Problem 3607: Power Grid Maintenance

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

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:

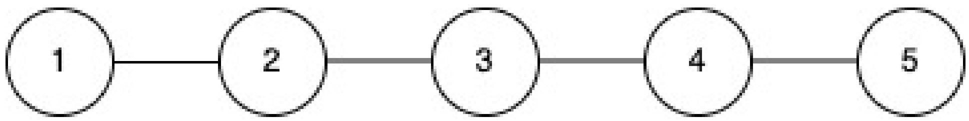
Input:

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

Code Snippets

C++:

```
class Solution {
public:
vector<int> processQueries(int c, vector<vector<int>>& connections,
vector<vector<int>>& queries) {

}
};
```

Java:

```
class Solution {
public int[] processQueries(int c, int[][] connections, int[][] queries) {

}
}
```

Python3:

```
class Solution:
def processQueries(self, c: int, connections: List[List[int]], queries:
List[List[int]]) -> List[int]:
```

Python:

```
class Solution(object):
def processQueries(self, c, connections, queries):
"""
:type c: int
:type connections: List[List[int]]
:type queries: List[List[int]]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number} c
* @param {number[][]} connections
* @param {number[][]} queries
* @return {number[]}
*/
var processQueries = function(c, connections, queries) {

};
```

TypeScript:

```
function processQueries(c: number, connections: number[][], queries:
number[][]): number[] {

};
```

C#:

```
public class Solution {
public int[] ProcessQueries(int c, int[][] connections, int[][] queries) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* processQueries(int c, int** connections, int connectionsSize, int*
connectionsColSize, int** queries, int queriesSize, int* queriesColSize, int*
returnSize) {

}
```

Go:

```
func processQueries(c int, connections [][]int, queries [][]int) []int {

}
```

Kotlin:

```
class Solution {
fun processQueries(c: Int, connections: Array<IntArray>, queries:
Array<IntArray>): IntArray {

}
}
```

Swift:

```
class Solution {
func processQueries(_ c: Int, _ connections: [[Int]], _ queries: [[Int]]) ->
[Int] {

}
}
```

Rust:

```
impl Solution {
pub fn process_queries(c: i32, connections: Vec<Vec<i32>>, queries:
Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer} c
# @param {Integer[][]} connections
# @param {Integer[][]} queries
# @return {Integer[]}
def process_queries(c, connections, queries)

end
```

PHP:

```
class Solution {

/**
* @param Integer $c
* @param Integer[][] $connections
* @param Integer[][] $queries
* @return Integer[]
*/
function processQueries($c, $connections, $queries) {

}
}
```

Dart:

```
class Solution {
List<int> processQueries(int c, List<List<int>> connections, List<List<int>>
queries) {

}
}
```

Scala:

```
object Solution {
def processQueries(c: Int, connections: Array[Array[Int]], queries:
Array[Array[Int]]): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec process_queries(c :: integer, connections :: [[integer]], queries ::
[[integer]]) :: [integer]
def process_queries(c, connections, queries) do

end
end
```

Erlang:

```
-spec process_queries(C :: integer(), Connections :: [[integer()]], Queries
:: [[integer()]]) -> [integer()].
process_queries(C, Connections, Queries) ->
.
```

Racket:

```
(define/contract (process-queries c connections queries)
(-> exact-integer? (listof (listof exact-integer?)) (listof (listof
exact-integer?)) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Power Grid Maintenance
* Difficulty: Medium
* Tags: array, graph, hash, search, queue, heap
*
* Approach: Use two pointers or sliding window technique
```

```
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
vector<int> processQueries(int c, vector<vector<int>>& connections,
vector<vector<int>>& queries) {

}
};
```

Java Solution:

```
/**
* Problem: Power Grid Maintenance
* Difficulty: Medium
* Tags: array, graph, hash, search, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int[] processQueries(int c, int[][] connections, int[][] queries) {

}
}
```

Python3 Solution:

```
"""
Problem: Power Grid Maintenance
Difficulty: Medium
Tags: array, graph, hash, search, queue, heap

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""
```

```
class Solution:
def processQueries(self, c: int, connections: List[List[int]], queries:
List[List[int]]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def processQueries(self, c, connections, queries):
"""
:type c: int
:type connections: List[List[int]]
:type queries: List[List[int]]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Power Grid Maintenance
* Difficulty: Medium
* Tags: array, graph, hash, search, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number} c
* @param {number[][]} connections
* @param {number[][]} queries
* @return {number[]}
*/
var processQueries = function(c, connections, queries) {

};
```

TypeScript Solution:

```
/**
* Problem: Power Grid Maintenance
* Difficulty: Medium
* Tags: array, graph, hash, search, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function processQueries(c: number, connections: number[][], queries:
number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Power Grid Maintenance
* Difficulty: Medium
* Tags: array, graph, hash, search, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int[] ProcessQueries(int c, int[][] connections, int[][] queries) {

}
}
```

C Solution:

```
/*
* Problem: Power Grid Maintenance
* Difficulty: Medium
* Tags: array, graph, hash, search, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
```

```
* Space Complexity: O(n) for hash map
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* processQueries(int c, int** connections, int connectionsSize, int*
connectionsColSize, int** queries, int queriesSize, int* queriesColSize, int*
returnSize) {

}
```

Go Solution:

```
// Problem: Power Grid Maintenance
// Difficulty: Medium
// Tags: array, graph, hash, search, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func processQueries(c int, connections [][]int, queries [][]int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun processQueries(c: Int, connections: Array<IntArray>, queries:
Array<IntArray>): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func processQueries(_ c: Int, _ connections: [[Int]], _ queries: [[Int]]) ->
[Int] {
```

```
}
}
```

Rust Solution:

```
// Problem: Power Grid Maintenance
// Difficulty: Medium
// Tags: array, graph, hash, search, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn process_queries(c: i32, connections: Vec<Vec<i32>>, queries:
Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer} c
# @param {Integer[][]} connections
# @param {Integer[][]} queries
# @return {Integer[]}
def process_queries(c, connections, queries)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $c
* @param Integer[][] $connections
* @param Integer[][] $queries
* @return Integer[]
*/
function processQueries($c, $connections, $queries) {
```

```
}
}
```

Dart Solution:

```
class Solution {
List<int> processQueries(int c, List<List<int>> connections, List<List<int>>
queries) {

}
}
```

Scala Solution:

```
object Solution {
def processQueries(c: Int, connections: Array[Array[Int]], queries:
Array[Array[Int]]): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec process_queries(c :: integer, connections :: [[integer]], queries ::
[[integer]]) :: [integer]
def process_queries(c, connections, queries) do

end
end
```

Erlang Solution:

```
-spec process_queries(C :: integer(), Connections :: [[integer()]], Queries
:: [[integer()]]) -> [integer()].
process_queries(C, Connections, Queries) ->
.
```

Racket Solution:

```
(define/contract (process-queries c connections queries)
(-> exact-integer? (listof (listof exact-integer?)) (listof (listof
exact-integer?)) (listof exact-integer?))
)
```