```
# Amruta Sool
# BBCO19128
```

```
"""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"""
```

```
'Design and implement Parallel Breadth First Search and Depth First Search based on
existing\nalgorithms using OpenMP. Use a Tree or an undirected graph for BFS and DF
S'
```

```
!/usr/local/cuda/bin/nvcc --version
```

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2022 NVIDIA Corporation
Built on Wed_Sep_21_10:33:58_PDT_2022
Cuda compilation tools, release 11.8, V11.8.89
Build cuda_11.8.r11.8/compiler.31833905_0
```

```
!nvcc --version
```

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2022 NVIDIA Corporation
Built on Wed_Sep_21_10:33:58_PDT_2022
Cuda compilation tools, release 11.8, V11.8.89
Build cuda_11.8.r11.8/compiler.31833905_0
```

```
!pip install git+https://github.com/andreinechaev/nvcc4jupyter.git
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting git+https://github.com/andreinechaev/nvcc4jupyter.git
  Cloning https://github.com/andreinechaev/nvcc4jupyter.git to /tmp/pip-req-build-k4n66nax
  Running command git clone --filter=blob:none --quiet https://github.com/andreinechaev/nvcc4jupyter.git /tmp
  Resolved https://github.com/andreinechaev/nvcc4jupyter.git to commit aac710a35f52bb78ab34d2e52517237941399e
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: NVCCPlugin
  Building wheel for NVCCPlugin (setup.py) ... done
  Created wheel for NVCCPlugin: filename=NVCCPlugin-0.0.2-py3-none-any.whl size=4287 sha256=5ca2fdf38be421efe
  Stored in directory: /tmp/pip-ephem-wheel-cache-ijlz3o5t/wheels/a8/b9/18/23f8ef71ceb0f63297dd1903aedd067e62
Successfully built NVCCPlugin
Installing collected packages: NVCCPlugin
Successfully installed NVCCPlugin-0.0.2
```

Saved successfully! ✕

```
ure Python compatibility.
```

```
created output directory at /content/src
Out bin /content/result.out
```

```
# Commented out IPython magic to ensure Python compatibility.
%%cu

#include <iostream>
#include <queue>
#include <omp.h>
#include <vector>
#include <stack>

void parallelBFS(int startNode, int numThreads, const std::vector<std::vector<int>>& graph)
{
    int numNodes = graph.size();
    std::vector<bool> visited(numNodes, false);
    std::queue<int> q;

    visited[startNode] = true;
    q.push(startNode);
    while (!q.empty())
    {
        #pragma omp parallel num_threads(numThreads)
        {
            #pragma omp for
            for (int i = 0; i < q.size(); i++) {
                int currentNode;
                #pragma omp critical
                {
                    currentNode = q.front();
                    q.pop();
                }

                // Process current node
                std::cout << "Visited: " << currentNode << std::endl;

                // Explore neighbors in parallel
                #pragma omp for
                for (int j = 0; j < graph[currentNode].size(); j++) {
                    int neighbor = graph[currentNode][j];

                    // Visit unvisited neighbors
                    #pragma omp critical
                    {
                        if (!visited[neighbor])
                        {
                            visited[neighbor] = true;
                            q.push(neighbor);
                        }
                    }
                        }
                    }
                }
            }
}


void parallelDFS(int startNode, int numThreads, const std::vector<std::vector<int>>& graph) {
    int numNodes = graph.size();
    std::vector<bool> visited(numNodes, false);
    std::stack<int> st;

    #pragma omp parallel num_threads(numThreads)
    {
        #pragma omp single

                                    e;
            st.push(startNode);
        }
        while (!st.empty())
        {
            int currentNode;
            #pragma omp critical
            {
                currentNode = st.top();
                st.pop();
            }
            // Process current node
            std::cout << "Visited: " << currentNode << std::endl;

            // Explore neighbors in parallel
            #pragma omp for
            for (int i = 0; i < graph[currentNode].size(); i++) {
                int neighbor = graph[currentNode][i];

                // Visit unvisited neighbors
                #pragma omp critical
                {
                    if (!visited[neighbor])
```

Saved successfully! ✕

```
                    {
                        visited[neighbor] = true;
                        st.push(neighbor);
                    }
                }
            }
        }
    }
}
int main()
{
    // Example graph representation (adjacency list)
    std::vector<std::vector<int>> graph = {
        {1, 2},
        {0, 3, 4},
        {0, 5, 6},
        {1},
        {1},
        {2},
        {2}
    };

    int startNode = 0;
    int numThreads = 4;

    std::cout << "Parallel BFS:" << std::endl;
    parallelBFS(startNode, numThreads, graph);

    std::cout << "\nParallel DFS:" << std::endl;
    parallelDFS(startNode, numThreads, graph);

    return 0;
}
```

```
Parallel BFS:
Visited: 0
Visited: 1
Visited: 2
Visited: 3
Visited: 4
Visited: 5
Visited: 6

Parallel DFS:
Visited: 0
Visited: 2
```

Saved successfully! ✕

```
Visited: 1
Visited: 4
Visited: 3
```

Saved successfully!