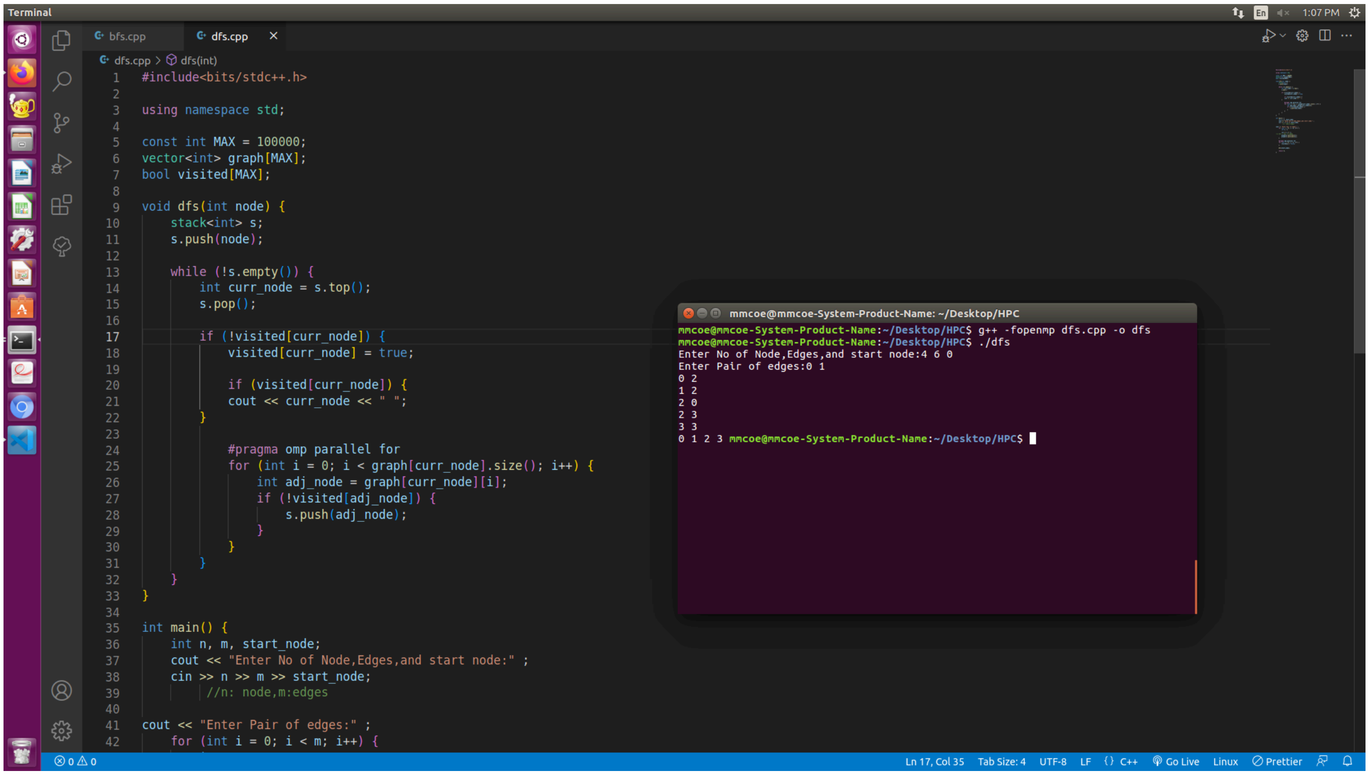
DFS

```
#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;
}
```

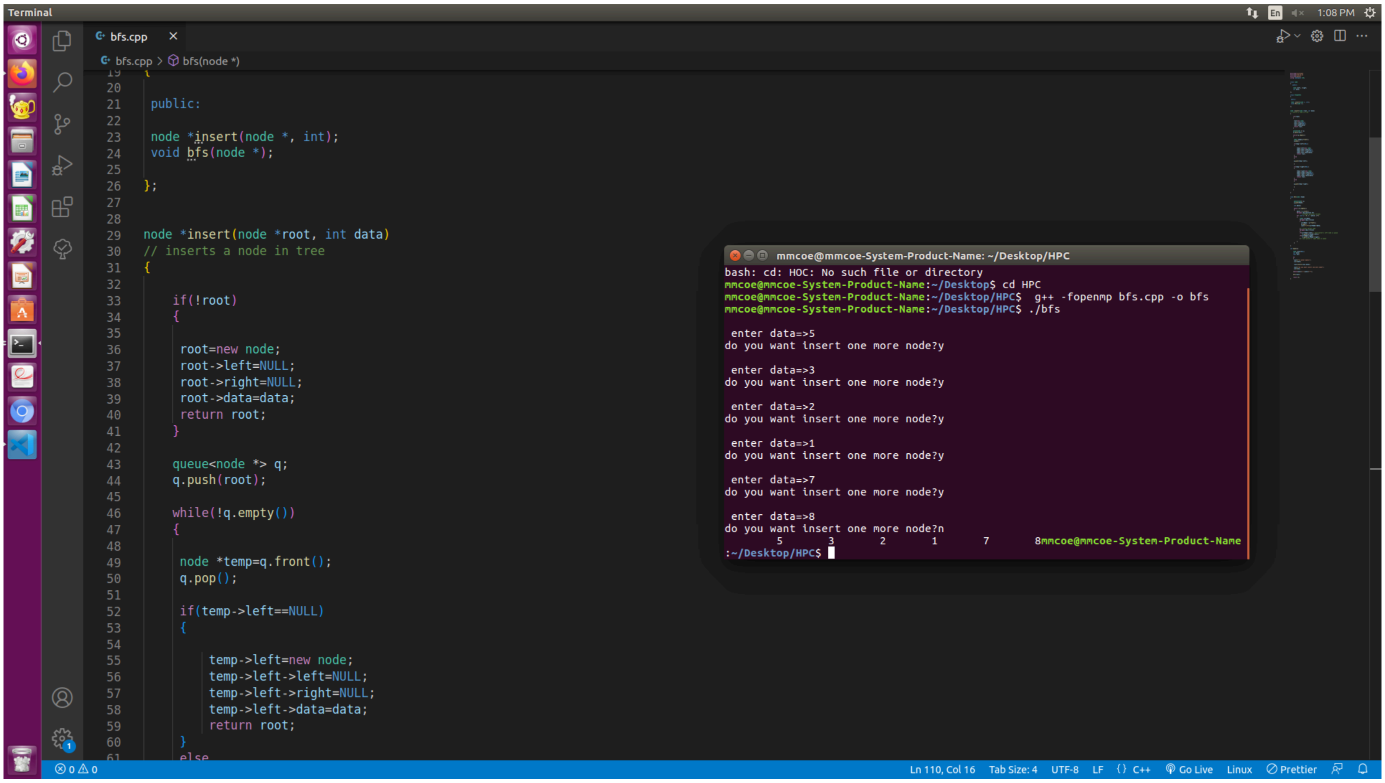


```
#include<bits/stdc++.h>

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++) {
```

```
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;
}
```



```
 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
```

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$ 
```