

ASSIGNMENT NO 1

Title of the Assignment:

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

Objective:

1. Implement parallel BFS and DFS algorithms using OpenMP.
2. Utilize either tree or an undirected graph data structure.

Outcome:

Students should understand basic of OPENMP programming module using BFS and DFS

Prerequisites:

Understanding of graph theory knowledge of openMP, data structure, knowledge.

Software Requirement:

OpenMP, Programming language, c/c++ compiler

Hardware Requirement:

x86_64 bit, 2 – 2/4 GB DDR RAM, 80 - 500 GB SATA HD.

Source

```
#include<iostream>

#include<omp.h>

#include<bits/stdc++.h>

using namespace std;

class Graph{

public:

// vector<vector<int>> graph;

// vector<bool> visited;

// int vertices = 0;

// int edges = 0;

int vertices = 6;

int edges = 5;

vector<vector<int>>graph={{1},{0,2,3},{1,4,5},{1,4},{2,3},{2}};

vector<bool> visited;

// Graph(){

// cout << "Enter number of nodes: ";

// cin >> vertices;

// cout << "Enter number of edges: ";

// cin >> edges;

// graph.assign(vertices,vector<int>());

// for(int i = 0 ; i < edges;i++){

// int a,b;

// cout << "Enter adjacent nodes: ";

// cin >> a >> b;

// addEdge(a,b);

// }

// }

void addEdge(int a, int b){

graph[a].push_back(b);
```

```
graph[b].push_back(a);
}
void printGraph(){
for(int i = 0; i < vertices; i++){
cout << i << " -> ";
for(auto j = graph[i].begin(); j != graph[i].end();j++){
cout << *j << " ";
}
cout << endl;
}
}
void initialize_visited(){
visited.assign(vertices,false);
}
void dfs(int i){
stack<int> s;
s.push(i);
visited[i] = true;
while(s.empty() != true){
int current = s.top();
cout << current << " ";
s.pop();
for(auto j = graph[current].begin(); j != graph[current].end();j++){
if(visited[*j] == false){
s.push(*j);
visited[*j] = true;
}
}
}
}
```

```

void parallel_dfs(int i){
    stack<int> s;
    s.push(i);
    visited[i] = true;
    while(s.empty() != true){
        int current = s.top();
        cout << current << " ";
        #pragma omp critical
        s.pop();
        #pragma omp parallel for
        for(auto j=graph[current].begin();j!=graph[current].end();j++){
            if(visited[*j] == false){
                #pragma omp critical
                {
                    s.push(*j);
                    visited[*j] = true;
                }
            }
        }
    }
}

void bfs(int i){
    queue<int> q;
    q.push(i);
    visited[i] = true;
    while(q.empty() != true){
        int current = q.front();
        q.pop();
        cout << current << " ";
        for(auto j = graph[current].begin(); j != graph[current].end();j++){

```

```

if(visited[*j] == false){
    q.push(*j);
    visited[*j] = true;
}
}
}
}

void parallel_bfs(int i){
    queue<int> q;
    q.push(i);
    visited[i] = true;
    while(q.empty() != true){
        int current = q.front();
        cout << current << " ";
        #pragma omp critical
        q.pop();
        #pragma omp parallel for
        for(auto j = graph[current].begin(); j != graph[current].end();j++){
            if(visited[*j] == false){
                #pragma omp critical
                q.push(*j);
                visited[*j] = true;
            }
        }
    }
};

int main(int argc, char const *argv[])
{
    Graph g;

