1. Parallel BFS using OpenMP

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

using namespace std;

const int MAXN = 100005;
vector<int> adj[MAXN];
bool visited[MAXN];

void bfs(int start) {
   queue<int> q;
   q.push(start);
   visited[start] = true;

   while (!q.empty()) {
      int v = q.front();
      q.pop();

      // Process node v here

      #pragma omp parallel for
      for (int i = 0; i < adj[v].size(); i++) {
         int u = adj[v][i];

         if (!visited[u]) {
            visited[u] = true;
            q.push(u);
         }
      }
   }
}

int main() {
   int n, m, start;
   cin >> n >> m >> start;

   for (int i = 0; i < m; i++) {
      int u, v;
      cin >> u >> v;
      adj[u].push_back(v);
      adj[v].push_back(u);
   }

   bfs(start);

   // Output visited nodes
   cout << "Visited nodes: ";
```

```
    for (int i = 1; i <= n; i++) {
        if (visited[i]) {
            cout << i << " ";
        }
    }
    cout << endl;

    return 0;
}
```

Output:

```
6 7 1
1 2
1 3
2 4
2 5
3 5
4 6
5 6
Visited nodes: 1 2 3 4 5 6
```

2. Parallel dfs using OpenMP

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

using namespace std;

const int MAXN = 100005;
vector<int> adj[MAXN];
bool visited[MAXN];

void dfs(int v) {
    visited[v] = true;

    // Process node v here

    #pragma omp parallel for
    for (int i = 0; i < adj[v].size(); i++) {
        int u = adj[v][i];

        if (!visited[u]) {
            dfs(u);
        }
    }
}

int main() {
    int n, m, start;
```

```
    cin >> n >> m >> start;

    for (int i = 0; i < m; i++) {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    dfs(start);

    // Output visited nodes
    cout << "Visited nodes: ";
    for (int i = 1; i <= n; i++) {
        if (visited[i]) {
            cout << i << " ";
        }
    }
    cout << endl;

    return 0;
}
```

Output:

6 7 1
1 2
1 3
2 4
2 5
3 5
4 6
5 6
Visited nodes: 1 2 4 6 5 3