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)

Notes:

- 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.

Credits:
Special thanks to @stellari for adding this problem and creating all test cases.

Solution 1

Use hp[i][j] to store the min hp needed at position (i, j), then do the calculation from right-bottom to left-up.

Note: adding dummy row and column would make the code cleaner.

```
class Solution {
public:
    int calculateMinimumHP(vector<vector<int> > &dungeon) {
        int M = dungeon.size();
        int N = dungeon[0].size();
        // hp[i][j] represents the min hp needed at position (i, j)
        // Add dummy row and column at bottom and right side
        vector<vector<int> > hp(M + 1, vector<int>(N + 1, INT_MAX));
        hp[M][N - 1] = 1;
        hp[M - 1][N] = 1;
        for (int i = M - 1; i >= 0; i--) {
            for (int j = N - 1; j >= 0; j--) {
                int need = min(hp[i + 1][j], hp[i][j + 1]) - dungeon[i][j];
                hp[i][j] = need <= 0 ? 1 : need;
            }
        }
        return hp[0][0];
    }
};
```

written by sidbai original link here

Solution 2

```
public int calculateMinimumHP(int[][] dungeon) {
    if (dungeon == null || dungeon.length == 0 || dungeon[0].length == 0) return
0;

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

    int[][] health = new int[m][n];

    health[m - 1][n - 1] = Math.max(1 - dungeon[m - 1][n - 1], 1);

    for (int i = m - 2; i >= 0; i--) {
        health[i][n - 1] = Math.max(health[i + 1][n - 1] - dungeon[i][n - 1], 1)
;
    }

    for (int j = n - 2; j >= 0; j--) {
        health[m - 1][j] = Math.max(health[m - 1][j + 1] - dungeon[m - 1][j], 1);
    }

    for (int i = m - 2; i >= 0; i--) {
        for (int j = n - 2; j >= 0; j--) {
            int down = Math.max(health[i + 1][j] - dungeon[i][j], 1);
            int right = Math.max(health[i][j + 1] - dungeon[i][j], 1);
            health[i][j] = Math.min(right, down);
        }
    }

    return health[0][0];
}
```

written by vime2 original link here

Solution 3

Here is my solution using dp and rolling array --Dungeon Game:

```
int calculateMinimumHP(vector<vector<int> > &dungeon) {
    const int m = dungeon.size();
    const int n = dungeon[0].size();
    vector<int> dp(n + 1, INT_MAX);
    dp[n - 1] = 1;
    for(int i = m - 1; i >= 0; --i)
        for(int j = n - 1; j >= 0; --j)
            dp[j] = getMin(min(dp[j], dp[j + 1]) - dungeon[i][j]);
    return dp[0];
}
int getMin(int n){
    return n <= 0 ? 1 : n;
}
```

Note: Update from right to left and from bottom up.

written by flyingPig original link here

From Leetcoder.