

**RAJASTHAN INSTITUTE OF ENGINEERING AND TECHNOLOGY, JAIPUR**

**Lab Record**

**Analysis of Algorithms Lab**

**Submitted To: Submitted By:**

Mr. Dinesh Swami Vicky Kumar

(Assistant Prof.) Roll No. 18ERECS080

CSE (3rd year)



RAJASTHAN TECHNICAL UNIVERSITY, KOTA

**Syllabus**

**III Year-V Semester: B.Tech. Computer Science and Engineering**

**5CS4-23: Analysis of Algorithms Lab**

**Credit: 1 Max. Marks:50 (IA:30, ETE:20)**

**0L+0T+2P End Term Exam: 2 Hours**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **S.no** | **Name of Experiment** | **Date** | **Attenance(10)** | **Perfor-**  **mance**  **(10)** | **Rec**  **ord**  **(10)** | **Viva**  **(20)** | **Total**  **(50)** |
| 1. | 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. |  |  |  |  |  |  |
| 2. | Implement a parallelized 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. |  |  |  |  |  |  |
| 3. | a. Obtain the Topological ordering of vertices in a given digraph. b. Compute the transitive closure of a given directed graph using Warshall's algorithm. |  |  |  |  |  |  |
| 4. | Implement 0/1 Knapsack problem using Dynamic Programming. |  |  |  |  |  |  |
| 5. | From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm. |  |  |  |  |  |  |
| 6. | Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm. |  |  |  |  |  |  |
| 7. | a. Print all the nodes reachable from a given starting node in a digraph using BFS method. b. Check whether a given graph is connected or not using DFS method. |  |  |  |  |  |  |
| 8. | Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s algorithm. |  |  |  |  |  |  |
| 9. | Implement All-Pairs Shortest Paths Problem using Floyd's algorithm. |  |  |  |  |  |  |
| 10. | Implement N Queen's problem using Back Tracking. |  |  |  |  |  |  |

**Experiment – 1**

**Objective:**

Sort a given set of elements using the Quick sort 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.

**Resources:**

Dev C++

**Program logic:**

Quick Sort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot.

There are many different versions of Quick Sort that pick pivot in different ways.

1. Always pick the first element as pivot.

2. Always pick last element as pivot (implemented below)

3. Pick a random element as pivot.

4. Pick median as pivot.

The key process in Quick Sort is partition. Target of partitions is, given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x.

**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:**

#include <stdio.h>

#include <time.h>

#include<stdlib.h>

void Exch(int \*p, int \*q)

{

int temp = \*p;

\*p = \*q;

\*q = temp;

}

void QuickSort(int a[], int low, int high)

{

int i, j, key, k;

if(low>=high)

return;

key=low;

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, j-1);

QuickSort(a, j+1, high);

}

main()

{

int n, a[1000],k;

int clock\_t,st,et;

double ts;

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 ");

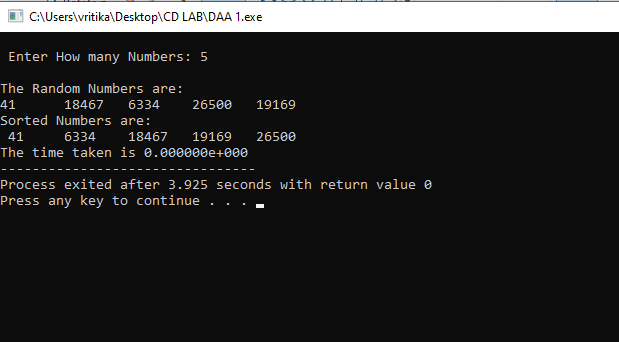
for(k=1; k<=n; k++)

printf("%d\t", a[k]);

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

}

**Output:**



**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.

**Resources:**

Dev C++

**Program logic:**

Merge Sort is a Divide and Conquer algorithm. It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves.

The merge() function is used for merging two halves. The merge(a, low, mid, high) is key process that assumes that a[low..mid] and a[mid+1..high] are sorted and merges the two sorted sub-arrays into one.

**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:**

#include <stdio.h>

#include<time.h>

#include<stdlib.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++] ;

}

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];

}

void MergeSort(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);

}

main()

{

int n, a[50000],k;

int clock\_tst,ts,st;

double et;

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

scanf("%d", &n);

printf("\n The 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 ");

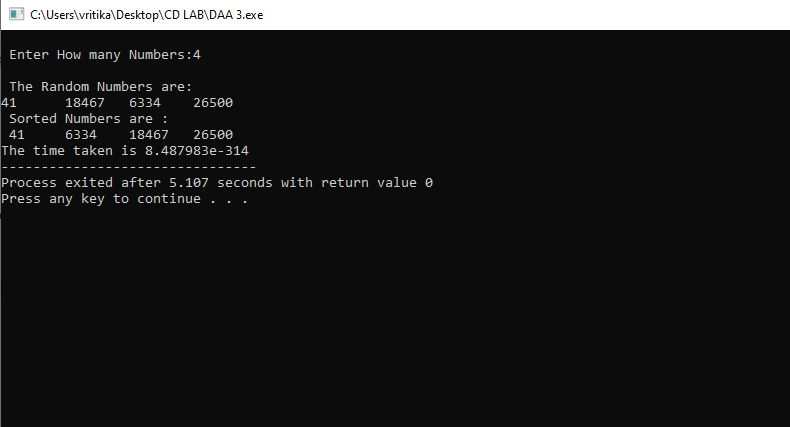
for(k=1; k<=n; k++)

printf("%d\t", a[k]);

