**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.**

**1) BFS Graph**

**Code:**

#include <iostream>

#include <queue>

#include <vector>

#include <omp.h>

using namespace std;

const int MAX\_NODES = 100;

vector<int> graph[MAX\_NODES];

bool bfs(int start\_node, int target\_node, int n, int\* visited\_nodes) {

bool visited[MAX\_NODES] = {false};

queue<int> q;

visited[start\_node] = true;

q.push(start\_node);

int num\_visited = 0;

visited\_nodes[num\_visited++] = start\_node;

while (!q.empty()) {

bool found = false;

#pragma omp parallel for

for (int i = 0; i < q.size(); i++) {

int node = q.front();

q.pop();

if (node == target\_node) {

found = true;

}

for (int j = 0; j < graph[node].size(); j++) {

int neighbor = graph[node][j];

if (!visited[neighbor]) {

#pragma omp critical

{

visited[neighbor] = true;

q.push(neighbor);

visited\_nodes[num\_visited++] = neighbor;

}

}

}

}

if (found) {

return true;

}

}

return false;

}

int main() {

graph[0] = {1, 2};

graph[1] = {0, 3, 4};

graph[2] = {0, 5};

graph[3] = {1};

graph[4] = {1};

graph[5] = {2};

int num\_nodes = 6;

int start\_node = 0;

int target\_node = 5;

int visited\_nodes[MAX\_NODES];

bool found = bfs(start\_node, target\_node, num\_nodes, visited\_nodes);

if (found) {

cout << "Node " << target\_node << " found in the graph!" << endl;

} else {

cout << "Node " << target\_node << " not found in the graph." << endl;

}

cout << "Nodes visited in order: ";

for (int i = 0; i < num\_nodes; i++) {

if (visited\_nodes[i] != -1) {

cout << visited\_nodes[i] << " ";

}

}

cout << endl;

return 0;

}

**Output:**

**Case 1 :**

hardik@hardik-Vector-GP66-12UGS:~/HPC\_Practical/Practical\_1$ ***g++ -fopenmp BFS\_GRAPH.cpp -o BFS\_GRAPH***

hardik@hardik-Vector-GP66-12UGS:~/HPC\_Practical/Practical\_1$ ***./BFS\_GRAPH***

***Node 5 found in the graph!***

***Nodes visited in order: 0 1 2 3 4 5***

**Case 2 :**

hardik@hardik-Vector-GP66-12UGS:~/HPC\_Practical/Practical\_1$ ***g++ -fopenmp BFS\_GRAPH.cpp -o BFS\_GRAPH***

hardik@hardik-Vector-GP66-12UGS:~/HPC\_Practical/Practical\_1$ ***./BFS\_GRAPH***

***Node 6 not found in the graph.***

***Nodes visited in order: 0 1 2 3 4 4***

**2) DFS Graph**

**Code:**

#include <iostream>

#include <vector>

#include <stack>

#include <omp.h>

using namespace std;

class Graph {

private:

int num\_vertices;

vector<vector<int>> adjacency\_list;

public:

Graph(int n) : num\_vertices(n), adjacency\_list(n) {}

void add\_edge(int u, int v) {

adjacency\_list[u].push\_back(v);

adjacency\_list[v].push\_back(u);

}

vector<int> get\_neighbors(int v) const {

return adjacency\_list[v];

}

int get\_num\_vertices() const {

return num\_vertices;

}

};

bool dfs(const Graph& graph, int start\_node, int target\_val) {

stack<int> s;

s.push(start\_node);

vector<bool> visited(graph.get\_num\_vertices(), false);

visited[start\_node] = true;

bool found = false;

while (!s.empty()) {

vector<int> visited\_nodes;

#pragma omp parallel shared(found)

{

vector<int> local\_stack;

#pragma omp single

{

local\_stack.push\_back(s.top());

s.pop();

}

while (!local\_stack.empty()) {

int node = local\_stack.back();

local\_stack.pop\_back();

visited\_nodes.push\_back(node);

if (node == target\_val) {

#pragma omp critical

{

found = true;

}

}

vector<int> neighbors = graph.get\_neighbors(node);

for (int i = 0; i < neighbors.size(); i++) {

int neighbor = neighbors[i];

if (!visited[neighbor]) {

#pragma omp critical

{

visited[neighbor] = true;

local\_stack.push\_back(neighbor);

}}}}}

cout << "Visited nodes: ";

for (int i = 0; i < visited\_nodes.size(); i++) {

cout << visited\_nodes[i] << " ";

}

cout << endl;

if (found) {

return true;

}}

return false;

}

int main() {

Graph g(7);

g.add\_edge(0, 1);

g.add\_edge(0, 2);

g.add\_edge(1, 3);

g.add\_edge(1, 4);

g.add\_edge(2, 5);

g.add\_edge(2, 6);

int start\_node = 0;

int target\_val = 5;

bool found = dfs(g, start\_node, target\_val);

if (found) {

cout << "Node with value " << target\_val << " found in the graph!" << endl;

} else {

cout << "Node with value " << target\_val << " not found in the graph." << endl;

}

return 0;

}

**Output:**

**Case 1 :**

hardik@hardik-Vector-GP66-12UGS:~/HPC\_Practical/Practical\_1$ ***g++ -fopenmp DFS\_GRAPH.cpp -o DFS\_GRAPH***

hardik@hardik-Vector-GP66-12UGS:~/HPC\_Practical/Practical\_1$ ***./DFS\_GRAPH***

***Visited nodes: 0 2 6 5 1 4 3***

***Node with value 5 found in the graph!***

**Case 2 :**

hardik@hardik-Vector-GP66-12UGS:~/HPC\_Practical/Practical\_1$ ***g++ -fopenmp DFS\_GRAPH.cpp -o DFS\_GRAPH***

hardik@hardik-Vector-GP66-12UGS:~/HPC\_Practical/Practical\_1$ ***./DFS\_GRAPH***

***Visited nodes: 0 2 6 5 1 4 3***

***Node with value 8 not found in the graph.***

**3) BFS Tree**

**Code:**

#include <iostream>

#include <queue>

#include <vector>

#include <omp.h>

using namespace std;

struct TreeNode {

int val;

TreeNode\* left;

TreeNode\* right;

TreeNode(int v) : val(v), left(nullptr), right(nullptr) {}

};

bool bfs(TreeNode\* root, int target\_val, int\* visited\_nodes) {

queue<TreeNode\*> q;

q.push(root);

int num\_visited = 0;

visited\_nodes[num\_visited++] = root->val;

while (!q.empty()) {

bool found = false;

#pragma omp parallel for

for (int i = 0; i < q.size(); i++) {

TreeNode\* node = q.front();

q.pop();

if (node->val == target\_val) {

found = true;

}

if (node->left) {

#pragma omp critical

{

q.push(node->left);

visited\_nodes[num\_visited++] = node->left->val;

}

}

if (node->right) {

#pragma omp critical

{

q.push(node->right);

visited\_nodes[num\_visited++] = node->right->val;

}

}}

if (found) {

return true;

}}

return false;

}

int main() {

TreeNode\* root = new TreeNode(1);

root->left = new TreeNode(2);

root->right = new TreeNode(3);

root->left->left = new TreeNode(4);

root->left->right = new TreeNode(5);

root->right->left = new TreeNode(6);

root->right->right = new TreeNode(7);

int target\_val = 10;

int visited\_nodes[7];

for (int i = 0; i < 7; i++) {

visited\_nodes[i] = -1;

}

bool found = bfs(root, target\_val, visited\_nodes);

if (found) {

cout << "Node with value " << target\_val << " found in the tree!" << endl;

} else {

cout << "Node with value " << target\_val << " not found in the tree." << endl;

