HPC Lab Exp No.1

import java.util.\*;

import java.util.concurrent.\*;

import java.util.Scanner;

public class Main {

static class Graph {

private final Map<Integer, List<Integer>> adjList;

public Graph() {

adjList = new HashMap<>();

}

public void addEdge(int u, int v) {

adjList.computeIfAbsent(u, k -> new ArrayList<>()).add(v);

adjList.computeIfAbsent(v, k -> new ArrayList<>()).add(u);

}

public List<Integer> getNeighbors(int u) {

return adjList.getOrDefault(u, new ArrayList<>());

}

public void printGraph() {

System.out.println("\nGraph Structure:");

for (Map.Entry<Integer, List<Integer>> entry : adjList.entrySet()) {

System.out.print(entry.getKey() + " -> ");

for (int neighbor : entry.getValue()) {

System.out.print(neighbor + " ");

}

System.out.println();

}

}

}

// Parallel BFS

public static List<Integer> parallelBFS(Graph graph, int start) {

Set<Integer> visited = Collections.synchronizedSet(new HashSet<>());

Queue<Integer> queue = new ConcurrentLinkedQueue<>();

List<Integer> bfsTraversal = new ArrayList<>();

queue.add(start);

visited.add(start);

ExecutorService executor = Executors.newCachedThreadPool();

while (!queue.isEmpty()) {

int levelSize = queue.size();

List<Future<?>> futures = new ArrayList<>();

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

final int currentNode = queue.poll();

bfsTraversal.add(currentNode);

futures.add(executor.submit(() -> {

for (int neighbor : graph.getNeighbors(currentNode)) {

if (visited.add(neighbor)) {

queue.add(neighbor);

}

}

}));

}

for (Future<?> future : futures) {

try {

future.get();

} catch (InterruptedException | ExecutionException e) {

e.printStackTrace();

}

}

}

executor.shutdown();

return bfsTraversal;

}

// Parallel DFS

public static List<Integer> parallelDFS(Graph graph, int start) {

Set<Integer> visited = Collections.synchronizedSet(new HashSet<>());

List<Integer> dfsTraversal = new ArrayList<>();

ExecutorService executor = Executors.newCachedThreadPool();

dfsRecursive(graph, start, visited, executor, dfsTraversal);

executor.shutdown();

return dfsTraversal;

}

private static void dfsRecursive(Graph graph, int node, Set<Integer> visited, ExecutorService executor, List<Integer> dfsTraversal) {

visited.add(node);

dfsTraversal.add(node);

List<Integer> neighbors = graph.getNeighbors(node);

List<Future<?>> futures = new ArrayList<>();

for (int neighbor : neighbors) {

if (!visited.contains(neighbor)) {

futures.add(executor.submit(() -> {

dfsRecursive(graph, neighbor, visited, executor, dfsTraversal);

}));

}

}

for (Future<?> future : futures) {

try {

future.get();

} catch (InterruptedException | ExecutionException e) {

e.printStackTrace();

}

}

}

public static void printTraversalAsTree(List<Integer> traversal, String traversalType) {

System.out.println("\n" + traversalType + " Traversal Output: ");

for (int node : traversal) {

System.out.print(node + " ");

}

System.out.println();

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

Graph graph = new Graph();

System.out.println("Enter the number of edges you want to add to the graph:");

int numEdges = scanner.nextInt();

System.out.println("Enter edges as pairs of nodes (e.g., 0 1 for an edge between 0 and 1): ");

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

int u = scanner.nextInt();

int v = scanner.nextInt();

graph.addEdge(u, v);

}

graph.printGraph();

boolean continueRunning = true;

while (continueRunning) {

System.out.println("\nChoose an option: ");

System.out.println("1. Parallel BFS");

System.out.println("2. Parallel DFS");

System.out.println("3. Exit");

System.out.print("Enter your choice: ");

int choice = scanner.nextInt();

switch (choice) {

case 1:

System.out.print("Enter starting node for BFS: ");

int startBFS = scanner.nextInt();

System.out.println("\nRunning Parallel BFS...");

List<Integer> bfsTraversal = parallelBFS(graph, startBFS);

printTraversalAsTree(bfsTraversal, "BFS");

break;

case 2:

System.out.print("Enter starting node for DFS: ");

int startDFS = scanner.nextInt();

System.out.println("\nRunning Parallel DFS...");

List<Integer> dfsTraversal = parallelDFS(graph, startDFS);

printTraversalAsTree(dfsTraversal, "DFS");

break;

case 3:

continueRunning = false;

System.out.println("Exited the program successfully");

break;

default:

System.out.println("Invalid choice! Please choose a valid option.");

break;

}

if (continueRunning) {

System.out.print("\nDo you want to continue? (yes/no): ");

String userChoice = scanner.next();

if (userChoice.equalsIgnoreCase("no")) {

continueRunning = false;

System.out.println("Exited the program successfully");

}

}

}

scanner.close();

}

}

OUTPUT:



