Problem 1642: Furthest Building You Can Reach

Problem Information

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

Problem Description

You are given an integer array

heights

representing the heights of buildings, some

bricks

, and some

ladders

.

You start your journey from building

0

and move to the next building by possibly using bricks or ladders.

While moving from building

i

to building

i+1

(

0-indexed

),

If the current building's height is

greater than or equal

to the next building's height, you do

not

need a ladder or bricks.

If the current building's height is

less than

the next building's height, you can either use

one ladder

or

(h[i+1] - h[i])

bricks

.

Return the furthest building index (0-indexed) you can reach if you use the given ladders and bricks optimally.

Example 1:

Input:

heights = [4,2,7,6,9,14,12], bricks = 5, ladders = 1

Output:

4

Explanation:

Starting at building 0, you can follow these steps: - Go to building 1 without using ladders nor bricks since 4 >= 2. - Go to building 2 using 5 bricks. You must use either bricks or ladders because 2 < 7. - Go to building 3 without using ladders nor bricks since 7 >= 6. - Go to building 4 using your only ladder. You must use either bricks or ladders because 6 < 9. It is impossible to go beyond building 4 because you do not have any more bricks or ladders.

Example 2:

Input:

heights = [4,12,2,7,3,18,20,3,19], bricks = 10, ladders = 2

Output:

7

Example 3:

Input:

heights = [14,3,19,3], bricks = 17, ladders = 0

Output:

3

Constraints:

1 <= heights.length <= 10

5

1 <= heights[i] <= 10

6

0 <= bricks <= 10

9

0 <= ladders <= heights.length

Code Snippets

C++:

```
class Solution {
public:
int furthestBuilding(vector<int>& heights, int bricks, int ladders) {

}
};
```

Java:

```
class Solution {
public int furthestBuilding(int[] heights, int bricks, int ladders) {

}
}
```

Python3:

```
class Solution:
def furthestBuilding(self, heights: List[int], bricks: int, ladders: int) ->
int:
```

Python:

```
class Solution(object):
def furthestBuilding(self, heights, bricks, ladders):
"""
:type heights: List[int]
:type bricks: int
:type ladders: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} heights
* @param {number} bricks
* @param {number} ladders
* @return {number}
*/
var furthestBuilding = function(heights, bricks, ladders) {

};
```

TypeScript:

```
function furthestBuilding(heights: number[], bricks: number, ladders:
number): number {

};
```

C#:

```
public class Solution {
public int FurthestBuilding(int[] heights, int bricks, int ladders) {

}
}
```

C:

```
int furthestBuilding(int* heights, int heightsSize, int bricks, int ladders)
{

}
```

Go:

```
func furthestBuilding(heights []int, bricks int, ladders int) int {

}
```

Kotlin:

```
class Solution {
fun furthestBuilding(heights: IntArray, bricks: Int, ladders: Int): Int {

}
}
```

Swift:

```
class Solution {
func furthestBuilding(_ heights: [Int], _ bricks: Int, _ ladders: Int) -> Int
{

}
```

```
}
```

Rust:

```
impl Solution {
pub fn furthest_building(heights: Vec<i32>, bricks: i32, ladders: i32) -> i32
{

}
}
```

Ruby:

```
# @param {Integer[]} heights
# @param {Integer} bricks
# @param {Integer} ladders
# @return {Integer}
def furthest_building(heights, bricks, ladders)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $heights
* @param Integer $bricks
* @param Integer $ladders
* @return Integer
*/
function furthestBuilding($heights, $bricks, $ladders) {

}
}
```

Dart:

```
class Solution {
int furthestBuilding(List<int> heights, int bricks, int ladders) {

}
```

```
}
```

Scala:

```
object Solution {
def furthestBuilding(heights: Array[Int], bricks: Int, ladders: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec furthest_building(heights :: [integer], bricks :: integer, ladders ::
integer) :: integer
def furthest_building(heights, bricks, ladders) do

end
end
```

Erlang:

```
-spec furthest_building(Heights :: [integer()], Bricks :: integer(), Ladders
:: integer()) -> integer().
furthest_building(Heights, Bricks, Ladders) ->
.
```

Racket:

```
(define/contract (furthest-building heights bricks ladders)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Furthest Building You Can Reach
* Difficulty: Medium
```

```
* Tags: array, greedy, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int furthestBuilding(vector<int>& heights, int bricks, int ladders) {

}
};
```

Java Solution:

```
/**
* Problem: Furthest Building You Can Reach
* Difficulty: Medium
* Tags: array, greedy, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int furthestBuilding(int[] heights, int bricks, int ladders) {

}
}
```

Python3 Solution:

```
"""
Problem: Furthest Building You Can Reach
Difficulty: Medium
Tags: array, greedy, queue, heap

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
```

```
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def furthestBuilding(self, heights: List[int], bricks: int, ladders: int) ->
int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def furthestBuilding(self, heights, bricks, ladders):
"""
:type heights: List[int]
:type bricks: int
:type ladders: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Furthest Building You Can Reach
* Difficulty: Medium
* Tags: array, greedy, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number[]} heights
* @param {number} bricks
* @param {number} ladders
* @return {number}
*/
var furthestBuilding = function(heights, bricks, ladders) {

};
```

TypeScript Solution:

```
/**
* Problem: Furthest Building You Can Reach
* Difficulty: Medium
* Tags: array, greedy, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function furthestBuilding(heights: number[], bricks: number, ladders:
number): number {

};
```

C# Solution:

```
/*
* Problem: Furthest Building You Can Reach
* Difficulty: Medium
* Tags: array, greedy, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int FurthestBuilding(int[] heights, int bricks, int ladders) {

}
}
```

C Solution:

```
/*
* Problem: Furthest Building You Can Reach
* Difficulty: Medium
* Tags: array, greedy, queue, heap
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

int furthestBuilding(int* heights, int heightsSize, int bricks, int ladders)
{

}
```

Go Solution:

```
// Problem: Furthest Building You Can Reach
// Difficulty: Medium
// Tags: array, greedy, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func furthestBuilding(heights []int, bricks int, ladders int) int {

}
```

Kotlin Solution:

```
class Solution {
fun furthestBuilding(heights: IntArray, bricks: Int, ladders: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func furthestBuilding(_ heights: [Int], _ bricks: Int, _ ladders: Int) -> Int
{

}
}
```

Rust Solution:

```
// Problem: Furthest Building You Can Reach
// Difficulty: Medium
// Tags: array, greedy, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn furthest_building(heights: Vec<i32>, bricks: i32, ladders: i32) -> i32
{

}
}
```

Ruby Solution:

```
# @param {Integer[]} heights
# @param {Integer} bricks
# @param {Integer} ladders
# @return {Integer}
def furthest_building(heights, bricks, ladders)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $heights
* @param Integer $bricks
* @param Integer $ladders
* @return Integer
*/
function furthestBuilding($heights, $bricks, $ladders) {

}
}
```

Dart Solution:

```
class Solution {
int furthestBuilding(List<int> heights, int bricks, int ladders) {

}
}
```

Scala Solution:

```
object Solution {
def furthestBuilding(heights: Array[Int], bricks: Int, ladders: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec furthest_building(heights :: [integer], bricks :: integer, ladders ::
integer) :: integer
def furthest_building(heights, bricks, ladders) do

end
end
```

Erlang Solution:

```
-spec furthest_building(Heights :: [integer()], Bricks :: integer(), Ladders
:: integer()) -> integer().
furthest_building(Heights, Bricks, Ladders) ->
.
```

Racket Solution:

```
(define/contract (furthest-building heights bricks ladders)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```