3607. Power Grid Maintenance

Solved

Medium | Topics | Companies | Hint

You are given an integer `c` representing `c` power stations, each with a unique identifier `id` from 1 to `c` (1-based indexing).

These stations are interconnected via `n` **bidirectional** cables, represented by a 2D array `connections`, where each element `connections[i] = [u_i, v_i]` indicates a connection between station `u_i` and station `v_i`. Stations that are directly or indirectly connected form a **power grid**.

Initially, **all** stations are online (operational).

You are also given a 2D array `queries`, where each query is one of the following *two* types:

- `[1, x]`: A maintenance check is requested for station `x`. If station `x` is online, it resolves the check by itself. If station `x` is offline, the check is resolved by the operational station with the smallest `id` in the same **power grid** as `x`. If **no operational** station *exists* in that grid, return -1.
- `[2, x]`: Station `x` goes offline (i.e., it becomes non-operational).

Return an array of integers representing the results of each query of type `[1, x]` in the **order** they appear.

Note: The power grid preserves its structure; an offline (non-operational) node remains part of its grid and taking it offline does not alter connectivity.

Example 1:

Input: `c = 5, connections = [[1,2],[2,3],[3,4],[4,5]], queries = [[1,3],[2,1],[1,1],[2,2],[1,2]]`

Output: `[3,2,3]`

Explanation:

- Initially, all stations `{1, 2, 3, 4, 5}` are online and form a single power grid.
- Query `[1,3]`: Station 3 is online, so the maintenance check is resolved by station 3.
- Query `[2,1]`: Station 1 goes offline. The remaining online stations are `{2, 3, 4, 5}`.
- Query `[1,1]`: Station 1 is offline, so the check is resolved by the operational station with the smallest `id` among `{2, 3, 4, 5}`, which is station 2.
- Query `[2,2]`: Station 2 goes offline. The remaining online stations are `{3, 4, 5}`.
- Query `[1,2]`: Station 2 is offline, so the check is resolved by the operational station with the smallest `id` among `{3, 4, 5}`, which is station 3.

Example 2:

Input: `c = 3, connections = [], queries = [[1,1],[2,1],[1,1]]`

Output: `[1,-1]`

Explanation:

- There are no connections, so each station is its own isolated grid.
- Query `[1,1]`: Station 1 is online in its isolated grid, so the maintenance check is resolved by station 1.
- Query `[2,1]`: Station 1 goes offline.
- Query `[1,1]`: Station 1 is offline and there are no other stations in its grid, so the result is -1.

Constraints:

- $1 <= c <= 10^5$
- $0 <= n == connections.length <= min(10^5, c * (c - 1) / 2)$
- $connections[i].length == 2$
- $1 <= u_i, v_i <= c$
- $u_i != v_i$
- $1 <= queries.length <= 2 * 10^5$
- $queries[i].length == 2$
- $queries[i][0]$ is either 1 or 2.
- $1 <= queries[i][1] <= c$

Python:

```
from collections import defaultdict
import heapq

class DSU:
    def __init__(self, n):
        self.parent = list(range(n + 1))

    def find(self, x):
        if self.parent[x] != x: self.parent[x] = self.find(self.parent[x])
        return self.parent[x]

    def union(self, x, y):
        px, py = self.find(x), self.find(y)
        if px == py: return False
        self.parent[py] = px
        return True

class Solution:
    def processQueries(self, n: int, connections: List[List[int]], queries: List[List[int]]) -> List[int]:
        dsu = DSU(n)
        online = [True] * (n + 1)
```

```
        for u, v in connections: dsu.union(u, v)

        component_heap = defaultdict(list)
        for station in range(1, n + 1):
            root = dsu.find(station)
            heapq.heappush(component_heap[root], station)

        result = []

        for typ, x in queries:
            if typ == 2:
                online[x] = False
            else:
                if online[x]:
                    result.append(x)
                else:
                    root = dsu.find(x)
                    heap = component_heap[root]
                    while heap and not online[heap[0]]:
                        heapq.heappop(heap)
                    result.append(heap[0] if heap else -1)

        return result
```

JavaScript:

```
/**
 * @param {number} c
 * @param {number[][]} connections
 * @param {number[][]} queries
 * @return {number[]}
 */
var processQueries = function(c, connections, queries) {
  const parent = Array.from({ length: c + 1 }, (_, i) => i);
  const size = new Array(c + 1).fill(1);
  const find = (x) => (parent[x] === x ? x : (parent[x] = find(parent[x])));
  const union = (a, b) => {
    a = find(a); b = find(b);
    if (a === b) return;
    if (size[a] < size[b]) [a, b] = [b, a];
    parent[b] = a;
    size[a] += size[b];
  };
  for (const [u, v] of connections) union(u, v);
```

```
  const members = Array(c + 1);
  for (let i = 1; i <= c; i++) {
    const r = find(i);
    if (!members[r]) members[r] = [];
    members[r].push(i);
  }
  const ptr = new Array(c + 1).fill(0);
  const offline = new Array(c + 1).fill(false);
  const ans = [];
  for (const [t, x] of queries) {
    if (t === 1) {
      if (!offline[x]) {
        ans.push(x);
      } else {
        const r = find(x);
        const arr = members[r] || [];
        let p = ptr[r] || 0;
        while (p < arr.length && offline[arr[p]]) p++;
        ptr[r] = p;
        ans.push(p < arr.length ? arr[p] : -1);
      }
    } else {
      offline[x] = true;
    }
  }
  return ans;
};
```

Java:

```
class DSU {
    int[] parent;

    public DSU(int n) {
        parent = new int[n + 1];
        for (int i = 0; i <= n; i++)
            parent[i] = i;
    }

    public int find(int x) {
        if (parent[x] != x)
            parent[x] = find(parent[x]); // path compression
        return parent[x];
    }
```

```
    public boolean union(int x, int y) {
        int px = find(x), py = find(y);
        if (px == py)
            return false;
        parent[py] = px;
        return true;
    }
}

class Solution {
    public int[] processQueries(int c, int[][] connections, int[][] queries) {
        DSU dsu = new DSU(c);
        boolean[] online = new boolean[c + 1];
        Arrays.fill(online, true);

        for (int[] conn : connections)
            dsu.union(conn[0], conn[1]);

        Map<Integer, PriorityQueue<Integer>> componentHeap = new HashMap<>();
        for (int station = 1; station <= c; station++) {
            int root = dsu.find(station);
            componentHeap.putIfAbsent(root, new PriorityQueue<>());
            componentHeap.get(root).offer(station);
        }

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

        for (int[] query : queries) {
            int type = query[0], x = query[1];

            if (type == 2) {
                online[x] = false;
            } else {
                if (online[x]) {
                    result.add(x);
                } else {
                    int root = dsu.find(x);
                    PriorityQueue<Integer> heap = componentHeap.get(root);

                    while (heap != null && !heap.isEmpty() && !online[heap.peek()]) {
                        heap.poll();
                    }

                    result.add((heap == null || heap.isEmpty()) ? -1 : heap.peek());
```

```
            }
        }
    }

    int[] ans = new int[result.size()];
    for (int i = 0; i < result.size(); i++) {
        ans[i] = result.get(i);
    }
    return ans;
  }
}
```