/\*

**1192. Critical Connections in a Network**

Hard

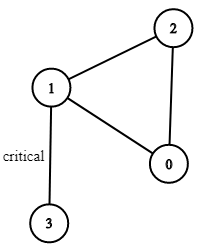
10317FavoriteShare

There are n servers numbered from 0 to n-1 connected by undirected server-to-server connections forming a network where connections[i] = [a, b] represents a connection between servers a and b. Any server can reach any other server directly or indirectly through the network.

A *critical connection* is a connection that, if removed, will make some server unable to reach some other server.

Return all critical connections in the network in any order.

**Example 1:**

****

**Input:** n = 4, connections = [[0,1],[1,2],[2,0],[1,3]]

**Output:** [[1,3]]

**Explanation:** [[3,1]] is also accepted.

\*/

import java.util.\*;

public class CriticalConnectionsinANetwork {

public List<List<Integer>> criticalConnections(int n, List<List<Integer>> connections) {

int[] seen = new int[n];

int[] low = new int[n];

Arrays.fill(seen, -1);

List<Integer>[] graph = new ArrayList[n];

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

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

graph[i] = new ArrayList<Integer>();

}

int len = connections.size();

// graph

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

// p -> q

int p = connections.get(i).get(0);

int q = connections.get(i).get(1);

graph[p].add(q);

graph[q].add(p);

}

// not seen, dfs

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

if (seen[i] == -1) {

dfs(i, low, seen, graph, res, 0);

}

}

return res;

}

int time = 0;

private void dfs(int u, int[] low, int[] seen, List<Integer>[] graph, List<List<Integer>> res, int node) {

seen[u] = low[u] = ++time;

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

int v = graph[u].get(j);

if (v == node) {

continue;

}

if (seen[v] == -1) {

dfs(v, low, seen, graph, res, u);

low[u] = Math.min(low[u], low[v]);

if (low[v] > seen[u]) {

res.add(Arrays.asList(u, v));

}

}

else {

low[u] = Math.min(low[u], seen[v]);

}

}

}

}