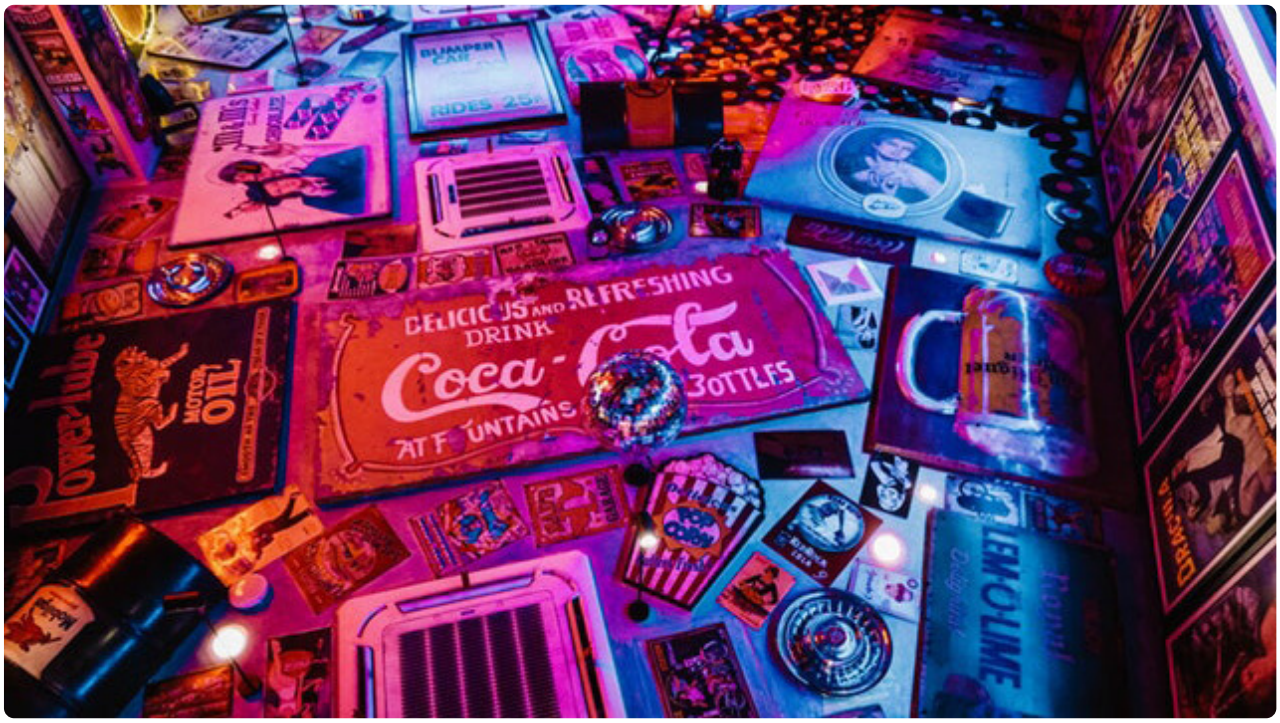
35 地下城游戏

更新时间：2019-09-25 09:46:48



> 宝剑锋从磨砺出，梅花香自苦寒来。
>
> ——佚名

刷题内容

> 难度: **Hard**
>
> 题目链接：https://leetcode.com/problems/dungeon-game/

题目描述

一些恶魔抓住了公主（P）并将她关在了地下城的右下角。地下城是由 M x N 个房间组成的二维网格。我们英勇的骑士（K）最初被安置在左上角的房间里，他必须穿过地下城并通过对抗恶魔来拯救公主。

骑士的初始健康点数为一个正整数。如果他的健康点数在某一时刻降至 0 或以下，他会立即死亡。

有些房间由恶魔守卫，因此骑士在进入这些房间时会失去健康点数（若房间里的值为负整数，则表示骑士将损失健康点数）；其他房间要么是空的（房间里的值为 0），要么包含增加骑士健康点数的魔法球（若房间里的值为正整数，则表示骑士将增加健康点数）。

为了尽快到达公主，骑士决定每次只向右或向下移动一步。

编写一个函数来计算确保骑士能够拯救到公主所需的最低初始健康点数。

例如，考虑到如下布局的地下城，如果骑士遵循最佳路径 右 -> 右 -> 下 -> 下，则骑士的初始健康点数至少为 7。

-2 (K) -3 3
-5 -10 1
10 30 -5 (P)

说明:

骑士的健康点数没有上限。

任何房间都可能对骑士的健康点数造成威胁，也可能增加骑士的健康点数，包括骑士进入的左上角房间以及公主被监禁的右下角房间。

解题方案

思路1 时间复杂度: O(row * col) 空间复杂度: O(row * col)

设左上角坐标为(0,0)

假设我们已经知道了从(0,1)和从(1,0)出发所需要的最少血量dp[0][1]和dp[1][0]，那自然dp[0][0]就知道了，其中dp[i][j] 代表从dungeon[i][j]出发救出公主所需要的最少血量

