Problem 675: Cut Off Trees for Golf Event

Problem Information

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

Problem Description

You are asked to cut off all the trees in a forest for a golf event. The forest is represented as an

m x n

matrix. In this matrix:

0

means the cell cannot be walked through.

1

represents an empty cell that can be walked through.

A number greater than

1

represents a tree in a cell that can be walked through, and this number is the tree's height.

In one step, you can walk in any of the four directions: north, east, south, and west. If you are standing in a cell with a tree, you can choose whether to cut it off.

You must cut off the trees in order from shortest to tallest. When you cut off a tree, the value at its cell becomes

1

(an empty cell).

Starting from the point

(0, 0)

, return

the minimum steps you need to walk to cut off all the trees

. If you cannot cut off all the trees, return

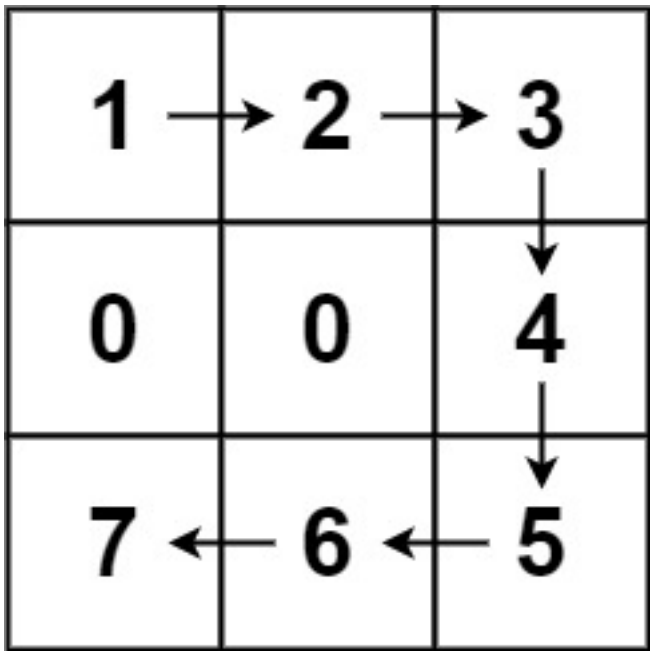
-1

.

Note:

The input is generated such that no two trees have the same height, and there is at least one tree needs to be cut off.

Example 1:



Input:

forest = [[1,2,3],[0,0,4],[7,6,5]]

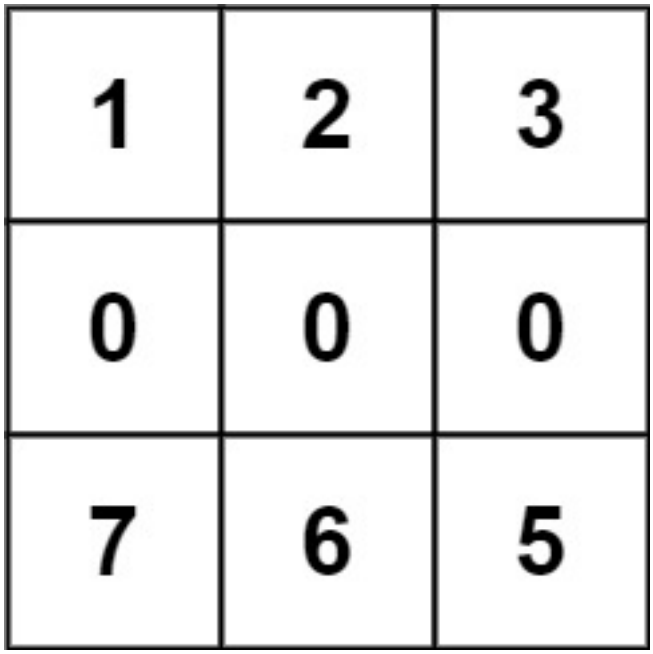
Output:

6

Explanation:

Following the path above allows you to cut off the trees from shortest to tallest in 6 steps.

Example 2:



Input:

forest = [[1,2,3],[0,0,0],[7,6,5]]

Output:

-1

Explanation:

The trees in the bottom row cannot be accessed as the middle row is blocked.

Example 3:

Input:

forest = [[2,3,4],[0,0,5],[8,7,6]]

Output:

6

Explanation:

You can follow the same path as Example 1 to cut off all the trees. Note that you can cut off the first tree at (0, 0) before making any steps.

