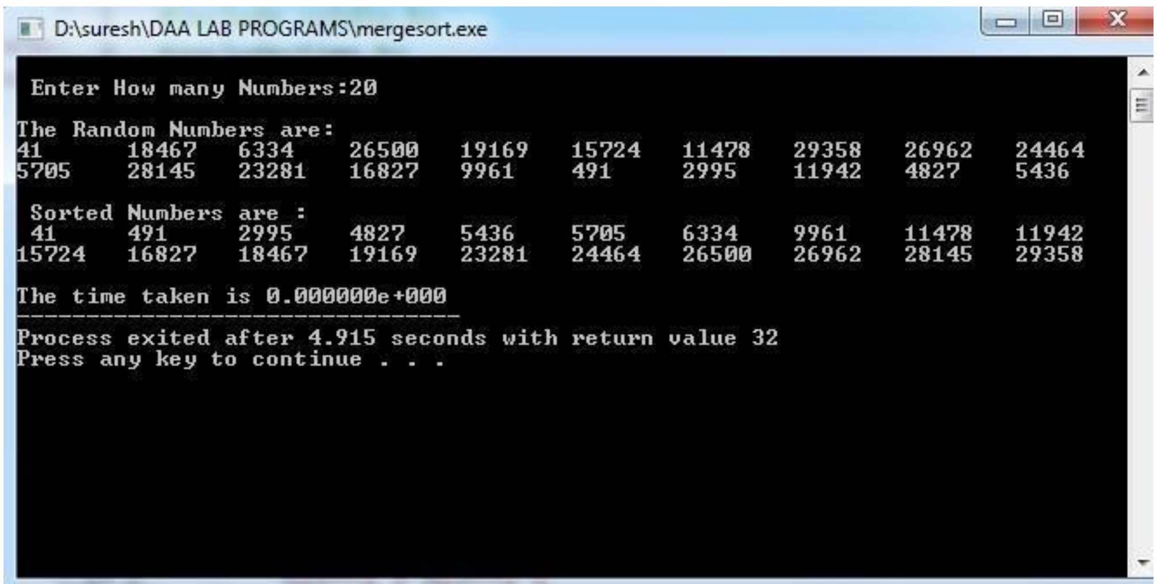
Implement merge sort algorithm to sort a given set of elements and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

PROGRAM

```
#include
<stdio.h> #include<time. h> int b[50000];
void Merge(int a[], int low, int mid, int high){ int i, j, k;
i=low; j=mid+1; k=low;
while ( i<=mid && j<=high ) { if( a[i] <= a[j] ) b[k++] = a[i++] ;

else

}


b[k++] = a[j++] ;
 while (i<=mid)
b[k++] = a[i++] ;
while (j<=high)
b[k++] = a[j++] ;
for(k=low; k<=high;
k++) a[k] =
b[k];

}
voidMergeSort(int a[], int low, int high)
{ int mid;
if(low >= high) return;
mid = (low+high)/2 ;
MergeSort(a, low, mid);
MergeSort(a, mid+1, high);
Merge(a, low, mid, high);
}
void main(){
int n, a[50000],k;
clock_tst,et; doublets;
printf("\n Enter How many Numbers:");
scanf("%d", &n); printf("\nThe Random Numbers are:\n");
for(k=1; k<=n; k++) {
a[k]=rand(); printf("%d\t", a[k]);
```

```
}
st=clock(); MergeSort(a, 1, n);
et=clock(); ts=(double)(et-
st)/CLOCKS_PER_SEC;
printf("\n Sorted Numbers are : \n ");
for(k=1; k<=n; k++)
printf("%d\t", a[k]);
printf("\nThe time taken is %e",ts);
}
```

OUTPUT

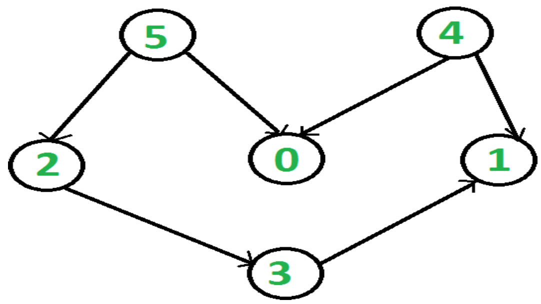


EXPERIMENT-3

OBJECTIVE

a.) Obtain the Topological ordering of vertices in a given digraph.

