**Dungeon game**

The demons had captured the princess (**P**) and imprisoned her in the bottom-right corner of a dungeon. The dungeon consists of M x N rooms laid out in a 2D grid. Our valiant knight (**K**) was initially positioned in the top-left room and must fight his way through the 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, so the knight loses health (*negative* integers) upon entering these rooms; other rooms are either empty (*0's*) or contain magic orbs that increase the knight's health (*positive* integers).

In order to reach the princess as quickly as possible, the knight decides to move only rightward or downward in each step.

**Write a function to determine the knight's minimum initial health so that he is able to rescue the princess.**

For example, given the dungeon below, the initial health of the knight must be at least **7** if he follows the optimal path RIGHT-> RIGHT -> DOWN -> DOWN.

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

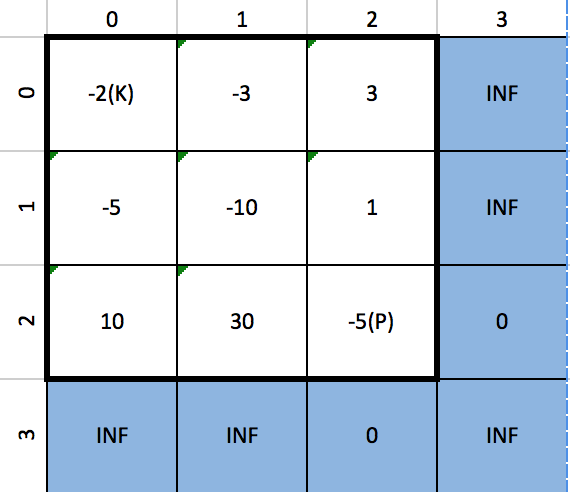
**Note:**

* The knight's health has no upper bound.
* 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.

**Solution**

**Dynamic Programming**  
First, we need to define the subproblem somewhat a little clever. If we define:  
*dp[i][j] = minimum cost from (0, 0) to (i, j)*  
It won't help solving the problem, because the result of dp[i + 1][j + 1] does not depends only on previous solve subproblems, but also future unsolved subproblems. So, how about let's define the subproblem from the other end of the puzzle?  
***dp[i][j] = minimum health level required to reach the princess when entering (i, j)***

So, what is dp[i + 1][j + 1] then? It depends on the minimum between dp[i][j + 1] and dp[i + 1][j], because we want to choose the cheapest way to go. Of course we also need to add or deduct the value from dungeon matrix. But be careful, if we find that the minimum required health level is less that 0, we need to set it to 0, because we are not allowed to overdraft health. With that said:  
***dp[i + 1][j + 1] = max(min(dp[i][j + 1], dp[i + 1][j]) - dungeon[i + 1][j + 1], 0);***

**Implementation**  
To get the code cleaner, I created the dp matrix 1 row and 1 column bigger that the original input. But we need to be careful when initializing the extra row and column, everything is initialized to Infinite except cell (m, n - 1) and (m - 1, n), which should be initialized to 0.  
I attached a picture to illustrate the idea (based on the test case given in the problem statement). Then code becomes very readable.

public class Solution {

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

int m = dungeon.length;

int n = m == 0 ? 0 : dungeon[0].length;

int[][] minRequred = new int[m + 1][n + 1];

for (int i = 0; i < m + 1; i++) {

minRequred[i][n] = Integer.MAX\_VALUE;

}

for (int j = 0; j < n + 1; j++) {

minRequred[m][j] = Integer.MAX\_VALUE;

}

minRequred[m][n - 1] = 0;

minRequred[m - 1][n] = 0;

for (int i = m - 1; i >= 0; i--) {

for (int j = n - 1; j >= 0; j--) {

minRequred[i][j] = Math.max(

Math.min(minRequred[i + 1][j], minRequred[i][j + 1]) - dungeon[i][j], 0);

}

}

return minRequred[0][0] + 1;

}

}