CS 301 Assignment 4

Beyzagul Demir, 28313

December 11, 2022

1 (a) Recursive formulation

We wan to return maximum number of weeds for each function. Since our robot can only go right or down, there are two possibilities to go. Moving to right or down: for each case we call our function itself. That means we call 2 new functions for each possible case. We should have a control condition to check if we reach to the end. Let's say we have a nxm matrix called A. A[1][1] gives the left upper cell and A[n][m] gives the right bottom cell.

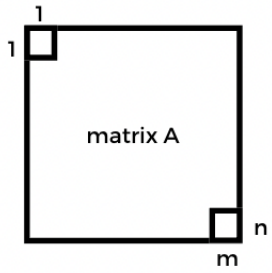
Our algorithm:
maxWeed(x,y) = max(maxWeed(x+1, y), maxWeed(x, y+1)) + hasWeed(x,y)

x and y keep the indexes of the matrix. Initially it starts from A[1][1] and we will reach to A[n][m]. For each call, the maximum one will be returned. Also, we check if there is a Weed at the current cell with hasWeed function. If there is a weed, add 1. Otherwise, add 0.

Pseudocode of the Recursive Algorithm:

```
maxWeed(A, x, y, n, m):
  #x and y keep indexes, n is number of rows, m is number of cols
  if(x = n and y = m):
    if (A[x][y] == 1):
       return 1
    else:
       return 0
  if(A[x][y] == 1):
    return 1 + max(maxWeed(A,x,y-1,n,m), maxWeed(A,x-1,y,n,m))
  else:
    return max(maxWeed(A,x,y-1,n,m), maxWeed(A,x-1,y,n,m))
```

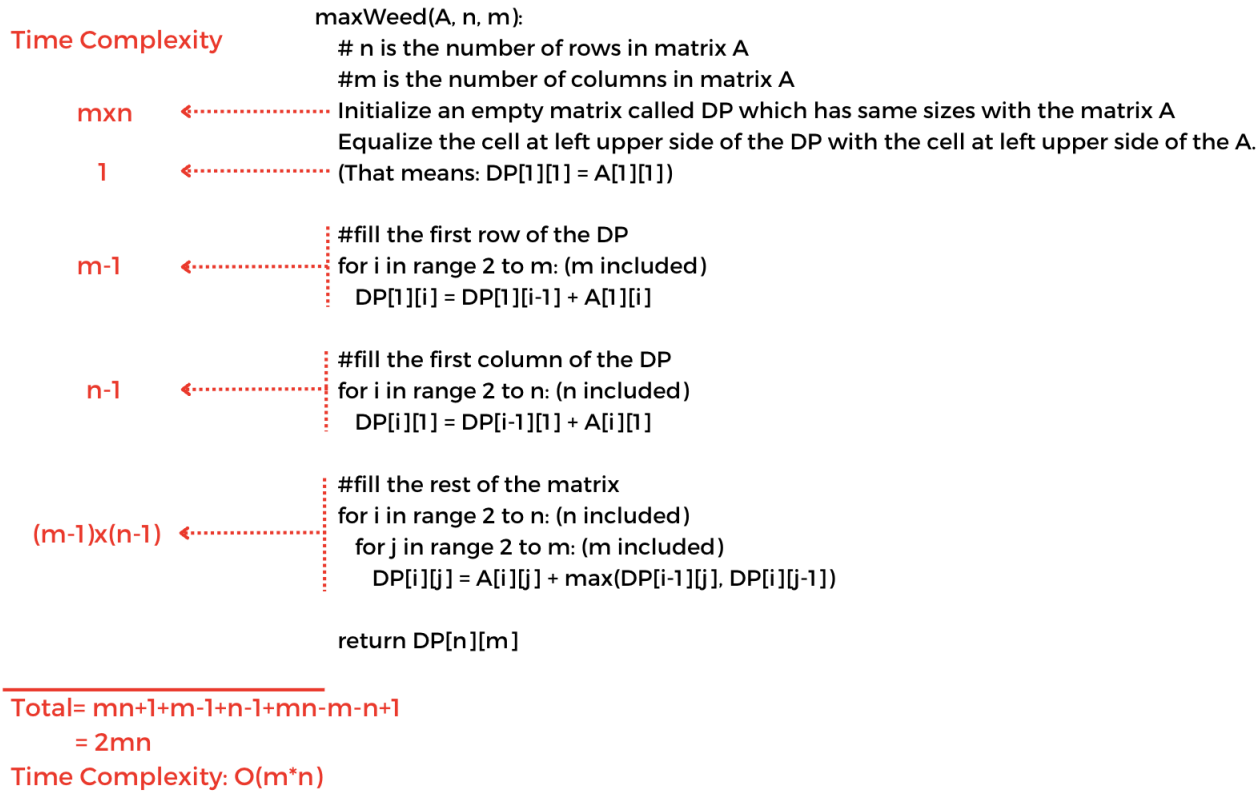
2 (b) Pseudocode of your algorithm




```
maxWeed(A, n, m):
  # n is the number of rows in matrix A
  #m is the number of columns in matrix A
  Initialize an empty matrix called DP which has same sizes with the matrix A
  Equalize the cell at left upper side of the DP with the cell at left upper side of the A.
  (That means: DP[1][1] = A[1][1])

  #fill the first row of the DP
  for i in range 2 to m): (m included)
    DP[1][i] = DP[1][i-1] + A[1][i]

  #fill the first column of the DP
  for i in range 2 to n): (n included)
    DP[i][1] = DP[i-1][1] + A[i][1]

  #fill the rest of the matrix
  for i in range 2 to n): (n included)
    for j in range 2 to m): (m included)
      DP[i][j] = A[i][j] + max(DP[i-1][j], DP[i][j-1])

  return DP[n][m]
```