}

cout << "Nodes visited in order: ";

for (int i = 0; i < 7; i++) {

if (visited\_nodes[i] != -1) {

cout << visited\_nodes[i] << " ";

}

}

cout << endl;

return 0;

}

**Output:**

**Case 1 :**

hardik@hardik-Vector-GP66-12UGS:~/HPC\_Practical/Practical\_1$ ***g++ -fopenmp BFS\_TREE.cpp -o BFS\_TREE***

hardik@hardik-Vector-GP66-12UGS:~/HPC\_Practical/Practical\_1$ ***./BFS\_TREE***

***Node with value 10 not found in the tree.***

***Nodes visited in order: 1 2 3 4 5 4 5***

**Case 2 :**

hardik@hardik-Vector-GP66-12UGS:~/HPC\_Practical/Practical\_1$ ***g++ -fopenmp BFS\_TREEcpp -o BFS\_TREE***

hardik@hardik-Vector-GP66-12UGS:~/HPC\_Practical/Practical\_1$ ***./BFS\_TREE***

***Node with value 5 found in the tree!***

***Nodes visited in order: 1 2 3 4 5 4 5***

**4) DFS Tree**

**Code:**

#include <iostream>

#include <stack>

#include <vector>

#include <omp.h>

using namespace std;

struct TreeNode {

int val;

TreeNode\* left;

TreeNode\* right;

TreeNode(int v) : val(v), left(nullptr), right(nullptr) {}

};

bool dfs(TreeNode\* root, int target\_val) {

stack<TreeNode\*> s;

s.push(root);

bool found = false;

while (!s.empty()) {

vector<int> visited\_nodes;

#pragma omp parallel shared(found)

{

vector<TreeNode\*> local\_stack;

#pragma omp single

{

local\_stack.push\_back(s.top());

s.pop();

}

while (!local\_stack.empty()) {

TreeNode\* node = local\_stack.back();

local\_stack.pop\_back();

visited\_nodes.push\_back(node->val);

if (node->val == target\_val) {

#pragma omp critical

{

found = true;

}

}

if (node->right) {

local\_stack.push\_back(node->right);

}

if (node->left) {

local\_stack.push\_back(node->left);

}

}

}

cout << "Visited nodes: ";

for (int i = 0; i < visited\_nodes.size(); i++) {

cout << visited\_nodes[i] << " ";

}

cout << endl;

if (found) {

return true;

}

}

return false;

}

int main() {

TreeNode\* root = new TreeNode(1);

root->left = new TreeNode(2);

root->right = new TreeNode(3);

root->left->left = new TreeNode(4);

root->left->right = new TreeNode(5);

root->right->left = new TreeNode(6);

root->right->right = new TreeNode(7);

int target\_val = 8;

bool found = dfs(root, target\_val);

if (found) {

cout << "Node with value " << target\_val << " found in the tree!" << endl;

} else {

cout << "Node with value " << target\_val << " not found in the tree." << endl;

}

return 0;

}

**Output:**

**Case 1 :**

hardik@hardik-Vector-GP66-12UGS:~/HPC\_Practical/Practical\_1$ ***g++ -fopenmp DFS\_TREE.cpp -o DFS\_TREE***

hardik@hardik-Vector-GP66-12UGS:~/HPC\_Practical/Practical\_1$ ***./DFS\_TREE***

***Visited nodes: 1 2 4 5 3 6 7***

***Node with value 8 not found in the tree.***

**Case 2 :**

hardik@hardik-Vector-GP66-12UGS:~/HPC\_Practical/Practical\_1$ ***g++ -fopenmp DFS\_TREEcpp -o DFS\_TREE***

hardik@hardik-Vector-GP66-12UGS:~/HPC\_Practical/Practical\_1$ ***./DFS\_TREE***

***Visited nodes: 1 2 4 5 3 6 7***

***Node with value 4 found in the tree!***