Constraints:

m == forest.length

n == forest[i].length

1 <= m, n <= 50

0 <= forest[i][j] <= 10

9

Heights of all trees are

distinct

.

Code Snippets

C++:

```
class Solution {
public:
int cutOffTree(vector<vector<int>>& forest) {

}
};
```

Java:

```
class Solution {
public int cutOffTree(List<List<Integer>> forest) {

}
}
```

Python3:

```
class Solution:
def cutOffTree(self, forest: List[List[int]]) -> int:
```

Python:

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

JavaScript:

```
/**
* @param {number[][]} forest
* @return {number}
*/
var cutOffTree = function(forest) {

};
```

TypeScript:

```
function cutOffTree(forest: number[][]): number {

};
```

C#:

```
public class Solution {
public int CutOffTree(IList<IList<int>> forest) {

}
}
```

C:

```
int cutOffTree(int** forest, int forestSize, int* forestColSize) {

}
```

Go:

```
func cutOffTree(forest [][]int) int {

}
```

Kotlin:

```
class Solution {
fun cutOffTree(forest: List<List<Int>>): Int {

}
}
```

Swift:

```
class Solution {
func cutOffTree(_ forest: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn cut_off_tree(forest: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} forest
# @return {Integer}
def cut_off_tree(forest)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $forest
* @return Integer
*/
function cutOffTree($forest) {

}
}
```

Dart:

```
class Solution {
int cutOffTree(List<List<int>> forest) {

}
}
```

Scala:

```
object Solution {
def cutOffTree(forest: List[List[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec cut_off_tree(forest :: [[integer]]) :: integer
def cut_off_tree(forest) do

end
end
```

Erlang:

```
-spec cut_off_tree(Forest :: [[integer()]]) -> integer().
cut_off_tree(Forest) ->
.
```

Racket:

```
(define/contract (cut-off-tree forest)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Cut Off Trees for Golf Event
* Difficulty: Hard
* Tags: array, tree, search, 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 cutOffTree(vector<vector<int>>& forest) {

}
};
```

Java Solution:

```
/**
* Problem: Cut Off Trees for Golf Event
* Difficulty: Hard
* Tags: array, tree, search, 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 cutOffTree(List<List<Integer>> forest) {

}
}
```

Python3 Solution:

```
"""
Problem: Cut Off Trees for Golf Event
Difficulty: Hard
Tags: array, tree, search, 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 cutOffTree(self, forest: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```javascript
/**
* Problem: Cut Off Trees for Golf Event
* Difficulty: Hard
* Tags: array, tree, search, 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[][]} forest
* @return {number}
*/
var cutOffTree = function(forest) {

};
```

TypeScript Solution:

```typescript
/**
* Problem: Cut Off Trees for Golf Event
* Difficulty: Hard
* Tags: array, tree, search, 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 cutOffTree(forest: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Cut Off Trees for Golf Event
* Difficulty: Hard
* Tags: array, tree, search, 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 CutOffTree(IList<IList<int>> forest) {

}
}
```

C Solution:

```
/*
* Problem: Cut Off Trees for Golf Event
* Difficulty: Hard
* Tags: array, tree, search, 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
*/

int cutOffTree(int** forest, int forestSize, int* forestColSize) {

}
```

Go Solution:

```
// Problem: Cut Off Trees for Golf Event
// Difficulty: Hard
// Tags: array, tree, search, 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 cutOffTree(forest [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun cutOffTree(forest: List<List<Int>>): Int {

}
}
```

Swift Solution:

```
class Solution {
func cutOffTree(_ forest: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Cut Off Trees for Golf Event
// Difficulty: Hard
// Tags: array, tree, search, 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 cut_off_tree(forest: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} forest
# @return {Integer}
def cut_off_tree(forest)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $forest
* @return Integer
*/
function cutOffTree($forest) {

}
}
```

Dart Solution:

```
class Solution {
int cutOffTree(List<List<int>> forest) {

}
}
```

Scala Solution:

```
object Solution {
def cutOffTree(forest: List[List[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec cut_off_tree(forest :: [[integer]]) :: integer
def cut_off_tree(forest) do

end
end
```

Erlang Solution:

```
-spec cut_off_tree(Forest :: [[integer()]]) -> integer().
cut_off_tree(Forest) ->
.
```

Racket Solution:

```
(define/contract (cut-off-tree forest)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```