

Problem 1488: Avoid Flood in The City

Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

Your country has 10

9

lakes. Initially, all the lakes are empty, but when it rains over the

n

th

lake, the

n

th

lake becomes full of water. If it rains over a lake that is

full of water

, there will be a

flood

. Your goal is to avoid floods in any lake.

Given an integer array

rains

where:

$rains[i] > 0$

means there will be rains over the

$rains[i]$

lake.

$rains[i] == 0$

means there are no rains this day and you

must

choose

one lake

this day and

dry it

.

Return

an array

ans

where:

$ans.length == rains.length$

ans[i] == -1

if

rains[i] > 0

.

ans[i]

is the lake you choose to dry in the

ith

day if

rains[i] == 0

.

If there are multiple valid answers return

any

of them. If it is impossible to avoid flood return

an empty array

.

Notice that if you chose to dry a full lake, it becomes empty, but if you chose to dry an empty lake, nothing changes.

Example 1:

Input:

rains = [1,2,3,4]

Output:

$[-1, -1, -1, -1]$

Explanation:

After the first day full lakes are [1] After the second day full lakes are [1,2] After the third day full lakes are [1,2,3] After the fourth day full lakes are [1,2,3,4] There's no day to dry any lake and there is no flood in any lake.

Example 2:

Input:

$\text{rains} = [1, 2, 0, 0, 2, 1]$

Output:

$[-1, -1, 2, 1, -1, -1]$

Explanation:

After the first day full lakes are [1] After the second day full lakes are [1,2] After the third day, we dry lake 2. Full lakes are [1] After the fourth day, we dry lake 1. There is no full lakes. After the fifth day, full lakes are [2]. After the sixth day, full lakes are [1,2]. It is easy that this scenario is flood-free. $[-1, -1, 1, 2, -1, -1]$ is another acceptable scenario.

Example 3:

Input:

$\text{rains} = [1, 2, 0, 1, 2]$

Output:

$[]$

Explanation:

After the second day, full lakes are [1,2]. We have to dry one lake in the third day. After that, it will rain over lakes [1,2]. It's easy to prove that no matter which lake you choose to dry in the 3rd day, the other one will flood.

Constraints:

$1 \leq \text{rains.length} \leq 10$

5

$0 \leq \text{rains}[i] \leq 10$

9

Code Snippets

C++:

```
class Solution {
public:
vector<int> avoidFlood(vector<int>& rains) {
    }
};
```

Java:

```
class Solution {
public int[] avoidFlood(int[] rains) {
    }
}
```

Python3:

```
class Solution:
def avoidFlood(self, rains: List[int]) -> List[int]:
```

Python:

```
class Solution(object):
    def avoidFlood(self, rains):
        """
        :type rains: List[int]
        :rtype: List[int]
        """

```

JavaScript:

```
/**
 * @param {number[]} rains
 * @return {number[]}
 */
var avoidFlood = function(rains) {
};


```

TypeScript:

```
function avoidFlood(rains: number[]): number[] {
};


```

C#:

```
public class Solution {
    public int[] AvoidFlood(int[] rains) {
        }
}
```

C:

```
/**
 * Note: The returned array must be malloced, assume caller calls free().
 */
int* avoidFlood(int* rains, int rainsSize, int* returnSize) {
}


```

Go:

```
func avoidFlood(rains []int) []int {  
}  
}
```

Kotlin:

```
class Solution {  
    fun avoidFlood(rains: IntArray): IntArray {  
        }  
    }  
}
```

Swift:

```
class Solution {  
    func avoidFlood(_ rains: [Int]) -> [Int] {  
        }  
    }  
}
```

Rust:

```
impl Solution {  
    pub fn avoid_flood(rains: Vec<i32>) -> Vec<i32> {  
        }  
    }  
}
```

Ruby:

```
# @param {Integer[]} rains  
# @return {Integer[]}  
def avoid_flood(rains)  
  
end
```

PHP:

```
class Solution {  
  
    /**  
     * @param Integer[] $rains  
     * @return Integer[]  
     */
```

```
*/  
function avoidFlood($rains) {  
  
}  
}  
}
```

