Avl tree-insertion,deletion:

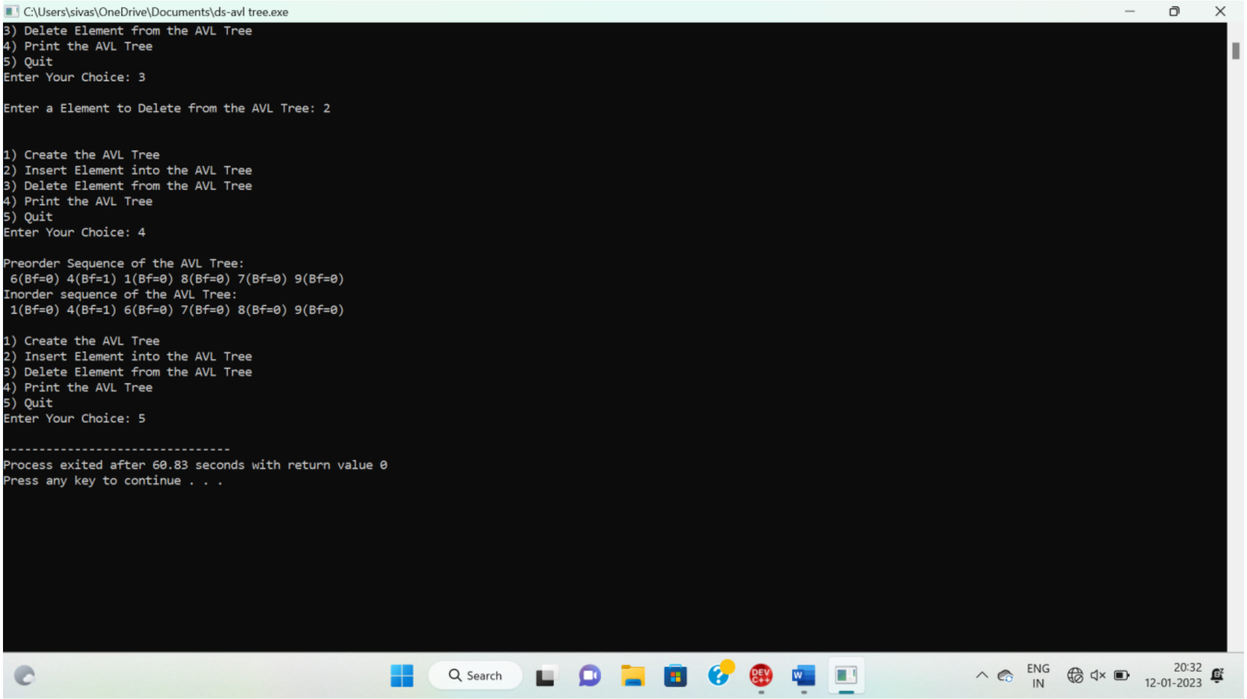
```
#include<conio.h>
#include<stdio.h>
#include<stdlib.h>
typedef struct node
{  int data;
   struct node *left,*right;
   int ht;
}node;
  node *insert(node *,int);
  node *Delete(node *,int);
  void  preorder(node *);
  void  inorder(node *);
  int   height( node *);
  node *rotateright(node *);
  node *rotateleft(node *);
  node *RR(node *);
  node *LL(node *);
  node *LR(node *);
  node *RL(node *);
  int BF(node *);

int main()
{
    node *root=NULL;
    int x,n,i,op;
    do
        {
            printf("\n");
            printf("\n1) Create the AVL Tree");
printf("\n2) Insert Element into the AVL Tree");
printf("\n3) Delete Element from the AVL Tree ");
printf("\n4) Print the AVL Tree");
printf("\n5) Quit");
printf("\nEnter Your Choice: ");
scanf("%d",&op);
switch(op)
  {
  case 1:printf("\nEnter Total Number of Elements in the AVL Tree: ");
       scanf("%d",&n);
       printf("\n Enter AVL Tree Elements: ");
       root=NULL;
       for(i=0;i<n;i++)
       {
        scanf("%d",&x);
        root=insert(root,x);
       }
       break;
  case 2:printf("\nEnter a Element to Insert in the AVL Tree: ");
       scanf("%d",&x);
       root=insert(root,x);
       break;
  case 3:printf("\nEnter a Element to Delete from the AVL Tree: ");
       scanf("%d",&x);
       root=Delete(root,x);
       break;
  case 4:    printf("\nPreorder Sequence of the AVL Tree:\n");
     preorder(root);
     printf("\nInorder sequence of the AVL Tree:\n");
     inorder(root);
     break;
```

```
                }
    }while(op!=5);
}

node * insert(node *T,int x)
{
    if(T==NULL)
    {
        T=(node*)malloc(sizeof(node));
        T->data=x;
        T->left=NULL;
        T->right=NULL;
    }
    else
        if(x > T->data)
        {
            T->right=insert(T->right,x);
            if(BF(T)==-2)
                if(x>T->right->data)
                    T=RR(T);
                else
                    T=RL(T);
        }
        else
            if(x<T->data)
            {
                T->left=insert(T->left,x);
                if(BF(T)==2)
                    if(x < T->left->data)
