# 1579. Remove Max Number of Edges to Keep Graph Fully Traversable

## SOLUTION IN JAVA

class UnionFind {

public UnionFind(int n) {

count = n;

id = new int[n];

rank = new int[n];

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

id[i] = i;

}

public boolean unionByRank(int u, int v) {

final int i = find(u);

final int j = find(v);

if (i == j)

return false;

if (rank[i] < rank[j]) {

id[i] = j;

} else if (rank[i] > rank[j]) {

id[j] = i;

} else {

id[i] = j;

++rank[j];

}

--count;

return true;

}

public int getCount() {

return count;

}

private int count;

private int[] id;

private int[] rank;

private int find(int u) {

return id[u] == u ? u : (id[u] = find(id[u]));

}

}

class Solution {

public int maxNumEdgesToRemove(int n, int[][] edges) {

UnionFind alice = new UnionFind(n);

UnionFind bob = new UnionFind(n);

int requiredEdges = 0;

// Greedily put type 3 edges in the front.

Arrays.sort(edges, (a, b) -> b[0] - a[0]);

for (int[] edge : edges) {

final int type = edge[0];

final int u = edge[1] - 1;

final int v = edge[2] - 1;

switch (type) {

case 3: // Can be traversed by Alice and Bob.

// Note that we should use | instead of || because if the first

// expression is true, short-circuiting will skip the second

// expression.

if (alice.unionByRank(u, v) | bob.unionByRank(u, v))

++requiredEdges;

break;

case 2: // Can be traversed by Bob.

if (bob.unionByRank(u, v))

++requiredEdges;

case 1: // Can be traversed by Alice.

if (alice.unionByRank(u, v))

++requiredEdges;

}

}

return alice.getCount() == 1 && bob.getCount() == 1 ? edges.length - requiredEdges : -1;

}

}