

DFS

```
#include<bits/stdc++.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);
```

```

return 0;
}

```

The screenshot shows a C++ IDE with a file named `dfs.cpp` open. The code implements a Depth-First Search (DFS) algorithm on a graph. It includes a stack to manage the nodes, a visited array to track visited nodes, and a recursive function `dfs` that explores the graph. The main function prompts the user to enter the number of nodes (`n`), edges (`m`), and a start node, then calls the `dfs` function. The output window shows the execution results for a graph with 4 nodes and 6 edges, starting from node 0. The output displays the nodes visited in the order: 0, 2, 1, 2, 2, 0, 3, 3, 0, 1, 2, 3.

```

Terminal
dfs.cpp x
1 #include<bits/stdc++.h>
2
3 using namespace std;
4
5 const int MAX = 100000;
6 vector<int> graph[MAX];
7 bool visited[MAX];
8
9 void dfs(int node) {
10     stack<int> s;
11     s.push(node);
12
13     while (!s.empty()) {
14         int curr_node = s.top();
15         s.pop();
16
17         if (!visited[curr_node]) {
18             visited[curr_node] = true;
19
20             if (visited[curr_node]) {
21                 cout << curr_node << " ";
22             }
23
24             #pragma omp parallel for
25             for (int i = 0; i < graph[curr_node].size(); i++) {
26                 int adj_node = graph[curr_node][i];
27                 if (!visited[adj_node]) {
28                     s.push(adj_node);
29                 }
30             }
31         }
32     }
33 }
34
35 int main() {
36     int n, m, start_node;
37     cout << "Enter No of Node,Edges,and start node:" ;
38     cin >> n >> m >> start_node;
39     //n: node,m:edges
40
41     cout << "Enter Pair of edges:" ;
42     for (int i = 0; i < m; i++) {

```

```

mmcoe@mmcoe-System-Product-Name: ~/Desktop/HPC
mmcoe@mmcoe-System-Product-Name:~/Desktop/HPC$ g++ -fopenmp dfs.cpp -o dfs
mmcoe@mmcoe-System-Product-Name:~/Desktop/HPC$ ./dfs
Enter No of Node,Edges,and start node:4 6 0
Enter Pair of edges:0 1
0 2
1 2
2 0
2 3
3 3
0 1 2 3 mmcoe@mmcoe-System-Product-Name:~/Desktop/HPC$

```

BFS

```

#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)
    {
        temp->left=new node;
        temp->left->left=NULL;
        temp->left->right=NULL;
        temp->left->data=data;
        return root;
    }
    else
    {
        q.push(temp->left);
    }
    if(temp->right==NULL)
    {
        temp->right=new node;
        temp->right->left=NULL;
        temp->right->right=NULL;
        temp->right->data=data;
        return root;
    }
    else
    {
        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
            {