                        T=LL(T);
                    else
```

```
                        T=LR(T);
            }
            T->ht=height(T);
            return(T);
}

node * Delete(node *T,int x)
{      node *p;

    if(T==NULL)
    {
        return NULL;
    }
    else

        if(x > T->data)
        {
            T->right=Delete(T->right,x);
            if(BF(T)==2)
                if(BF(T->left)>=0)
                    T=LL(T);
                else
                    T=LR(T);
        }
        else
            if(x<T->data)
                {
                    T->left=Delete(T->left,x);
                    if(BF(T)==-2)
                        if(BF(T->right)<=0)
                            T=RR(T);
            else
              T=RL(T);
        }
      else
       {
         if(T->right !=NULL)
         {
           p=T->right;
           while(p->left != NULL)
           p=p->left;

           T->data=p->data;
           T->right=Delete(T->right,p->data);
           if(BF(T)==2)
            if(BF(T->left)>=0)
              T=LL(T);
            else
              T=LR(T);
          }
         else
          return(T->left);

       }
  T->ht=height(T);
  return(T);
}
int height(node *T)
{
  int lh,rh;
  if(T==NULL)
    return(0);
    if(T->left==NULL)
        lh=0;
    else
        lh=1+T->left->ht;
    if(T->right==NULL)
        rh=0;
    else
        rh=1+T->right->ht;
    if(lh>rh)
        return(lh);
    return(rh);
}
node * rotateright(node *x)
{
    node *y;
    y=x->left;
    x->left=y->right;
    y->right=x;
    x->ht=height(x);
    y->ht=height(y);
    return(y);
}
node * rotateleft(node *x)
{
    node *y;
    y=x->right;
    x->right=y->left;
    y->left=x;
    x->ht=height(x);
    y->ht=height(y);
    return(y);
```

```
}
node * RR(node *T)
{
   T=rotateleft(T);
   return(T);
}
node * LL(node *T)
{
   T=rotateright(T);
   return(T);
}
node * LR(node *T)
{
   T->left=rotateleft(T->left);
   T=rotateright(T);
   return(T);
}
node * RL(node *T)
{
   T->right=rotateright(T->right);
   T=rotateleft(T);
   return(T);
}
int BF(node *T)
{
   int lh,rh;
   if(T==NULL)
   return(0);
   if(T->left==NULL)
      lh=0;
   else
```