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

}

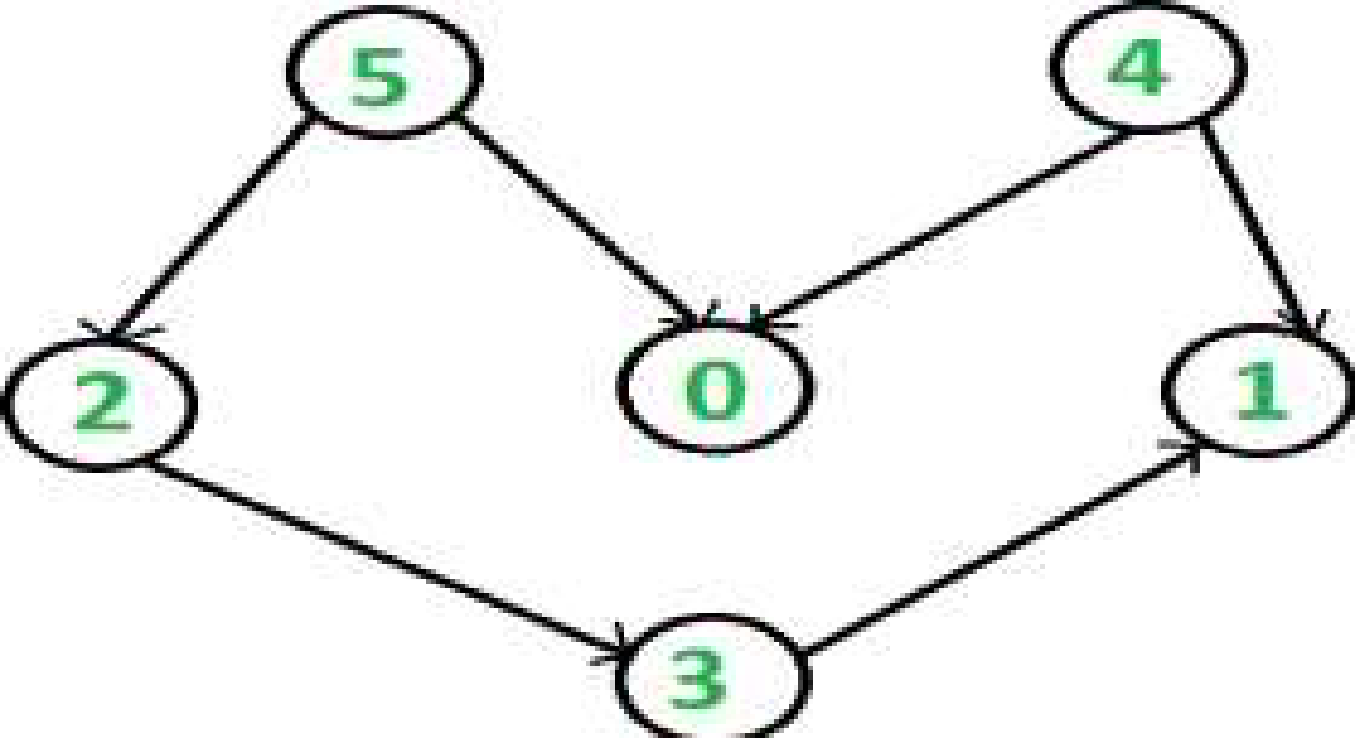
**Output:**



**Experiment – 3**

**Objective:**

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



2. Compute the transitive closure of a given directed graph using Warshall's algorithm

**Resources:**

Dev C++

**Program logic:**

**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];

void find\_indegre ()

{

int j,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()

{

int i,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]);

}

main()

{

int i,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]);

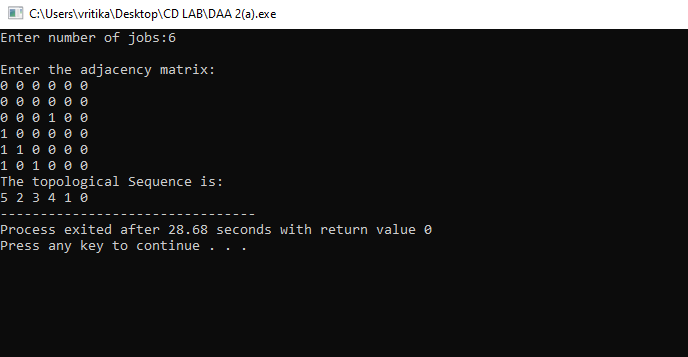
}

topology();

}

**Input / Output:**

**Topological ordering**



**2. //Transitive closure of a graph using Warshall's algorithm**

#include <stdio.h>

int n,a[10][10],p[10][10];

void path()

{

int i,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;

}

main()

{

int i,j,k;

for(j=0;j<n;j++)

if(p[i][k]==1&&p[k][j]==1)

p[i][j]=1;

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]);