```

```
cout << "Adjacency List:\n";
g.printGraph();
g.initialize_visited();
cout << "Depth First Search: \n";
auto start = chrono::high_resolution_clock::now();
g.dfs(0);
cout << endl;
auto end = chrono::high_resolution_clock::now();
cout << "Time taken: " << chrono::duration_cast<chrono::microseconds>(end
- start).count() << " microseconds" << endl;
cout << "Parallel Depth First Search: \n";
g.initialize_visited();
start = chrono::high_resolution_clock::now();
g.parallel_dfs(0);
cout << endl;
end = chrono::high_resolution_clock::now();
cout << "Time taken: " << chrono::duration_cast<chrono::microseconds>(end
- start).count() << " microseconds" << endl;
start = chrono::high_resolution_clock::now();
cout << "Breadth First Search: \n";
g.initialize_visited();
g.bfs(0);
cout << endl;
end = chrono::high_resolution_clock::now();
cout << "Time taken: " << chrono::duration_cast<chrono::microseconds>(end
- start).count() << " microseconds" << endl;
start = chrono::high_resolution_clock::now();
cout << "Parallel Breadth First Search: \n";
g.initialize_visited();
g.parallel_bfs(0);
```

```

cout << endl;

end = chrono::high_resolution_clock::now();

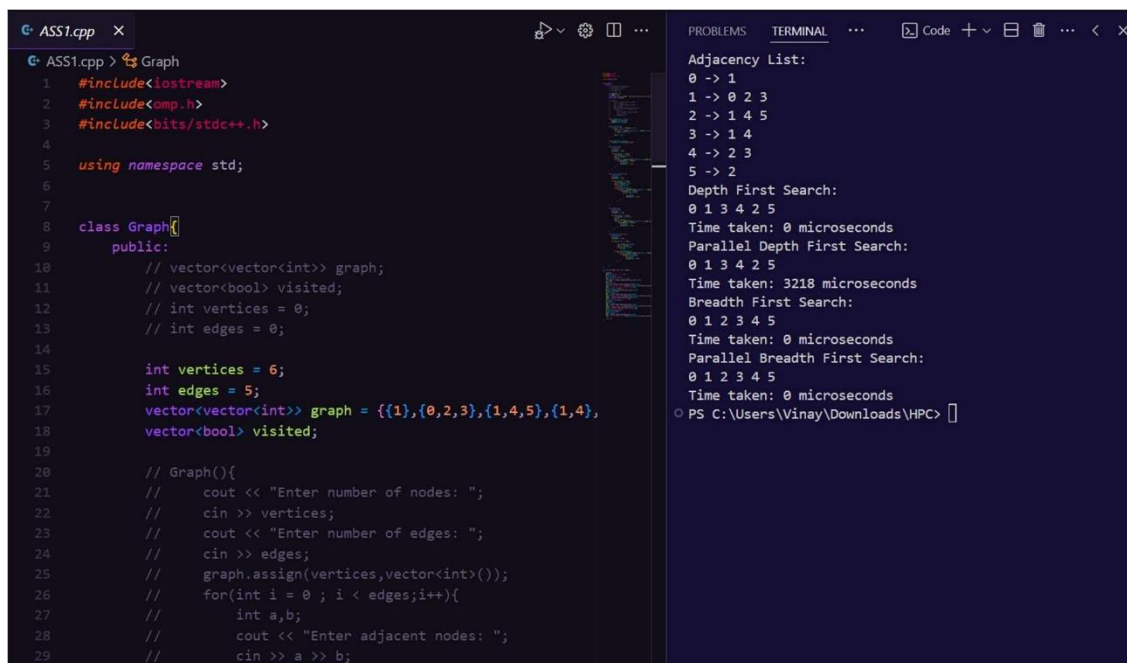
cout << "Time taken: " << chrono::duration_cast<chrono::microseconds>(end
- start).count() << " microseconds" << endl;

return 0;

}

```

OUTPUT:



The screenshot shows a C++ IDE with a file named `ASS1.cpp`. The code defines a `Graph` class with methods for Depth First Search (DFS) and Breadth First Search (BFS). The terminal output displays the results of these searches on a graph with 6 vertices and 5 edges.

```

ASS1.cpp > Graph
1  #include<iostream>
2  #include<omp.h>
3  #include<bits/stdc++.h>
4
5  using namespace std;
6
7
8  class Graph{
9  public:
10     // vector<vector<int>> graph;
11     // vector<bool> visited;
12     // int vertices = 0;
13     // int edges = 0;
14
15     int vertices = 6;
16     int edges = 5;
17     vector<vector<int>> graph = {{1},{0,2,3},{1,4,5},{1,4},
18     vector<bool> visited;
19
20     // Graph(){
21     //     cout << "Enter number of nodes: ";
22     //     cin >> vertices;
23     //     cout << "Enter number of edges: ";
24     //     cin >> edges;
25     //     graph.assign(vertices,vector<int>());
26     //     for(int i = 0 ; i < edges;i++){
27     //         int a,b;
28     //         cout << "Enter adjacent nodes: ";
29     //         cin >> a >> b;

```

```

PROBLEMS  TERMINAL  ...  Code  +  -  ...  <  x
Adjacency List:
0 -> 1
1 -> 0 2 3
2 -> 1 4 5
3 -> 1 4
4 -> 2 3
5 -> 2
Depth First Search:
0 1 3 4 2 5
Time taken: 0 microseconds
Parallel Depth First Search:
0 1 3 4 2 5
Time taken: 3218 microseconds
Breadth First Search:
0 1 2 3 4 5
Time taken: 0 microseconds
Parallel Breadth First Search:
0 1 2 3 4 5
Time taken: 0 microseconds
PS C:\Users\Vinay\Downloads\HPC>

```

Conclusion:

Implemented parallel BFS and DFS using OpenMP.