PROGRAM



Topological ordering

In topological sorting, a temporary stack is used with the name “s”. The node number is not printed immediately; first iteratively call topological sorting for all its adjacent vertices, then push adjacent vertex to stack. Finally, print contents of stack. Note that a vertex is pushed to stack only when all of its adjacent vertices (and their adjacent vertices and so on) are already in stack.

Transitive closure

Given a directed graph, find out if a vertex j is reachable from another vertex i for all vertex pairs (i, j) in the given graph. Here reachable mean that there is a path from vertex i to j. The reach- ability matrix is called transitive closure of a graph.

PROCEDURE:

1. Create: Open Dev C++, write a program after that save the program with .c extension.
2. Compile: Alt + F9
3. Execute: Ctrl + F10

SOURCE CODE:

```
// Topological ordering
#include<stdio.h>

int a[10][10],n,indegre[10]; voidfind_indegre (){

intj,i,sum; for(j=0;j<n;j++) {
sum=0; for(i=0;i<n;i
++)
sum+=a[i][j]; indegre[j]=sum;
}
}
void topology(){
inti,u,v,t[10],s[10],top=- 1,k=0; find_indegre(); for(i=0;i<n;i++){
if(indegre[i]==0) s[++top]=i;
}
while(top!=-1) {
u=s[top--];
t[k++]=u; //top element of stack is stored in temporary array for(v=0;v<n;v++){
if(a[u][v]==1){
indegre[v]--; if(indegre[v]==0)
s[++top]=v;     //Pushing adjacent vertex to stack
}
}
}
printf ("The topological Sequence is:\n"); for(i=0;i<n;i++)
printf ("%d ",t[i]);
}
void main(){
inti,j;
printf("Enter number of jobs:"); scanf("%d",&n);
printf("\nEnter the adjacency matrix:\n"); for(i=0;i<n;i++){
for(j=0;j<n;j++)
scanf("%d",&a[i][j]);
}
```

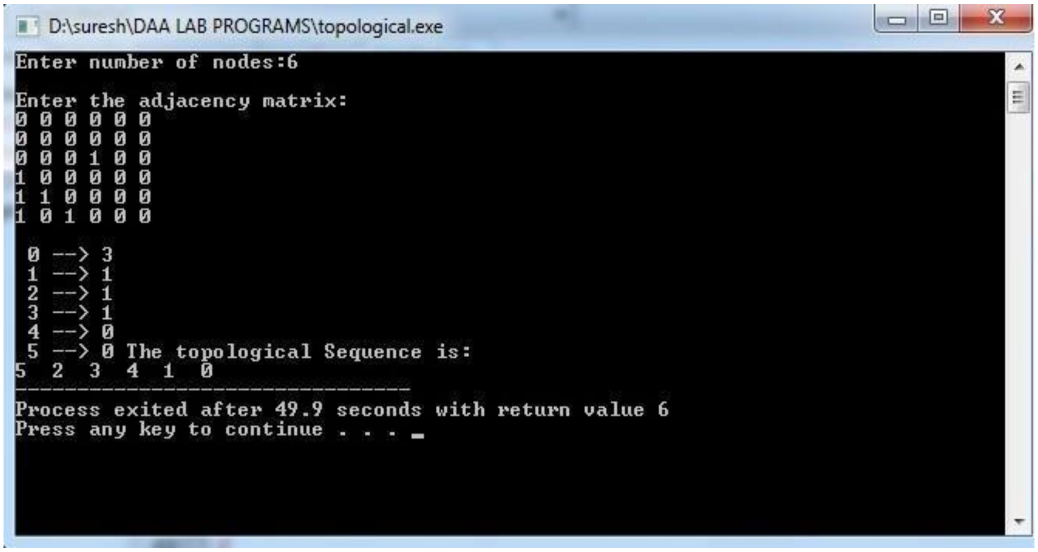
```
topology();
}
```

b.) Transitive closure of a graph using Warshall's algorithm

PROGRAM

```
#include <stdio.h> intn,a[10][10],p[10][10];
void path(){
inti,j,k; for(i=0;i<n;i
++)
for(j=0;j<n;j
++)
p[i][j]=a[i][j]; for(k=0;k<n;k++)
for(i=0;i<n;i++)
for(j=0;j<n;j++)
if(p[i][k]==1&&p[k][j]==1) p[i][j]=1;
}
void main(){ int i,j;
printf("Enter the number of nodes:"); scanf("%d",&n);
printf("\nEnter the adjacency matrix:\n"); for(i=0;i<n;i++)
for(j=0;j<n;j++)
scanf("%d",&a[i][j]);
path();
printf("\nThe path matrix is shown below\n"); for(i=0;i<n;i++){
for(j=0;j<n;j++)
printf("%d ",p[i][j]); printf("\n");
}
}
```