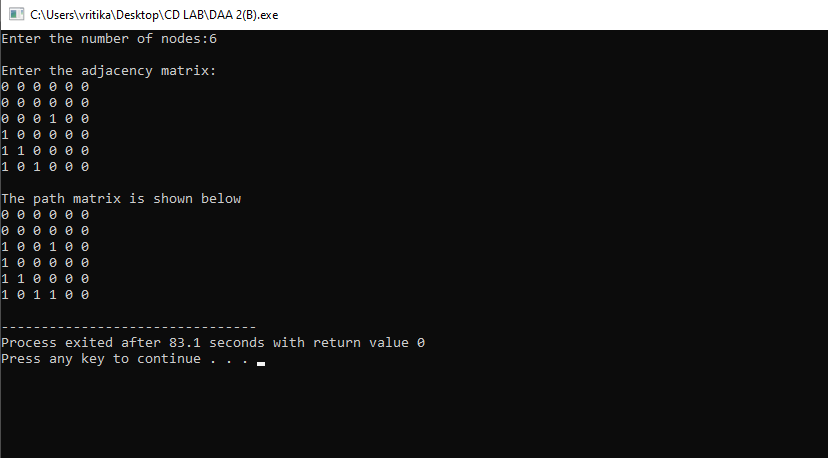
printf("\n");

}

}

**Input / Output:**

**Transitive closure**



**Experiment – 4**

**Objective:**

Implement 0/1 Knapsack problem using Dynamic Programming.

**Resources:**

Dev C++

**Program logic:**

Given some items, pack the knapsack to get the maximum total profit. Each item has some

Weight and some profit. Total weight that we can carry is no more than some fixed number W.

**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:**

#include<stdio.h>

int w[10],p[10],v[10][10],n,i,j,cap,x[10]={0};

int max(int i,int j)

{

return ((i>j)?i:j);

}

int knap(int i,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()

{

int profit,count=0;

printf("\nEnter the number of objects: ");

scanf("%d",&n);

printf("\nEnter 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",&cap);

for(i=0;i<=n;i++)

for(j=0;j<=cap;j++)

if((i==0)||(j==0))

v[i][j]=0;

else

v[i][j]=-1;

profit=knap(n,cap);

i=n;

j=cap;

while(j!=0&&i!=0)

{

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");

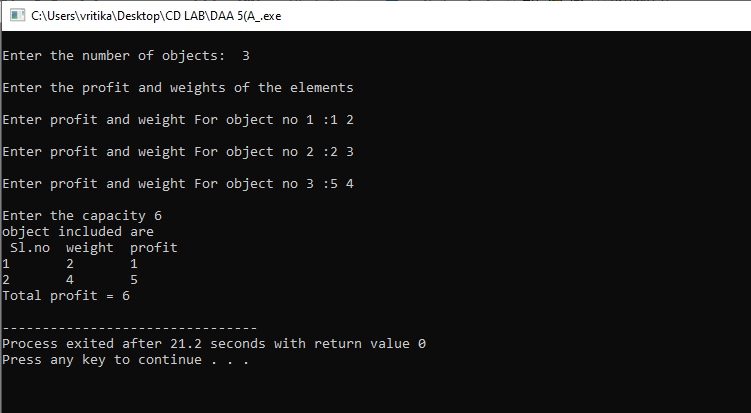
for(i=1;i<=n;i++)

if(x[i])

printf("%d\t%d\t%d\n",++count,w[i],p[i]);

printf("Total profit = %d\n",profit);

