1. **ATTAINMENT OF PROGRAM OUTCOMES AND PROGRAM SPECIFICOUTCOMES:**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **Experiment** |  |  |
| **L** | **QUICK SORT** |  |  |
| 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 1st 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** | **MERGE SORT** |  |  |
| 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 tobesortedandplotagraphofthetimetakenversusn.Theelements  can be read from a file or can be generated using the random number generator. |
| **3** | **WARSHALL’S ALGORITHM** |  |  |
| 1. Obtain the Topological ordering of vertices in a givendigraph.   http://d1gjlxt8vb0knt.cloudfront.net/wp-content/uploads/graph.png   1. Compute the transitive closure of a given directed graph using Warshall'salgorithm. |
|  |  |  |  |
|  |
| **4** | **SHORTEST PATHS ALGORITHM** |  |  |
| From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra’s algorithm. |

|  |  |  |  |
| --- | --- | --- | --- |
|  | http://d1gjlxt8vb0knt.cloudfront.net/wp-content/uploads/Fig-11.jpg |  |  |
| **5** | **MINIMUM COST SPANNING TREE** |  |  |
| Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal’s algorithm.  http://cs.geneseo.edu/~baldwin/csci242/fall2011/mst.png |

|  |  |  |  |
| --- | --- | --- | --- |
| **6** | **TRAVELLING SALES PERSON PROBLEM** |  |  |
| Implement any scheme to find the optimal solution for the Traveling Sales Person problem and then solve the same problem instance using any approximation algorithm and determine the error in the approximation. |
| **7** | **MINIMUM COST SPANNING TREE** |  |  |
| Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s algorithm. |

|  |  |  |  |
| --- | --- | --- | --- |
|  | images/lecture19/Kruskalexample.png |  |  |

|  |  |  |
| --- | --- | --- |
| **SR.NO-** | **Experiment** | **Page No** |
| **1** | **QICK SORT** |  |
| **2** | **MERGE SORT** |  |
| **3** | **WARSHALL’S ALGORITHM** |  |
| **4** | **SHORTEST PATHS ALGORITHM** |  |
| **5** | **MINIMUM COST SPANNING TREE** |  |
| **6** | **TRAVELLING SALES PERSON PROBLEM** |  |
| **7** | **MINIMUM COST SPANNING TREE** |  |

**1**

**QUICK SORT**

* 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.

## PROGRAMLOGIC:

QuickSort 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 QuickSort that pick pivot in different ways.

* + 1. Always pick first element aspivot.
    2. Always pick last element as pivot (implementedbelow)
    3. Pick a random element aspivot.
    4. Pick median aspivot.

The key process in QuickSort 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) afterx.

## PROCEDURE:

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

## SOURCECODE:

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)

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);

}

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]);

}

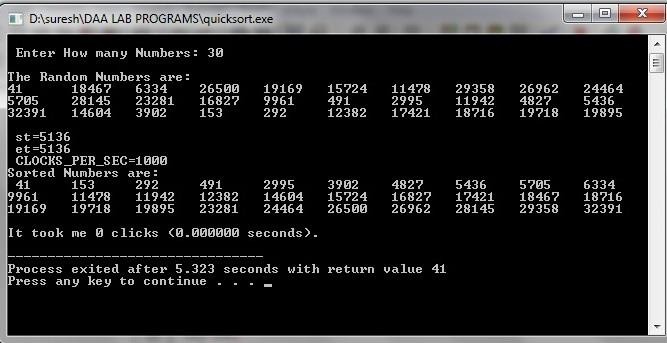
st=clock(); QuickSort(a, 1, n); et=clock();

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);

}

## INPUT/OUTPUT



# 2

**MERGE SORT**

## 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 numbergenerator.

## RESOURCES:

Dev C++

## PROGRAMLOGIC:

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 .cextension.
    2. Compile: Alt +F9
    3. Execute: Ctrl +F10

## SOURCECODE:

#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 Numbersare:\n"); for(k=1;k<=n;k++) {

