LeetCode Training Day 13 DP III 2D Grid

Today we will discuss Dynamic Programming with the input as 2D grid. Basically there is no much difference between 2D grid and 1D array, we just need to calculate the subarray result based on row and column.

## 361. Bomb Enemy

Medium

Given an m x n matrix grid where each cell is either a wall 'W', an enemy 'E' or empty '0', return *the maximum enemies you can kill using one bomb*. You can only place the bomb in an empty cell.

The bomb kills all the enemies in the same row and column from the planted point until it hits the wall since it is too strong to be destroyed.

**Example 1:**

A picture containing text, building, window, clipart

Description automatically generated

**Input:** grid = [["0","E","0","0"],["E","0","W","E"],["0","E","0","0"]]

**Output:** 3

**Example 2:**

A picture containing graphical user interface

Description automatically generated

**Input:** grid = [["W","W","W"],["0","0","0"],["E","E","E"]]

**Output:** 1

**Constraints:**

* m == grid.length
* n == grid[i].length
* 1 <= m, n <= 500
* grid[i][j] is either 'W', 'E', or '0'.

### Analysis:

We scan the grid from left to right, top to bottom, right to left and bottom to top. On every position, we know how many enemies on left, right, up and down. We choose the maximum number in the matrix.

/// <summary>

/// Leet code #361. Bomb Enemy

///

/// Given a 2D grid, each cell is either a wall 'W', an enemy 'E' or empty '0'

/// (the number zero), return the maximum enemies you can kill using one bomb.

/// The bomb kills all the enemies in the same row and column from the planted

/// point until it hits the wall since the wall is too strong to be destroyed.

/// Note that you can only put the bomb at an empty cell.

///

/// Example:

///

/// For the given grid

/// 0 E 0 0

/// E 0 W E

/// 0 E 0 0

///

/// return 3. (Placing a bomb at (1,1) kills 3 enemies)

/// </summary>

int LeetCodeDP::maxKilledEnemies(vector<vector<char>>& grid)

{

if (grid.size() == 0 || grid[0].size() == 0) return 0;

int max\_enemies = 0;

pair<int, int> max\_pos;

vector<vector<pair<int, int>>> sum(grid.size(), vector<pair<int, int>>(grid[0].size()));

for (size\_t i = 0; i < grid.size(); i++)

{

for (size\_t j = 0; j < grid[i].size(); j++)

{

pair<int, int> count;

if (grid[i][j] == 'W')

{

count = make\_pair(0, 0);

}

else if (grid[i][j] == '0')

{

if (i == 0) count.first = 0;

else count.first = sum[i - 1][j].first;

if (j == 0) count.second = 0;

else count.second = sum[i][j - 1].second;

}

else if (grid[i][j] == 'E')

{

if (i == 0) count.first = 1;

else count.first = sum[i - 1][j].first + 1;

if (j == 0) count.second = 1;

else count.second = sum[i][j - 1].second + 1;

}

sum[i][j] = count;

}

}

for (int i = grid.size() - 1; i >= 0; i--)

{

for (int j = grid[i].size() - 1; j >= 0; j--)

{

if (grid[i][j] == 'W') continue;

if (i < (int)grid.size() - 1)

{

sum[i][j].first = max(sum[i + 1][j].first, sum[i][j].first);

}

if (j < (int)grid[i].size() - 1)

{

sum[i][j].second = max(sum[i][j + 1].second, sum[i][j].second);

}

if (grid[i][j] == '0')

{

int enemies = sum[i][j].first + sum[i][j].second;

if (enemies > max\_enemies)

{

max\_enemies = enemies;

max\_pos = make\_pair(i, j);

}

}

}

}

return max\_enemies;

}

## 221. Maximal Square

Medium

Given an m x n binary matrix filled with 0's and 1's, *find the largest square containing only* 1's *and return its area*.

**Example 1:**

Calendar

Description automatically generated with medium confidence

**Input:** matrix = [["1","0","1","0","0"],["1","0","1","1","1"],["1","1","1","1","1"],["1","0","0","1","0"]]

**Output:** 4

**Example 2:**

A picture containing text, clock

Description automatically generated

**Input:** matrix = [["0","1"],["1","0"]]

**Output:** 1

**Example 3:**

**Input:** matrix = [["0"]]

**Output:** 0

**Constraints:**

* m == matrix.length
* n == matrix[i].length
* 1 <= m, n <= 300
* matrix[i][j] is '0' or '1'.

### Analysis:

To calculate maximum square, we need to get from minimum accumulated square from upper point, left point or left upper point and add 1, if current cell is 1. If current cell is 0 we reset it.

/// <summary>

/// Leet code #221. Maximal Square

///

/// Given a 2D binary matrix filled with 0's and 1's, find the largest square

/// containing only 1's and return its area.

/// For example, given the following matrix:

/// 1 0 1 0 0

/// 1 0 1 1 1

/// 1 1 1 1 1

/// 1 0 0 1 0

/// Return 4.

/// </summary>

int LeetCodeDP::maximalSquare(vector<vector<char>>& matrix)

{

int result = 0;

if (matrix.size() == 0 || matrix[0].size() == 0) return result;

vector<vector<int>> dp(matrix.size(), vector<int>(matrix[0].size()));

for (size\_t i = 0; i < matrix.size(); i++)

{

for (size\_t j = 0; j < matrix[0].size(); j++)

{

if (matrix[i][j] == '0') dp[i][j] = 0;

else

{

if ((i == 0) || (j == 0))

{

dp[i][j] = 1;

}

else

{

dp[i][j] = min(dp[i - 1][j - 1], dp[i - 1][j]);

dp[i][j] = min(dp[i][j], dp[i][j - 1]);

dp[i][j]++;

}

result = max(result, dp[i][j] \* dp[i][j]);

}

}

}

return result;

}

## 1277. Count Square Submatrices with All Ones

Medium

Given a m \* n matrix of ones and zeros, return how many **square** submatrices have all ones.