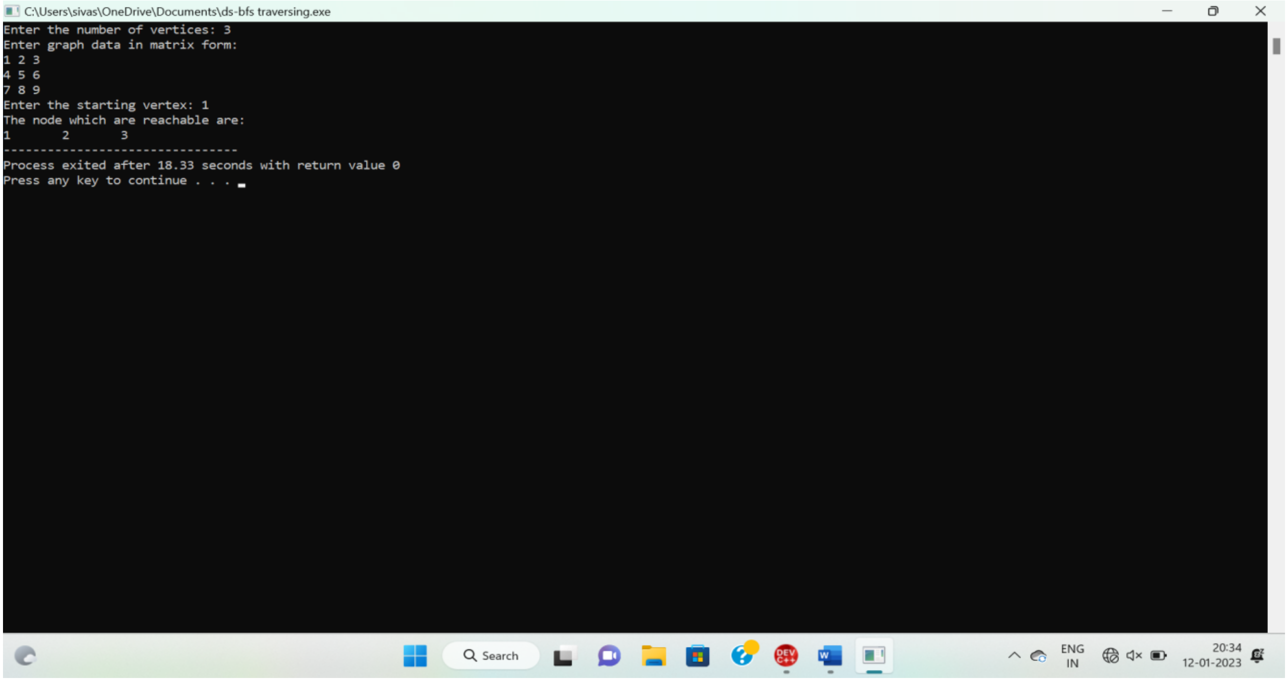
```
        lh=1+T->left->ht;
    if(T->right==NULL)
        rh=0;
    else
        rh=1+T->right->ht;
    return(lh-rh);
}

void preorder(node *T)
{
    if(T!=NULL)
    {
        printf(" %d(Bf=%d)",T->data,BF(T));
        preorder(T->left);
        preorder(T->right);
    }
}
void inorder(node *T)
{
    if(T!=NULL)
    {
        inorder(T->left);
        printf(" %d(Bf=%d)",T->data,BF(T));
        inorder(T->right);
    }
}
```



```
3) Delete Element from the AVL Tree
4) Print the AVL Tree
5) Quit
Enter Your Choice: 3

Enter a Element to Delete from the AVL Tree: 2


1) Create the AVL Tree
2) Insert Element into the AVL Tree
3) Delete Element from the AVL Tree
4) Print the AVL Tree
5) Quit
Enter Your Choice: 4

Preorder Sequence of the AVL Tree:
 6(Bf=0) 4(Bf=1) 1(Bf=0) 8(Bf=0) 7(Bf=0) 9(Bf=0)
Inorder sequence of the AVL Tree:
 1(Bf=0) 4(Bf=1) 6(Bf=0) 7(Bf=0) 8(Bf=0) 9(Bf=0)

1) Create the AVL Tree
2) Insert Element into the AVL Tree
3) Delete Element from the AVL Tree
4) Print the AVL Tree
5) Quit
Enter Your Choice: 5

--------------------------------
Process exited after 60.83 seconds with return value 0
Press any key to continue . . .
```