}

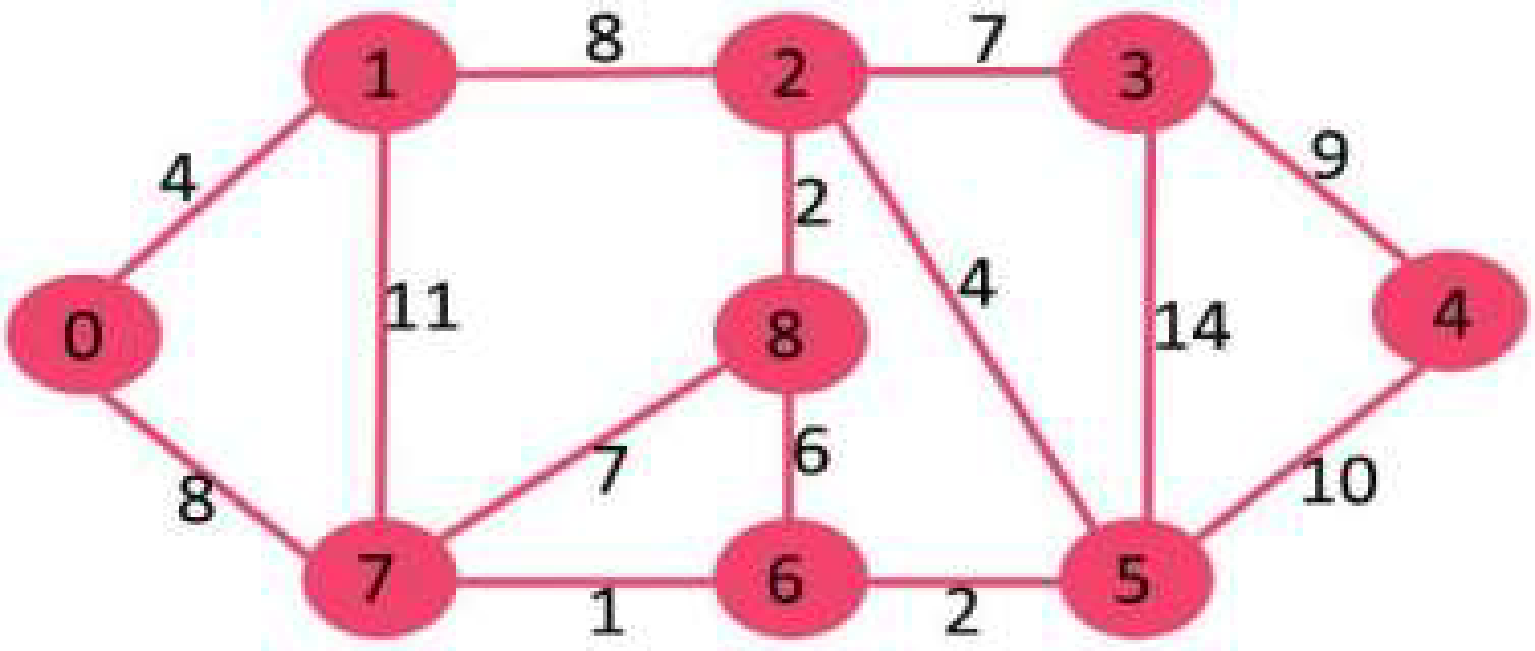
**Output:**

**Experiment – 5**

**Objective:**

From a given vertex in a weighted connected graph, find shortest paths to other vertices using

Dijkstra’s algorithm.



**Resource:**

Dev C++

**Program logic:**

**1)** Create a set S that keeps track of vertices included in shortest path tree, i.e., whose minimum distance from source is calculated and finalized. Initially, this set is empty.

**2)** Assign a distance value to all vertices in the input graph. Initialize all distance values as

INFINITE. Assign distance value as 0 for the source vertex so that it is picked first.

**3)** While *S* doesn’t include all vertices

**a)** Pick a vertex u which is not there in *S* and has minimum distance value.

**b)** Include u to *S*.

**c)** Update distance value of all adjacent vertices of u.

To update the distance values, iterate through all adjacent vertices. For every adjacent vertex v, if sum of distance value of u (from source) and weight of edge u-v, is less than the distance value of v, then update the distance value of v.

**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:**

#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");

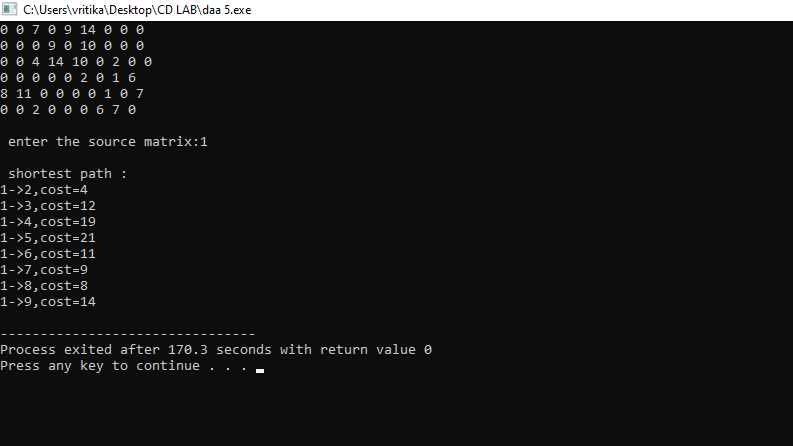
for(i=1;i<=n;i++)

if(i!=v)

printf("%d->%d,cost=%d\n",v,i,dist[i]);

}

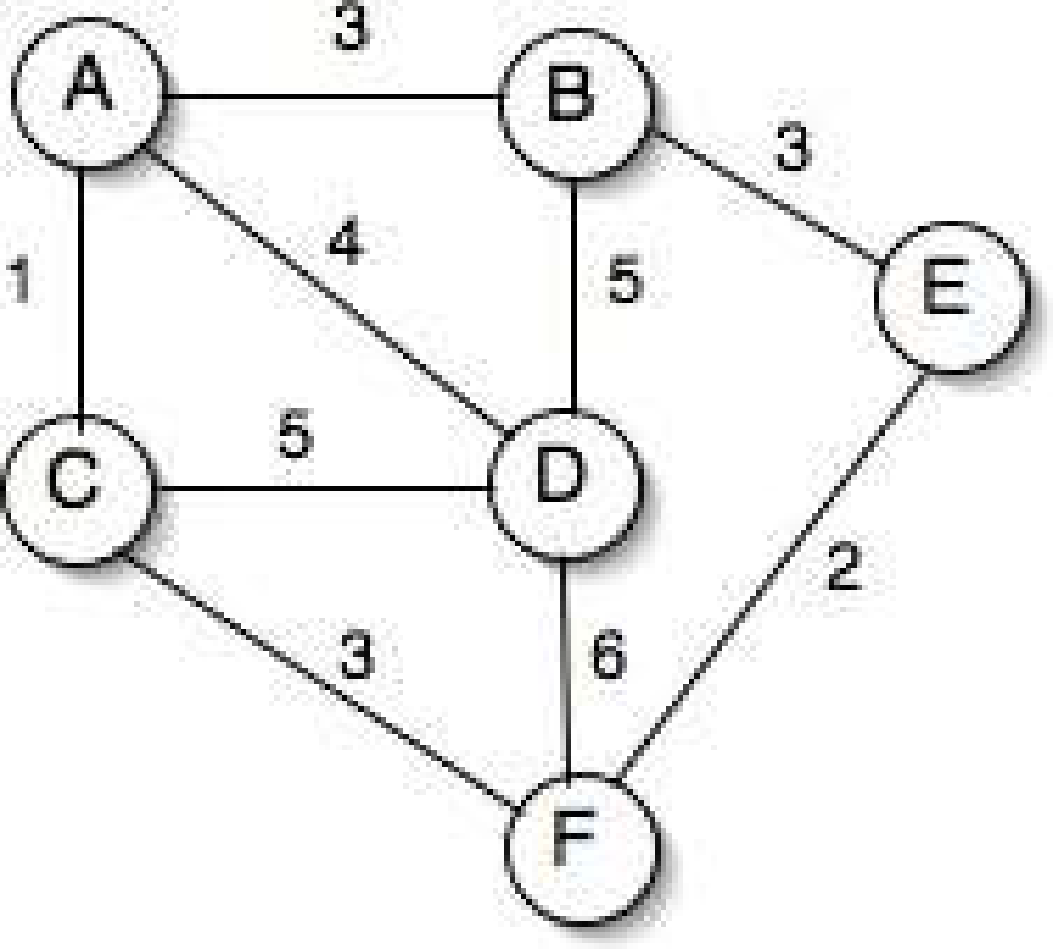
**Output:**



**Experiment – 6**

**Objective:**

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



**Resources:**

Dev C++

**Program logic:**

**1.** Sort all the edges in non-decreasing order of their weight.

**2.** Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.

**3.** Repeat step#2 until there are (V-1) edges in the spanning tree.

**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:**

#include<stdio.h>

#include<stdlib.h>

int i,j,k,a,b,u,v,n,ne=1;

int min,mincost=0,cost[9][9],parent[9];

int find(int);

int uni(int,int);

main()

{

printf("\n Implementation of Kruskal's algorithm\n\n");

printf("\nEnter the no. of vertices: ");

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;

}

int uni(int i,int j)

{

if(i!=j)

{

parent[j]=i;

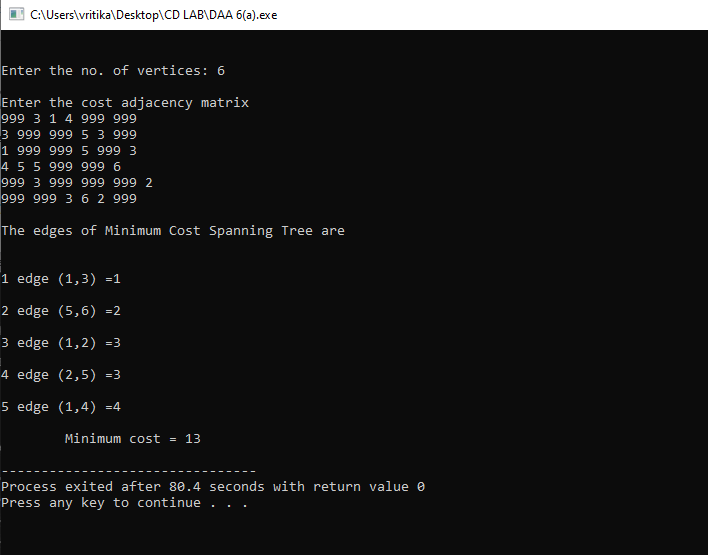
return 1;

}

return 0;

}

**Output:**



**Experiment – 7**

**Objective:**

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











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



**Resources:**

Dev C++

**Program logic:**

**Breadth first traversal**

Breadth First Search (BFS) algorithm traverses a graph in a breadth ward motion and uses a

queue to remember to get the next vertex to start a search.

1. Visit the adjacent unvisited vertex. Mark it as visited. Display it. Insert it in a queue.

2. If no adjacent vertex is found, remove the first vertex from the queue.

3. Repeat Rule 1 and Rule 2 until the queue is empty.

**Depth first traversal**

Depth First Search (DFS) algorithm traverses a graph in a depth ward motion and uses a stack to

remember to get the next vertex to start a search.

1. Visit the adjacent unvisited vertex. Mark it as visited. Display it. Push it in a stack.

2. If no adjacent vertex is found, pop up a vertex from the stack. (It will pop up all the vertices

from the stack, which do not have adjacent vertices.)

