**Binary tree creation and traversal**

#include<iostream>

using namespace std;

//binary tree node declaration

class bintree\_node{

public:

    bintree\_node \*left;

    bintree\_node \*right;

    char data;

} ;

class bintree\_class{

    bintree\_node \*root;

    public:

    bintree\_class(){

        root=NULL;

    }

    int isempty() {

        return(root==NULL);

    }

    void insert\_node(int item);

    void inorder\_seq();

    void inorder(bintree\_node \*);

    void postorder\_seq();

    void postorder(bintree\_node \*);

    void preorder\_seq();

    void preorder(bintree\_node \*);

};

void bintree\_class::insert\_node(int item){

    bintree\_node \*p=new bintree\_node;

    bintree\_node \*parent;

    p->data=item;

    p->left=NULL;

    p->right=NULL;

    parent=NULL;

    if(isempty())

        root=p;

    else{

        bintree\_node \*ptr;

        ptr=root;

        while(ptr!=NULL)        {

            parent=ptr;

            if(item>ptr->data)

                ptr=ptr->right;

            else

                ptr=ptr->left;

        }

        if(item<parent->data)

            parent->left=p;

        else

            parent->right=p;

    }

}

void bintree\_class::inorder\_seq()

{

    inorder(root);

}

void bintree\_class::inorder(bintree\_node \*ptr)

{

    if(ptr!=NULL){

        inorder(ptr->left);

        cout<<"  "<<ptr->data<<"     ";

        inorder(ptr->right);

    }

}

void bintree\_class::postorder\_seq()

{

    postorder(root);

}

void bintree\_class::postorder(bintree\_node \*ptr)

{

    if(ptr!=NULL){

        postorder(ptr->left);

        postorder(ptr->right);

        cout<<"  "<<ptr->data<<"     ";

    }

}

void bintree\_class::preorder\_seq()

{

    preorder(root);

}

void bintree\_class::preorder(bintree\_node \*ptr)

{

    if(ptr!=NULL){

        cout<<"  "<<ptr->data<<"     ";

        preorder(ptr->left);

        preorder(ptr->right);

    }

}

int main()

{

    bintree\_class bintree;

    bintree.insert\_node('A');

    bintree.insert\_node('B');

    bintree.insert\_node('C');

    bintree.insert\_node('D');

    bintree.insert\_node('E');

    bintree.insert\_node('F');

    bintree.insert\_node('G');

    cout<<"Inorder traversal:"<<endl;

    bintree.inorder\_seq();

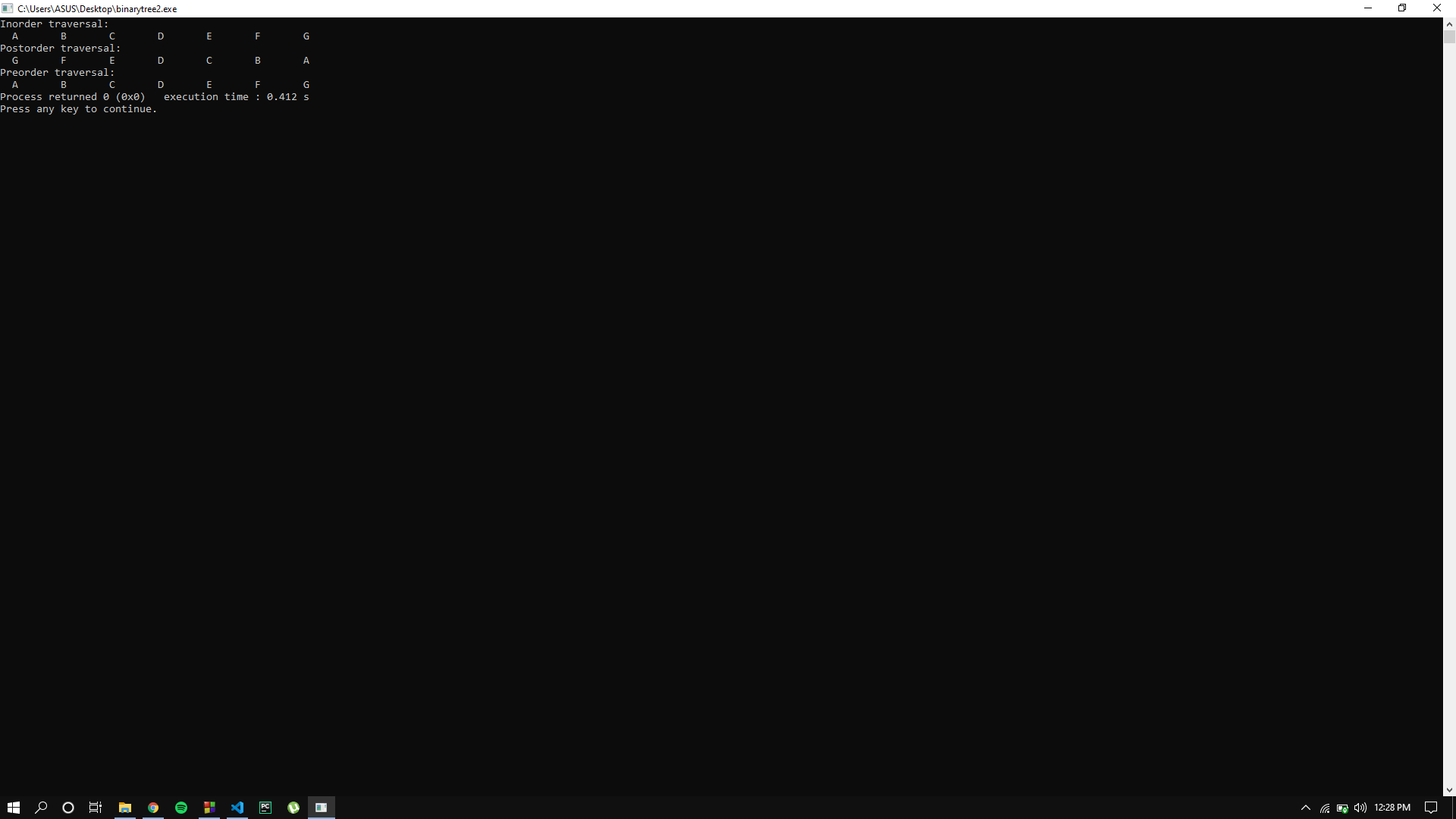
    cout<<endl<<"Postorder traversal:"<<endl;

