Name: Swarnabh Paul

Section: Y

Roll no: 19CS8122

Assignment no: 9

Questions attempted: 1,2

**Question (1)**

**Code:**

#include<iostream>

#include<stdbool.h>

#include"MyLinkedList.h"

#include"MyStack.h"

**using** **namespace** std;

**class** **graph**

{

sll \*adjl;

**int** N;

**public:**

graph(**int**);

~graph();

**void** **DFS**();

**void** **Display**();

**void** **connected\_components**();

**void** **insert\_edge**(**int**,**int**);

**void** **delete\_edge**(**int**,**int**);

};

graph::graph(**int** n)

{

cout<<"Graph created"<<endl;

N=n;

adjl=**new** sll[N];

}

graph::~graph()

{

**int** i;

**for**(i=**0**;i<N;i++)

adjl[i].deletesll();

**delete** []adjl;

cout<<"Graph destroyed"<<endl;

}

**void** graph::Display()

{

**int** i;

**for**(i=**0**;i<N;i++)

{

cout<<i+**1**<<": ";

adjl[i].display();

}

}

**void** graph::insert\_edge(**int** u,**int** v)

{

adjl[u-**1**].insertBeg(v);

}

**void** graph::delete\_edge(**int** u,**int** v)

{

**if**(!adjl[u-**1**].search(v))

{

cout<<"Error! Edge not present!"<<endl;

}

**else**

{

adjl[u-**1**].Delete(v);

}

}

**void** graph::DFS()

{

**int** i, cnt=**0**, u, j;

node \*t;

**for**(i=**0**;i<N;i++)

cnt+=adjl[i].head.data;

stack **s**(cnt+N);

**for**(i=**0**;i<N;i++)

adjl[i].head.extra=**0**;

**for**(i=**0**;i<N;i++)

{

**if**(!adjl[i].head.extra)

{

s.push(i+**1**);

**while**(!s.isempty())

{

u=s.pop();

**if**(!adjl[u-**1**].head.extra)

{

cout<<u<<' ';

adjl[u-**1**].head.extra=**1**;

}

**for**(t=adjl[u-**1**].head.link,j=**0**;j<adjl[u-**1**].size();j++,t=t->link)

**if**(!adjl[t->data-**1**].head.extra)

s.push(t->data);

}

}

}

}

**void** graph::connected\_components()

{

**int** i, cnt=**0**, u, j, n;

node \*t;

**for**(i=**0**;i<N;i++)

cnt+=adjl[i].head.data;

stack **s**(cnt+N);

**for**(i=**0**;i<N;i++)

adjl[i].head.extra=**0**;

cnt=**0**;

**for**(i=**0**;i<N;i++)

{

**if**(!adjl[i].head.extra)

{

cnt++;

n=**0**;

cout<<"Connected component "<<cnt<<": ";

s.push(i+**1**);

**while**(!s.isempty())

{

u=s.pop();

**if**(!adjl[u-**1**].head.extra)

{

n++;

cout<<u<<' ';

adjl[u-**1**].head.extra=**1**;

}

**for**(t=adjl[u-**1**].head.link,j=**0**;j<adjl[u-**1**].size();j++,t=t->link)

**if**(!adjl[t->data-**1**].head.extra)

s.push(t->data);

}

cout<<"[ Size: "<<n<<" ]";

cout<<endl;

}

}

cout<<"Number of connected components: "<<cnt<<endl;

}

**int** main()

{

**int** n, u, v;

**char** ch;

cout<<"How many vertices?: ";

cin>>n;

graph **g1**(n);

cout<<"Want to enter edge? <y/n>: ";

cin>>ch;

**while**(ch=='y'||ch=='Y')

{

cout<<"Enter edge: ";

cin>>u>>v;

g1.insert\_edge(u,v);

**if**(u!=v)

g1.insert\_edge(v,u);

cout<<"Want to enter more edges? <y/n>: ";

cin>>ch;

