Experiment 1

Parallel BFS

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

// Define the graph using adjacency list
struct Graph {
    int V; // number of vertices
    vector<vector<int>> adj; // adjacency list

    Graph(int v) {
        V = v;
        adj.resize(V);
    }

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

void parallelBFS(Graph g, int source) {
    queue<int> q;
    vector<bool> visited(g.V, false);

    // Initialize with the source node
    q.push(source);
    visited[source] = true;

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

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

            // If v is not visited, mark it visited and enqueue it
            if (!visited[v]) {
                visited[v] = true;
                q.push(v);
            }
```

```
        }
    }

    // Print the visited array to see the result
    cout << "Visited nodes: ";
    for (int i = 0; i < visited.size(); i++) {
        if (visited[i]) {
            cout << i << " ";
        }
    }
    cout << endl;
}

int main() {

    cout<<"\n---Parallel Breadth First Search---";
    int n;
    cout<<"\n Enter total number of pairs: ";
    cin>>n;

    Graph g(n);

    int u,v;
    cout<<" Enter edges: \n";
    for(int i=0; i<n; i++){
        cout<<" ";
        cin>>u>>v;
        g.addEdge(u, v);
    }
    int source = 2;
    parallelBFS(g, source);

    return 0;
}
```

```
---Parallel Breadth First Search---
 Enter total number of pairs: 6
 Enter edges:
 0 1
 0 2
 1 2
 2 0
 2 3
 3 3
Visited nodes: 0 1 2 3
```

Parallel DFS

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

// Define the graph using adjacency list
struct Graph {
    int V; // number of vertices
    vector<vector<int>> adj; // adjacency list

    Graph(int v) {
        V = v;
        adj.resize(V);
    }

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

void parallelDFS(Graph g, int source) {
    stack<int> s;
    vector<bool> visited(g.V, false);

    // Initialize with the source node
    s.push(source);

    while (!s.empty()) {
        int u = s.top();
        s.pop();

        if (!visited[u]) {
            visited[u] = true;

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

                // If v is not visited, add it to the stack
                if (!visited[v]) {
                    s.push(v);
                }
            }
```

```
        }
    }

    // Print the visited array to see the result
    cout << "Visited nodes: ";
    for (int i = 0; i < visited.size(); i++) {
        if (visited[i]) {
            cout << i << " ";
        }
    }
    cout << endl;
}

int main() {
    cout<<"\n---Parallel Depth First Search---";
    int n;
    cout<<"\n Enter total number of pairs: ";
    cin>>n;

    Graph g(n);

    int u,v;
    cout<<" Enter edges: \n";
    for(int i=0; i<n; i++){
        cout<<" ";
        cin>>u>>v;
        g.addEdge(u, v);
    }
    int source = 2;
    parallelDFS(g, source);

    return 0;
}
```

```
---Parallel Depth First Search---
 Enter total number of pairs: 6
 Enter edges:
 0 1
 0 2
 1 2
 2 0
 2 3
 3 3
Visited nodes: 0 1 2 3
```