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

}

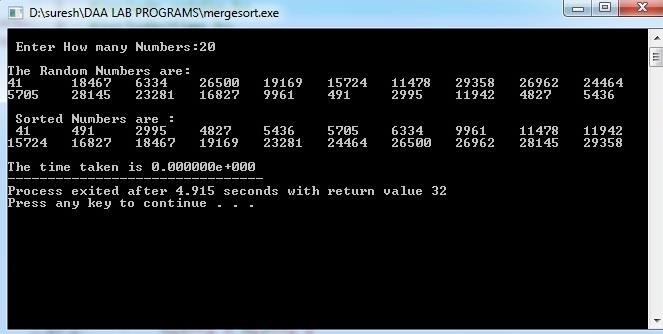
st=clock(); MergeSort(a, 1, n); et=clock();

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);

}

## INPUT/OUTPUT

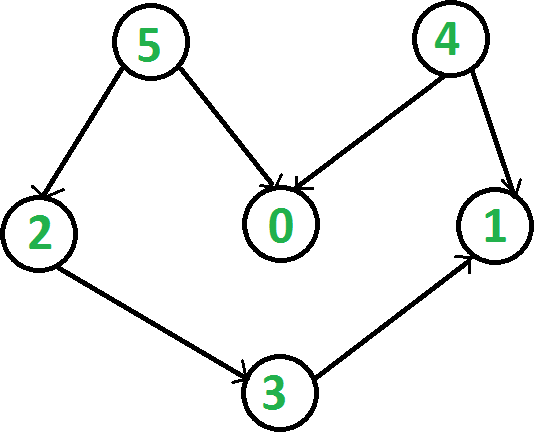


# 3

**WARSHALL’S ALGORITHM**

## OBJECTIVE:

* + 1. Obtain the Topological ordering of vertices in a givendigraph.



* + 1. Compute the transitive closure of a given directed graph using Warshall'salgorithm.

## RESOURCES:

Dev C++

## PROGRAMLOGIC:

**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 agraph.

## PROCEDURE:

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

## SOURCECODE:

**// 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 tostack

}

}

}

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();

}

## //Transitive closure of a graph using Warshall's algorithm

#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++)

}

void main(){

inti,j;

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++){

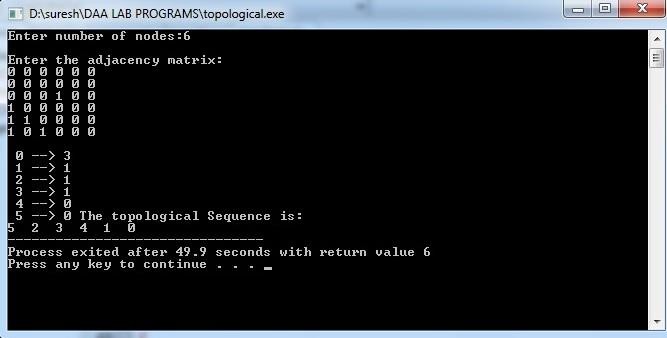
for(j=0;j<n;j++)