- 如果在cell dungeon[i][j]中能够加的血量足够进入下一个格子（即从dungeon[i][j]出来后还剩下的血量达到了从dp[i+1][j], dp[i][j+1]出发的最小血量），那么我们在进入dungeon[i][j]前只要血量够活就行，即1滴血
- 如果在cell dungeon[i][j]中能够加的血量不能进入下一个格子（即从dungeon[i][j]出来后还剩下的血量没有达到从dp[i+1][j], dp[i][j+1]出发的最小血量），假设差k滴血，那么我们在进入dungeon[i][j]前就需要拥有差的那些血量，即k滴血

Python

beats 74.12%

```python
class Solution:
    def calculateMinimumHP(self, dungeon: List[List[int]]) -> int:
        row = len(dungeon)
        col = len(dungeon[0]) if row else 0

        dp = [[sys.maxsize] * (col+1) for i in range(row+1)]
        # 初始化右下角下方和右上角右边这两个格子，即从右下角出来都只需要1滴血即可存活
        dp[-1][-2], dp[-2][-1] = 1, 1

        for i in range(row-1, -1, -1):
            for j in range(col-1, -1, -1):
                # 在dungeon[i][j]中加的血是否达到了从dungeon[i+1][j]和dungeon[i][j+1]两者中出发的最小血量
                tmp = dungeon[i][j] - min(dp[i+1][j], dp[i][j+1])
                if tmp >= 0: # 够, 只要1滴血能进dungeon[i][j]就行
                    dp[i][j] = 1
                else:      # 不够，初始就要这么多血的差值才能进来
                    dp[i][j] = -tmp

        return dp[0][0]
```

Go

beats 100%

```go
import "math"
func calculateMinimumHP(dungeon [][]int) int {
    row := len(dungeon)
    col := len(dungeon[0])

    dp := make([][]int, row+1)
    for i := 0; i < row + 1; i++ {
        dp[i] = make([]int, col+1)
        for j := 0; j < col + 1; j++ {
            dp[i][j] = math.MaxInt32
        }
    }
    // 初始化右下角下方和右上角右边这两个格子，即从右下角出来都只需要1滴血即可存活
    dp[row][col-1], dp[row-1][col] = 1, 1
    for i := row-1; i > -1; i-- {
        for j := col-1; j > -1; j-- {
            // 在dungeon[i][j]中加的血是否达到了从dungeon[i+1][j]和dungeon[i][j+1]两者中出发的最小血量
            tmp := dungeon[i][j] - int(math.Min(float64(dp[i+1][j]), float64(dp[i][j+1])))
            if tmp >= 0 { // 够, 只要1滴血能进dungeon[i][j]就行
                dp[i][j] = 1
            } else { // 不够，初始就要这么多血的差值才能进来
                dp[i][j] = -tmp
            }
        }
    }
    return dp[0][0]
}
```

Java

beats 100%

```java
class Solution {
    public int calculateMinimumHP(int[][] dungeon) {
        int row = dungeon.length;
        int col = dungeon[0].length;

        int[][] dp = new int[row+1][col+1];
        for (int i = 0; i < row + 1; i++) {
            dp[i] = new int[col+1];
            for (int j = 0; j < col + 1; j++) {
                dp[i][j] = Integer.MAX_VALUE;
            }
        }
        // 初始化右下角下方和右上角右边这两个格子，即从右下角出来都只需要1滴血即可存活
        dp[row][col-1] = 1;
        dp[row-1][col] = 1;
        for (int i = row-1; i > -1; i--) {
            for (int j = col-1; j > -1; j--) {
                // 在dungeon[i][j]中加的血是否达到了从dungeon[i+1][j]和dungeon[i][j+1]两者中出发的最小血量
                int tmp = dungeon[i][j] - (int) (Math.min(dp[i+1][j], dp[i][j+1]));
                if (tmp >= 0) { // 够，只要1滴血能进dungeon[i][j]就行
                    dp[i][j] = 1;
                } else { // 不够，初始就要这么多血的差值才能进来
                    dp[i][j] = -tmp;
                }
            }
        }
        return dp[0][0];
    }
}
```

C++

beats 100%

```cpp
class Solution {
public:
    int calculateMinimumHP(vector<vector<int>>& dungeon) {
        int row = dungeon.size();
        int col = dungeon[0].size();

        int dp[row+1][col+1];
        for (int i = 0; i < row + 1; i++) {
            for (int j = 0; j < col + 1; j++) {
                dp[i][j] = INT_MAX;
            }
        }
        // 初始化右下角下方和右上角右边这两个格子，即从右下角出来都只需要1滴血即可存活
        dp[row][col-1] = 1;
        dp[row-1][col] = 1;
        for (int i = row-1; i > -1; i--) {
            for (int j = col-1; j > -1; j--) {
                // 在dungeon[i][j]中加的血是否达到了从dungeon[i+1][j]和dungeon[i][j+1]两者中出发的最小血量
                int tmp = dungeon[i][j] - (int) (min(dp[i+1][j], dp[i][j+1]));
                if (tmp >= 0) { // 够，只要1滴血能进dungeon[i][j]就行
                    dp[i][j] = 1;
                } else { // 不够，初始就要这么多血的差值才能进来
                    dp[i][j] = -tmp;
                }
            }
        }
        return dp[0][0];
    }
};
```