Dart:

```
class Solution {  
List<int> avoidFlood(List<int> rains) {  
  
}  
}  
}
```

Scala:

```
object Solution {  
def avoidFlood(rains: Array[Int]): Array[Int] = {  
  
}  
}
```

Elixir:

```
defmodule Solution do  
@spec avoid_flood(rains :: [integer]) :: [integer]  
def avoid_flood(rains) do  
  
end  
end
```

Erlang:

```
-spec avoid_flood(Rains :: [integer()]) -> [integer()].  
avoid_flood(Rains) ->  
.
```

Racket:

```
(define/contract (avoid-flood rains)  
(-> (listof exact-integer?) (listof exact-integer?))  
)
```

Solutions

C++ Solution:

```
/*
 * Problem: Avoid Flood in The City
 * Difficulty: Medium
 * Tags: array, greedy, 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> avoidFlood(vector<int>& rains) {

}
};
```

Java Solution:

```
/**
 * Problem: Avoid Flood in The City
 * Difficulty: Medium
 * Tags: array, greedy, 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[] avoidFlood(int[] rains) {

}
};
```

Python3 Solution:

```

"""
Problem: Avoid Flood in The City
Difficulty: Medium
Tags: array, greedy, 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 avoidFlood(self, rains: List[int]) -> List[int]:
        # TODO: Implement optimized solution
        pass

```

Python Solution:

```

class Solution(object):
    def avoidFlood(self, rains):
        """
        :type rains: List[int]
        :rtype: List[int]
        """

```

JavaScript Solution:

```

/**
 * Problem: Avoid Flood in The City
 * Difficulty: Medium
 * Tags: array, greedy, 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[]} rains
 * @return {number[]}
 */
var avoidFlood = function(rains) {

```

```
};
```

TypeScript Solution:

```
/**  
 * Problem: Avoid Flood in The City  
 * Difficulty: Medium  
 * Tags: array, greedy, 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 avoidFlood(rains: number[]): number[] {  
  
};
```

C# Solution:

```
/*  
 * Problem: Avoid Flood in The City  
 * Difficulty: Medium  
 * Tags: array, greedy, 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[] AvoidFlood(int[] rains) {  
  
    }  
}
```

C Solution:

```
/*  
 * Problem: Avoid Flood in The City  
 * Difficulty: Medium
```

```

* Tags: array, greedy, 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* avoidFlood(int* rains, int rainsSize, int* returnSize) {

}

```

Go Solution:

```

// Problem: Avoid Flood in The City
// Difficulty: Medium
// Tags: array, greedy, 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 avoidFlood(rains []int) []int {
}

```

Kotlin Solution:

```

class Solution {
    fun avoidFlood(rains: IntArray): IntArray {
        }
    }
}
```

Swift Solution:

```

class Solution {
    func avoidFlood(_ rains: [Int]) -> [Int] {

```

```
}
```

```
}
```

Rust Solution:

```
// Problem: Avoid Flood in The City
// Difficulty: Medium
// Tags: array, greedy, 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 avoid_flood(rains: Vec<i32>) -> Vec<i32> {
        }

    }
}
```

Ruby Solution:

```
# @param {Integer[]} rains
# @return {Integer[]}
def avoid_flood(rains)

end
```

PHP Solution:

```
class Solution {

    /**
     * @param Integer[] $rains
     * @return Integer[]
     */
    function avoidFlood($rains) {
        }

    }
}
```

Dart Solution:

```
class Solution {  
    List<int> avoidFlood(List<int> rains) {  
        }  
    }  
}
```

Scala Solution:

```
object Solution {  
    def avoidFlood(rains: Array[Int]): Array[Int] = {  
        }  
    }  
}
```

Elixir Solution:

```
defmodule Solution do  
    @spec avoid_flood(rains :: [integer]) :: [integer]  
    def avoid_flood(rains) do  
  
    end  
end
```

Erlang Solution:

```
-spec avoid_flood(Rains :: [integer()]) -> [integer()].  
avoid_flood(Rains) ->  
.
```

Racket Solution:

```
(define/contract (avoid-flood rains)  
  (-> (listof exact-integer?) (listof exact-integer?))  
)
```