OUTPUT


D:\suresh\DAA LAB PROGRAMS\topological.exe
Enter number of nodes:6
Enter the adjacency matrix:
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 1 0 0
1 0 0 0 0 0
1 1 0 0 0 0
1 0 1 0 0 0
0 --> 3
1 --> 1
2 --> 1
3 --> 1
4 --> 0
5 --> 0 The topological Sequence is:
5 2 3 4 1 0

Process exited after 49.9 seconds with return value 6
Press any key to continue . . .

EXPERIMENT-4

OBJECTIVE

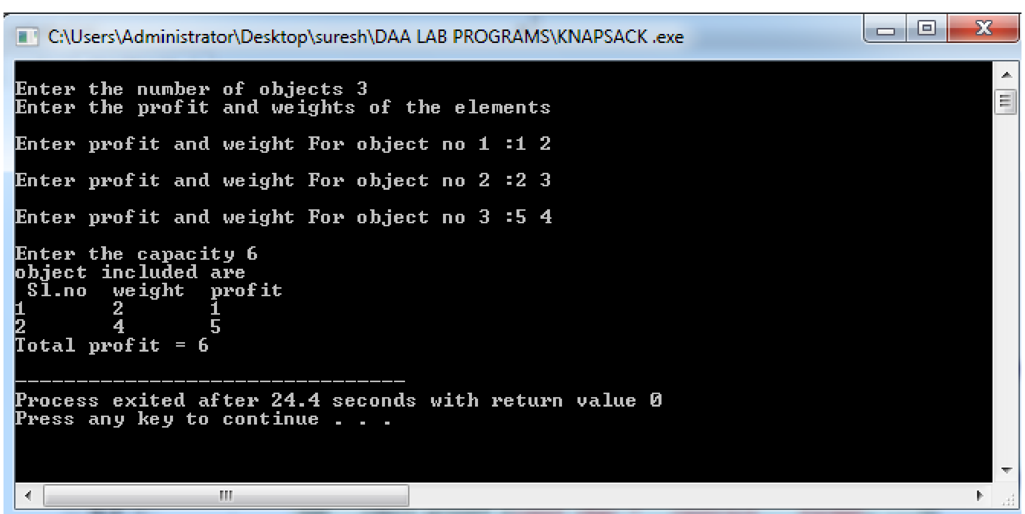
Implement 0/1 Knapsack problem using Dynamic Programming.

PROGRAM

```
#include<stdio.h>
int w[10],p[10],v[10][10],n,i,j,cap,x[10]={0};
int max(inti,int j){
return ((i>j)?i:j);
}
int knap(inti,int j){
int value; if(v[i][j]<0){
if(j<w[i])
value=knap(i-1,j);
else
value=max(knap(i-1,j),p[i]+knap(i-1,j-w[i]));
v[i][j]=value;
}
return(v[i][j]);
}
int main(){
intprofit,count=0;
printf("\nEnter the number of objects "); scanf("%d",&n);
printf("Enter the profit and weights of the elements
\n "); for(i=1;i<=n;i++){
printf("\nEnter profit and weight For object no %d :",i); scanf("%d%d",&p[i],&w[i]);
}
printf("\nEnter the capacity ");
scanf("%d",&ca p); for(i=0;i<=n;i+
+)
for(j=0;j<=cap;j
++)
if((i==0)||(j==0))
```

```
v[i][j]=0;

else

profit=knap(n,cap); i=n;
j=cap; while(j!=0&&i!=0){
v[i][j]=-1;
 if(v[i][j]!=v[i-1][j]){
x[i]=1;
j=j-w[i];
i--;
}
else
}
 i--;
 printf("object included are
\n        ");
printf("Sl.no\tweight\tprofit\ n"); for(i=1;i<=n;i++)
if(x[i])
printf("%d\t%d\t%d\n",++count,w[i],p[i])
; printf("Total profit = %d\n",profit);
}
```

