|  |  |
| --- | --- |
| 1a | Design and implement Parallel Breadth First Search based on existing algorithms using OpenMP. Use a Tree or an undirected graph for BFS. |

#include <iostream>

#include <stdlib.h>

#include <queue>

#include <omp.h>

using namespace std;

class Node {

public:

Node \*left, \*right;

int data;

Node(int data) : data(data), left(nullptr), right(nullptr) {}

};

class BreadthFirstSearch {

public:

Node \*insert(Node \*root, int data);

void bfs(Node \*root);

};

Node \*BreadthFirstSearch::insert(Node \*root, int data) {

if (!root) {

return new Node(data);

}

queue<Node \*> q;

q.push(root);

while (!q.empty()) {

Node \*temp = q.front();

q.pop();

if (!temp->left) {

temp->left = new Node(data);

return root;

} else {

q.push(temp->left);

}

if (!temp->right) {

temp->right = new Node(data);

return root;

} else {

q.push(temp->right);

}

}

return root;

}

void BreadthFirstSearch::bfs(Node \*root) {

if (!root) {

cout << "The tree is empty." << endl;

return;

}

queue<Node \*> q;

q.push(root);

while (!q.empty()) {

int qSize = q.size();

#pragma omp parallel for

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

Node \*currNode;

#pragma omp critical

{

currNode = q.front();

q.pop();

cout << "\t" << currNode->data;

}

#pragma omp critical

{

if (currNode->left) {

q.push(currNode->left);

}

if (currNode->right) {

q.push(currNode->right);

}

}

}

cout << endl; // Separate BFS levels visually

}

}

int main() {

Node \*root = nullptr;

BreadthFirstSearch bfs;

int data;

char ans;

do {

cout << "\nEnter data: ";

cin >> data;

root = bfs.insert(root, data);

cout << "Do you want to insert another node? (y/n): ";

cin >> ans;

} while (ans == 'y' || ans == 'Y');

cout << "\nBFS Output:";

bfs.bfs(root);

return 0;

}