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

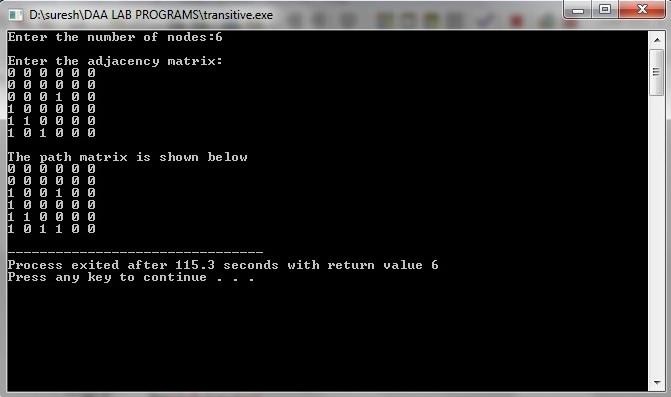
}

}

## INPUT/ OUTPUT Topologicalordering



**Transitive closure of a graph using Warshall's algorithm**

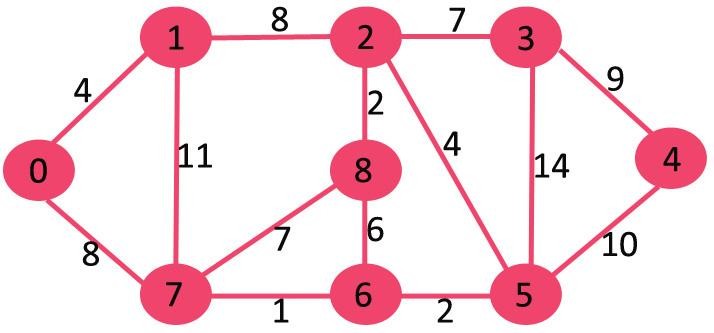


# 4

**SHORTEST PATHS ALGORITHM**

## OBJECTIVE:

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



## RESOURCES:

Dev C++

## PROGRAMLOGIC:

* + 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 pickedfirst.
    3. While *S* doesn’t include allvertices

1. Pick a vertex u which is not there in *S* and has minimum distance value.
2. Include u to*S*.
3. 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 ofv.

## PROCEDURE:

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

## SOURCECODE:

#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 sourcematrix:"); scanf("%d",&v);

dij(n,v,cost,dist);

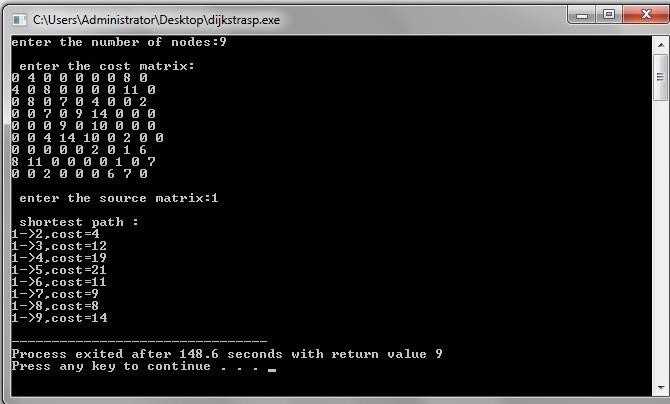
printf("\n shortest path : \n"); for(i=1;i<=n;i++)

if(i!=v)

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

}

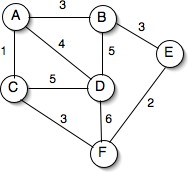
## INPUT/OUTPUT



# 5

**MINIMUM COST SPANNING TREE**

## OBJECTIVE:

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

## RESOURCES:

Dev C++

## PROGRAMLOGIC:

* + 1. Sort all the edges in non-decreasing order of theirweight.
    2. Pickthesmallestedge.Checkifitforms acyclewiththespanningtreeformedsofar.Ifcycleis not formed, include this edge. Else, discardit.
    3. Repeat step#2 until there are (V-1) edges in the spanningtree.

## PROCEDURE:

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

## SOURCECODE:

#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) {

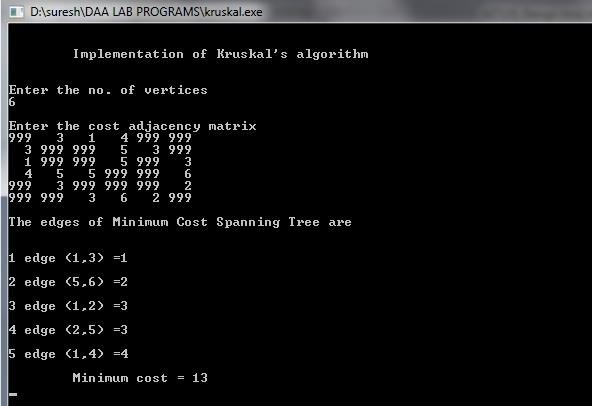
parent[j]=i; return 1;

}

return 0;

}

## INPUT/OUTPUT



**TRAVELLING SALES PERSON PROBLEM**

## OBJECTIVE:

Implement any scheme to find the optimal solution for the Traveling Sales Person problem and then solve the same problem instance using any approximation algorithm and determine the error in the approximation

