Name : Akshata Jadhav

BE-A-25

Practical No 1: 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 .

Code:

```
#include <iostream>
#include <vector>
#include <queue>
#include <stack>
#include <omp.h>
using namespace std;

class Graph {
   int V;
   vector<vector<int>> adj;
public:
   Graph(int V) : V(V), adj(V) {}
   void addEdge(int u, int v) { adj[u].push_back(v), adj[v].push_back(u); }
   void BFS(int start) {
      vector<bool> vis(V, false);
      queue<int> q; vis[start] = true, q.push(start);
      cout << "BFS: ";
      while (!q.empty()) {
         int u = q.front(); q.pop(); cout << u << " ";
         #pragma omp parallel for
         for (int v : adj[u]) if (!vis[v]) vis[v] = true, q.push(v);
      }
      cout << endl;
   }
```

```
    void DFS(int start) {
        vector<bool> vis(V, false);
        stack<int> s; s.push(start);
        cout << "DFS: ";
        while (!s.empty()) {
            int u = s.top(); s.pop();
            if (!vis[u]) { vis[u] = true, cout << u << " "; }
            #pragma omp parallel for
            for (int v : adj[u]) if (!vis[v]) s.push(v);
        }
        cout << endl;
    }
};

int main() {
    Graph g(7);
    g.addEdge(0,1), g.addEdge(0,2), g.addEdge(1,3), g.addEdge(1,4), g.addEdge(2,5), g.addEdge(2,6);
    g.BFS(0);
    g.DFS(0);
    return 0;
}
```

Output: