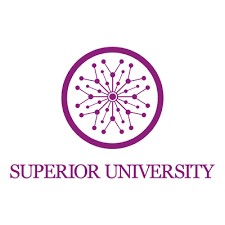
**TASK NO 13**



**M ZUHAIB ANWAR**

**SU92-BSSEM-S24-005**

**SE-3A**

**DSA LAB**

//DFS TREE

#include<iostream>

using namespace std;

struct TreeNode

{

int val;

TreeNode\*left;

TreeNode\*right;

TreeNode(int x):val(x),left(nullptr),right(nullptr){}

};

class TreeDFS

{

TreeNode\*root;

TreeNode\*insertHelper(TreeNode\*node,int val)

{

if(!node)

return new TreeNode(val);

if(val<node->val)

node->left=insertHelper(node->left,val);

else if(val>node->val)node->right=insertHelper(node->right,val);

return node;

}

public:

TreeDFS():root(nullptr){}

void insert(int val)

{

root=insertHelper(root,val);

}

void dfsPreorder()

{

if(!root)return;

class stack{

struct Node

{

TreeNode\*data;

Node\*next;

Node(TreeNode\*d):data(d),next(nullptr){}

};

Node\*top;

public:

stack():top(nullptr){}

void push(TreeNode\*d){Node\*n=new Node(d);n->next=top;top=n;}

void pop()

{if(top)

{

Node\*temp=top;top=top->next;delete temp;

}}

TreeNode\* peek(){return top?top->data:nullptr;}

bool empty(){return top==nullptr;}

}s;

s.push(root);

while(!s.empty())

{

TreeNode\*current=s.peek();

s.pop();

cout<<current->val<<" ";

if(current->right)s.push(current->right);

if(current->left)s.push(current->left);

}

cout<<endl;

}

};

int main()

{

TreeDFS dfs;

dfs.insert(5);

dfs.insert(3);

dfs.insert(7);

dfs.insert(2);

dfs.insert(4);

dfs.insert(6);

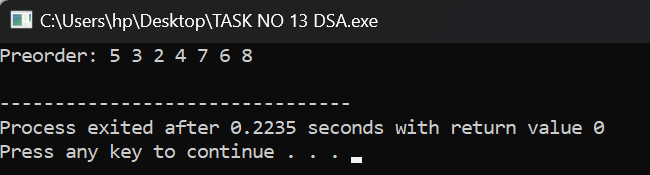
dfs.insert(8);

cout<<"Preorder: ";

dfs.dfsPreorder();

return 0;

}



//DFS GRAPH

#include<iostream>

using namespace std;

class GraphDFS

{

int V;

struct Node

{

int data;

Node\*next;

Node(int d):data(d),next(nullptr){}

};

Node\*\*adj;

void dfsUtil(int v,bool\*visited)

{

visited[v]=true;

cout<<v<<" ";

for(Node\*curr=adj[v];curr;curr=curr->next)

if(!visited[curr->data])dfsUtil(curr->data,visited);

}

public:

GraphDFS(int V):V(V)

{

adj=new Node\*[V];

for(int i=0;i<V;i++)adj[i]=nullptr;

}

void addEdge(int v,int w)

{

Node\*n=new Node(w);

n->next=adj[v];

adj[v]=n;

}

void dfs(int start)

{

bool\*visited=new bool[V]{false};

dfsUtil(start,visited);

cout<<endl;

delete[] visited;

}

};

int main(){

GraphDFS g(5);

g.addEdge(0,1);

g.addEdge(0,2);

g.addEdge(1,3);

g.addEdge(2,4);

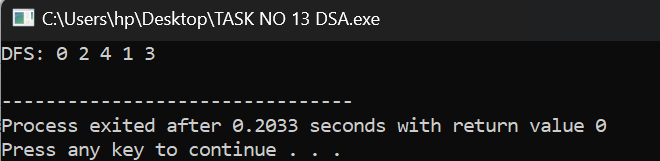
g.addEdge(3,4);

cout<<"DFS: ";

g.dfs(0);

return 0;