**Example 1:**

**Input:** matrix =

[

  [0,1,1,1],

  [1,1,1,1],

  [0,1,1,1]

]

**Output:** 15

**Explanation:**

There are **10** squares of side 1.

There are **4** squares of side 2.

There is **1** square of side 3.

Total number of squares = 10 + 4 + 1 = **15**.

**Example 2:**

**Input:** matrix =

[

[1,0,1],

[1,1,0],

[1,1,0]

]

**Output:** 7

**Explanation:**

There are **6** squares of side 1.

There is **1** square of side 2.

Total number of squares = 6 + 1 = **7**.

**Constraints:**

* 1 <= arr.length <= 300
* 1 <= arr[0].length <= 300
* 0 <= arr[i][j] <= 1

### Analysis:

Find the maximum square K at every point, add size of 1-K to the count, (add K).

/// <summary>

/// Leetcode #1277. Count Square Submatrices with All Ones

///

/// Given a m \* n matrix of ones and zeros, return how many square

/// submatrices have all ones.

///

/// Example 1:

///

/// Input: matrix =

/// [

/// [0,1,1,1],

/// [1,1,1,1],

/// [0,1,1,1]

/// ]

/// Output: 15

/// Explanation:

/// There are 10 squares of side 1.

/// There are 4 squares of side 2.

/// There is 1 square of side 3.

/// Total number of squares = 10 + 4 + 1 = 15.

///

/// Example 2:

///

/// Input: matrix =

/// [

/// [1,0,1],

/// [1,1,0],

/// [1,1,0]

/// ]

/// Output: 7

/// Explanation:

/// There are 6 squares of side 1.

/// There is 1 square of side 2.

/// Total number of squares = 6 + 1 = 7.

///

/// Constraints:

/// 1. 1 <= arr.length <= 300

/// 2. 1 <= arr[0].length <= 300

/// 3. 0 <= arr[i][j] <= 1

/// </summary>

int LeetCodeDP::countSquares(vector<vector<int>>& matrix)

{

vector<vector<int>> count(matrix.size(), vector<int>(matrix[0].size()));

int result = 0;

for (size\_t i = 0; i < matrix.size(); i++)

{

for (size\_t j = 0; j < matrix[0].size(); j++)

{

if (matrix[i][j] == 0)

{

count[i][j] = 0;

}

else

{

if (i == 0 || j == 0) count[i][j] = 1;

else

{

int prev\_count = count[i - 1][j];

prev\_count = min(prev\_count, count[i][j - 1]);

prev\_count = min(prev\_count, count[i - 1][j - 1]);

count[i][j] = 1 + prev\_count;

}

result += count[i][j];

}

}

}

return result;

}

# Advanced Problems

## 174. Dungeon Game

Hard

The demons had captured the princess 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 was initially positioned in **the top-left room** and must fight his way through 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 (represented by negative integers), so the knight loses health upon entering these rooms; other rooms are either empty (represented as 0) or contain magic orbs that increase the knight's health (represented by positive integers).

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

Return *the knight's minimum initial health so that he can rescue the princess*.

**Note** that 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.

**Example 1:**

A picture containing text, crossword puzzle, clipart

Description automatically generated

**Input:** dungeon = [[-2,-3,3],[-5,-10,1],[10,30,-5]]

**Output:** 7

**Explanation:** The initial health of the knight must be at least 7 if he follows the optimal path: RIGHT-> RIGHT -> DOWN -> DOWN.

**Example 2:**

**Input:** dungeon = [[0]]

**Output:** 1

**Constraints:**

* m == dungeon.length
* n == dungeon[i].length
* 1 <= m, n <= 200
* -1000 <= dungeon[i][j] <= 1000

### Analysis:

To reach the right bottom cell, you need at least 1 blood, which can also be known on the 1 cell up and 1 cell left how much blood you need, reversely walk through the 2-D grid, calculate the minimum blood you need in every cell, you will know at initial position, how much blood you need. Just remember when entering in every room, you need at least 1 blood, so if the room has +4 blood, you will need 1 bloods before in this room, if the room has -4 blood, you will need 5 bloods before in this room.

/// <summary>

/// Leet code #174. 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:

/// 1. The knight's health has no upper bound.

/// 2. 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.

/// </summary>

int LeetCodeDP::calculateMinimumHP(vector<vector<int>>& dungeon)

{

if ((dungeon.size() == 0) || dungeon[0].size() == 0) return 1;

vector<vector<int>> min\_hp(dungeon.size());

for (int i = dungeon.size() - 1; i >= 0; i--)

{

min\_hp[i] = vector<int>(dungeon[i].size());

for (int j = dungeon[i].size() - 1; j >= 0; j--)

{

if ((i == dungeon.size() - 1) && (j == dungeon[i].size() - 1))

{

min\_hp[i][j] = max(1, 0 - dungeon[i][j] + 1);

}

else if (i == dungeon.size() - 1)

{

min\_hp[i][j] = max(1, min\_hp[i][j + 1] - dungeon[i][j]);

}

else if (j == dungeon[i].size() - 1)

{

min\_hp[i][j] = max(1, min\_hp[i + 1][j] - dungeon[i][j]);

}

else

{

min\_hp[i][j] = min(min\_hp[i + 1][j], min\_hp[i][j + 1]) - dungeon[i][j];

min\_hp[i][j] = max(1, min\_hp[i][j]);

}

}

}

return min\_hp[0][0];

}