Bfs traversing:

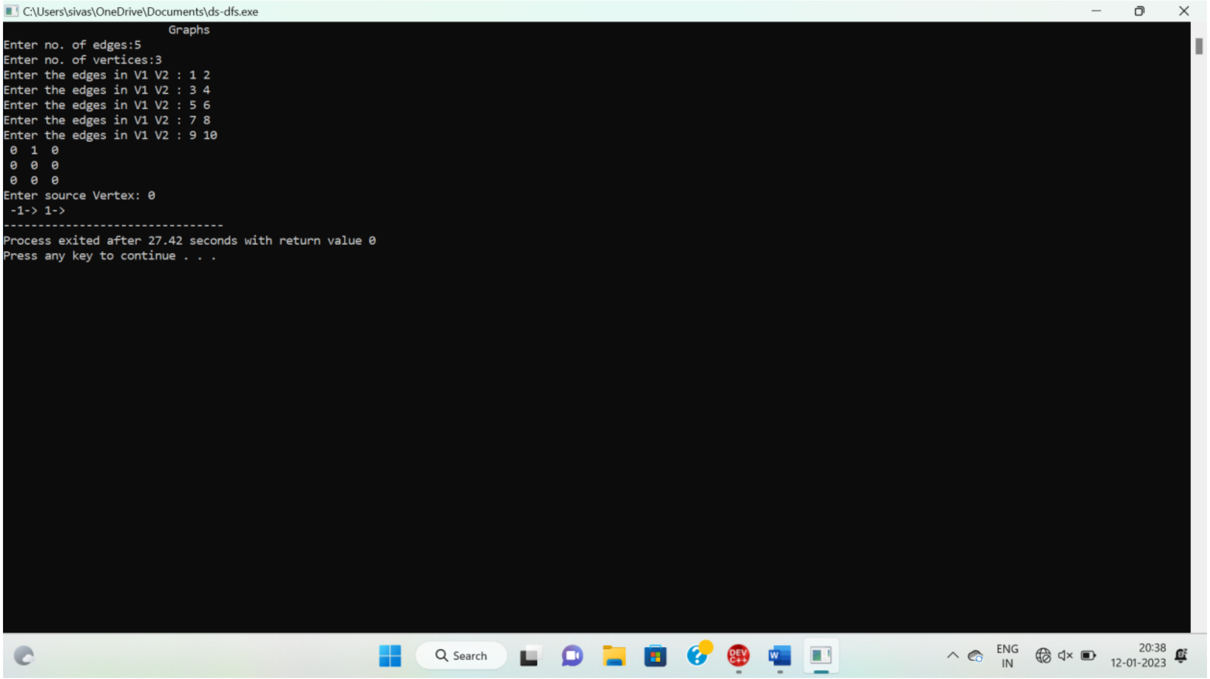
```
#include <stdio.h>

int n, i, j, visited[10], queue[10], front = -1, rear = -1;

int adj[10][10];


void bfs(int v)

{

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

        if (adj[v][i] && !visited[i])

            queue[++rear] = i;

    if (front <= rear)

    {

        visited[queue[front]] = 1;

        bfs(queue[front++]);

    }

}

int main()

{
    int v;
    printf("Enter the number of vertices: ");
    scanf("%d", &n);
    for (i = 1; i <= n; i++)
    {
        queue[i] = 0;
        visited[i] = 0;
    }
    printf("Enter graph data in matrix form:    \n");
    for (i = 1; i <= n; i++)
        for (j = 1; j <= n; j++)
            scanf("%d", &adj[i][j]);
    printf("Enter the starting vertex: ");
    scanf("%d", &v);
    bfs(v);
    printf("The node which are reachable are:    \n");
    for (i = 1; i <= n; i++)
        if (visited[i])
            printf("%d\t", i);
        else
            printf("BFS is not possible. Not all nodes are reachable");
    return 0;
}
```



```
Enter the number of vertices: 3
Enter graph data in matrix form:
1 2 3
4 5 6
7 8 9
Enter the starting vertex: 1
The node which are reachable are:
1       2       3
--------------------------------
Process exited after 18.33 seconds with return value 0
Press any key to continue . . .
```

Dfs traversing:

```
#include <stdio.h>
#include <stdlib.h>
int sourceV,Vertex,Edge,time,visited[10],Graph[10][10];
void DepthFirstSearch(int i)
{
int j;
visited[i]=1;
printf(" %d->",i++);
for(j=0;j<Vertex;j++)
{
if(Graph[i][j]==1&&visited[j]==0)
DepthFirstSearch(j);
}
}
int main()
{
int i,j,vertex1,vertex2;
printf("\t\t\tGraphs\n");
printf("Enter no. of edges:");
scanf("%d",&Edge);
printf("Enter no. of vertices:");
scanf("%d",&Vertex);
for(i=0;i<Vertex;i++)
{
for(j=0;j<Vertex;j++)
Graph[i][j]=0;
}
for(i=0;i<Edge;i++)
{
printf("Enter the edges in V1 V2 : ");
scanf("%d%d",&vertex1,&vertex2);
Graph[vertex1-1][vertex2-1]=1;
}
for(i=0;i<Vertex;i++)
{
for(j=0;j<Vertex;j++)
printf(" %d ",Graph[i][j]);
printf("\n");
}
printf("Enter source Vertex: ");
scanf("%d",&sourceV);
DepthFirstSearch(sourceV-1);
return 0;
}
```

