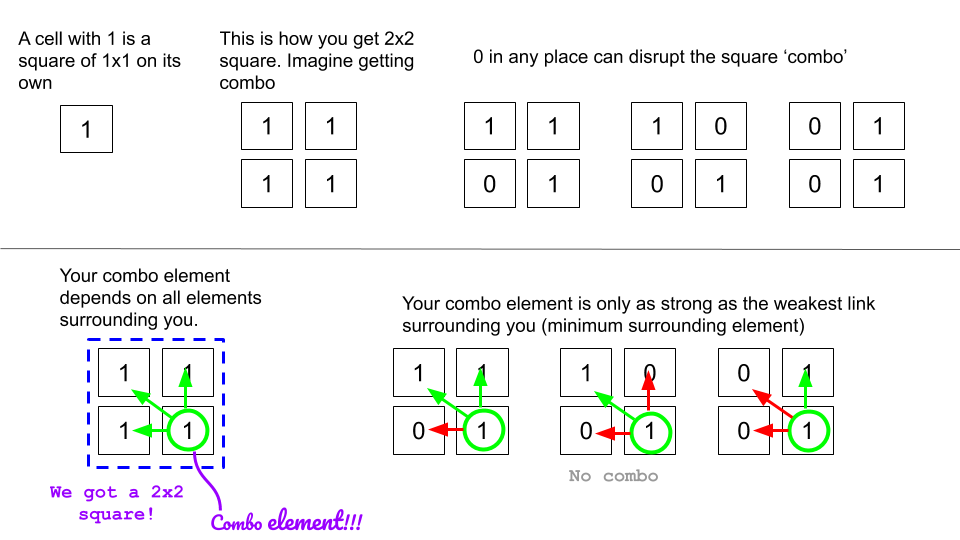
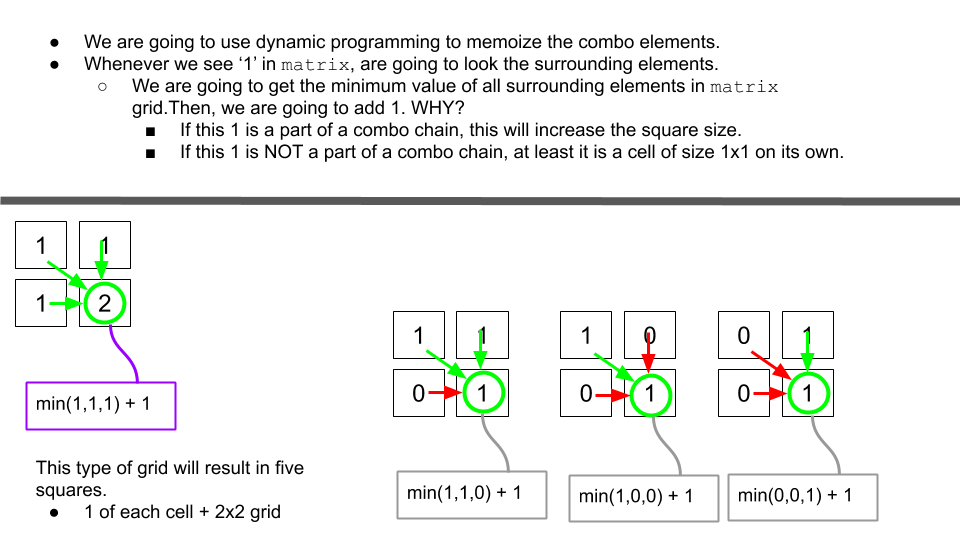
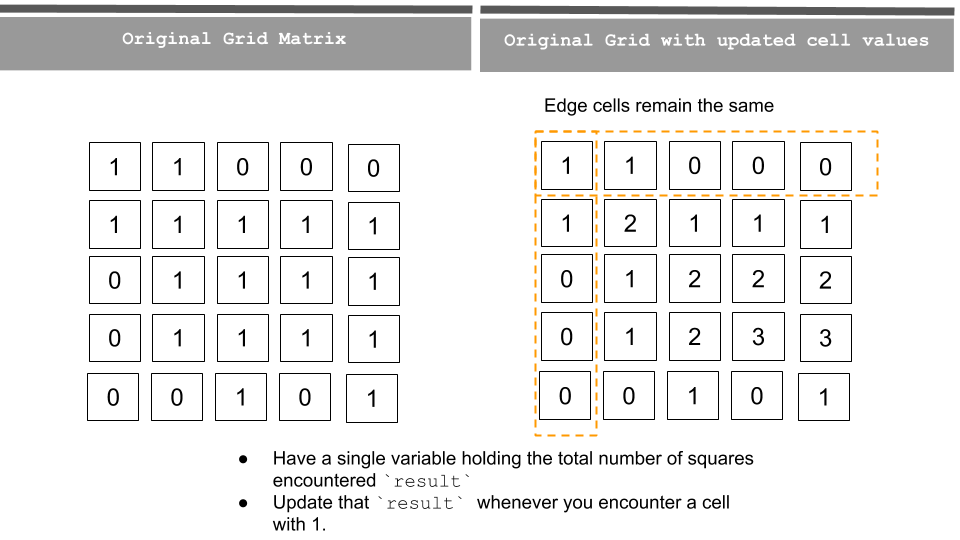
<https://leetcode.com/problems/count-square-submatrices-with-all-ones/discuss/643429/Python-DP-Solution-%2B-Thinking-Process-Diagrams-(O(mn)-runtime-O(1)-space)>

The idea is to scan each cell in the matrix to update the placeholder result variable with the number of squares that can be formed from the currently looking cell (when it is the bottom right corner cell of the **any possible** square).

The main workhorse of this Dynamic Programming approach is Line 17. Let's try to understand that:



Since we don't create any additional grid to hold our results, and if we don't account for the original input matrix, the space complexity is O(1) and runtime complexity is O(mn) where m and n are rows and cols for the matrix.

**Python**

1. class Solution:

2. def countSquares(self, matrix: List[List[int]]) -> int:

3. if matrix is None or len(matrix) == 0:

4. return 0

5.

6. rows = len(matrix)

7. cols = len(matrix[0])

8.

9. result = 0

10.

11. for r in range(rows):

12. for c in range(cols):

13. if matrix[r][c] == 1:

14. if r == 0 or c == 0: # Cases with first row or first col

15. result += 1 # The 1 cells are square on its own

16. else: # Other cells

17. cell\_val = min(matrix[r-1][c-1], matrix[r][c-1], matrix[r-1][c]) + matrix[r][c]

18. result += cell\_val

19. matrix[r][c] = cell\_val #\*\*memoize the updated result\*\*

20. return result