OUTPUT

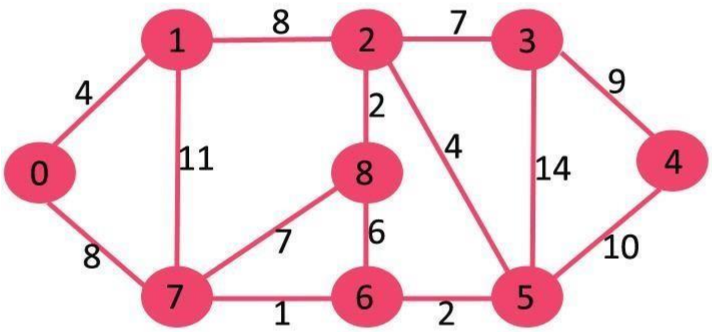


EXPERIMENT-5

OBJECTIVE

From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.

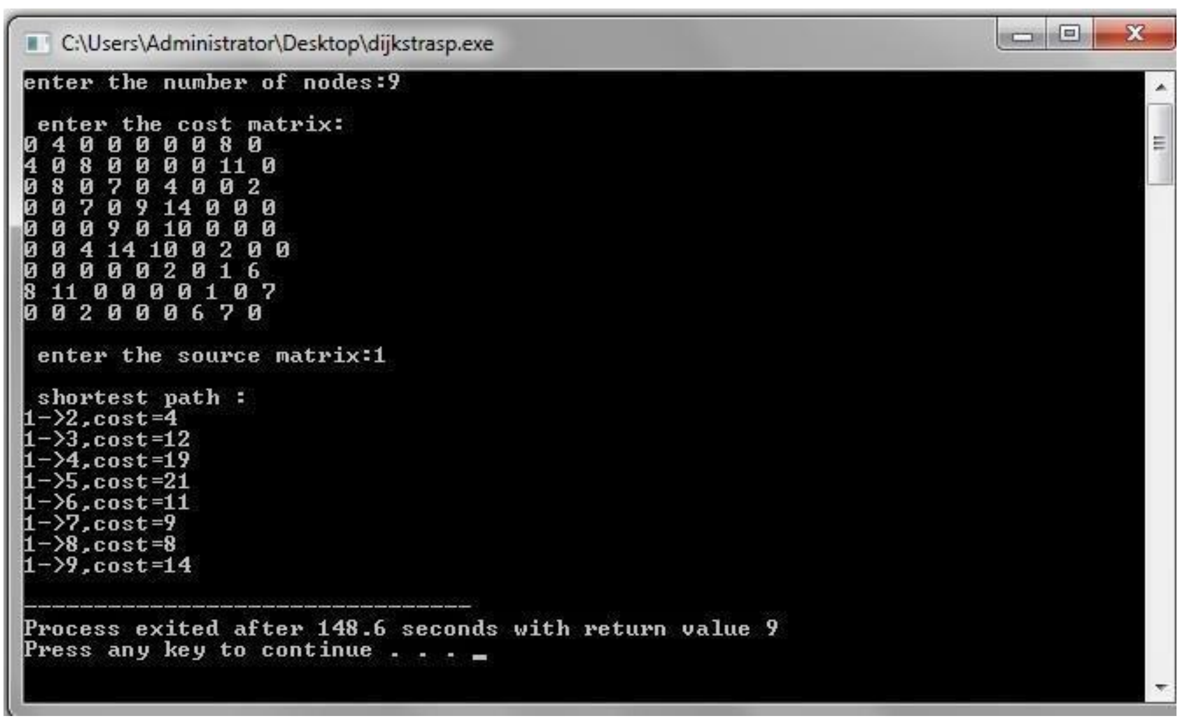
PROGRAM




```
#include<stdio. h>
#define infinity 999

void dij(int n, int v,int cost[20][20], int dist[]){

int i,u,count,w,flag[20],min; for(i=1;i<=n;i++)
flag[i]=0, dist[i]=cost[v][i]; count=2; while(count<=n){
min=99; for(w=1;w<=n;w++)
if(dist[w]<min && !flag[w])  { min=dist[w]; u=w;
}
flag[u
]=1;
count
++;
for(w=1;w<=n;w++)
if((dist[u]+cost[u][w]<dist[w]) &&
!flag[w]) dist[w]=dist[u]+cost[u][w];
}
```

```
}
int main(){
int n,v,i,j,cost[20][20],dist[20]; printf("enter the number of nodes:"); scanf("%d",&n);
printf("\n enter the cost matrix:\n"); for(i=1;i<=n;i++)
for(j=1;j<=n;j++){
scanf("%d",&cost[i][j]); if(cost[i][j] == 0)
cost[i][j]=infinity;
}
printf("\n enter the source matrix:"); scanf("%d",&v); dij(n,v,cost,dist);
printf("\n shortest path : \n"); for(i=1;i<=n;i++)
if(i!=v)
printf("%d->%d,cost=%d\n",v,i,dist[i]);
}
```