}

cout<<"Displaying adjacency lists for each vertex:"<<endl;

g1.Display();

cout<<"DFS traversal: ";

g1.DFS();

cout<<endl;

g1.connected\_components();

**return** **0**;

}

**Output:**

How many vertices?: 10

Graph created

Want to enter edge? <y/n>: y

Enter edge: 1 2

Want to enter more edges? <y/n>: y

Enter edge: 1 3

Want to enter more edges? <y/n>: y

Enter edge: 2 3

Want to enter more edges? <y/n>: y

Enter edge: 4 5

Want to enter more edges? <y/n>: y

Enter edge: 4 6

Want to enter more edges? <y/n>: y

Enter edge: 5 7

Want to enter more edges? <y/n>: y

Enter edge: 6 7

Want to enter more edges? <y/n>: y

Enter edge: 8 9

Want to enter more edges? <y/n>: n

Displaying adjacency lists for each vertex:

1: 3 --> 2 --> ||

2: 3 --> 1 --> ||

3: 2 --> 1 --> ||

4: 6 --> 5 --> ||

5: 7 --> 4 --> ||

6: 7 --> 4 --> ||

7: 6 --> 5 --> ||

8: 9 --> ||

9: 8 --> ||

10: ||

DFS traversal: 1 2 3 4 5 7 6 8 9 10

Connected component 1: 1 2 3 [ Size: 3 ]

Connected component 2: 4 5 7 6 [ Size: 4 ]

Connected component 3: 8 9 [ Size: 2 ]

Connected component 4: 10 [ Size: 1 ]

Number of connected components: 4

Graph destroyed

**Question (2)**

**Code:**

#include<iostream>

**using** **namespace** std;

**class** **node**

{

**public:**

**int** data;

node \*left, \*right;

node(**int**,node\*,node\*);

};

node::node(**int** d=**0**,node \*l=NULL,node \*r=NULL)

{

data=d;

left=l;

right=r;

}

**class** **tree**

{

node head;

**int** **depth\_of\_subtree**(node \*root);

**void** **Inorder**(node \*root);

**void** **Preorder**(node \*root);

**void** **create\_subtree**(node \*root);

**void** **deallocate\_nodes**(node \*root);

**public:**

tree();

~tree();

**int** **Find\_depth**();

**void** **Display**();

**void** **Longest\_path**();

**void** **create\_tree**();

**void** **delete\_tree**();

};

tree::tree()

{

cout<<"Binary tree created"<<endl;

}

tree::~tree()

{

deallocate\_nodes(head.left);

head.data=**0**;

head.left=head.right=NULL;

cout<<"Binary tree destroyed"<<endl;

}

**void** tree::create\_tree()

{

**char** ch;

**int** d;

cout<<"Does tree have root node? <y/n>: ";

cin>>ch;

**if**(ch=='y'||ch=='Y')

{

cout<<"Enter data for root node: ";

cin>>d;

head.left=head.right=**new** node(d,NULL,NULL);

head.data++;

create\_subtree(head.left);

}

}

**int** tree::Find\_depth()

{

**return** depth\_of\_subtree(head.left);

}

**void** tree::Display()

{

cout<<"Inorder traversal: ";

Inorder(head.left);

cout<<endl;

cout<<"Preorder traversal: ";

Preorder(head.left);

cout<<endl;

}

**void** tree::Longest\_path()

{

**int** n=Find\_depth();

**if**(n==**0**)

{

cout<<"No path possible! Tree is empty!"<<endl;

**return**;

}

**int** \*a=**new** **int**[n], i=**0**, d1, d2;

node \*root=head.left;

**while**(root!=NULL)

{

a[i]=root->data;

i++;

d1=depth\_of\_subtree(root->left);

d2=depth\_of\_subtree(root->right);

**if**(d1>=d2)

root=root->left;

**else**

root=root->right;

