Name: Mayur Goraksha Gaikwad

Roll No.: 19121028

Subject: High Performance Computing

**Experiment: 1**

**Title :**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 :**

**Implementation of Parallel Breath first Search:**

#include<iostream>

#include<stdlib.h>

#include<queue>

#include<omp.h>

using namespace std;

class node

{

   public:

    node \*left, \*right;

    int data;

};

class Breadthfs

{

 public:

 node \*insert(node \*, int);

 void bfs(node \*);

};

node \*insert(node \*root, int data)

// inserts a node in tree

{

    if(!root)

    {

     root=new node;

     root->left=NULL;

     root->right=NULL;

     root->data=data;

     return root;

    }

    queue<node \*> q;

    q.push(root);

    while(!q.empty())

    {

     node \*temp=q.front();

     q.pop();

     if(temp->left==NULL)

     {

         temp->left=new node;

         temp->left->left=NULL;

         temp->left->right=NULL;

         temp->left->data=data;

         return root;

     }

     else

     {

     q.push(temp->left);

     }

     if(temp->right==NULL)

     {

         temp->right=new node;

         temp->right->left=NULL;

         temp->right->right=NULL;

         temp->right->data=data;

         return root;

     }

     else

     {

     q.push(temp->right);

     }

    }

}

void bfs(node \*head)

{

     queue<node\*> q;

     q.push(head);

     int qSize;

     while (!q.empty())

     {

         qSize = q.size();

         #pragma omp parallel for

                //creates parallel threads

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

         {

             node\* currNode;

             #pragma omp critical

             {

               currNode = q.front();

               q.pop();

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

             }// prints parent node

             #pragma omp critical

             {

             if(currNode->left)// push parent's left node in queue

                 q.push(currNode->left);

             if(currNode->right)

                 q.push(currNode->right);

             }// push parent's right node in queue

         }

     }

}

int main() {

    // Create a binary tree

    node\* root = NULL;

    root = insert(root, 7);

    root->left = insert(root->left, 6);

    root->right = insert(root->right, 1);

    root->left->left = insert(root->left->left, 4);

    root->left->right = insert(root->left->right, 5);

    root->right->left = insert(root->right->left, 8);

    root->right->right = insert(root->right->right, 9);

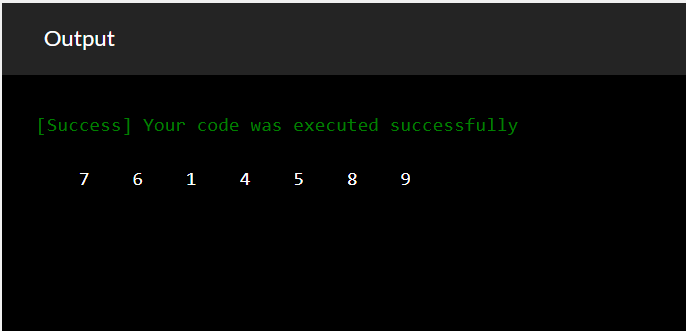
    // Perform Breadth First Search on the tree

    bfs(root);

    return 0;

}

Output :



**Implementation of Parallel Breath first Search:**

**Code:**

#include <iostream>

#include <vector>

#include <stack>

#include <omp.h>

using namespace std;

const int MAX = 100000;

vector<int> graph[MAX];

bool visited[MAX];

void dfs(int node) {

    stack<int> s;

    s.push(node);

    while (!s.empty()) {

        int curr\_node = s.top();

        s.pop();

        if (!visited[curr\_node]) {

            visited[curr\_node] = true;

            if (visited[curr\_node]) {

            cout << curr\_node << " ";

        }

            #pragma omp parallel for

            for (int i = 0; i < graph[curr\_node].size(); i++) {

                int adj\_node = graph[curr\_node][i];

                if (!visited[adj\_node]) {

                    s.push(adj\_node);

                }

            }

        }

    }

}

int main() {

    int n, m, start\_node;

    cout << "Enter No of Node,Edges,and start node:" ;

    cin >> n >> m >> start\_node;

         //n: node,m:edges

cout << "Enter Pair of edges:" ;

    for (int i = 0; i < m; i++) {

        int u, v;

        cin >> u >> v;

//u and v: Pair of edges

        graph[u].push\_back(v);

        graph[v].push\_back(u);

    }

    #pragma omp parallel for

    for (int i = 0; i < n; i++) {

        visited[i] = false;

    }

    dfs(start\_node);

/\*  for (int i = 0; i < n; i++) {

        if (visited[i]) {

            cout << i << " ";

        }

    }\*/

    return 0;

}

**Output:**

[Success] Your code was executed successfully

Enter No of Node,Edges,and start node: 6 7 1

Enter Pair of edges: 1 2 1 4 2 3 2 4 2 5 3 6 4 5