OUTPUT

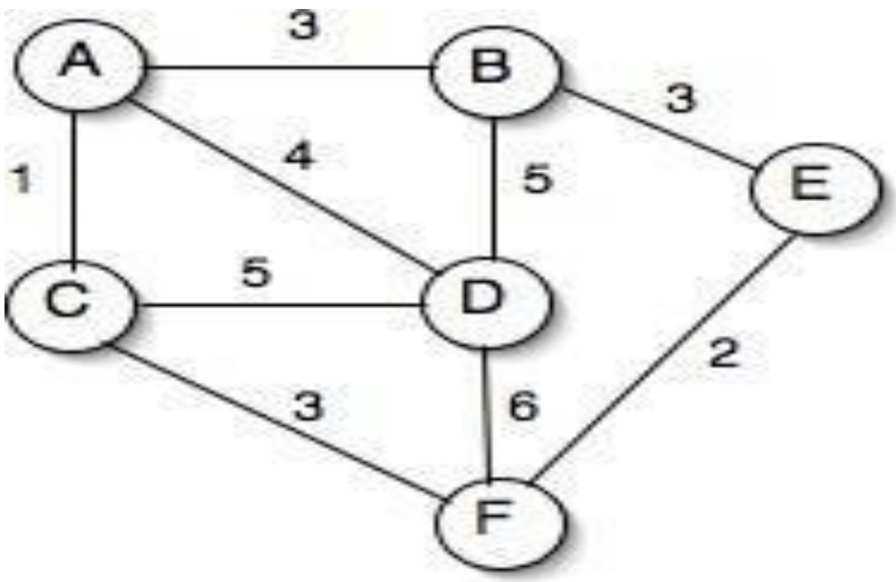


EXPERIMENT-6

OBJECTIVE

Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm.

