Problem 3161: Block Placement Queries

Problem Information

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

Problem Description

There exists an infinite number line, with its origin at 0 and extending towards the

positive

x-axis.

You are given a 2D array

queries

, which contains two types of queries:

For a query of type 1,

queries[i] = [1, x]

. Build an obstacle at distance

x

from the origin. It is guaranteed that there is

no

obstacle at distance

x

when the query is asked.

For a query of type 2,

queries[i] = [2, x, sz]

. Check if it is possible to place a block of size

sz

anywhere

in the range

[0, x]

on the line, such that the block

entirely

lies in the range

[0, x]

. A block

cannot

be placed if it intersects with any obstacle, but it may touch it. Note that you do

not

actually place the block. Queries are separate.

Return a boolean array

results

, where

results[i]

is

true

if you can place the block specified in the

i

th

query of type 2, and

false

otherwise.

Example 1:

Input:

queries = [[1,2],[2,3,3],[2,3,1],[2,2,2]]

Output:

[false,true,true]

Explanation:

For query 0, place an obstacle at

x = 2

. A block of size at most 2 can be placed before

x = 3

.

Example 2:

Input:

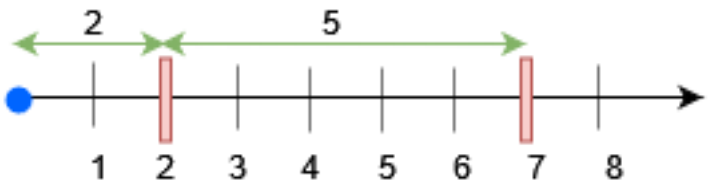
queries =

[[1,7],[2,7,6],[1,2],[2,7,5],[2,7,6]]

Output:

[true,true,false]

Explanation:



Place an obstacle at

x = 7

for query 0. A block of size at most 7 can be placed before

x = 7

.

Place an obstacle at

x = 2

for query 2. Now, a block of size at most 5 can be placed before

x = 7

, and a block of size at most 2 before

x = 2

.

Constraints:

1 <= queries.length <= 15 * 10

4

2 <= queries[i].length <= 3

1 <= queries[i][0] <= 2

1 <= x, sz <= min(5 * 10

4

, 3 * queries.length)

The input is generated such that for queries of type 1, no obstacle exists at distance

x

when the query is asked.

The input is generated such that there is at least one query of type 2.

Code Snippets

C++:

```
class Solution {
public:
vector<bool> getResults(vector<vector<int>>& queries) {

}
};
```

Java:

```
class Solution {
public List<Boolean> getResults(int[][] queries) {

}
}
```

Python3:

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

Python:

```
class Solution(object):
def getResults(self, queries):
"""
:type queries: List[List[int]]
:rtype: List[bool]
"""
```

JavaScript:

```
/**
* @param {number[][]} queries
* @return {boolean[]}
*/
var getResults = function(queries) {

};
```

TypeScript:

```
function getResults(queries: number[][]): boolean[] {

};
```

C#:

```
public class Solution {
public IList<bool> GetResults(int[][] queries) {

}
}
```

C:

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

}
```

Go:

```
func getResults(queries [][]int) []bool {

}
```

Kotlin:

```
class Solution {
fun getResults(queries: Array<IntArray>): List<Boolean> {

}
}
```

Swift:

```
class Solution {
func getResults(_ queries: [[Int]]) -> [Bool] {

}
}
```

Rust:

```
impl Solution {
pub fn get_results(queries: Vec<Vec<i32>>) -> Vec<bool> {

}
}
```

Ruby:

```
# @param {Integer[][]} queries
# @return {Boolean[]}
def get_results(queries)

end
```

PHP:

```
class Solution {

/**
```

```
* @param Integer[][] $queries
* @return Boolean[]
*/
function getResults($queries) {

}
}
```

Dart:

```
class Solution {
List<bool> getResults(List<List<int>> queries) {

}
}
```

Scala:

```
object Solution {
def getResults(queries: Array[Array[Int]]): List[Boolean] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec get_results(queries :: [[integer]]) :: [boolean]
def get_results(queries) do

end
end
```

Erlang:

```
-spec get_results(Queries :: [[integer()]]) -> [boolean()].
get_results(Queries) ->
.
```

Racket:

```
(define/contract (get-results queries)
(-> (listof (listof exact-integer?)) (listof boolean?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Block Placement Queries
* Difficulty: Hard
* Tags: array, tree, search
*
* 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<bool> getResults(vector<vector<int>>& queries) {

}
};
```

Java Solution:

```
/**
* Problem: Block Placement Queries
* Difficulty: Hard
* Tags: array, tree, search
*
* 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 List<Boolean> getResults(int[][] queries) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Block Placement Queries
Difficulty: Hard
Tags: array, tree, search

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 getResults(self, queries: List[List[int]]) -> List[bool]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def getResults(self, queries):
"""
:type queries: List[List[int]]
:rtype: List[bool]
"""
```

JavaScript Solution:

```
/**
* Problem: Block Placement Queries
* Difficulty: Hard
* Tags: array, tree, search
*
* 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[][]} queries
* @return {boolean[]}
*/
var getResults = function(queries) {

};
```

TypeScript Solution:

```
/**
* Problem: Block Placement Queries
* Difficulty: Hard
* Tags: array, tree, search
*
* 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 getResults(queries: number[][]): boolean[] {

};
```

C# Solution:

```
/*
* Problem: Block Placement Queries
* Difficulty: Hard
* Tags: array, tree, search
*
* 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 IList<bool> GetResults(int[][] queries) {

}
}
```

C Solution:

```
/*
* Problem: Block Placement Queries
* Difficulty: Hard
* Tags: array, tree, search
*
* 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().
*/
bool* getResults(int** queries, int queriesSize, int* queriesColSize, int*
returnSize) {

}
```

Go Solution:

```
// Problem: Block Placement Queries
// Difficulty: Hard
// Tags: array, tree, search
//
// 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 getResults(queries [][]int) []bool {

}
```

Kotlin Solution:

```
class Solution {
fun getResults(queries: Array<IntArray>): List<Boolean> {

}
}
```

Swift Solution:

```
class Solution {
func getResults(_ queries: [[Int]]) -> [Bool] {

}
}
```

Rust Solution:

```
// Problem: Block Placement Queries
// Difficulty: Hard
// Tags: array, tree, search
//
// 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 get_results(queries: Vec<Vec<i32>>) -> Vec<bool> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} queries
# @return {Boolean[]}
def get_results(queries)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $queries
* @return Boolean[]
*/
function getResults($queries) {
```

```
}
}
```

Dart Solution:

```
class Solution {
List<bool> getResults(List<List<int>> queries) {

}
}
```

Scala Solution:

```
object Solution {
def getResults(queries: Array[Array[Int]]): List[Boolean] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec get_results(queries :: [[integer]]) :: [boolean]
def get_results(queries) do

end
end
```

Erlang Solution:

```
-spec get_results(Queries :: [[integer()]]) -> [boolean()].
get_results(Queries) ->
.
```

Racket Solution:

```
(define/contract (get-results queries)
(-> (listof (listof exact-integer?)) (listof boolean?))
)
```