}

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

cout<<a[i]<<" -> ";

cout<<"||";

**delete** []a;

}

**int** tree::depth\_of\_subtree(node \*root)

{

**if**(root==NULL)

**return** **0**;

**int** d1=depth\_of\_subtree(root->left)+**1**, d2=depth\_of\_subtree(root->right)+**1**;

**if**(d1>=d2)

**return** d1;

**return** d2;

}

**void** tree::Inorder(node \*root)

{

**if**(root!=NULL)

{

Inorder(root->left);

cout<<root->data<<' ';

Inorder(root->right);

}

}

**void** tree::Preorder(node \*root)

{

**if**(root!=NULL)

{

cout<<root->data<<' ';

Preorder(root->left);

Preorder(root->right);

}

}

**void** tree::create\_subtree(node \*root)

{

**char** ch;

**int** d;

cout<<"Does node have left subtree? <y/n>: ";

cin>>ch;

**if**(ch=='y'||ch=='Y')

{

cout<<"Enter data: ";

cin>>d;

root->left=**new** node(d,NULL,NULL);

head.data++;

create\_subtree(root->left);

}

cout<<"Does node have right subtree? <y/n>: ";

cin>>ch;

**if**(ch=='y'||ch=='Y')

{

cout<<"Enter data: ";

cin>>d;

root->right=**new** node(d,NULL,NULL);

head.data++;

create\_subtree(root->right);

}

}

**void** tree::deallocate\_nodes(node \*root)

{

**if**(root!=NULL)

{

deallocate\_nodes(root->left);

deallocate\_nodes(root->right);

**delete** root;

}

}

**void** tree::delete\_tree()

{

deallocate\_nodes(head.left);

head.data=**0**;

head.left=head.right=NULL;

}

**int** main()

{

tree t1;

t1.create\_tree();

t1.Display();

cout<<"Depth: "<<t1.Find\_depth()<<endl;

cout<<"Longest path: ";

t1.Longest\_path();

cout<<endl;

**return** **0**;

}

**Output:**

Binary tree created

Does tree have root node? <y/n>: y

Enter data for root node: 1

Does node have left subtree? <y/n>: y

Enter data: 2

Does node have left subtree? <y/n>: y

Enter data: 4

Does node have left subtree? <y/n>: y

Enter data: 8

Does node have left subtree? <y/n>: n

Does node have right subtree? <y/n>: n

Does node have right subtree? <y/n>: y

Enter data: 9

Does node have left subtree? <y/n>: y

Enter data: 12

Does node have left subtree? <y/n>: n

Does node have right subtree? <y/n>: n

Does node have right subtree? <y/n>: y

Enter data: 13

Does node have left subtree? <y/n>: n

Does node have right subtree? <y/n>: n

Does node have right subtree? <y/n>: y

Enter data: 5

Does node have left subtree? <y/n>: n

Does node have right subtree? <y/n>: y

Enter data: 10

Does node have left subtree? <y/n>: n

Does node have right subtree? <y/n>: n

Does node have right subtree? <y/n>: y

Enter data: 3

Does node have left subtree? <y/n>: y

Enter data: 6

Does node have left subtree? <y/n>: n

Does node have right subtree? <y/n>: n

Does node have right subtree? <y/n>: y

Enter data: 7

Does node have left subtree? <y/n>: y

Enter data: 11

Does node have left subtree? <y/n>: y

Enter data: 14

Does node have left subtree? <y/n>: n

Does node have right subtree? <y/n>: n

Does node have right subtree? <y/n>: n

Does node have right subtree? <y/n>: n

Inorder traversal: 8 4 12 9 13 2 5 10 1 6 3 14 11 7

Preorder traversal: 1 2 4 8 9 12 13 5 10 3 6 7 11 14

Depth: 5

Longest path: 1 -> 2 -> 4 -> 9 -> 12 -> ||

Binary tree destroyed