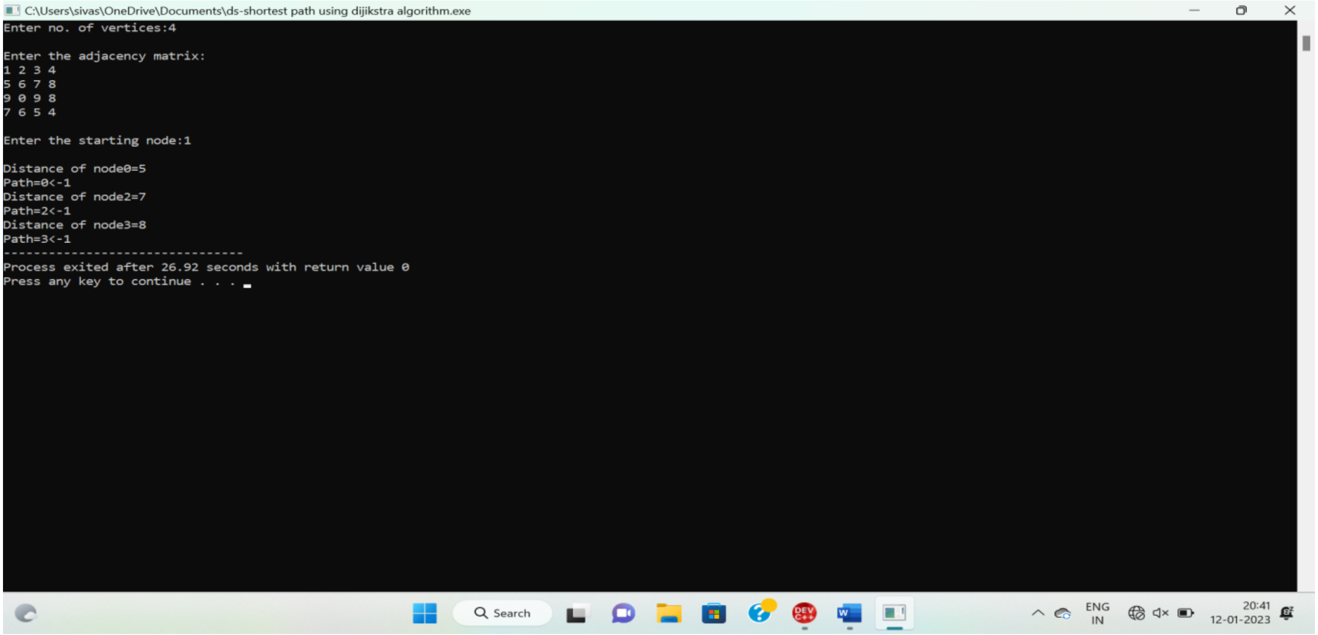


```
                          Graphs
Enter no. of edges:5
Enter no. of vertices:3
Enter the edges in V1 V2 : 1 2
Enter the edges in V1 V2 : 3 4
Enter the edges in V1 V2 : 5 6
Enter the edges in V1 V2 : 7 8
Enter the edges in V1 V2 : 9 10
 0  1  0
 0  0  0
 0  0  0
Enter source Vertex: 0
 -1-> 1->
--------------------------------
Process exited after 27.42 seconds with return value 0
Press any key to continue . . .
```

Shortest path using dijkstra algorithm:

```
#include<stdio.h>

#include<conio.h>

#define INFINITY 9999

#define MAX 10

void dijkstra(int G[MAX][MAX],int n,int startnode);

int main()

{

int G[MAX][MAX],i,j,n,u;

printf("Enter no. of vertices:");

scanf("%d",&n);

printf("\nEnter the adjacency matrix:\n");

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

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

scanf("%d",&G[i][j]);

printf("\nEnter the starting node:");

scanf("%d",&u);
```

```
dijkstra(G,n,u);
return 0;
}
void dijkstra(int G[MAX][MAX],int n,int startnode)
{
int cost[MAX][MAX],distance[MAX],pred[MAX];
int visited[MAX],count,mindistance,nextnode,i,j;
for(i=0;i<n;i++)
for(j=0;j<n;j++)
if(G[i][j]==0)
cost[i][j]=INFINITY;
else
cost[i][j]=G[i][j];
for(i=0;i<n;i++)
{
distance[i]=cost[startnode][i];
pred[i]=startnode;
visited[i]=0;
}
distance[startnode]=0;
visited[startnode]=1;
count=1;
while(count<n-1)
{
mindistance=INFINITY;
for(i=0;i<n;i++)
if(distance[i]<mindistance&&!visited[i])
{
mindistance=distance[i];
nextnode=i;
}
visited[nextnode]=1;
for(i=0;i<n;i++)
if(!visited[i])
if(mindistance+cost[nextnode][i]<distance[i])
{
distance[i]=mindistance+cost[nextnode][i];
pred[i]=nextnode;
}
count++;
}
for(i=0;i<n;i++)
if(i!=startnode)
{
printf("\nDistance of node%d=%d",i,distance[i]);
printf("\nPath=%d",i);
j=i;
do
{
j=pred[j];
printf("<-%d",j);
}while(j!=startnode);
}
}
```

