Name: Swarnabh Paul

Section: Y

Roll no: 19CS8122

Assignment no: 5

Questions attempted: a,b

**Question (a)**

**Contents of Linked List header file:**

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

**class** **node**

{

**public:**

**int** data;

node \*link;

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

};

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

{

data=x;

link=l;

}

**class** **sll**

{

**bool** isempty();

**public:**

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

node head;

sll(node \*l);

sll(**const** sll&);

~sll();

**void** **deletesll**();

**void** **insertBeg**(**int**);

**void** **Delete**(**int**);

**bool** **search**(**int**);

**void** **display**();

**int** **size**();

};

sll::sll(node \*l=NULL)

{

head.data=**0**;

head.link=l;

**if**(l!=NULL)

{

**int** cnt=**1**;

node \*t=l;

**while**(t->link!=NULL)

{

t=t->link;

cnt++;

}

head.data=cnt;

}

cout<<"List constructed"<<endl;

}

sll::sll(**const** sll &s)

{

head.data=s.head.data;

head.link=NULL;

node \*t=s.head.link;

**if**(t!=NULL)

{

insertBeg(t->data);

head.data--;

t=t->link;

node \*p=head.link;

**for**(**int** i=**1**;i<s.head.data;i++,t=t->link,p=p->link)

p->link=createNewNode(t->data,NULL);

}

}

**void** sll::deletesll()

{

node \*t;

**for**(**int** i=**0**;i<head.data;i++)

{

t=head.link;

head.link=head.link->link;

**delete** t;

}

head.data=**0**;

}

**bool** sll::isempty()

{

**return** (head.data==**0**);

}

**void** sll::insertBeg(**int** x)

{

head.link=createNewNode(x,head.link);

head.data++;

}

node\* sll::createNewNode(**int** x,node \*l)

{

node \*t=**new** node(x,l);

**return** t;

}

sll::~sll()

{

deletesll();

cout<<"List destroyed"<<endl;

}

**void** sll::Delete(**int** x)

{

**if**(isempty())

{

cout<<"List is empty**\n**";

**return**;

}

node \*p=head.link;

node \*q;

**if**(p->data==x)

{

head.link=p->link;

**delete** p;

head.data--;

}

**else**

{

**while**(p!=NULL&&p->data!=x)

{

q=p;

p=p->link;

}

**if**(p==NULL)

cout<<"No match :: deletion failed**\n**";

**else**

{

q->link=p->link;

**delete** p;

head.data--;

}

}

}

**bool** sll::search(**int** x)

{

node \*t=head.link;

**int** i;

**for**(i=**0**;i<head.data;i++,t=t->link)

**if**(t->data==x)

**return** true;

**return** false;

}

**void** sll::display()

{

node \*t=head.link;

**for**(**int** i=**0**;i<head.data;i++,t=t->link)

{

cout<<t->data<<" --> ";

}

cout<<"||"<<endl;

}

**int** sll::size()

{

**return** head.data;

}

**Contents of Stack header file:**

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

**class** **stack**

{

**private:**

**int** top, size;

**int** \*a;

**bool** **isfull**();

**void** **initialize**(**int**);

**void** **deconstruct**();

**public:**

stack(**int**);

stack(**int**,**int**);

stack();

~stack();

**bool** **isempty**();

**void** **push**(**int**);

**int** **pop**();

**void** **display**();

};

**void** stack::push(**int** x)

{

**if**(isfull())

{

cout<<"Stack overflow!!**\n**";

**return**;

}

top++;

a[top]=x;

}

**int** stack::pop()

{

**if**(isempty())

{

cout<<"Stack underflow!!**\n**";

**return** -**1**;

}

**int** x=a[top];

top--;

**return** x;

}

stack::stack(**int** n)

{

initialize(n);

cout<<"Constructed stack of size "<<n<<endl;

}

stack::stack(**int** n,**int** x)

{

initialize(n);

cout<<"Constructed stack of size "<<n<<endl;

top++;

a[top]=x;

}

stack::stack()

{

**int** n=**10**;

initialize(n);

cout<<"Constructed stack of size "<<n<<endl;

}

stack::~stack()

{

deconstruct();

cout<<"Destroyed stack of size "<<size<<endl;

}

**void** stack::display()

{

**if**(isempty())

{

cout<<"Stack is empty**\n**";

**return**;

}

cout<<"Displaying stack from top to bottom:**\n**";

**for**(**int** i=top;i>=**0**;i--)

cout<<a[i]<<' ';

cout<<endl;

}

**bool** stack::isempty()

{

**return** (top==-**1**);

}

**bool** stack::isfull()

{

**return** (top==(size-**1**));

}

**void** stack::initialize(**int** n)

{

a=**new** **int**[n];

top=-**1**;

size=n;

}

**void** stack::deconstruct()

{

**delete** []a;

}

**Code:**

#include<iostream>

#include<stdbool.h>

#include"MyLinkedList.h"

#include"MyStack.h"

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

**class** **Graph**

{

sll \*g;

**int** V;

**int** **minDist**(**int**\*,**bool**\*);

**public:**

Graph(**int**);

~Graph();

Graph(**const** Graph&);

**void** **insertEdge**(**int**,**int**);

**void** **deleteEdge**(**int**,**int**);

**void** **displayGraph**();

**void** **dfs**();

**void** **connectedComponents**();

**int** **shortestPath**(**int**,**int**);

};

Graph::Graph(**int** n=**10**)

{

V=n;

g=**new** sll[n];

cout<<"Graph constructed with "<<V<<" vertices"<<endl;

}

Graph::~Graph()

{

**for**(**int** i=**0**;i<V;i++)

g[i].deletesll();

**delete** []g;

cout<<"Graph with "<<V<<" vertices destroyed"<<endl;

}

**void** Graph::insertEdge(**int** u,**int** v)

{

**if**(u<**1**||u>V||v<**1**||v>V)

{

cout<<"Wrong input. Insertion not possible."<<endl;

**return**;

}

**if**(u!=v)

{

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

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

}

**else**

{

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

}

}

**void** Graph::displayGraph()

{

**for**(**int** i=**0**;i<V;i++)

{

cout<<"Adjacency list of vertex "<<i+**1**<<": ";

g[i].display();

}

}

**void** Graph::connectedComponents()

{

stack s(V\*V);

**int** top;

node \*t;

**bool** \*visited=**new** **bool**[V];

**int** cnt=**0**, N=**0**;

**for**(**int** i=**0**;i<V;i++)

visited[i]=false;

**for**(**int** i=**0**;i<V;i++)

{

**if**(!visited[i])

{

cnt++;

N=**0**;

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

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

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

{

top=s.pop();

**if**(!visited[top-**1**])

{

cout<<top<<' ';

visited[top-**1**]=true;

N++;

}

t=g[top-**1**].head.link;

**for**(**int** i=**0**;i<g[top-**1**].head.data;i++,t=t->link)

**if**(!visited[t->data-**1**])

{

s.push(t->data);

}

}

cout<<"[Size: "<<N<<"]"<<endl;

}

}

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

**delete** []visited;

}

**void** Graph::dfs()

{

stack s(V\*V);

**int** top;

node \*t;

**bool** \*visited=**new** **bool**[V];

**for**(**int** i=**0**;i<V;i++)

visited[i]=false;

**for**(**int** i=**0**;i<V;i++)

{

**if**(!visited[i])

{

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

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

{

top=s.pop();

**if**(!visited[top-**1**])

{

cout<<top<<' ';

visited[top-**1**]=true;

}

t=g[top-**1**].head.link;

**for**(**int** i=**0**;i<g[top-**1**].head.data;i++,t=t->link)

**if**(!visited[t->data-**1**])

{

s.push(t->data);

}

}

}

}

cout<<endl;

**delete** []visited;

}

**void** Graph::deleteEdge(**int** u,**int** v)

{

**if**(u<**1**||u>V||v<**1**||v>V)

{

cout<<"Wrong input. Deletion not possible."<<endl;

**return**;

}

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

{

cout<<"Edge does not exist. Deletion not possible."<<endl;

**return**;

}

**else**

{

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

**if**(u!=v)

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

}

}

**int** Graph::shortestPath(**int** u,**int** v)

{

**int** \*dist=**new** **int**[V];

**bool** \*spt=**new** **bool**[V];

node \*t;

**for**(**int** i=**0**;i<V;i++)

{

dist[i]=-**1**;

spt[i]=false;

}

dist[u-**1**]=**0**;

**for**(**int** i=**0**;i<V-**1**;i++)

{

u=minDist(dist,spt);

spt[u-**1**]=true;

t=g[u-**1**].head.link;

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

{

**if**((dist[t->data-**1**]==-**1**||(dist[t->data-**1**]>dist[u-**1**]+**1**))&&(dist[u-**1**]!=-**1**))

dist[t->data-**1**]=dist[u-**1**]+**1**;

}

}

**int** result=dist[v-**1**];

**delete** []spt;

**delete** []dist;

**return** result;

}

**int** Graph::minDist(**int** \*dist,**bool** \*spt)

{

**int** m=-**1**, index;

**for**(**int** i=**0**;i<V;i++)

{

**if**(!spt[i])

{

**if**(dist[i]==-**1**&&m==-**1**)

index=i;

**else** **if**((dist[i]<m||m==-**1**)&&dist[i]!=-**1**)

{

index=i;

m=dist[i];

}

}

}

**return** index+**1**;

}

Graph::Graph(**const** Graph &c)

{

V=c.V;

g=**new** sll[c.V];

node \*t, \*p;

**for**(**int** i=**0**;i<c.V;i++)

{

g[i].head.data=c.g[i].head.data;

g[i].head.link=NULL;

t=c.g[i].head.link;

**if**(t!=NULL)

{

g[i].head.link=g[i].createNewNode(t->data,NULL);

t=t->link;

p=g[i].head.link;

**for**(**int** j=**1**;j<c.g[i].head.data;j++,t=t->link,p=p->link)

p->link=g[i].createNewNode(t->data,NULL);

}

}

}

**int** main()

{

Graph g1(**9**);

g1.insertEdge(**1**,**2**);

g1.insertEdge(**2**,**3**);

g1.insertEdge(**3**,**4**);

g1.insertEdge(**4**,**5**);

g1.insertEdge(**5**,**1**);

g1.insertEdge(**2**,**5**);

g1.insertEdge(**2**,**4**);

g1.insertEdge(**6**,**7**);

g1.insertEdge(**7**,**7**);

g1.insertEdge(**7**,**8**);

g1.insertEdge(**8**,**9**);

cout<<"Displaying g1:**\n**";

g1.displayGraph();

cout<<"DFS of g1:**\n**";

g1.dfs();

Graph g2=g1;

g1.deleteEdge(**7**,**7**);

g1.deleteEdge(**7**,**8**);

cout<<"Displaying g1:**\n**";

g1.displayGraph();

cout<<"DFS of g1:**\n**";

g1.dfs();

cout<<"Connected components in g1:**\n**";

g1.connectedComponents();

**int** dist=g1.shortestPath(**1**,**4**);

**if**(dist!=-**1**)

cout<<"Length of shortest path between vertices 1 and 4 in g1 is: "<<dist<<endl;

**else**

cout<<"Path does not exist"<<endl;

cout<<"Displaying g2:**\n**";

g2.displayGraph();

cout<<"DFS of g2:**\n**";

g2.dfs();

cout<<"Connected components in g2:**\n**";

g2.connectedComponents();

**return** **0**;

}

**Output:**

List constructed

List constructed

List constructed

List constructed

List constructed

List constructed

List constructed

List constructed

List constructed

Graph constructed with 9 vertices

Displaying g1:

Adjacency list of vertex 1: 5 --> 2 --> ||

Adjacency list of vertex 2: 4 --> 5 --> 3 --> 1 --> ||

Adjacency list of vertex 3: 4 --> 2 --> ||

Adjacency list of vertex 4: 2 --> 5 --> 3 --> ||

Adjacency list of vertex 5: 2 --> 1 --> 4 --> ||

Adjacency list of vertex 6: 7 --> ||

Adjacency list of vertex 7: 8 --> 7 --> 6 --> ||

Adjacency list of vertex 8: 9 --> 7 --> ||

Adjacency list of vertex 9: 8 --> ||

DFS of g1:

Constructed stack of size 81

1 2 3 4 5 6 7 8 9

Destroyed stack of size 81

List constructed

List constructed

List constructed

List constructed

List constructed

List constructed

List constructed

List constructed

List constructed

Displaying g1:

Adjacency list of vertex 1: 5 --> 2 --> ||

Adjacency list of vertex 2: 4 --> 5 --> 3 --> 1 --> ||

Adjacency list of vertex 3: 4 --> 2 --> ||

Adjacency list of vertex 4: 2 --> 5 --> 3 --> ||

Adjacency list of vertex 5: 2 --> 1 --> 4 --> ||

Adjacency list of vertex 6: 7 --> ||

Adjacency list of vertex 7: 6 --> ||

Adjacency list of vertex 8: 9 --> ||

Adjacency list of vertex 9: 8 --> ||

DFS of g1:

Constructed stack of size 81

1 2 3 4 5 6 7 8 9

Destroyed stack of size 81

Connected components in g1:

Constructed stack of size 81

Connected component 1: 1 2 3 4 5 [Size: 5]

Connected component 2: 6 7 [Size: 2]

Connected component 3: 8 9 [Size: 2]

Number of connected components: 3

Destroyed stack of size 81

Length of shortest path between vertices 1 and 4 in g1 is: 2

Displaying g2:

