Name: Vaishnavi Pravin Kolse

Class: BE Div: A

Roll No.37

Practical No.1

Aim: Design and implement Parallel Breadth First Search and Depth First Search based on existing algorithms using OpenMP. Use a Tree or an undirected graph for BFS and DFS.

Program:

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

class Graph {
public:
    int V; // Number of nodes (vertices)
    std::vector<std::vector<int>> adj;

    Graph(int V) {
        this->V = V;
        adj.resize(V);
    }

    void addEdge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    void parallelBFS(int start) {
        std::vector<bool> visited(V, false);
```

```
std::queue<int> q;

visited[start] = true;
q.push(start);

std::cout << "Parallel BFS: ";

while (!q.empty()) {
    int size = q.size();

    #pragma omp parallel for
    for (int i = 0; i < size; i++) {
        int node;

        #pragma omp critical
        {
            node = q.front();
            q.pop();
        }

        std::cout << node << " ";

        for (int neighbor : adj[node]) {
            if (!visited[neighbor]) {
                visited[neighbor] = true;

                #pragma omp critical
                {
                    q.push(neighbor);
```

```
                }
            }
        }
    }
    std::cout << std::endl;
}

void parallelDFSUtil(int node, std::vector<bool>& visited) {
    visited[node] = true;
    std::cout << node << " ";

    #pragma omp parallel for
    for (size_t i = 0; i < adj[node].size(); i++) {
        int neighbor = adj[node][i];
        if (!visited[neighbor]) {
            parallelDFSUtil(neighbor, visited);
        }
    }
}

void parallelDFS(int start) {
    std::vector<bool> visited(V, false);
    std::cout << "Parallel DFS: ";
    parallelDFSUtil(start, visited);
    std::cout << std::endl;
}
};
```

```
int main() {
    int V = 7;
    Graph g(V);



    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 3);
    g.addEdge(1, 4);
    g.addEdge(2, 5);
    g.addEdge(2, 6);

    int startNode = 0;
    g.parallelBFS(startNode);
    g.parallelDFS(startNode);

    return 0;
}
```

Output:

Parallel BFS: 0 1 2 3 4 5 6

Parallel DFS: 0 1 3 4 2 5 6

=== Code Execution Successful ===