Documentation for "Dungeon Game" Problem Solution

Problem Statement

The knight is on a mission to rescue a princess imprisoned in the bottom-right corner of a dungeon grid. *The dungeon is represented as a 2D grid dungeon, where each cell may contain:*

- Negative values, representing demons that reduce the knight's health.
- Zero, representing an empty room.
- Positive values, representing magic orbs that increase the knight's health.

The knight starts at the top-left corner of the dungeon and can only move right or down at each step. The objective is to determine the minimum initial health the knight needs to start with so that he can reach the princess alive, regardless of the path chosen. If at any point the knight's health drops to 0 or below, he dies immediately.

Problem Constraints

- m (number of rows in the dungeon) and n (number of columns in the dungeon) satisfy (1 leq m, n leq 200).
- The dungeon grid cells contain integer values ranging from (-1000) to (1000).

Solution Overview

To solve this problem, we employ a Dynamic Programming (DP) approach. The key idea is to work backwards from the destination (bottom-right corner where the princess is) to the start (top-left corner where the knight begins). This reverse approach helps in determining the minimum health required at each room, considering the knight's possible future steps.

Key Points to Consider:

1. **Health Requirement:**

- The knight must always maintain a health value greater than 0. Therefore, for any room, the knight’s health upon entry must be sufficient to survive any negative values (demons) and end with at least 1 health point.

2. **Reverse Traversal Strategy:**

- Start calculating from the bottom-right room moving towards the top-left. This allows us to make optimal decisions at each step since the decisions are influenced by the rooms ahead.

3. **DP Table Definition:**

- Create a DP table dp where dp[i][j] represents the minimum initial health required for the knight to reach the princess starting from room (i, j).

4. **DP Transition Formula:**

- *For each cell (i, j), the minimum health required to enter is calculated based on the minimum health required in the next possible steps (right (i, j+1) and down (i+1, j)):*
  - [text{dp}[i][j] = max(min(text{dp}[i+1][j], text{dp}[i][j+1]) text{dungeon}[i][j], 1)]
- *This formula ensures:*
  - The knight has enough health to move to the next room (either right or down).
  - Health must always be at least 1 upon entering any room.

5. **Edge Cases:**

- The princess’s room (m-1, n-1) should be handled specifically because it doesn’t lead to any other room.
- Rooms at the boundaries (last row or last column) have limited possible movements (only right or down, respectively).

Detailed Steps

1. **Initialization:**

- Initialize a DP table dp with dimensions (m+1) x (n+1) to accommodate the boundary conditions.
- Set all values in dp to infinity (inf) initially to represent an unreachable state except for boundary conditions.
- Define the boundary conditions such that moving "beyond" the princess’s room requires at least 1 health (i.e., dp[m][n-1] = dp[m-1][n] = 1).

2. **Fill the DP Table:**

- Traverse the dungeon grid in reverse (starting from (m-1, n-1) to (0, 0)).
- For each cell (i, j), compute the required minimum health based on future steps (dp[i+1][j] for down and dp[i][j+1] for right).
- Apply the DP transition formula to compute dp[i][j].

3. **Final Result:**

- The result is the value at dp[0][0], which represents the minimum initial health required for the knight to survive starting from the top-left corner to the bottom-right corner of the dungeon.

Complexity Analysis

- **Time Complexity: (O(m times n))**
  - ➢ This results from filling up an m x n DP table where each cell computation is (O(1)).

- **Space Complexity: (O(m times n))**
  - ➢ The space is used for the DP table, which stores the minimum health requirements for each cell.

Conclusion

The solution utilizes dynamic programming to calculate the minimum initial health required for the knight to rescue the princess safely. By reversing the traversal order and carefully handling the health calculation at each cell, the approach ensures that the optimal path is determined efficiently. This method guarantees that the knight survives through all possible scenarios, ensuring a safe rescue of the princess.