    bintree.postorder\_seq();

    cout<<endl<<"Preorder traversal:"<<endl;

    bintree.preorder\_seq();

}



**Binary Search tree**

#include <iostream>

#include <stdlib.h>

using namespace std;

class Node {

public:

    int data;

    Node \*left;

    Node \*right;

};

Node\* create(int item)

{

    Node\* node = new Node;

    node->data = item;

    node->left = node->right = NULL;

    return node;

}

void inorder(Node \*root)

{

    if (root == NULL)

        return;

    inorder(root->left);

    cout<< root->data << "   ";

    inorder(root->right);

}

Node\* findMinimum(Node\* cur)

{

    while(cur->left != NULL) {

        cur = cur->left;

    }

    return cur;

}

Node\* insertion(Node\* root, int item)

{

    if (root == NULL)

        return create(item);

    if (item < root->data)

        root->left = insertion(root->left, item);

    else

        root->right = insertion(root->right, item);

    return root;

}

void search(Node\* &cur, int item, Node\* &parent)

{

    while (cur != NULL && cur->data != item)

    {

        parent = cur;

        if (item < cur->data)

            cur = cur->left;

        else

            cur = cur->right;

    }

}

void deletion(Node\*& root, int item)

{

    Node\* parent = NULL;

    Node\* cur = root;

    search(cur, item, parent);

    if (cur == NULL)

        return;

    if (cur->left == NULL && cur->right == NULL)

    {

        if (cur != root)

        {

            if (parent->left == cur)

                parent->left = NULL;

            else

                parent->right = NULL;

        }

        else

            root = NULL;

        free(cur);

    }

    else if (cur->left && cur->right)

    {

        Node\* succ  = findMinimum(cur->right);

        int val = succ->data;

        deletion(root, succ->data);

        cur->data = val;

    }

    else

    {

        Node\* child = (cur->left)?cur->left:cur->right;

        if (cur != root)

        {

            if (cur == parent->left)

                parent->left = child;

            else

                parent->right = child;

        }

        else

            root = child;

        free(cur);

    }

}

int main()

{

   Node\* root = NULL;

   int keys[8];

   for(int i=0;i<8;i++)

    {

    cout << "Enter value to be inserted : ";

    cin>>keys[i];

        root = insertion(root, keys[i]);

    }

    inorder(root);

