

Problem 2940: Find Building Where Alice and Bob Can Meet

Problem Information

Difficulty: **Hard**

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given a

0-indexed

array

heights

of positive integers, where

heights[i]

represents the height of the

i

th

building.

If a person is in building

i

, they can move to any other building

j

if and only if

$i < j$

and

$\text{heights}[i] < \text{heights}[j]$

.

You are also given another array

queries

where

$\text{queries}[i] = [a$

i

, b

i

]

. On the

i

th

query, Alice is in building

a

i

while Bob is in building

b

i

.

Return

an array

ans

where

ans[i]

is

the index of the leftmost building

where Alice and Bob can meet on the

i

th

query

.

If Alice and Bob cannot move to a common building on query

i

,

set

ans[i]

to

-1

.

Example 1:

Input:

heights = [6,4,8,5,2,7], queries = [[0,1],[0,3],[2,4],[3,4],[2,2]]

Output:

[2,5,-1,5,2]

Explanation:

In the first query, Alice and Bob can move to building 2 since heights[0] < heights[2] and heights[1] < heights[2]. In the second query, Alice and Bob can move to building 5 since heights[0] < heights[5] and heights[3] < heights[5]. In the third query, Alice cannot meet Bob since Alice cannot move to any other building. In the fourth query, Alice and Bob can move to building 5 since heights[3] < heights[5] and heights[4] < heights[5]. In the fifth query, Alice and Bob are already in the same building. For ans[i] != -1, It can be shown that ans[i] is the leftmost building where Alice and Bob can meet. For ans[i] == -1, It can be shown that there is no building where Alice and Bob can meet.

Example 2:

Input:

heights = [5,3,8,2,6,1,4,6], queries = [[0,7],[3,5],[5,2],[3,0],[1,6]]

Output:

[7,6,-1,4,6]

Explanation:

In the first query, Alice can directly move to Bob's building since $\text{heights}[0] < \text{heights}[7]$. In the second query, Alice and Bob can move to building 6 since $\text{heights}[3] < \text{heights}[6]$ and $\text{heights}[5] < \text{heights}[6]$. In the third query, Alice cannot meet Bob since Bob cannot move to any other building. In the fourth query, Alice and Bob can move to building 4 since $\text{heights}[3] < \text{heights}[4]$ and $\text{heights}[0] < \text{heights}[4]$. In the fifth query, Alice can directly move to Bob's building since $\text{heights}[1] < \text{heights}[6]$. For $\text{ans}[i] \neq -1$, It can be shown that $\text{ans}[i]$ is the leftmost building where Alice and Bob can meet. For $\text{ans}[i] == -1$, It can be shown that there is no building where Alice and Bob can meet.

Constraints:

$1 \leq \text{heights.length} \leq 5 * 10^4$

4

$1 \leq \text{heights}[i] \leq 10^9$

9

$1 \leq \text{queries.length} \leq 5 * 10^4$

4

$\text{queries}[i] = [a$

i

$, b$

i

$]$

$0 \leq a$

i

, b

i

<= heights.length - 1

Code Snippets

C++:

```
class Solution {
public:
    vector<int> leftmostBuildingQueries(vector<int>& heights,
    vector<vector<int>>& queries) {

    }
};
```

Java:

```
class Solution {
    public int[] leftmostBuildingQueries(int[] heights, int[][] queries) {

    }
}
```

Python3:

```
class Solution:
    def leftmostBuildingQueries(self, heights: List[int], queries:
    List[List[int]]) -> List[int]:
```

Python:

```
class Solution(object):
    def leftmostBuildingQueries(self, heights, queries):
        """
        :type heights: List[int]
        :type queries: List[List[int]]
```

```
:rtype: List[int]
"""
```

JavaScript:

```
/**
 * @param {number[]} heights
 * @param {number[][]} queries
 * @return {number[]}
 */
var leftmostBuildingQueries = function(heights, queries) {

};
```

TypeScript:

```
function leftmostBuildingQueries(heights: number[], queries: number[][]):
number[] {

};
```

C#:

```
public class Solution {
    public int[] LeftmostBuildingQueries(int[] heights, int[][] queries) {

    }
}
```

C:

```
/**
 * Note: The returned array must be malloced, assume caller calls free().
 */
int* leftmostBuildingQueries(int* heights, int heightsSize, int** queries,
int queriesSize, int* queriesColSize, int* returnSize) {

}
```

Go:

```

func leftmostBuildingQueries(heights []int, queries [][]int) []int {

}

```

Kotlin:

```

class Solution {
    fun leftmostBuildingQueries(heights: IntArray, queries: Array<IntArray>):
        IntArray {

    }
}

```

Swift:

```

class Solution {
    func leftmostBuildingQueries(_ heights: [Int], _ queries: [[Int]]) -> [Int] {

    }
}

```

Rust:

```

impl Solution {
    pub fn leftmost_building_queries(heights: Vec<i32>, queries: Vec<Vec<i32>>)
        -> Vec<i32> {

    }
}

```

Ruby:

```

# @param {Integer[]} heights
# @param {Integer[][]} queries
# @return {Integer[]}
def leftmost_building_queries(heights, queries)

end

```

PHP:

```

class Solution {

```

```

/**
 * @param Integer[] $heights
 * @param Integer[][] $queries
 * @return Integer[]
 */
function leftmostBuildingQueries($heights, $queries) {

}
}

```

Dart:

```

class Solution {
  List<int> leftmostBuildingQueries(List<int> heights, List<List<int>> queries)
  {

  }
}

```

Scala:

```

object Solution {
  def leftmostBuildingQueries(heights: Array[Int], queries: Array[Array[Int]]):
  Array[Int] = {

  }
}

```

Elixir:

```

defmodule Solution do
  @spec leftmost_building_queries(heights :: [integer], queries :: [[integer]])
  :: [integer]
  def leftmost_building_queries(heights, queries) do

  end
end

```

Erlang:

```

-spec leftmost_building_queries(Heights :: [integer()], Queries ::
[[integer()]]) -> [integer()].