3. Repeat Rule 1 and Rule 2 until the stack is empty.

**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:**

**//Breadth first traversal**

#include<stdio.h>

#include<conio.h>

int a[20][20],q[20],visited[20],n,i,j,f=-1,r=0;

void bfs(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;

}

f++;

v=q[f];

}

}

void main()

{

int v;

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]);

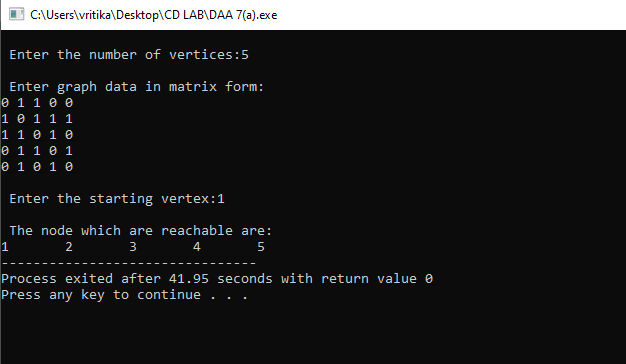
else

printf("\n Bfs is not possible");

}

**Input/Output:**

**Breadth first traversal**

****

**# Depth first traversal**

#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);

}

}

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");

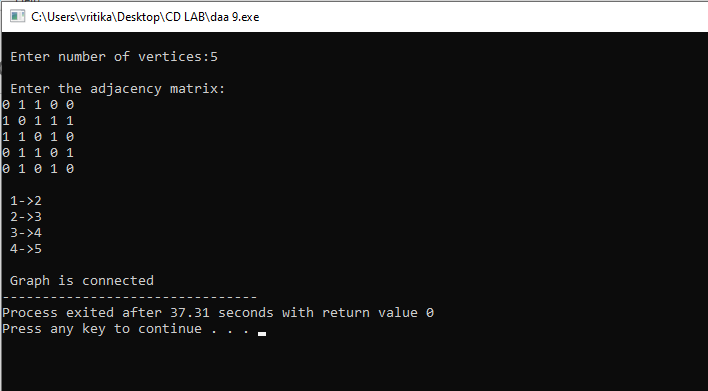
else

printf("\n Graph is not connected");

}

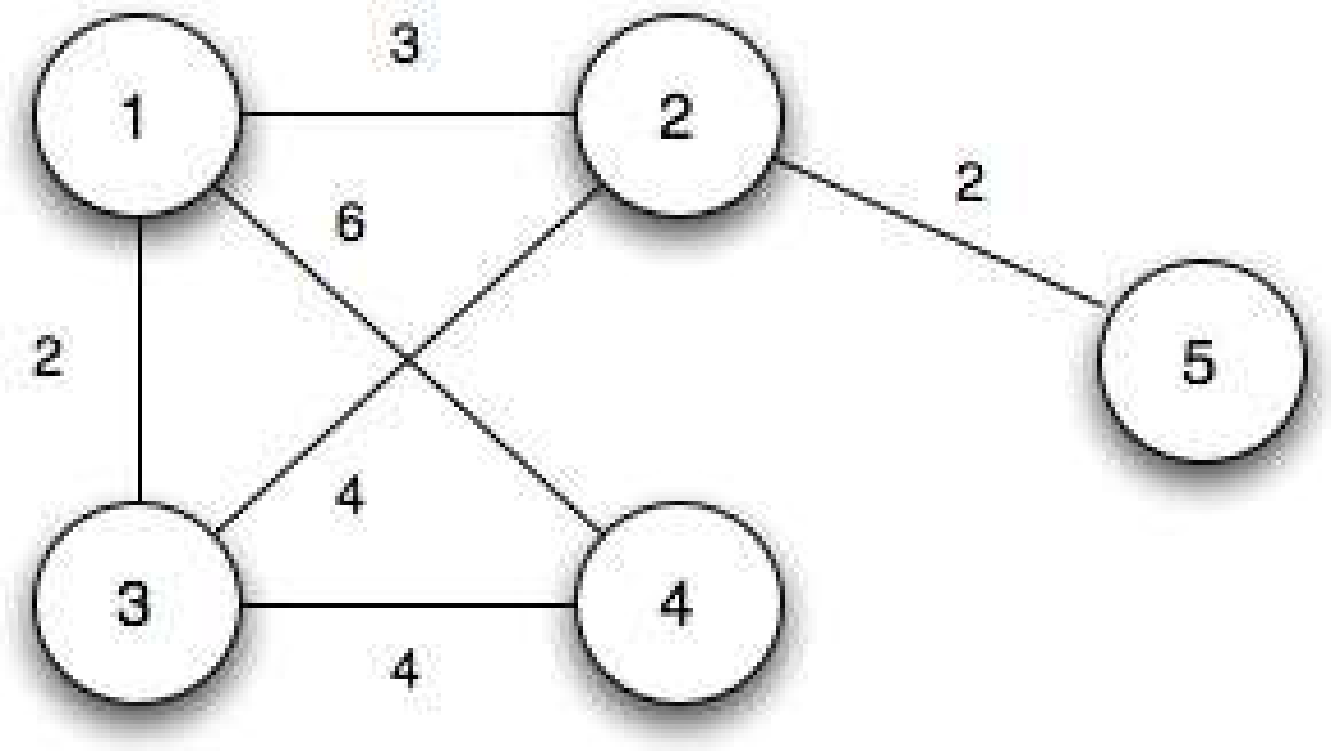
**Input/Output:**

**Depth first traversal**



**Experiment – 8**

**Objective:** Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s algorithm.



**Resources:**

Dev C++

**Program logic:**

**1)** Create a set Sthat keeps track of vertices already included in MST.

**2)** Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE.

Assign key value as 0 for the first vertex so that it is picked first.