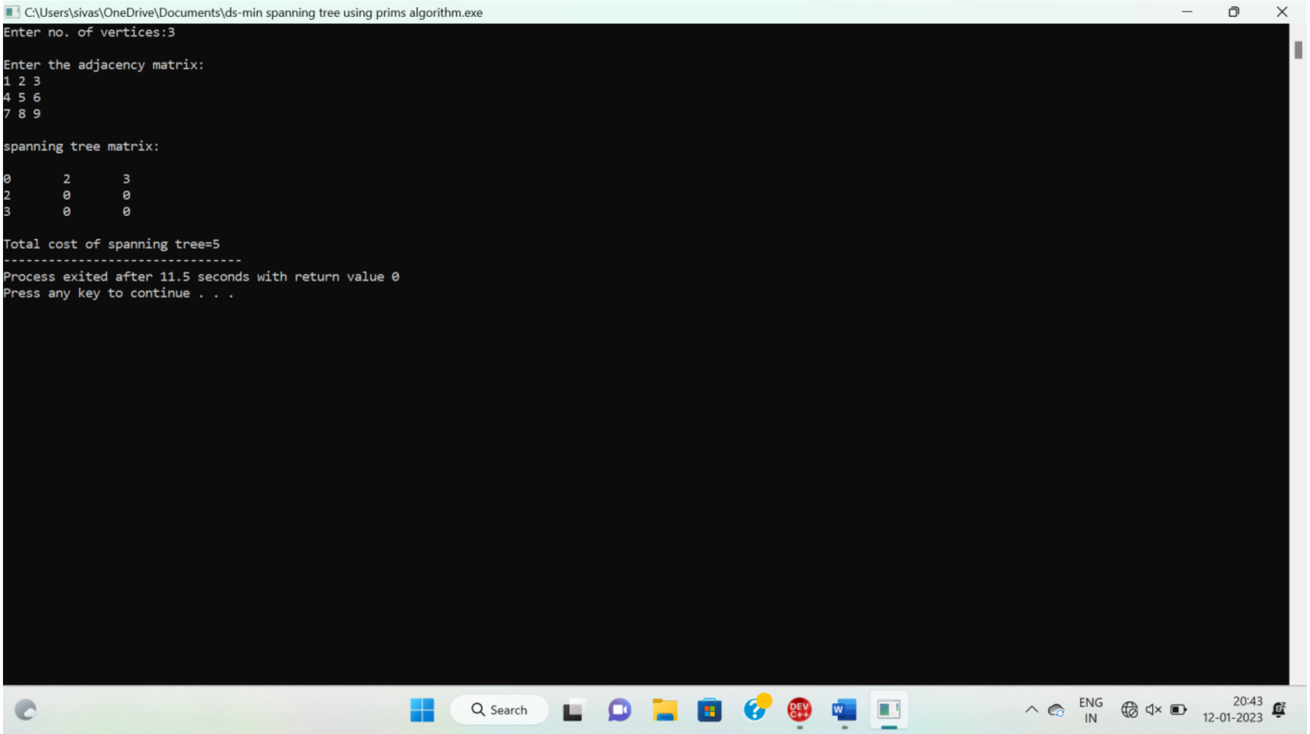

C:\Users\sivas\OneDrive\Documents\ds-shortest path using dijikstra algorithm.exe
Search
ENG
IN
20:41
12-01-2023

```
Enter no. of vertices:4

Enter the adjacency matrix:
1 2 3 4
5 6 7 8
9 0 9 8
7 6 5 4

Enter the starting node:1

Distance of node0=5
Path=0<-1
Distance of node2=7
Path=2<-1
Distance of node3=8
Path=3<-1
--------------------------------
Process exited after 26.92 seconds with return value 0
Press any key to continue . . .
```

