

Problem 174: Dungeon Game

Problem Information

Difficulty: Hard

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

The demons had captured the princess and imprisoned her in
the bottom-right corner
of a
dungeon
. The
dungeon
consists of
 $m \times n$
rooms laid out in a 2D grid. Our valiant knight was initially positioned in
the top-left room
and must fight his way through
dungeon
to rescue the princess.

The knight has an initial health point represented by a positive integer. If at any point his health point drops to

0

or below, he dies immediately.

Some of the rooms are guarded by demons (represented by negative integers), so the knight loses health upon entering these rooms; other rooms are either empty (represented as 0) or contain magic orbs that increase the knight's health (represented by positive integers).

To reach the princess as quickly as possible, the knight decides to move only

rightward

or

downward

in each step.

Return

the knight's minimum initial health so that he can rescue the princess

.

Note

that any room can contain threats or power-ups, even the first room the knight enters and the bottom-right room where the princess is imprisoned.

Example 1:

-2 →	-3 →	3 ↓
-5	-10	1 ↓
10	30	-5

Input:

dungeon = [[-2,-3,3],[-5,-10,1],[10,30,-5]]

Output:

7

Explanation:

The initial health of the knight must be at least 7 if he follows the optimal path: RIGHT-> RIGHT -> DOWN -> DOWN.

Example 2:

Input:

dungeon = [[0]]

Output:

1

Constraints:

m == dungeon.length

n == dungeon[i].length

1 <= m, n <= 200

-1000 <= dungeon[i][j] <= 1000

Code Snippets

C++:

```
class Solution {
public:
    int calculateMinimumHP(vector<vector<int>>& dungeon) {

    }
};
```

Java:

```
class Solution {
    public int calculateMinimumHP(int[][] dungeon) {

    }
}
```

Python3:

```
class Solution:
    def calculateMinimumHP(self, dungeon: List[List[int]]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function calculateMinimumHP(dungeon: number[][]): number {

};
```

C#:

```
public class Solution {
    public int CalculateMinimumHP(int[][] dungeon) {

    }
}
```

C:

```
int calculateMinimumHP(int** dungeon, int dungeonSize, int* dungeonColSize) {

}
```

Go:

```
func calculateMinimumHP(dungeon [][]int) int {

}
```

Kotlin:

```
class Solution {
    fun calculateMinimumHP(dungeon: Array<IntArray>): Int {

    }
}
```

Swift:

```
class Solution {  
    func calculateMinimumHP(_ dungeon: [[Int]]) -> Int {  
  
    }  
}
```

Rust:

```
impl Solution {  
    pub fn calculate_minimum_hp(dungeon: Vec<Vec<i32>>) -> i32 {  
  
    }  
}
```

Ruby:

```
# @param {Integer[][]} dungeon  
# @return {Integer}  
def calculate_minimum_hp(dungeon)  
  
end
```

PHP:

```
class Solution {  
  
    /**  
     * @param Integer[][] $dungeon  
     * @return Integer  
     */  
    function calculateMinimumHP($dungeon) {  
  
    }  
}
```

Dart:

```
class Solution {  
    int calculateMinimumHP(List<List<int>> dungeon) {  
  
    }  
}
```

```
}
```

Scala:

```
object Solution {  
  def calculateMinimumHP(dungeon: Array[Array[Int]]): Int = {  
  
  }  
}
```

Elixir:

```
defmodule Solution do  
  @spec calculate_minimum_hp(dungeon :: [[integer]]) :: integer  
  def calculate_minimum_hp(dungeon) do  
  
  end  
end
```

Erlang:

```
-spec calculate_minimum_hp(Dungeon :: [[integer()]]) -> integer().  
calculate_minimum_hp(Dungeon) ->  
.
```

Racket:

```
(define/contract (calculate-minimum-hp dungeon)  
  (-> (listof (listof exact-integer?)) exact-integer?)  
)
```

Solutions

C++ Solution:

```
/*  
 * Problem: Dungeon Game  
 * Difficulty: Hard  
 * Tags: array, dp  
 */
```

```

* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int calculateMinimumHP(vector<vector<int>>& dungeon) {

}
};

```

Java Solution:

```

/**
 * Problem: Dungeon Game
 * Difficulty: Hard
 * Tags: array, dp
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) or O(n * m) for DP table
 */

class Solution {
public int calculateMinimumHP(int[][] dungeon) {

}
}

```

Python3 Solution:

```

"""
Problem: Dungeon Game
Difficulty: Hard
Tags: array, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

```

```

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

```

Python Solution:

```

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

```

JavaScript Solution:

```

/**
 * Problem: Dungeon Game
 * Difficulty: Hard
 * Tags: array, dp
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) or O(n * m) for DP table
 */

/**
 * @param {number[][]} dungeon
 * @return {number}
 */
var calculateMinimumHP = function(dungeon) {

};

```

TypeScript Solution:

```

/**
 * Problem: Dungeon Game
 * Difficulty: Hard
 * Tags: array, dp

```

```

*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function calculateMinimumHP(dungeon: number[][]): number {

};

```

C# Solution:

```

/*
* Problem: Dungeon Game
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
    public int CalculateMinimumHP(int[][] dungeon) {

    }
}

```

C Solution:

```

/*
* Problem: Dungeon Game
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int calculateMinimumHP(int** dungeon, int dungeonSize, int* dungeonColSize) {

```

```
}
```

Go Solution:

```
// Problem: Dungeon Game
// Difficulty: Hard
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func calculateMinimumHP(dungeon [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
    fun calculateMinimumHP(dungeon: Array<IntArray>): Int {

    }
}
```

Swift Solution:

```
class Solution {
    func calculateMinimumHP(_ dungeon: [[Int]]) -> Int {

    }
}
```

Rust Solution:

```
// Problem: Dungeon Game
// Difficulty: Hard
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
```

```
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
    pub fn calculate_minimum_hp(dungeon: Vec<Vec<i32>>) -> i32 {

    }
}
```

Ruby Solution:

```
# @param {Integer[][]} dungeon
# @return {Integer}
def calculate_minimum_hp(dungeon)

end
```

PHP Solution:

```
class Solution {

    /**
     * @param Integer[][] $dungeon
     * @return Integer
     */
    function calculateMinimumHP($dungeon) {

    }
}
```

Dart Solution:

```
class Solution {
    int calculateMinimumHP(List<List<int>> dungeon) {

    }
}
```

Scala Solution:

```
object Solution {
    def calculateMinimumHP(dungeon: Array[Array[Int]]): Int = {
```

```
}  
}
```

Elixir Solution:

```
defmodule Solution do  
  @spec calculate_minimum_hp(dungeon :: [[integer]]) :: integer  
  def calculate_minimum_hp(dungeon) do  
  
  end  
end
```

Erlang Solution:

```
-spec calculate_minimum_hp(Dungeon :: [[integer()]]) -> integer().  
calculate_minimum_hp(Dungeon) ->  
.
```

Racket Solution:

```
(define/contract (calculate-minimum-hp dungeon)  
  (-> (listof (listof exact-integer?)) exact-integer?)  
)
```