Adjacency list of vertex 1: 5 --> 2 --> ||

Adjacency list of vertex 2: 4 --> 5 --> 3 --> 1 --> ||

Adjacency list of vertex 3: 4 --> 2 --> ||

Adjacency list of vertex 4: 2 --> 5 --> 3 --> ||

Adjacency list of vertex 5: 2 --> 1 --> 4 --> ||

Adjacency list of vertex 6: 7 --> ||

Adjacency list of vertex 7: 8 --> 7 --> 6 --> ||

Adjacency list of vertex 8: 9 --> 7 --> ||

Adjacency list of vertex 9: 8 --> ||

DFS of g2:

Constructed stack of size 81

1 2 3 4 5 6 7 8 9

Destroyed stack of size 81

Connected components in g2:

Constructed stack of size 81

Connected component 1: 1 2 3 4 5 [Size: 5]

Connected component 2: 6 7 8 9 [Size: 4]

Number of connected components: 2

Destroyed stack of size 81

List destroyed

List destroyed

List destroyed

List destroyed

List destroyed

List destroyed

List destroyed

List destroyed

List destroyed

Graph with 9 vertices destroyed

List destroyed

List destroyed

List destroyed

List destroyed

List destroyed

List destroyed

List destroyed

List destroyed

List destroyed

Graph with 9 vertices destroyed

**Question (b)**

**Code:**

#include<iostream>

#include<stdbool.h>

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

**class** **node**

{

**public:**

**int** data;

node \*left, \*right;

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

};

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

{

data=x;

left=l;

right=r;

}

**class** **BST**

{

node root;

**void** **deleteTreeRecursively**(node\*);

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

**void** **inorder**(node\*);

**bool** **isempty**();

**int** **depth**(node\*);

node\* **minValueNode**(node\*);

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

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

**void** **postorder**(node\*);

node\* **copytree**(node\*);

**public:**

BST();

~BST();

BST(**const** BST&);

**void** **deletebst**();

**void** **insert**(**int**);

**void** **display**(); //uses in-order traversal

node\* **search**(**int**);

**int** **findheight**();

**void** **Delete**(**int**);

**void** **bfs**(); //uses level-order traversal

**void** **dfs**(); //uses post-order traversal

};

**bool** BST::isempty()

{

**return** (root.data==**0**);

}

node\* BST::createNewNode(**int** x,node \*l,node \*r)

{

node \*t=**new** node(x,l,r);

**return** t;

}

BST::BST()

{

root.data=**0**;

root.left=root.right=NULL;

cout<<"BST constructed"<<endl;

}

BST::~BST()

{

deletebst();

cout<<"BST destroyed"<<endl;

}

**void** BST::deletebst()

{

deleteTreeRecursively(root.right);

root.data=**0**;

root.left=root.right=NULL;

}

**void** BST::deleteTreeRecursively(node \*n)

{

**if**(n!=NULL)

{

deleteTreeRecursively(n->left);

deleteTreeRecursively(n->right);

**delete** n;

}

}

**void** BST::insert(**int** x)

{

**if**(root.data==**0**)

{

root.left=root.right=createNewNode(x,NULL,NULL);

}

**else**

{

node \*p=root.right, \*q;

**while**(p!=NULL)

{

q=p;

**if**(x<p->data)

p=p->left;

**else**

p=p->right;

}

**if**(x<q->data)

q->left=createNewNode(x,NULL,NULL);

**else**

q->right=createNewNode(x,NULL,NULL);

}

root.data++;

}

**void** BST::display()

{

**if**(isempty())

{

cout<<"Tree is empty"<<endl;

**return**;

}

inorder(root.right);

cout<<endl;

}

**void** BST::inorder(node \*r)

{

**if**(r!=NULL)

{

inorder(r->left);

cout<<r->data<<' ';

inorder(r->right);

}

}

node\* BST::search(**int** x)

{

node \*t=root.right;

**while**(t!=NULL)

{

**if**(x<t->data)

t=t->left;

**else** **if**(x>t->data)

t=t->right;

**else**

**return** t;

}

**return** t;

}

**int** BST::findheight()

{

**return** depth(root.right);

}

**int** BST::depth(node \*n)

{

**if**(n==NULL)

**return** **0**;

**else**

{

**int** l=depth(n->left), r=depth(n->right);

**if**(l>r)

**return** l+**1**;

**else**

**return** r+**1**;

}

}

**void** BST::Delete(**int** x)

{

**if**(isempty())

{

cout<<"List is empty. Deletion not possible."<<endl;

