CS301 Assignment 4 Answers

Uğur Öztunç 28176

Recursive Formulation

Let's say $MaxWeed(i, j)$ is a function that returns the maximum number of weed can be cleared through the path from (0,0) location of the farm to the (i,j) by just moving left or down, and $hasWeed(i, j)$ is a function that returns 1 if there is weed in (i,j) in farm or 0 otherwise. Than the recursive formulation of $MaxWeed(i, j)$ function will be following:

$$MaxWeed(i, j) = max(MaxWeed(i-1, j), MaxWeed(i, j-1)) + hasWeed(i, j)$$

Pseudocode of Naive Algorithm

Inputs:

- *farmMatrix* : matrix of the given farm
- *m* and *n* : target locations

```
function MaxWeed(x, y, farmMatrix):
    if m is 0 and n is 0 do:
            return hasWeed(m, n)
    else if x is 0 do:
            return maxWeed(m, n-1, farmMatrix) + hasWeed(m, n)
    else if y is 0 do:
            return maxWeed(m-1, n, farmMatrix) + hasWeed(m, n)
    else do:
            return max(maxWeed(m-1, n, farmMatrix), maxWeed(m, n-1, farmMatrix)) +
        hasWeed(m, n)
```

Pseudocode of Dynamic Programming Algorithm

Inputs:

- *farmMatrix* : matrix of the given farm
- *x* and *y* : target locations

```
function MaxWeed(m, n, farmMatrix):
    Create a matrix  named table with x rows and y columns

    for i = 1 to m do:
        for j = 1 to n do:
            table[i][j] = hasWeed(i,j)

    for i = 2 to n do:
        table[1][i] = table[1][i] + table[1][i-1]

    for i = 2 to m do:
        table[i][1] = table[i][1] + table[i-1][1]

    for i = 2 to m do:
        for j = 2 to n do:
            table[i][j] = table[i][j] + max(table[i-1][j], table[i][j-1])

    return table[m][n]
```

Asymptotic Complexity Analysis of Naive Algorithm

Naive algorithm is recursive and in every step there are at most 2 recursive calls, and this goes until we reach m to 0 and n to 0. This means total time complexity of algorithm is $O(2^{m*n})$.

Asymptotic Complexity Analysis of Dynamic Programming Algorithm

Creating table = $m * n$
Filling first row = $n - 1$
Filling first column = $m - 1$
Filling rest of the table = $(m-1)*(n-1)$
Total = $m * n$ + n - 1 + m - 1 + $m * n$ - n - m + 1 = 2mn - 1 = $O(m * n)$
Time Complexity = $O(m * n)$

Space Complexity: An additional matrix with m rows and n columns is created, this means space complexity is $O(m * n)$

Experimental Evaluations of Dynamic Programming Algorithm

The algorithm was tested with different sizes and the average running times of the algorithm for each N size was plotted. In this test, at each N size, a random NxN farm was generated and the size N was increased with step size 10 every time until it reaches size 500, which means a matrix with 500 rows and columns. Also, for each N value the algorithm was tested with randomly generated NxN matrices that are different from each other 10 times and the average running time of 10 operations was saved for that N value in order to obtain more accurate average time.

Since $m = n = N$ in this test case, asymptotic time complexity of the algorithm is $O(m * n)$=$O(N^2)$, and it can be said that the graph shows that as the N value increases, running time of the algorithm increases quadratically, as we expected.