Implementaion of minimum spanning tree using prims algorithm:

```
#include<stdio.h>

#include<stdlib.h>

#define infinity 9999

#define MAX 20

int G[MAX][MAX],spanning[MAX][MAX],n;

int prims();

int main()

{

int i,j,total_cost;

printf("Enter no. of vertices:");

scanf("%d",&n);

printf("\nEnter the adjacency matrix:\n");

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

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

scanf("%d",&G[i][j]);

total_cost=prims();

printf("\nspanning tree matrix:\n");
```

```
for(i=0;i<n;i++)
{
printf("\n");
for(j=0;j<n;j++)
printf("%d\t",spanning[i][j]);
}
printf("\n\nTotal cost of spanning tree=%d",total_cost);
return 0;
}

int prims()
{
int cost[MAX][MAX];
int u,v,min_distance,distance[MAX],from[MAX];
int visited[MAX],no_of_edges,i,min_cost,j;
for(i=0;i<n;i++)
for(j=0;j<n;j++)
{
if(G[i][j]==0)
cost[i][j]=infinity;
else
cost[i][j]=G[i][j];
spanning[i][j]=0;
}
distance[0]=0;
visited[0]=1;
for(i=1;i<n;i++)
{
distance[i]=cost[0][i];
from[i]=0;
visited[i]=0;
}
min_cost=0;
no_of_edges=n-1;
while(no_of_edges>0)
{
min_distance=infinity;
for(i=1;i<n;i++)
if(visited[i]==0&&distance[i]<min_distance)
{
v=i;
min_distance=distance[i];
}
u=from[v];
spanning[u][v]=distance[v];
spanning[v][u]=distance[v];
no_of_edges--;
visited[v]=1;
for(i=1;i<n;i++)
if(visited[i]==0&&cost[i][v]<distance[i])
{
distance[i]=cost[i][v];
from[i]=v;
}
min_cost=min_cost+cost[u][v];
}
return(min_cost);
}
```



C:\Users\sivas\OneDrive\Documents\ds-min spanning tree using prims algorithm.exe
Search
ENG
IN
20:43
12-01-2023

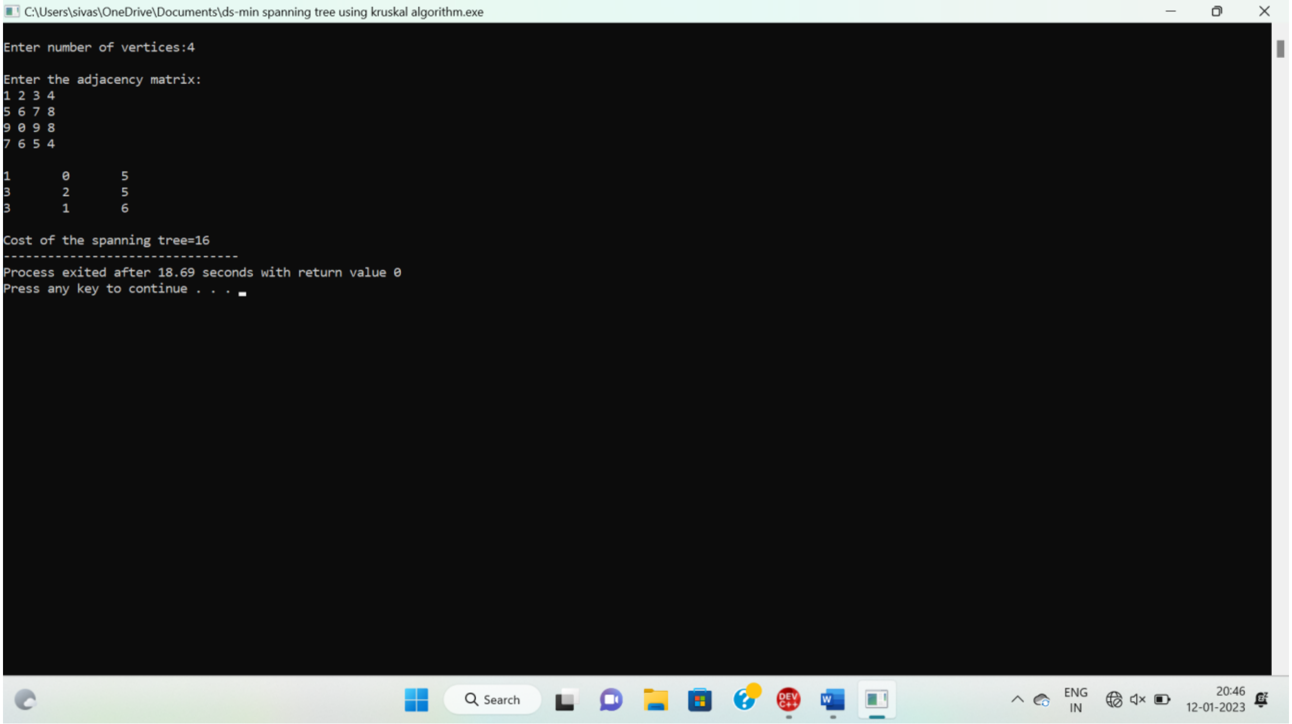
```
Enter no. of vertices:3

Enter the adjacency matrix:
1 2 3
4 5 6
7 8 9

spanning tree matrix:

0       2       3
2       0       0
3       0       0

Total cost of spanning tree=5
--------------------------------
Process exited after 11.5 seconds with return value 0
Press any key to continue . . .
```