```

```
leftmost_building_queries(Heights, Queries) ->
.
```

Racket:

```
(define/contract (leftmost-building-queries heights queries)
  (-> (listof exact-integer?) (listof (listof exact-integer?)) (listof
    exact-integer?))
  )
```

Solutions

C++ Solution:

```
/*
 * Problem: Find Building Where Alice and Bob Can Meet
 * Difficulty: Hard
 * Tags: array, tree, search, stack, queue, heap
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(h) for recursion stack where h is height
 */

class Solution {
public:
    vector<int> leftmostBuildingQueries(vector<int>& heights,
    vector<vector<int>>& queries) {

    }

};
```

Java Solution:

```
/**
 * Problem: Find Building Where Alice and Bob Can Meet
 * Difficulty: Hard
 * Tags: array, tree, search, stack, queue, heap
 *
 * Approach: Use two pointers or sliding window technique
```

```

* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public int[] leftmostBuildingQueries(int[] heights, int[][] queries) {

}
}

```

Python3 Solution:

```

"""
Problem: Find Building Where Alice and Bob Can Meet
Difficulty: Hard
Tags: array, tree, search, stack, queue, heap

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class Solution:
    def leftmostBuildingQueries(self, heights: List[int], queries:
List[List[int]]) -> List[int]:
        # TODO: Implement optimized solution
        pass

```

Python Solution:

```

class Solution(object):
    def leftmostBuildingQueries(self, heights, queries):
        """
        :type heights: List[int]
        :type queries: List[List[int]]
        :rtype: List[int]
        """

```

JavaScript Solution:

```

/**
 * Problem: Find Building Where Alice and Bob Can Meet
 * Difficulty: Hard
 * Tags: array, tree, search, stack, queue, heap
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(h) for recursion stack where h is height
 */

/**
 * @param {number[]} heights
 * @param {number[][]} queries
 * @return {number[]}
 */
var leftmostBuildingQueries = function(heights, queries) {

};

```

TypeScript Solution:

```

/**
 * Problem: Find Building Where Alice and Bob Can Meet
 * Difficulty: Hard
 * Tags: array, tree, search, stack, queue, heap
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(h) for recursion stack where h is height
 */

function leftmostBuildingQueries(heights: number[], queries: number[][]):
number[] {

};

```

C# Solution:

```

/*
 * Problem: Find Building Where Alice and Bob Can Meet
 * Difficulty: Hard
 * Tags: array, tree, search, stack, queue, heap

```

```

*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public int[] LeftmostBuildingQueries(int[] heights, int[][] queries) {

}

}

```

C Solution:

```

/*
* Problem: Find Building Where Alice and Bob Can Meet
* Difficulty: Hard
* Tags: array, tree, search, stack, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* leftmostBuildingQueries(int* heights, int heightsSize, int** queries,
int queriesSize, int* queriesColSize, int* returnSize) {

}

```

Go Solution:

```

// Problem: Find Building Where Alice and Bob Can Meet
// Difficulty: Hard
// Tags: array, tree, search, stack, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

```

```

func leftmostBuildingQueries(heights []int, queries [][]int) []int {

}

```

Kotlin Solution:

```

class Solution {
    fun leftmostBuildingQueries(heights: IntArray, queries: Array<IntArray>):
        IntArray {

    }
}

```

Swift Solution:

```

class Solution {
    func leftmostBuildingQueries(_ heights: [Int], _ queries: [[Int]]) -> [Int] {

    }
}

```

Rust Solution:

```

// Problem: Find Building Where Alice and Bob Can Meet
// Difficulty: Hard
// Tags: array, tree, search, stack, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
    pub fn leftmost_building_queries(heights: Vec<i32>, queries: Vec<Vec<i32>> )
        -> Vec<i32> {

    }
}

```

Ruby Solution:

```

# @param {Integer[]} heights
# @param {Integer[][]} queries
# @return {Integer[]}
def leftmost_building_queries(heights, queries)

end

```

PHP Solution:

```

class Solution {

    /**
     * @param Integer[] $heights
     * @param Integer[][] $queries
     * @return Integer[]
     */
    function leftmostBuildingQueries($heights, $queries) {

    }

}

```

Dart Solution:

```

class Solution {
  List<int> leftmostBuildingQueries(List<int> heights, List<List<int>> queries)
  {

  }

}

```

Scala Solution:

```

object Solution {
  def leftmostBuildingQueries(heights: Array[Int], queries: Array[Array[Int]]):
    Array[Int] = {

  }

}

```

Elixir Solution:

```

defmodule Solution do
  @spec leftmost_building_queries(heights :: [integer], queries :: [[integer]])
    :: [integer]
  def leftmost_building_queries(heights, queries) do

  end

end

```

Erlang Solution:

```

-spec leftmost_building_queries(Heights :: [integer()], Queries ::
[[integer()]]) -> [integer()].
leftmost_building_queries(Heights, Queries) ->
.

```

Racket Solution:

```

(define/contract (leftmost-building-queries heights queries)
  (-> (listof exact-integer?) (listof (listof exact-integer?)) (listof
exact-integer?))
  )

```