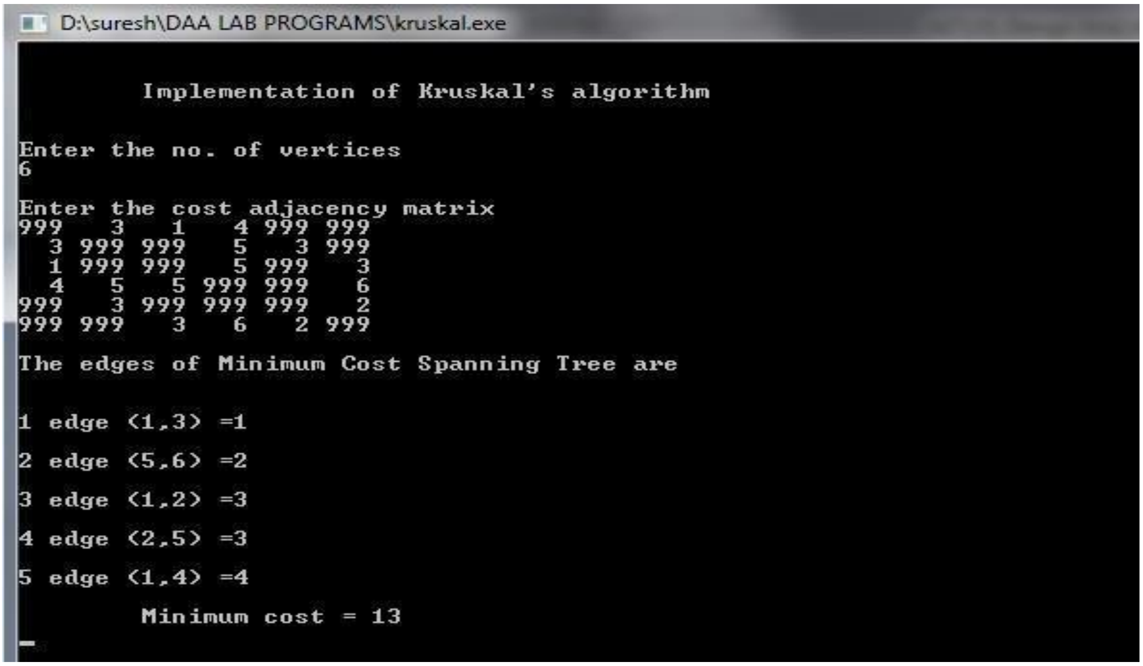
PROGRAM




```
#include<stdio.h> #include<stdlib.h> inti,j,k,a,b,u,v,n,ne=1;
intmin,mincost=0,cost[9][9],parent [9]; int find(int);
intuni(int,int
); void main() {
printf("\n Implementation of Kruskal's algorithm\n\n"); printf("\nEnter the no. of vertices\n");
scanf("%d",&n);
printf("\nEnter the cost adjacency matrix\n"); for(i=1;i<=n;i++){
for(j=1;j<=n;j++) { scanf("%d",&cost[i][j]); if(cost[i][j]==0)
cost[i][j]=999;
}
}
printf("\nThe edges of Minimum Cost Spanning Tree are\n\n"); while(ne<n){
for(i=1,min=999;i<=n;i++) { for(j=1;j<=n;j++){
if(cost[i][j]<min){
```

```
min=cost[i
][j]; a=u=i; b=v=j;
}
}
}
u=find(u
);
v=find( v); if(uni(u,
v)){
printf("\n%d edge (%d,%d)
=%d\n",ne++,a,b,min); mincost +=min;
}
cost[a][b]=cost[b][a]=999;
}
printf("\n\tMinimum cost = %d\n",mincost);
}
int find(int i){
while(parent[i])
i=parent[i];
return i;
}
 intuni(inti,int j){
if(i!=j) {
}
 parent[j]=i; return 1;
 return 0;
}
```

OUTPUT

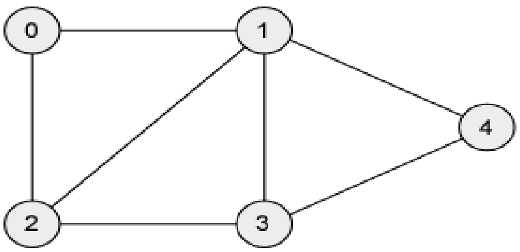

D:\suresh\DAA LAB PROGRAMS\kruskal.exe
Implementation of Kruskal's algorithm
Enter the no. of vertices
6
Enter the cost adjacency matrix
999 3 1 4 999 999
3 999 999 5 3 999
1 999 999 5 999 3
4 5 5 999 999 6
999 3 999 999 999 2
999 999 3 6 2 999
The edges of Minimum Cost Spanning Tree are
1 edge (1,3) =1
2 edge (5,6) =2
3 edge (1,2) =3
4 edge (2,5) =3
5 edge (1,4) =4
Minimum cost = 13

EXPERIMENT-7

OBJECTIVE

a.) Print all the nodes reachable from a given starting node in a digraph using BFS method.