Implementation of minimum spanning tree using kruskal algorithm:

```
#include<stdio.h>

#define MAX 30

typedef struct edge

{

int u,v,w;

}edge;

typedef struct edgelist

{

edge data[MAX];

int n;

}edgelist;

edgelist elist;

int G[MAX][MAX],n;

edgelist spanlist;

void kruskal();

int find(int belongs[],int vertexno);
```

```
void union1(int belongs[],int c1,int c2);
void sort();
void print();
int main()
{
int i,j,total_cost;
printf("\nEnter number of vertices:");
scanf("%d",&n);
printf("\nEnter the adjacency matrix:\n");
for(i=0;i<n;i++)
for(j=0;j<n;j++)
scanf("%d",&G[i][j]);
kruskal();
print();
}
void kruskal()
{
int belongs[MAX],i,j,cno1,cno2;
elist.n=0;
for(i=1;i<n;i++)
for(j=0;j<i;j++)
{
if(G[i][j]!=0)
{
elist.data[elist.n].u=i;
elist.data[elist.n].v=j;
elist.data[elist.n].w=G[i][j];
elist.n++;
}
}
sort();
```

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

belongs[i]=i;

spanlist.n=0;

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

{

cno1=find(belongs,elist.data[i].u);

cno2=find(belongs,elist.data[i].v);

if(cno1!=cno2)

{

spanlist.data[spanlist.n]=elist.data[i];

spanlist.n=spanlist.n+1;

union1(belongs,cno1,cno2);

}

}

}

int find(int belongs[],int vertexno)

{

return(belongs[vertexno]);

}

void union1(int belongs[],int c1,int c2)

{

int i;

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

if(belongs[i]==c2)

belongs[i]=c1;

}

void sort()

{

int i,j;

edge temp;

for(i=1;i<elist.n;i++)
for(j=0;j<elist.n-1;j++)
if(elist.data[j].w>elist.data[j+1].w)
{
temp=elist.data[j];
elist.data[j]=elist.data[j+1];
elist.data[j+1]=temp;
}
}
void print()
{
int i,cost=0;
for(i=0;i<spanlist.n;i++)
{
printf("\n%d\t%d\t%d",spanlist.data[i].u,spanlist.data[i].v,spanlist.data[i].w);
cost=cost+spanlist.data[i].w;
}
printf("\n\nCost of the spanning tree=%d",cost);
}
```


C:\Users\sivas\OneDrive\Documents\ds-min spanning tree using kruskal algorithm.exe
Search
ENG
IN
20:46
12-01-2023

```
Enter number of vertices:4

Enter the adjacency matrix:
1 2 3 4
5 6 7 8
9 0 9 8
7 6 5 4

1       0       5
3       2       5
3       1       6

Cost of the spanning tree=16
--------------------------------
Process exited after 18.69 seconds with return value 0
Press any key to continue . . .
```