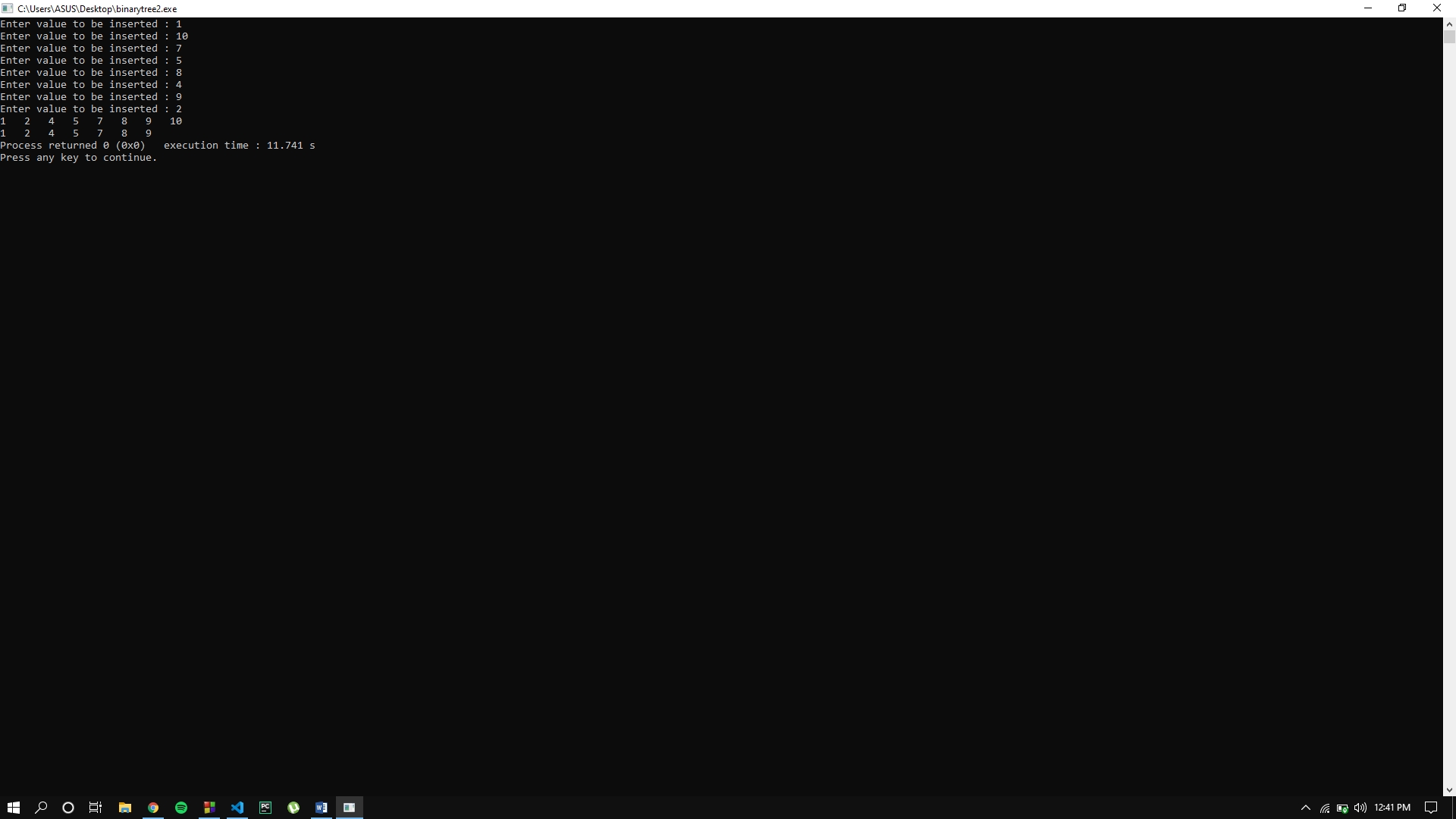
    cout<<"\n";

    deletion(root, 10);

    inorder(root);

    return 0;

}



**Graph on Breath first search**

#include<iostream>

#include<conio.h>

#include<stdlib.h>

using namespace std;

int cost[10][10],i,j,k,n,queue[10],front,rear,v,visit[10],visited[10];

int main()

{

    int m;

    cout <<"enterno of vertices";

    cin >> n;

    cout <<"ente no of edges";

    cin >> m;

    cout <<"\nEDGES \n";

    for(k=1;k<=m;k++)

    {

        cin >>i>>j;

        cost[i][j]=1;

    }

    cout <<"enter initial vertex";

    cin >>v;

    cout <<"Visitied vertices\n";

    cout << v;

    visited[v]=1;

    k=1;

    while(k<n)

    {

        for(j=1;j<=n;j++)

            if(cost[v][j]!=0 && visited[j]!=1 && visit[j]!=1)

            {

                visit[j]=1;

                queue[rear++]=j;

            }

        v=queue[front++];

        cout<<v << " ";

        k++;

