逆向动态规划——疯狂地下城问题

原创 邹嘉钰 LOA算法学习笔记 2021-01-07 15:02

01 问题描述

有一个公主(Q，queen)被困在了地下城的底部（右下角），现在派出一名勇士(K, knight)去解救这个公主。其中，地下城（dungeon）的结构用矩阵形式给出，地下城的每个房间里面可能有恶魔，如果勇士进入这个房间，那么就必须与恶魔战斗消耗血量；地下城的房间里面也可能有奖励，如果勇士进入这个房间，可以补充血量；或者这是一个空房间，如果勇士进入这个房间，不会消耗血量，同时也不能增加血量。勇士的初始位置在地下城的入口处（左上角），每次只能往右走一个房间或往下走一个房间。勇士的初始血量是一个正数，在营救过程中，勇士需要保证血量始终大于等于 1，否则营救失败。

请你设计一个算法，求得勇士成功营救公主所需的最小初始血量。

输入：地下城矩阵 dungeon

输出：勇士最小初始血量

02 解决方案

2.1 问题结构分析

首先，我们很自然会想到正向的动态规划算法，因为**勇士每次只能往右走一个房间或往下走一个房间**。换言之，当前的房间是从上面的房间或者左边的房间走过来的。用 dp[i][j]表示走到（i, j）这个房间所需要的最小血量，那么动态规划表达式写成：dp[i][j] = min(dp[i-1][j], dp[i][j-1])这样的做法表面上能够保证达到最后一个房间时候的最小血量，但是却忽略了一个重要的问题：**无法保证勇士在行进过程中的血量始终大于等于 1**。这种做法有可能会使勇士在与恶魔战斗的过程中耗尽血量，无法走到地下城的底部。下面举例说明：

dungeon（地下城矩阵）

2（K）	-6	6
0	-4	0（**）
-8	-10	-6（Q）

假设现在勇士要走到第二行第三列的位置（用**标记），按照正向动态规划算法，勇士从第一行第一列出发的走法是：下->右->右，对应最小初始血量为 3，如果勇士的走法是：右->右->下，对应的最小初始血量是 5，比 3 更大。

现在考虑走到公主所在的位置（用 Q 标记），从图中容易看出来这一步一定是从(**)的位置往下走，因为沿着最左边->最下面的这条路径上的地下城房间全部都是负数，所需的最小初始血量必定更大。如果延续(**)的走法，路径为下->右->右->下，对应最小血量为 9。然而，另一种走法：右->右->下->下，对应最小血量为 5，比 9 小。

因此，考虑了（i, j）房间的值之后，用正向的动态规划算法未必能找到最优解，甚至出现找到的路径不符合要求的情况（勇士在中途的某一个房间血量小于等于 0）。

那么**正向的动态规划不好做，能不能反向思考呢？**从地下城的右下角往左上角考虑，每次向上或者向左走一个房间，为了保证初始血量最小，到达最后一个房间（公主所在的房间）的血量设为 1，这样回溯得到的 dp[0][0]就是所求的最小血量。这里，dp[i][j]表示进入（i, j）房间，并能够走出这个房间去探索其他房间所需的最小血量。根据地下城房间的三种不同情况，分类讨论如下：

1）dungeon[i][j]<0，这个**房间里面有恶魔**，勇士与之战斗的过程中要消耗（-dungeon[i][j]）的血量，因此dp[i][j] = min(dp[i+1][j],dp[i][j+1])-dungeon[i][j]。

2）dungeon[i][j]=0, 这个**房间里面什么也没有**，勇士进入没有血量消耗或血量提升，因此dp[i][j] = min(dp[i+1][j],dp[i][j+1])。

3）dungeon[i][j]>0，这个**房间里面有奖励**，勇士获得 dungeon[i][j]的血量提升，因此勇士进入到这个房间之前的血量可以在 min(dp[i+1][j],dp[i][j+1])的基础上减少 dungeon[i][j]。与此同时，要注意这样相减可能会导致结果

