1.Parallel Breadth First Search:

```
#include<iostream>
#include<stdlib.h>
#include<queue>
#include<omp.h>

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)
{
   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
      for (int i = 0; i < qSize; i++)
      {
         node* currNode;
         #pragma omp critical
         {
            currNode = q.front();
            q.pop();
            cout << "\t" << currNode->data;
         }

         #pragma omp critical
         {
            if(currNode->left)
               q.push(currNode->left);
            if(currNode->right)
               q.push(currNode->right);
         }
      }
   }
}
```

```
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;
}
```

Output:

```
Enter data=>1
Do you want insert one more node?y

 Enter data=>2
Do you want insert one more node?y

 Enter data=>99
Do you want insert one more node?y

 Enter data=>110
Do you want insert one more node?y

 Enter data=>76
Do you want insert one more node?y

 Enter data=>51
Do you want insert one more node?y

 Enter data=>7
Do you want insert one more node?n

          1         2         99        110       76        51        7
```

2.Parallel Depth First Search

```
#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;
         cout << curr_node << " ";

         // Vector to hold unvisited neighbors
         vector<int> unvisited_neighbors;

         // Iterate over neighbors to find unvisited ones
         #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]) {
               // Store unvisited neighbors in the vector
               #pragma omp critical
               unvisited_neighbors.push_back(adj_node);
            }
         }

         // Push unvisited neighbors onto the stack
         for (int neighbor : unvisited_neighbors) {
            s.push(neighbor);
         }
      }
   }
```

```
}

int main() {
   int n, m, start_node;
   cout << "Enter no. of Nodes, no. of Edges, and Starting Node of the graph:\n";
   cin >> n >> m >> start_node;

   cout << "Enter pairs of nodes representing edges:\n";
   for (int i = 0; i < m; i++) {
      int u, v;
      cin >> u >> v;
      graph[u].push_back(v);
      graph[v].push_back(u);
   }

   // Initialize visited array
   for (int i = 0; i < n; i++) {
      visited[i] = false;
   }

   // Perform DFS from the start node
   dfs(start_node);

   return 0;
}
```

Output:

```
Enter no. of Nodes, no. of Edges, and Starting Node of the graph:
6 6 1
Enter pairs of nodes representing edges:
1 2
1 3
1 4
2 5
2 6
4 6

1 3 2 5 6 4
```