CODE:

```
#include <iostream>
#include <vector>
#include <queue>
#include <stack>
#include <omp.h>
using namespace std;
void parallelBFS(vector<vector<int>> &graph, int startNode) {
   int n = graph.size();
   vector<bool> visited(n, false);
   queue<int> q;
   q.push(startNode);
   visited[startNode] = true;
   cout << "Parallel BFS Traversal: ";
   #pragma omp parallel
   {
      while (!q.empty()) {
         int node;
         #pragma omp critical
         {
            if (!q.empty()) {
               node = q.front();
               q.pop();
               cout << node << " ";
            }
         }
         #pragma omp parallel for
         for (int i = 0; i < graph[node].size(); i++) {
            int neighbor = graph[node][i];
            if (!visited[neighbor]) {
               visited[neighbor] = true;
               #pragma omp critical
               q.push(neighbor);
            }
         }
      }
   }
   cout << endl;
}
void parallelDFS(vector<vector<int>> &graph, int startNode) {
   int n = graph.size();
```

```
    vector<bool> visited(n, false);
    stack<int> s;
    s.push(startNode);
    visited[startNode] = true;
    cout << "Parallel DFS Traversal: ";
    #pragma omp parallel
    {
        while (!s.empty()) {
            int node;
            #pragma omp critical
            {
                if (!s.empty()) {
                    node = s.top();
                    s.pop();
                    cout << node << " ";
                }
            }
            #pragma omp parallel for
            for (int i = graph[node].size() - 1; i >= 0; i--) {
                int neighbor = graph[node][i];
                if (!visited[neighbor]) {
                    visited[neighbor] = true;
                    #pragma omp critical
                    s.push(neighbor);
                }
            }
        }
    }
    cout << endl;
}
int main() {
    int nodes, edges;
    cout << "Enter number of nodes and edges: ";
    cin >> nodes >> edges;
    vector<vector<int>> graph(nodes);
    cout << "Enter edges (u v):" << endl;
    for (int i = 0; i < edges; i++) {
        int u, v;
        cin >> u >> v;
        graph[u].push_back(v);
        graph[v].push_back(u);
```

```
    }
    int startNode;
    cout << "Enter starting node: ";
    cin >> startNode;
    parallelBFS(graph, startNode);
    parallelDFS(graph, startNode);
    return 0;
}
```

INPUT:

```
Enter number of nodes and edges: 6 7
Enter edges (u v):
0 1
0 2
1 3
1 4
2 4
3 5
4 5
Enter starting node: 0
```

OUTPUT:

```
Parallel BFS Traversal: 0 1 2 3 4 5
Parallel DFS Traversal: 0 2 4 5 1 3
```