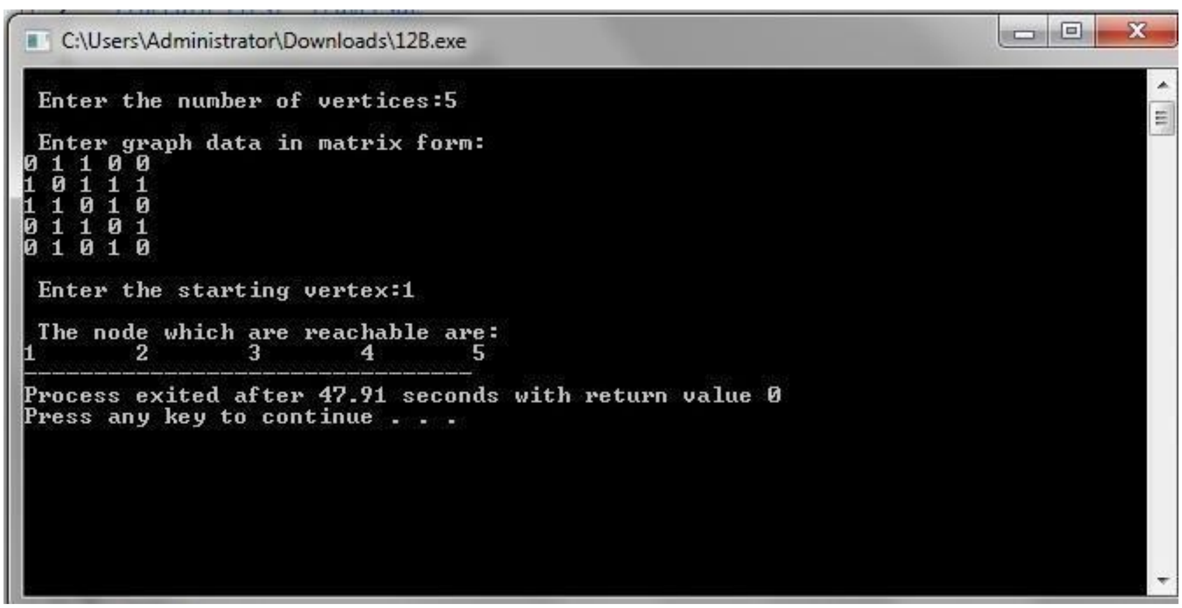
PROGRAM




```
#include<stdio.h> #include<conio.h>
int a[20][20],q[20],visited[20],n,i,j,f=- 1,r=0; voidbfs(int v){
q[++r]=v; visited[v]
=1;
while(f<
=r) {
for(i=1;i<=n;i++)
if(a[v][i] && !visited[i]){
visited[i]=1; q[++r]=i;
 }
void main(){
int v;
 }
f++;
v=q[f];
}
 printf("\n Enter the number of vertices:"); scanf("%d",&n);
for(i=1;i<=n;i++){
q[i]=0;
visited[i]=0;
```

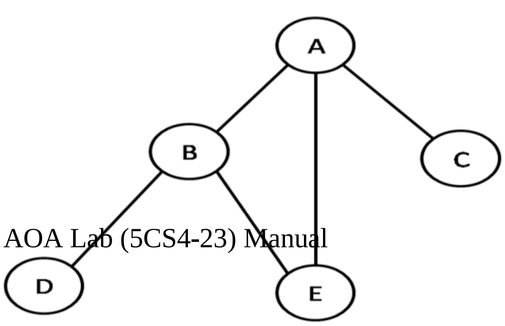
```
}
printf("\n Enter graph data in matrix form:\n"); for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
scanf("%d",&a[i] [j]); printf("\n Enter the starting
vertex:"); scanf("%d",&v);
bfs(v);
printf("\n The node which are reachable are:\n"); for(i=1;i<=n;i++) if(visited[i])
printf("%d\t",q[i]);
 else
}
 printf("\n Bfs is not possible");
```

OUTPUT



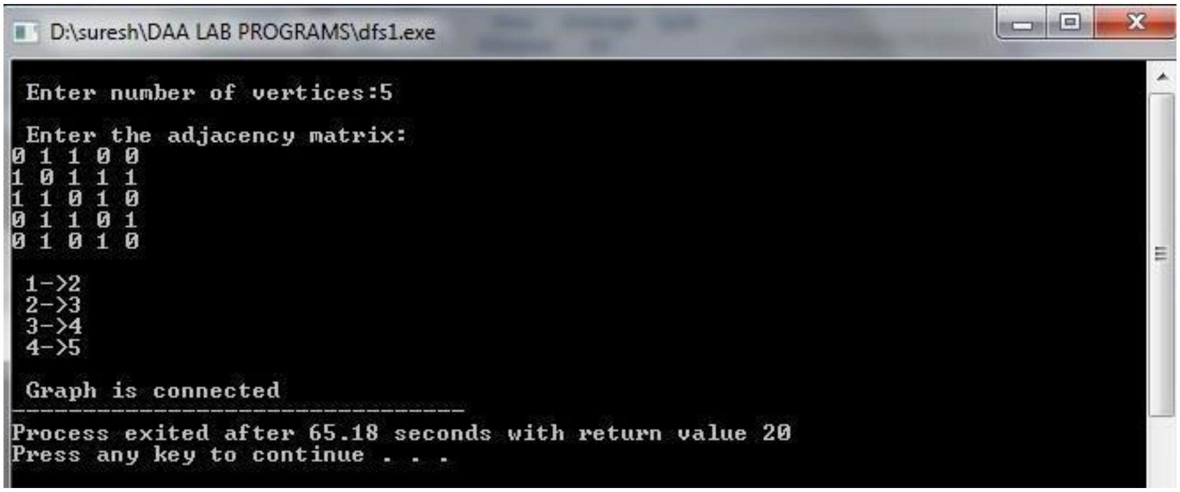
b.) Check whether a given graph is connected or not using DFS method.