为负，这是不被允许的，遇到这种情况，令 dp[i][j]取能够离开这个房间的最小值 1 就行了，因此 dp[i][j] = max(min(dp[i+1][j],dp[i][j+1])-dungeon[i][j],1)。

2.2 状态转移方程

综合上述三种情况，状态转移方程可以统一写成如下形式：

$$dp[i][j] = max(min(dp[i+1][j],dp[i][j+1])-dungeon[i][j],1)$$

03 代码实现

```python
def calculateMinimumHP(self, dungeon: List[List[int]]):
    m = len(dungeon)
    n = len(dungeon[0])
    dp = [[float('inf')]*(n+1) for i in range(m+1)]
    # 设置 dp[m][n-1]= dp[m-1][n] = 1 是为了保证勇士离开地下城最后一个房间血量为 1
    dp[m][n-1] = 1
    dp[m-1][n] = 1
    for i in range(m-1,-1,-1):
        for j in range(n-1,-1,-1):
            dp[i][j] = max(min(dp[i+1][j],dp[i][j+1])-dungeon[i][j],1)
    return dp[0][0]
```

时间复杂度：外层 for 循环有 m 个，内层 for 循环有 n 个，每次确定 dp[i][j]的值的时候只比较常数次（2 次），算法复杂度为 O(mn)，其中 m 和 n 分别为地下城的行数和列数。

04 动态规划数组的变化和最终路径demo

为了演示逆向动态规划的过程，以如下这个地下城dungeon矩阵为例，做成了一个demo，如下所示。

初始状态

dungeon

2	-6	6
0	-4	0
-8	-10	-6

dp

inf	inf	inf	inf
inf	inf	inf	inf
inf	inf	inf	1
inf	inf	1	inf

图1 初始状态

第一轮

dungeon

2	-6	6
0	-4	0
-8	-10	-6

dp

inf	inf	inf	inf
inf	inf	inf	inf
inf	inf	7	1
inf	inf	1	inf

图2 第一轮

第二轮

dungeon

2	-6	6
0	-4	0
-8	-10	-6

dp

inf	inf	inf	inf
inf	inf	inf	inf
inf	17	7	1
inf	inf	1	inf

图3 第二轮

第三轮

dungeon

2	-6	6
0	-4	0
-8	-10	-6

dp

inf	inf	inf	inf
inf	inf	inf	inf
25	17	7	1
inf	inf	1	inf

图4 第三轮

第四轮

dungeon

2	-6	6
0	-4	0
-8	-10	-6

dp

inf	inf	inf	inf
inf	inf	7	inf
25	17	7	1
inf	inf	1	inf

图5 第四轮

第五轮

dungeon

2	-6	6
0	-4	0
-8	-10	-6

dp

inf	inf	inf	inf
inf	11	7	inf
25	17	7	1
inf	inf	1	inf

图6 第五轮

第六轮

dungeon

2	-6	6
0	-4	0
-8	-10	-6

dp

inf	inf	inf	inf
11	11	7	inf
25	17	7	1
inf	inf	1	inf

图7 第六轮

第七轮

dungeon

2	-6	6
0	-4	0
-8	-10	-6

dp

inf	inf	1	inf
11	11	7	inf
25	17	7	1
inf	inf	1	inf

图8 第七轮

第八轮

dungeon

2	-6	6
0	-4	0
-8	-10	-6

dp

inf	7	1	inf
11	11	7	inf
25	17	7	1
inf	inf	1	inf

图9 第八轮

第九轮

dungeon

2	-6	6
0	-4	0
-8	-10	-6

dp

5	7	1	inf
11	11	7	inf
25	17	7	1
inf	inf	1	inf

图10 第九轮

路径图

dungeon

2	-6	6
0	-4	0
-8	-10	-6

逆向动态规划数组元素变化

dp

5	7	1	inf
11	11	7	inf
25	17	7	1
inf	inf	1	inf

图11 路径图