Problem name: Andres Iniesta Language: English

Solution

Many problems that require finding optimal solution are solvable by using dynamic programming approach. Here, we are going to present solution in time complexity $O(N \cdot M \cdot K^2)$.

Let's define matrix $left[N][M][K]$, where $left[r][c][l]$ represents maximal number of visible cells left from cell (r, c) if Iniesta can destroy up to l obstacles that are located left of him (same row with lower index of column). It not hard to see that following recursive formula is correct (if cell (r, c) contains obstacle, then we can destroy 1 less obstacle left of cell that cell):

$$left[r][c][l] = \begin{cases} left[r][c-1][l-1] + 1, & if\ there\ is\ obstacle\ at\ cell\ (r,c) \\ left[r][c-1][l] + 1, & otherwise \end{cases}$$

Where initial values are 0 or 1 based on that if cell contain obstacle or not.

In the similar way we can define matrixes $right[N][M][K], up[N][M][K], down[N][M][K]$. Formulas for constructing those matrix are very similar.

Now we have matrixes $left, right, up$ and $down$, but we still have to combine then to get a solution. We are going to make matrix $column[N][M][K]$, where $column[r][c][l]$ represents maximal number of visible cells in the same column where cell (r, c) is located in the matrix if we can destroy up to l obstacles in column c. We can find $column[r][c][l]$ by trying all combinations destroying p obstacles up from cell (r, c) and $l - p$ down of that cell. If cell (r, c) in the matrix contains an obstacle, then we can destroy 1 less obstacle when we observe all combinations.

$$column[r][c][l] = \begin{cases} \max\limits_{p=0..l-1} (left[r][c-1][p] + right[r][c+1][l-p] + 1), (r,c)\ has\ obstacle \\ \max\limits_{p=0..l} (left[r][c-1][p] + right[r][c+1][l-p] + 1), \quad otherwise \end{cases}$$

In similar way we can define matrix $row[N][M][K]$, where $row[r][c][l]$ represents maximal number of visible cells that are located in same row as cell (r, c) if Iniesta can destroy up to l obstacles in that same row.

Finally we are going to combine matrixes $column$ and row to get a result. For every cell in matrix we are going to calculate maximal field of view if we stand on that cell and can destroy K cells.

For each cell we should try all combinations of destroying obstacles, where combinations differ from each other by number of destroyed obstacles in row and in column. So if cell (r, c) doesn't contain an obstacle we can conclude that

$$solution[r][c] = \max_{l=0..K} column[r][c][l] + row[r][c][K-l] - 1$$

We have -1 in last formula because we are counting cell (r, c) 2 times, first time in sum $column[r][c][l]$ and second time in sum $row[r][c][K - l]$.

If cell (r, c) contains an obstacle, that means we have to destroy that obstacle. Formula becomes slightly different than the last one.

After calculating solution for each cell, we have to pick the best one. Calculated matrixes $left, right, up, down$ and $solution$ is done in time complexity $O(N \cdot M \cdot K)$, but matrixes $column$ and row are computed in time complexity $O(N \cdot M \cdot K^2)$ so time complexity of whole solution is $O(N \cdot M \cdot K^2)$.