}



//bfs tree

#include<iostream>

using namespace std;

struct TreeNode

{

int val;

TreeNode\*left;

TreeNode\*right;

TreeNode(int x):val(x),left(nullptr),right(nullptr){}

};

class TreeBFS

{

TreeNode\*root;

TreeNode\*insertHelper(TreeNode\*node,int val)

{

if(!node)

return new

TreeNode(val);

if(val<node->val)

node->left=insertHelper

(node->left,val);

else if(val>node->val)

node->right=insertHelper

(node->right,val);

return node;

}

public:

TreeBFS():root(nullptr){}

void insert(int val)

{

root=insertHelper(root,val);

}

void bfs()

{

if(!root)

return;

class queue

{

struct Node{

TreeNode\*data;

Node\*next;

Node(TreeNode\*d):data(d),next(nullptr){}

};

Node \*front,\*rear;

public:

queue():front(nullptr),rear(nullptr){}

void push(TreeNode\*d){

Node\*n=new Node(d);

if(!rear)

front=rear=n;

else

{

rear->next=n;rear=n;

}

}

void pop()

{

if(front)

{

Node\*temp=front;

front=front->next;

if(!front)

rear=nullptr;

delete temp;

}

}

TreeNode\* peek()

{

return front?front->data:nullptr;

}

bool empty()

{

return front==nullptr;

}

}q;

q.push(root);

while(!q.empty())

{

TreeNode\*current=q.peek();

q.pop();

cout<<current->val<<" ";

if(current->left)

q.push

(current->left);

if(current->right)

q.push

(current->right);

}

cout<<endl;

}

};

int main(){

TreeBFS bfs1;

bfs1.insert(5);

bfs1.insert(3);

bfs1.insert(7);

bfs1.insert(2);

bfs1.insert(4);

bfs1.insert(6);

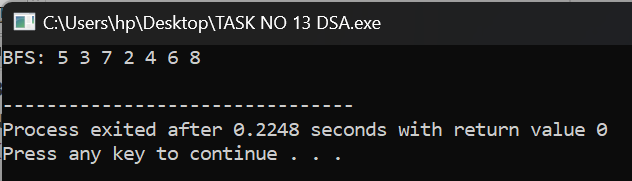
bfs1.insert(8);

cout<<"BFS: ";

bfs1.bfs();

return 0;

}



#include<iostream>

using namespace std;

class GraphBFS

{

int V;

struct Node

{

int data;

Node\*next;

Node(int d):data(d),next(nullptr){}

};

Node\*\*adj;

public:

GraphBFS(int V):V(V)

{

adj=new Node\*[V];

for(int i=0;i<V;i++)adj[i]=nullptr;

}

void addEdge(int v,int w)

{

Node\*n=new Node(w);

n->next=adj[v];

adj[v]=n;

}

void bfs(int start)

{

bool\*visited=new bool[V]{false};

class queue

{

struct QNode

{

int data;

QNode\*next;

QNode(int d):data(d),next(nullptr){}

};

QNode \*front,\*rear;

public:

queue():front(nullptr),rear(nullptr){}

void push(int d)

{

QNode\*n=new QNode(d);

if(!rear)

front=rear=n;

else

{

rear->next=n;

rear=n;

}

}

void pop()

{

if(front)

{

QNode\*temp=front;

front=front->next;

if(!front)

rear=nullptr;

delete temp;

}

}

int peek()

{

return

front?front->data:-1;

}

bool empty()

{

return

front==nullptr;

}

}q;

visited[start]=true;

q.push(start);

while(!q.empty())

{

int v=q.peek();

q.pop();

cout<<v<<" ";

for(Node\*curr=adj[v];curr;curr=curr->next)

{

if(!visited[curr->data])

{

visited[curr->data]=true;

q.push(curr->data);

}

}

}

cout<<endl;

delete[] visited;

}

};

int main(){

GraphBFS bg(5);

bg.addEdge(0,1);

bg.addEdge(0,2);

bg.addEdge(1,3);

bg.addEdge(2,4);

bg.addEdge(3,4);

cout<<"BFS: ";

bg.bfs(0);

return 0;

}