## 6.1 RESOURCES:

Dev C++

## PROGRAMLOGIC:

* + 1. Check for the disconnection between the current city and the nextcity
    2. Check whether the travelling sales person has visited all thecities
    3. Find the next city to bevisited
    4. Find the solution andterminate

## PROCEDURE:

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

## SOURCE CODE:

#include<stdio.h> ints,c[100][100],ver; float optimum=999,sum;

/\* function to swap array elements \*/ void swap(int v[], int i, int j) {

int t;

t = v[i]; v[i] = v[j]; v[j] = t;

}

/\* recursive function to generate permutations \*/ voidbrute\_force(int v[], int n, int i) {

// this function generates the permutations of the array from element i to element n-1 int j,sum1,k;

//if we are at the end of the array, we have one permutation if (i == n) {

if(v[0]==s) {

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

printf ("%d ", v[j]); sum1=0;

for( k=0;k<n-1;k++) {

sum1=sum1+c[v[k]][v[k+1]];

}

}

else

}

sum1=sum1+c[v[n-1]][s]; printf("sum = %d\n",sum1); if (sum1<optimum)

optimum=sum1;

// recursively explore the permutations starting at index i going through index n-1\*/ for (j=i; j<n; j++) { /\* try the array with i and j switched \*/

swap (v, i, j); brute\_force (v, n, i+1);

/\* swap them back the way they were \*/ swap (v, i, j);

}

}

voidnearest\_neighbour(intver) { intmin,p,i,j,vis[20],from; for(i=1;i<=ver;i++)

vis[i]=0;

vis[s]=1; from=s; sum=0;

for(j=1;j<ver;j++) { min=999; for(i=1;i<=ver;i++)

if(vis[i] !=1 &&c[from][i]<min && c[from][i] !=0 ) { min= c[from][i];

p=i;

}

vis[p]=1; from=p; sum=sum+min;

}

sum=sum+c[from][s];

}

void main () {

intver,v[100],i,j;

printf("Enter n : ");

scanf("%d",&ver); for (i=0; i<ver;i++)

v[i] = i+1; printf("Enter costmatrix\n"); for(i=1;i<=ver;i++)

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

scanf("%d",&c[i][j]); printf("\nEnter source : "); scanf("%d",&s);

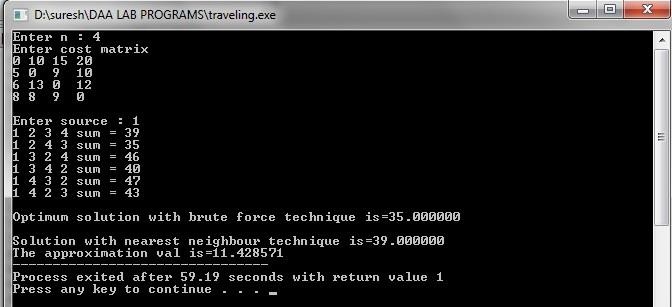
brute\_force (v, ver, 0);

printf("\nOptimum solution with brute force technique is=%f\n",optimum); nearest\_neighbour(ver);

printf("\nSolution with nearest neighbour technique is=%f\n",sum); printf("The approximation val is=%f",((sum/optimum)-1)\*100); printf(" % ");

}

## INPUT/OUTPUT

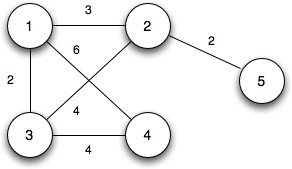


**7**

**MINIMUM COST SPANNING TREE**

* 1. **OBJECTIVE:**

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



## RESOURCES:

Dev C++

## PROGRAMLOGIC:

* + 1. Create a set Sthat keeps track of vertices already included inMST.
    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 pickedfirst.
    3. While S doesn’t include allvertices.
       1. Pick a vertex *u* which is not there in Sand has minimum keyvalue.
       2. Include *u* toS.
       3. 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.

## PROCEDURE:

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

## SOURCECODE.

#include<stdio.h> inta,b,u,v,n,i,j,ne=1;

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

{

}

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

{

min=cost[i][j]; a=u=i;

b=v=j;

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

