Assignment No. 1

**BFS**

Program-

#include<iostream> #include<stdlib.h> #include<queue> using namespace std; class node

{

public:

node \*left, \*right; int data;

};

class Breadthfs

{

public:

node \*insert(node \*, int); void bfs(node \*);

};

node \*insert(node \*root, int data)

// inserts a node in tree

{

if(!root)

{

root=new node; root->left=NULL; root->right=NULL; root->data=data; return root;

}

queue<node \*> q; q.push(root);

while(!q.empty())

{

node \*temp=q.front(); q.pop();

if(temp->left==NULL)

{

}

else

{

temp->left=new node; temp->left->left=NULL; temp->left->right=NULL; temp->left->data=data; return root;

q.push(temp->left);

}

if(temp->right==NULL)

{

}

else

{

temp->right=new node; temp->right->left=NULL; temp->right->right=NULL; temp->right->data=data; return root;

q.push(temp->right)

}

}

}

void bfs(node \*head)

{

queue<node\*> q;

q.push(head);

int qSize;

while (!q.empty())

{

qSize = q.size(); #pragma omp parallel for

//creates parallel threads for (int i = 0; i < qSize; i++)

{

node\* currNode; #pragma omp critical

{

currNode = q.front(); q.pop(); cout<<"\t"<<currNode->data;

}// prints parent node #pragma omp critical

{

if(currNode->left)// push parent's left node in queue q.push(currNode->left);

if(currNode->right)

q.push(currNode->right);

}// push parent's right node in queue

}

}

}

int main(){

node \*root=NULL; int data;

char ans;

do

{

cout<<"\n enter data=>"; cin>>data;

root=insert(root,data);

cout<<"do you want insert one more node?"; cin>>ans;

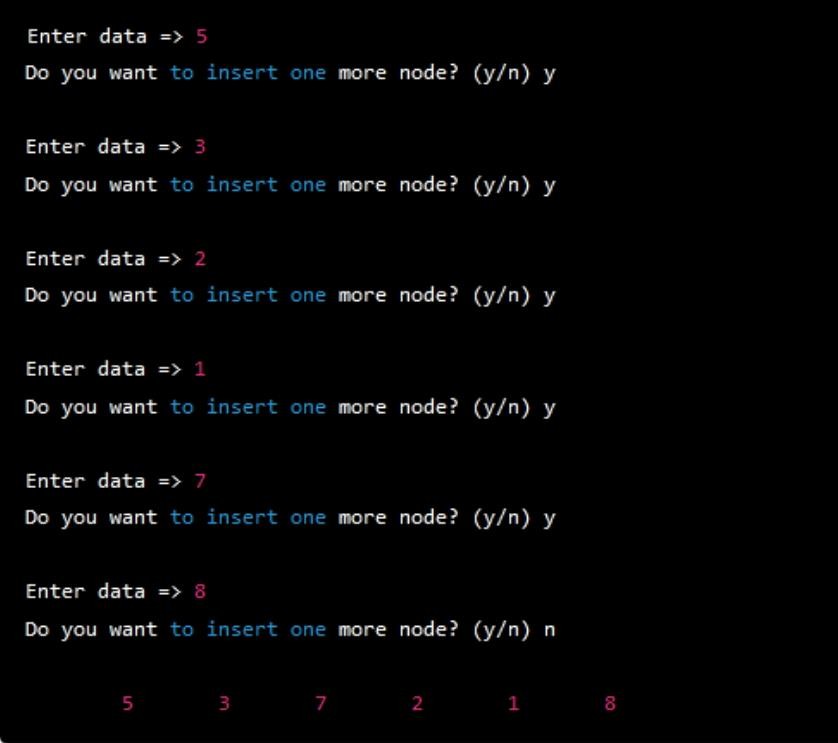
}while(ans=='y'||ans=='Y');

bfs(root); return 0;

}

Run Commands:

1. g++ -fopenmp bfs.cpp -o bfs
2. ./bfs

OUTPUT-

DFS

Program-

#include <iostream> #include <vector> #include <stack> #include <omp.h>

using namespace std; const int MAX = 100000; vector<int> graph[MAX]; bool visited[MAX];

void dfs(int node) {

stack<int> s; s.push(node);

while (!s.empty()) {

int curr\_node = s.top(); s.pop();

if (!visited[curr\_node]) { visited[curr\_node] = true;

if (visited[curr\_node]) { cout << curr\_node << " ";

}

#pragma omp parallel for

for (int i = 0; i < graph[curr\_node].size(); i++) { int adj\_node = graph[curr\_node][i];

if (!visited[adj\_node]) { s.push(adj\_node);

}

}

}

}

}

int main() {

int n, m, start\_node;

cout << "Enter No of Node,Edges,and start node:" ; cin >> n >> m >> start\_node;

//n: node,m:edges

cout << "Enter Pair of edges:" ;

for (int i = 0; i < m; i++) { int u, v;

cin >> u >> v;

//u and v: Pair of edges graph[u].push\_back(v); graph[v].push\_back(u);

}

#pragma omp parallel for for (int i = 0; i < n; i++) { visited[i] = false;

}

dfs(start\_node);

/\* for (int i = 0; i < n; i++) { if (visited[i]) {

cout << i << " ";

}

}\*/

return 0;

}