**3)** While S doesn’t include all vertices.

**a)** Pick a vertex *u* which is not there in Sand has minimum key value.

**b)** Include *u* to S.

**c)** Update key value of all adjacent vertices of *u*.

To update the key values, iterate through all adjacent vertices. For every adjacent vertex *v*, if

weight of edge *u-v* is less than the previous key value of *v*, update the key value as weight of *u-v.*

The idea of using key values is to pick the minimum weight edge from cut. The key values are used only for vertices which are not yet included in MST, the key value for these vertices indicate the minimum weight edges connecting them to the set of vertices included in MST.

**Procedures:**

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:**

#include<stdio.h>

int a,b,u,v,n,i,j,ne=1;

int visited[10]={0},min,mincost=0,cost[10][10];

main()

{

printf("\n Enter the number of nodes:");

scanf("%d",&n);

printf("\n Enter the 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;

}

visited[1]=1;

printf("\n");

while(ne<n)

{

for(i=1,min=999;i<=n;i++)

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

if(cost[i][j]<min)

if(visited[i]!=0)

{

min=cost[i][j];

a=u=i;

b=v=j;

}

if(visited[u]==0 || visited[v]==0)

{

printf("\n Edge %d:(%d %d) cost:%d",ne++,a,b,min);

mincost+=min;

visited[b]=1;

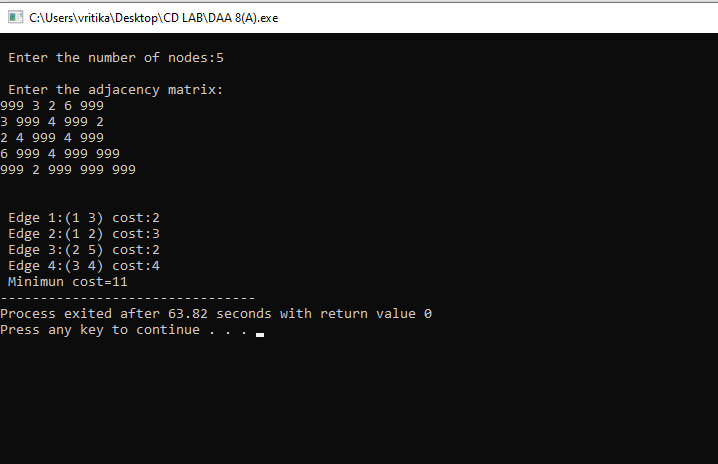
}

cost[a][b]=cost[b][a]=999;

}

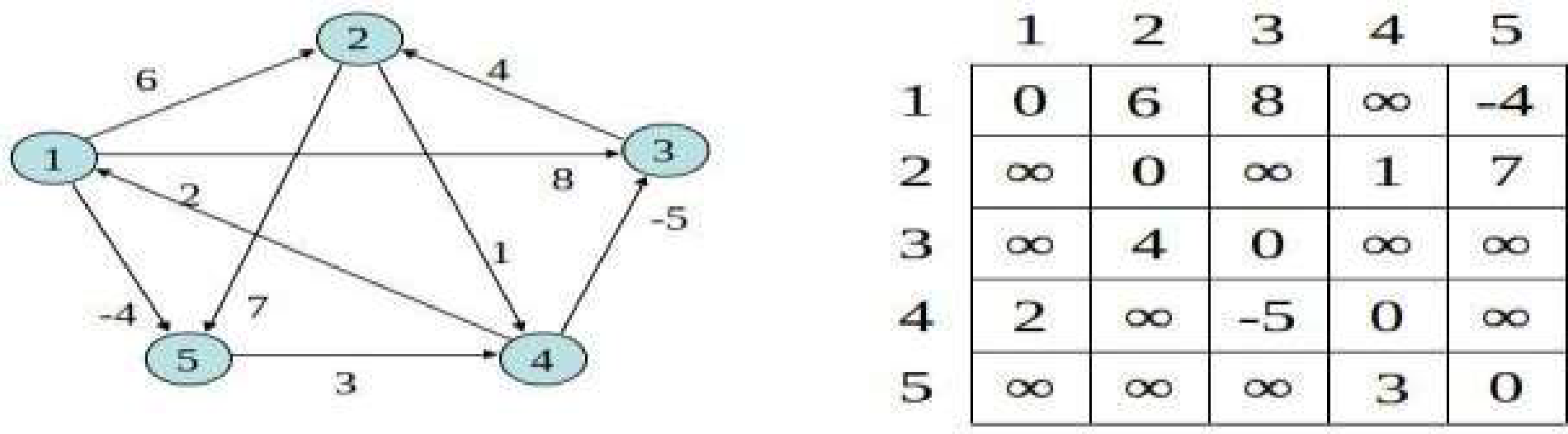
printf("\n Minimun cost=%d",mincost);

}

**Input/ Output:**

**Experiment – 9**

**Objective:** Implement All-Pairs Shortest Paths Problem using Floyd's algorithm.



**Resources:**

Dev C++

**Program Logic:**

Initialize the solution matrix same as the input graph matrix as a first step. Then we update the solution matrix by considering all vertices as an intermediate vertex. The ideas is to one by one pick all vertices and update all shortest paths which include the picked vertex as an intermediate vertex in the shortest path.

When we pick vertex number k as an intermediate vertex, we already have considered vertices {0, 1, 2, .. k-1} as intermediate vertices.