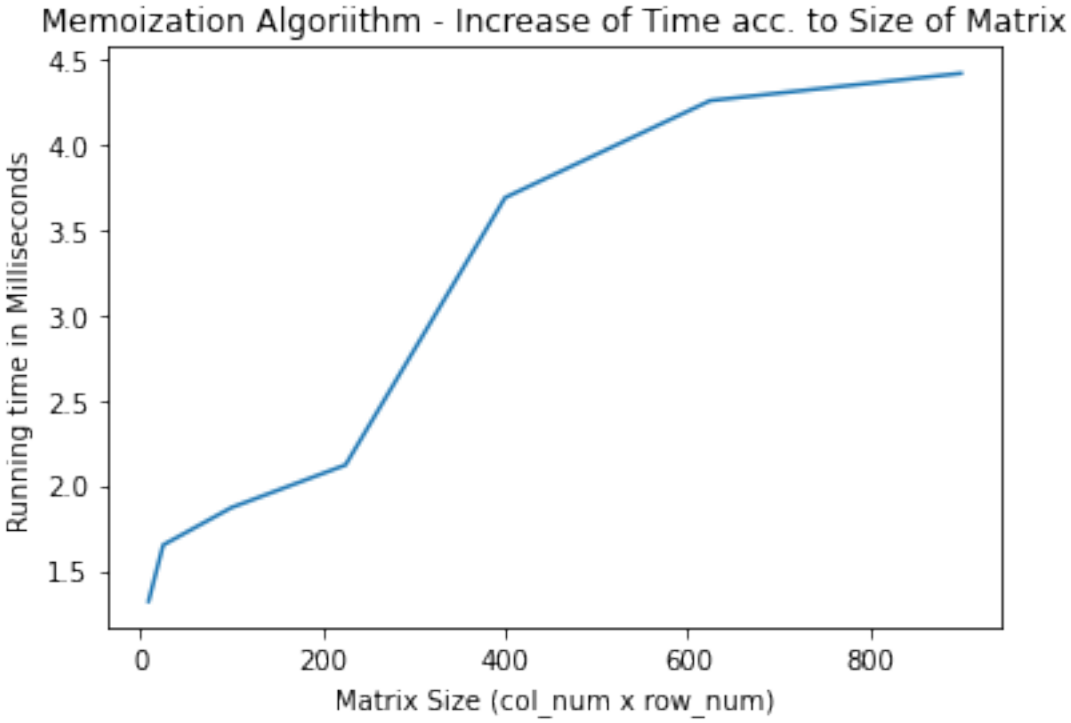
In that pseudocode, matrix A refers to the original table. To indicate whether there is a weed or not, we use 0 and 1. If there is a weed in that cell, the value for this cell is 1. Otherwise, it is 0. Here I used the memoization algorithm. In that type of dynamic programming algorithm, we use the top-down method. We use the extra matrix DP to keep notes and we return the right bottom cell of DP as a result.

3 (c) Asymptotic time and space complexity analysis



Time Complexity: As can be seen above, the total time complexity is $\theta(m*n)$.
Space Complexity: We use an additional nxm matrix. That costs n$*$m space for us. As a result, totally we have $\theta(n*m)$ space complexity.

4 (d) Experimental evaluations of your algorithm



For the x-axis, I used matrix size which is calculated as the number of columns times the number of rows. For the y-axis, I used time as milliseconds. I used the output runtimes of 3x3, 5x5, 10x10, 15x15, 20x20, 25x25, 30x30 matrices. So, sample matrix sizes are 9, 25, 100, 225, 400, 625, and 900. As you can see in the graph above, runtime increases when the matrix size becomes bigger. The result supports the time complexity that we found in part c. So it is not surprising, we were expecting that increase in runtime when the matrix size is increased. Sample data used for the graph and other test cases can be seen from the code file.