PROGRAM



```
#include<stdio.h> #include<conio.h> int
a[20][20],reach[20],n; void dfs(int v){
int i; reach[v]=1; for(i=1;i<=n;i++) if(a[v][i] &&
!reach[i]) {
printf("\n %d-
>%d",v,i); dfs(i);
}
}
void main(){
int i,j,count=0;
printf("\n Enter number of vertices:"); scanf("%d",&n);
for(i=1;i<=n;i++){
reach[i]=0; for(j=1;j<=n;j
++) a[i][j]=0;
}
printf("\n Enter the adjacency matrix:\n"); for(i=1;i<=n;i++) for(j=1;j<=n;j++)
scanf("%d",&a[i][j]);
dfs(1); printf("\n"); for(i=1;i<=n;i
++){
if(reach[i]) count++;
}
if(count==n)
printf("\n Graph is connected");

else
}
printf("\n Graph is not connected");
```

OUTPUT


D:\suresh\DAA LAB PROGRAMS\dfs1.exe
Enter number of vertices:5
Enter the adjacency matrix:
0 1 1 0 0
1 0 1 1 1
1 1 0 1 0
0 1 1 0 1
0 1 0 1 0
1->2
2->3
3->4
4->5
Graph is connected

Process exited after 65.18 seconds with return value 20
Press any key to continue . . .

EXPERIMENT-8

OBJECTIVE

Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm.

PROGRAM

```
#include <stdio.h>
#include <limits.h>

#define V 5

int minKey(int key[], int mstSet[]) {
    int min = INT_MAX, min_index;
    int v;
    for (v = 0; v < V; v++)
        if (mstSet[v] == 0 && key[v] < min)
            min = key[v], min_index = v;

    return min_index;
}

int printMST(int parent[], int n, int graph[V][V]) {
    int i;
    printf("Edge   Weight\n");
    for (i = 1; i < V; i++)
        printf("%d - %d    %d \n", parent[i], i, graph[i][parent[i]]);
}

void primMST(int graph[V][V]) {
    int parent[V]; // Array to store constructed MST
    int key[V], i, v, count; // Key values used to pick minimum weight edge in cut
    int mstSet[V]; // To represent set of vertices not yet included in MST

    // Initialize all keys as INFINITE
    for (i = 0; i < V; i++)
```

```
        key[i] = INT_MAX, mstSet[i] = 0;

    // Always include first 1st vertex in MST.
    key[0] = 0; // Make key 0 so that this vertex is picked as first vertex
    parent[0] = -1; // First node is always root of MST

    // The MST will have V vertices
    for (count = 0; count < V - 1; count++) {
        int u = minKey(key, mstSet);
        mstSet[u] = 1;

        for (v = 0; v < V; v++)

            if (graph[u][v] && mstSet[v] == 0 && graph[u][v] < key[v])
                parent[v] = u, key[v] = graph[u][v];
    }

    // print the constructed MST
    printMST(parent, V, graph);
}

int main() {
    /* Let us create the following graph
     2    3
    (0)--(1)--(2)
     |   / \   |
    6| 8/   \5 |7
     | /     \ |
    (3)-------(4)
     9          */
    int graph[V][V] = { { 0, 2, 0, 6, 0 }, { 2, 0, 3, 8, 5 },
            { 0, 3, 0, 0, 7 }, { 6, 8, 0, 0, 9 }, { 0, 5, 7, 9, 0 }, };

    primMST(graph);
```

```
    return 0;
}
```

OUTPUT

```
$ gcc PrimsMST.c
$ ./a.out

Edge   Weight
0 - 1    2
1 - 2    3
0 - 3    6
1 - 4    5
```