For every pair (i, j) of source and destination vertices respectively, there are two possible cases.

**1)** k is not an intermediate vertex in shortest path from i to j. We keep the value of dist[i][j] as it is.

**2)** k is an intermediate vertex in shortest path from i to j. We update the value of dist[i][j] as

dist[i][k] + dist[k][j].

**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:**

#include<stdio.h>

int min(int,int);

void floyds(int p[10][10],int n)

{

int i,j,k;

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

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

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

if(i==j)

p[i][j]=0;

else

p[i][j]=min(p[i][j],p[i][k]+p[k][j]);

}

int min(int a,int b){

if(a<b)

return(a);

else

return(b);

}

main()

{

int p[10][10],w,n,e,u,v,i,j;

printf("\n Enter the number of vertices:");

scanf("%d",&n);

printf("\n Enter the number of edges: ");

scanf("%d",&e);

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

{

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

p[i][j]=999;

}

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

{

printf("\n Enter the end vertices of edge%d with its weight: ",i);

scanf("%d%d%d",&u,&v,&w);

p[u][v]=w;

}

printf("\n Matrix of input data:\n");

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

{

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

printf("%d \t",p[i][j]);

printf("\n");

}

floyds(p,n);

printf("\n Transitive closure:\n");

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

{

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

printf("%d \t",p[i][j]);

printf("\n");

}

printf("\n The shortest paths are:\n");

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

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

{

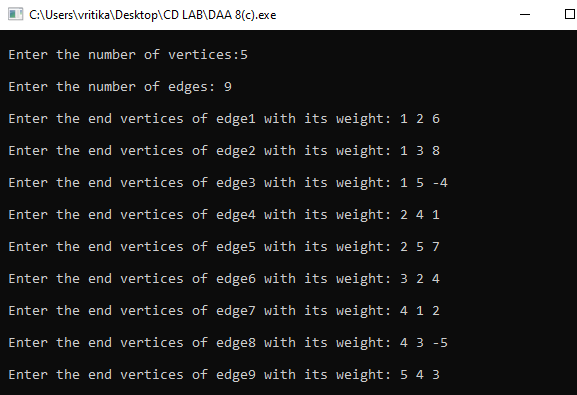
if(i!=j)

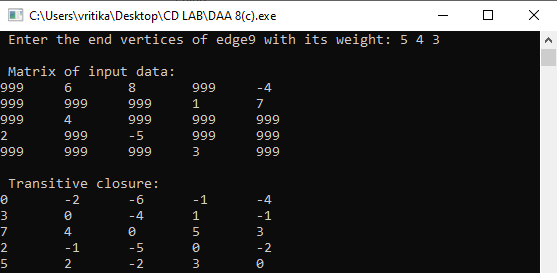
printf("\n <%d,%d>=%d",i,j,p[i][j]);

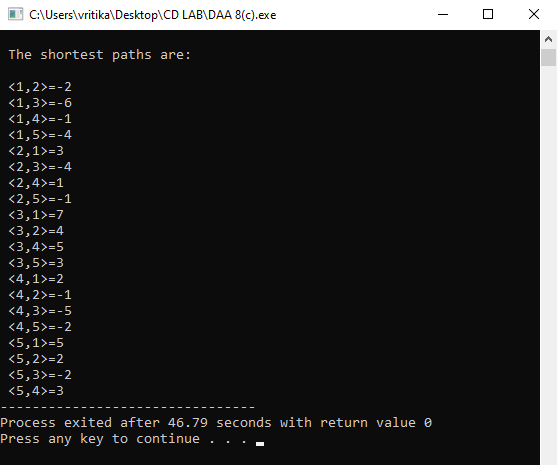
}

}

**Output:**







**Experiment – 10**

**Objective:** Implement N Queen's problem using Back Tracking.

**Resources:**

Dev C++

**Program Logic:**

1) Start in the leftmost column

2) If all queens are placedreturn true

3) Try all rows in the current column. Do following for every tried row.

a) If the queen can be placed safely in this row then mark this [row, column] as part of the solution and recursively check if placing queen here leads to a solution.

b) If placing queen in [row, column] leads to a solution then return true.

c) If placing queen doesn't lead to a solution then unmark this [row, column] (Backtrack) and go to step (a) to try other rows.

3) If all rows have been tried and nothing worked, return false to trigger Backtracking.

**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:**

#include<stdio.h>

#include<math.h>

int a[30],count=0;

int place(int pos)

{

int i;

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

{

if((a[i]==a[pos])||((abs(a[i]-a[pos])==abs(i-pos))))

return 0;

}

return 1;

}

void print\_sol(int n){

int i,j; count++;

printf("\n\nSolution #%d:\n",count);

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

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

if(a[i]==j)

printf("Q\t");

else

printf("\*\t"); }

printf("\n");}

}

void queen(int n){

int k=1;

a[k]=0;

while(k!=0){

a[k]=a[k]+1;

while((a[k]<=n)&&!place(k))

a[k]++;

if(a[k]<=n){

if(k==n)

print\_sol(n);

else{

k++;

a[k]=0;

} }

else

k--;

} }

main(){

int i,n;

printf("Enter the number of Queens :");

scanf("%d",&n);

queen(n);

printf("\nTotal solutions=%d",count); }

**Output:**