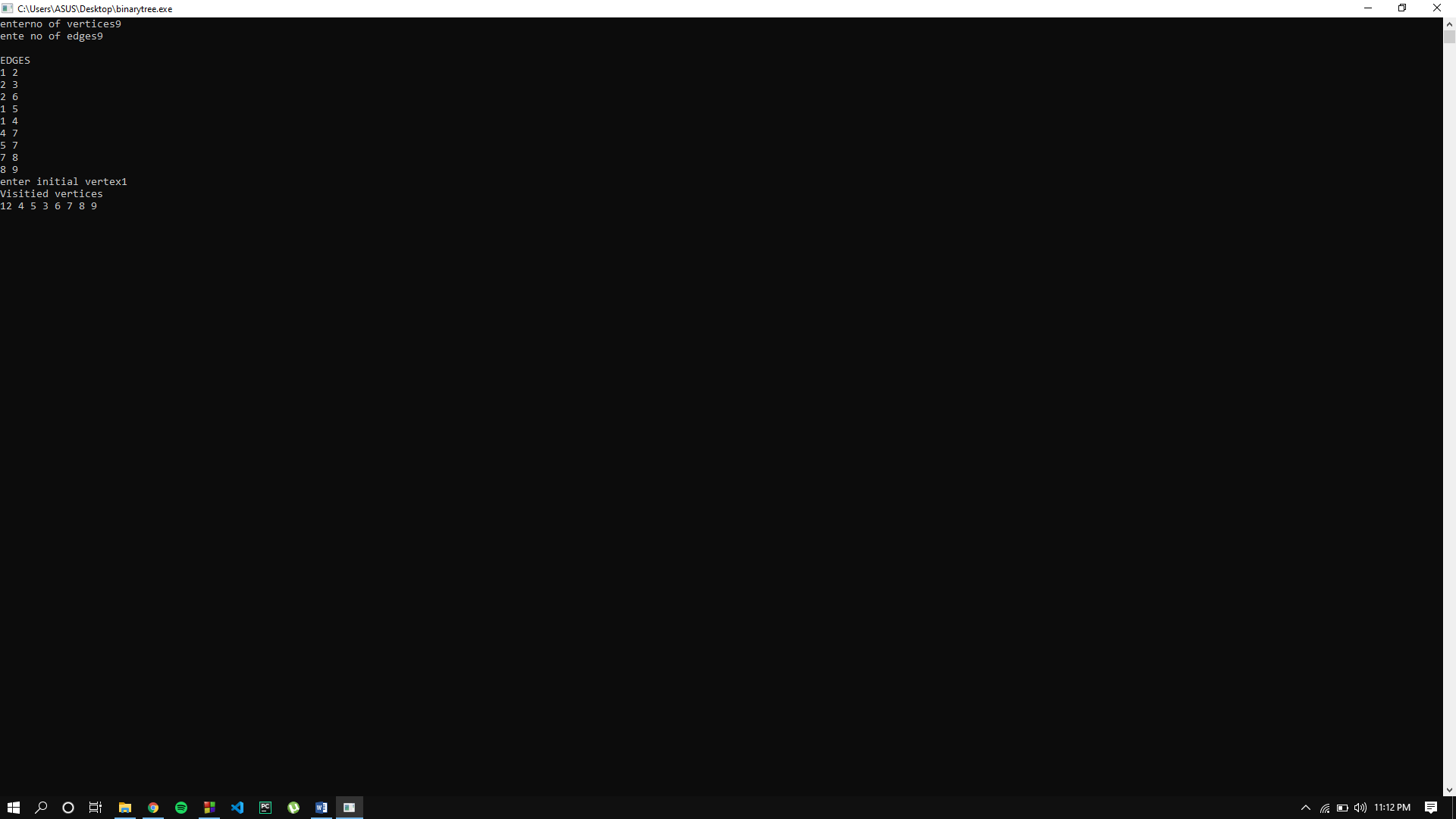
        visit[v]=0;

        visited[v]=1;

    }

    getch();

}



**Graph on Breath first search**

#include<iostream>

#include<conio.h>

#include<stdlib.h>

using namespace std;

int cost[10][10],i,j,k,n,stack[10],top,v,visit[10],visited[10];

int main()

{

    int m;

    cout <<"enterno of vertices";

    cin >> n;

    cout <<"ente no of edges";

    cin >> m;

    cout <<"\nEDGES \n";

    for(k=1;k<=m;k++)

    {

        cin >>i>>j;

        cost[i][j]=1;

    }

    cout <<"enter initial vertex";

    cin >>v;

    cout <<"ORDER OF VISITED VERTICES";

    cout << v <<" ";

    visited[v]=1;

    k=1;

    while(k<n)

    {

        for(j=n;j>=1;j--)

            if(cost[v][j]!=0 && visited[j]!=1 && visit[j]!=1)

            {

                visit[j]=1;

                stack [top]=j;

                top++;

            }

        v= stack [--top];

        cout<<v << " ";

        k++;

        visit[v]=0; visited[v]=1;

    }

    getch();

}