```

```

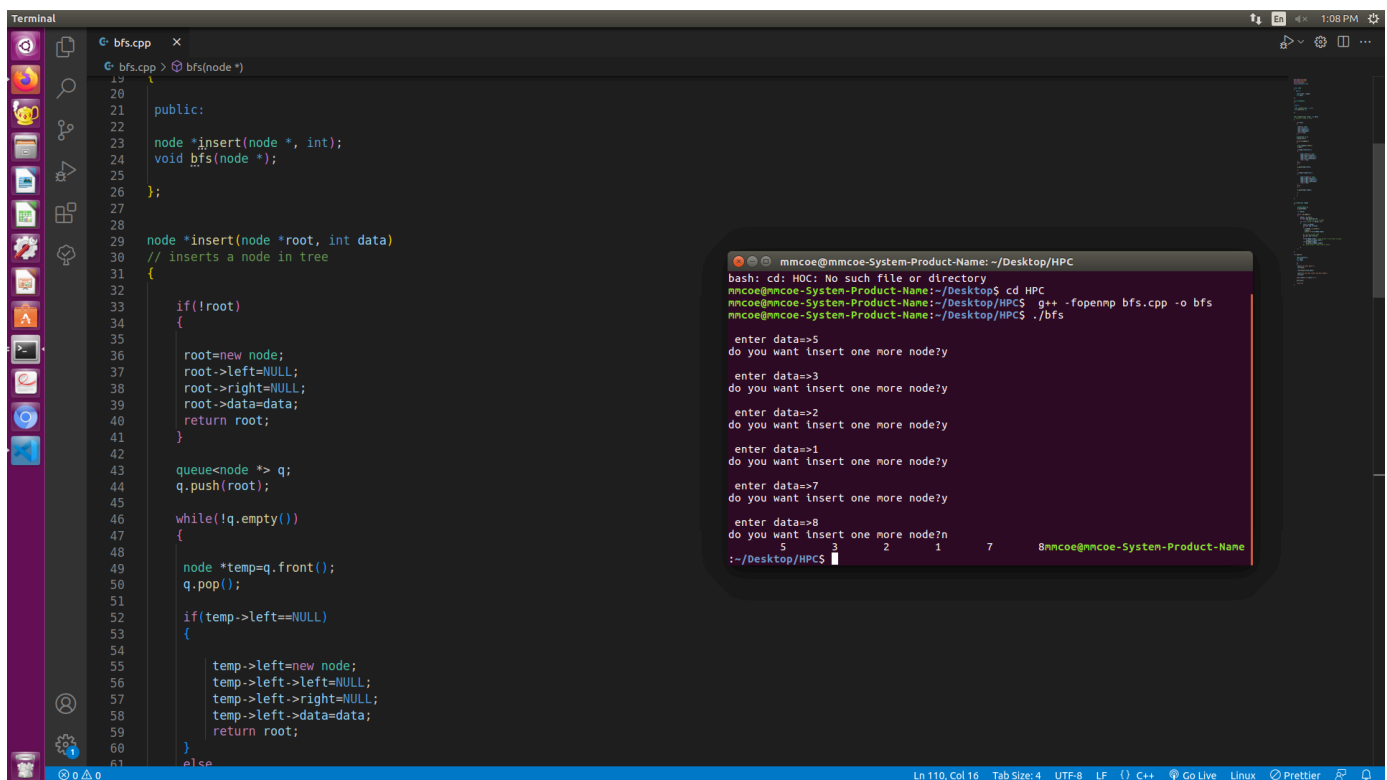
        currNode = q.front();
        q.pop();
        cout<<"\t"<<currNode->data;

    }// 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;
}

```



The screenshot displays a C++ IDE with a file named `bfs.cpp`. The code implements a Binary Search Tree (BST) using Breadth-First Search (BFS). It includes functions for inserting a node and performing a BFS traversal. The terminal window shows the program's execution with input data (5, 3, 2, 1, 7) and the resulting tree structure printed as a level-order traversal (5 3 2 1 7).

```

bfs.cpp
bfs(node *)
19
20
21 public:
22
23 node *insert(node *, int);
24 void bfs(node *);
25
26 };
27
28
29 node *insert(node *root, int data)
30 // inserts a node in tree
31 {
32
33     if(!root)
34     {
35
36         root=new node;
37         root->left=NULL;
38         root->right=NULL;
39         root->data=data;
40         return root;
41     }
42
43     queue<node *> q;
44     q.push(root);
45
46     while(!q.empty())
47     {
48
49         node *temp=q.front();
50         q.pop();
51
52         if(temp->left==NULL)
53         {
54
55             temp->left=new node;
56             temp->left->left=NULL;
57             temp->left->right=NULL;
58             temp->left->data=data;
59             return root;
60         }
61         else

```

```

mmcoe@mmcoe-System-Product-Name: ~/Desktop/HPC
bash: cd: HOC: No such file or directory
mmcoe@mmcoe-System-Product-Name: ~/Desktop$ cd HPC
mmcoe@mmcoe-System-Product-Name: ~/Desktop/HPC$ g++ -fopenmp bfs.cpp -o bfs
mmcoe@mmcoe-System-Product-Name: ~/Desktop/HPC$ ./bfs

enter data=>5
do you want insert one more node?y

enter data=>3
do you want insert one more node?y

enter data=>2
do you want insert one more node?y

enter data=>1
do you want insert one more node?y

enter data=>7
do you want insert one more node?y

enter data=>8
do you want insert one more node?n
5 3 2 1 7 8mmcoe@mmcoe-System-Product-Name
~/Desktop/HPC$

```

Ln 110, Col 16 Tab Size: 4 UTF-8 LF C++ Go Live Linux Prettier