**return**;

}

node \*n=search(x);

**if**(n==NULL)

{

cout<<"Element not found. Deletion not possible."<<endl;

**return**;

}

node \*r, \*parent;

**if**(n==root.right&&(n->left==NULL||n->right==NULL))

{

r=n;

**if**(n->left==NULL)

{

r=n->right;

}

**else**

{

r=n->left;

}

**delete** n;

root.left=root.right=r;

root.data--;

}

**else**

{

parent=searchparent(root.right,x);

**if**(n->left==NULL)

{

**if**(parent->right==n)

parent->right=n->right;

**else**

parent->left=n->right;

**delete** n;

}

**else** **if**(n->right==NULL)

{

**if**(parent->right==n)

parent->right=n->left;

**else**

parent->left=n->left;

**delete** n;

}

**else**

{

node \*t=minValueNode(n->right);

**int** temp=t->data;

parent=searchparent(root.right,temp);

**if**(t->left==NULL)

{

**if**(parent->right==t)

parent->right=t->right;

**else**

parent->left=t->right;

**delete** t;

}

**else** **if**(t->right==NULL)

{

**if**(parent->right==t)

parent->right=t->left;

**else**

parent->left=t->left;

**delete** t;

}

n->data=temp;

}

root.data--;

}

}

node\* BST::minValueNode(node \*t)

{

**if**(t==NULL)

**return** NULL;

**else**

{

**while**(t->left!=NULL)

t=t->left;

**return** t;

}

}

node\* BST::searchparent(node \*n,**int** v)

{

**if**(n==NULL)

**return** NULL;

**else**

{

**if**(n->left!=NULL&&n->left->data==v)

**return** n;

**else** **if**(n->right!=NULL&&n->right->data==v)

**return** n;

node \*l=searchparent(n->left,v);

**if**(l!=NULL)

**return** l;

**else**

**return** **searchparent**(n->right,v);

}

}

**void** BST::bfs()

{

**if**(isempty())

{

cout<<"Tree is empty"<<endl;

**return**;

}

**int** h=findheight();

**for**(**int** i=**1**;i<=h;i++)

printlevel(root.right,i);

cout<<endl;

}

**void** BST::printlevel(node \*n,**int** l)

{

**if**(n==NULL)

{

**return**;

}

**if**(l==**1**)

cout<<n->data<<' ';

**else**

{

printlevel(n->left,l-**1**);

printlevel(n->right,l-**1**);

}

}

**void** BST::dfs()

{

**if**(isempty())

{

cout<<"Tree is empty"<<endl;

**return**;

}

postorder(root.right);

cout<<endl;

}

**void** BST::postorder(node \*n)

{

**if**(n!=NULL)

{

postorder(n->left);

postorder(n->right);

cout<<n->data<<' ';

}

}

BST::BST(**const** BST &b)

{

root.data=b.root.data;

root.left=root.right=copytree(b.root.right);

}

node\* BST::copytree(node \*b)

{

**if**(b!=NULL)

{

node \*t=createNewNode(b->data,NULL,NULL);

t->left=copytree(b->left);

t->right=copytree(b->right);

**return** t;

}

**else**

{

**return** NULL;

}

}

**int** main()

{

BST t1;

t1.insert(**3**);

t1.insert(**12**);

t1.insert(**18**);

t1.insert(**22**);

t1.insert(**7**);

t1.insert(**0**);

t1.insert(-**98**);

t1.insert(**123**);

t1.insert(**77**);

cout<<"Displaying t1: ";

t1.display();

cout<<"BFS of t1: ";

t1.bfs();

cout<<"DFS of t1: ";

t1.dfs();

BST t2=t1;

t1.Delete(**3**);

cout<<"Displaying t1: ";

t1.display();

cout<<"Displaying t2: ";

t2.display();

cout<<"Searching for 12 in t1.**\n**";

**if**(t1.search(**12**)==NULL)

cout<<"Search unsuccessful. Element not found."<<endl;

**else**

cout<<"Search successful. Element found."<<endl;

cout<<"Height of tree t1 is: "<<t1.findheight()<<endl;

**return** **0**;

}

**Output:**

BST constructed

Displaying t1: -98 0 3 7 12 18 22 77 123

BFS of t1: 3 0 12 -98 7 18 22 123 77

DFS of t1: -98 0 7 77 123 22 18 12 3

Displaying t1: -98 0 7 12 18 22 77 123

Displaying t2: -98 0 3 7 12 18 22 77 123

Searching for 12 in t1.

Search successful. Element found.

Height of tree t1 is: 6

BST destroyed

BST destroyed