LeetCode Training Day 12 DP III

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;

}

# Advanced Problems