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

using namespace std;

class Graph {
    int V; // Number of vertices

    // Adjacency list representation of graph
    vector<vector<int>> adj;

public:
    Graph(int V) : V(V), adj(V) {}

    // Add an edge to the graph
    void addEdge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u); // Assuming undirected graph
    }

    // Parallel Breadth First Search
    void parallelBFS(int start) {
        vector<bool> visited(V, false);
        queue<int> q;

        #pragma omp parallel
        {
            #pragma omp single
            {
                q.push(start);
                visited[start] = true;
            }

            while (!q.empty()) {
                int u;
                #pragma omp critical
                {
                    u = q.front();
                    q.pop();
                }
```

```
            #pragma omp for
            for (int v : adj[u]) {
                if (!visited[v]) {
                    #pragma omp critical
                    {
                        q.push(v);
                        visited[v] = true;
                    }
                }
            }
            #pragma omp barrier
            #pragma omp master
            {
                cout << u << " ";
            }
        }
    }

    // Parallel Depth First Search
    void parallelDFS(int start) {
        vector<bool> visited(V, false);

        #pragma omp parallel
        {
            parallelDFSUtil(start, visited);
        }
    }

private:
    // DFS utility function
    void parallelDFSUtil(int u, vector<bool>& visited) {
        if (visited[u]) return;
        visited[u] = true;
        cout << u << " "; // Process the current node

        #pragma omp for
        for (int v : adj[u]) {
            parallelDFSUtil(v, visited);
        }
    }
};
```

```
int main() {
    Graph g(6);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 3);
    g.addEdge(1, 4);
    g.addEdge(2, 5);

    cout << "Parallel BFS starting from node 0:" << endl;
    g.parallelBFS(0);
    cout << endl;

    cout << "Parallel DFS starting from node 0:" << endl;
    g.parallelDFS(0);
    cout << endl;

    return 0;
}
```