思路2 时间复杂度: O(row * col) 空间复杂度: O(col)

观察上一种思路的dp公式可发现，对于dp表中的任意一格，只依赖它右边和下面的格子，所以可以压缩整个dp表到一维从而达成线性空间复杂度。

Python

beats 97.58%

```python
class Solution:
    def calculateMinimumHP(self, dungeon: List[List[int]]) -> int:
        row = len(dungeon)
        col = len(dungeon[0]) if row else 0

        dp = [sys.maxsize] * (col+1)
        dp[-2] = 1
        for i in range(row-1, -1, -1):
            for j in range(col-1, -1, -1):
                # 在dungeon[i][j]中加的血是否达到了从dungeon[i+1][j]和dungeon[i][j+1]两者中出发的最小血量
                tmp = dungeon[i][j] - min(dp[j], dp[j + 1])
                if tmp >= 0: # 够, 只要1滴血能进dungeon[i][j]就行
                    dp[j] = 1
                else:       # 不够，初始就要这么多血的差值才能进来
                    dp[j] = -tmp

        return dp[0]
```

Go

beats 100%

```go
import "math"
func calculateMinimumHP(dungeon [][]int) int {
    row := len(dungeon)
    col := len(dungeon[0])

    dp := make([]int, col+1)
    for i := 0; i < col + 1; i++ {
        dp[i] = math.MaxInt32
    }
    // 初始化右下角下方和右上角右边这两个格子，即从右下角出来都只需要1滴血即可存活
    dp[col-1] = 1
    for i := row-1; i > -1; i-- {
        for j := col-1; j > -1; j-- {
            // 在dungeon[i][j]中加的血是否达到了从dungeon[i+1][j]和dungeon[i][j+1]两者中出发的最小血量
            tmp := dungeon[i][j] - int(math.Min(float64(dp[j]), float64(dp[j+1])))
            if tmp >= 0 { // 够, 只要1滴血能进dungeon[i][j]就行
                dp[j] = 1
            } else { // 不够，初始就要这么多血的差值才能进来
                dp[j] = -tmp
            }
        }
    }
    return dp[0]
}
```

Java

beats 100%

```java
class Solution {
    public int calculateMinimumHP(int[][] dungeon) {
        int row = dungeon.length;
        int col = dungeon[0].length;

        int[] dp = new int[col+1];
        for (int i = 0; i < col + 1; i++) {
            dp[i] = Integer.MAX_VALUE;
        }
        // 初始化右下角下方和右上角右边这两个格子，即从右下角出来都只需要1滴血即可存活
        dp[col-1] = 1;
        for (int i = row-1; i > -1; i--) {
            for (int j = col-1; j > -1; j--) {
                // 在dungeon[i][j]中加的血是否达到了从dungeon[i+1][j]和dungeon[i][j+1]两者中出发的最小血量
                int tmp = dungeon[i][j] - (int) (Math.min(dp[j], dp[j+1]));
                if (tmp >= 0) { // 够, 只要1滴血能进dungeon[i][j]就行
                    dp[j] = 1;
                } else { // 不够，初始就要这么多血的差值才能进来
                    dp[j] = -tmp;
                }
            }
        }
        return dp[0];
    }
}
```

C++

beats 100%

```java
class Solution {
public:
    int calculateMinimumHP(vector<vector<int>>& dungeon) {
        int row = dungeon.size();
        int col = dungeon[0].size();

        int dp[col+1];
        for (int i = 0; i < col + 1; i++) {
            dp[i] = INT_MAX;
        }
        // 初始化右下角下方和右上角右边这两个格子，即从右下角出来都只需要1滴血即可存活
        dp[col-1] = 1;
        for (int i = row-1; i > -1; i--) {
            for (int j = col-1; j > -1; j--) {
                // 在dungeon[i][j]中加的血是否达到了从dungeon[i+1][j]和dungeon[i][j+1]两者中出发的最小血量
                int tmp = dungeon[i][j] - (int) (min(dp[j], dp[j+1]));
                if (tmp >= 0) { // 够, 只要1滴血能进dungeon[i][j]就行
                    dp[j] = 1;
                } else { // 不够，初始就要这么多血的差值才能进来
                    dp[j] = -tmp;
                }
            }
        }
        return dp[0];
    }
};
```

总结

- 假设大法好，这也是dp或者递归中的一个基础思想，我先假设我知道其中一种情况，有了已知条件我们如果可以继续往下推出更多的信息，那么就能够解决问题了。
- 通常dp的方法分为三部曲，先想到状态怎么表示，然后是状态怎么转变，最后考虑一下状态的初始化问题即可。这道题我们先想到p[i][j] 代表从dungeon[i][j]出发救出公主所需要的最少血量；然后也知道了dp[i][j]和dp[i][j+1],

dp[i+1][j]之间的关系；最后就是初始化了，我们知道右下角下方和右上角右边这两个格子，即从右下角出来都只需要1滴血即可存活，所以设置dp[-1][-2], dp[-2][-1] = 1, 1

}

34 寻找最近的回文数

36 学生出勤记录