CS301 Assignment 4

Ahmet Bilal Yıldız

December 2022

1 Agricultural Robotics Problem

(a)

Recursive Formulation of the Agricultural Robotics Problem:

$$C[i,j] = \begin{cases} max(c[1,j-1], c[i-1,j]) + 1 & \text{if c[i,j] has weed, i} \neq \text{1, j} \neq \text{1} \\ max(c[1,j-1], c[i-1,j]) & \text{if c[i,j] not have weed, i} \neq \text{1, j} \neq \text{1} \\ 1 & \text{if c[i,j] has weed, i = 1, j = 1} \\ 0 & \text{if c[i,j] not have weed, i = 1, j = 1} \end{cases} \quad (1)$$

(b)

To find a dynamic programming algorithm first we need to consider if there is optimal substructure property and overlapping subproblems.

Optimal Substructure: There is optimal substructure property because an optimal solution to the main problem can be based on the optimal solutions to the sub-problems. In this example, a path that contains maximum number of weeds can be based on the sub-paths that contains maximum number of weeds. This can also verified by the cut and paste method as well.

Overlapping Sub-problems: There are overlapping sub-problems because the inputs are overlaps with each other.

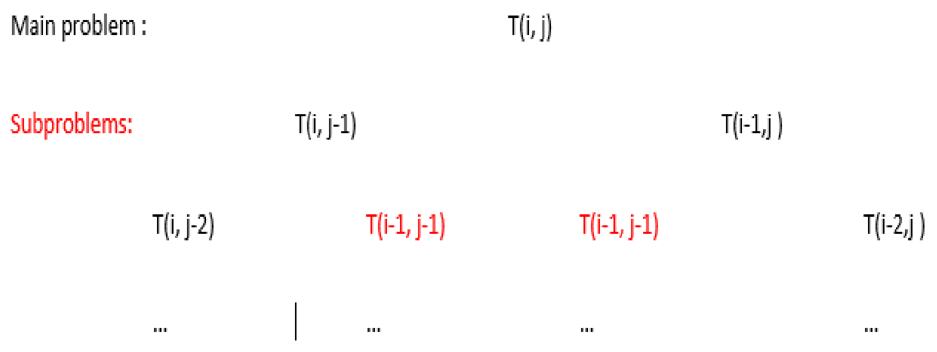

Figure 1: Illustration of Overlapping Sub-problems

As it can be seen from the above illustration, all the lines after the red parts are overlaps with each other, and therefore; a dynamic proramming algorithm can be created.

Pseudo Code for the Algorithm:

Create ixj matrix called farm which consists of 1's for the cells that contains weeds and 0's for the cells that does not contain weeds according to given information. (Actually, this matrix is not created but this is the input matrix)

Create an ixj matrix called maxFarm consisting of all -1's later that is used by the max weed function and all the cells will filled by the values that equals to the number of maximum weed that a path can contain.

Then MW (max weed) is called with the i and j values of given matrix.
MW():

if the cell not equals to -1, then return maxFarm(i-1, j-1), else go with next steps

if i = 1 and j = 1, assign maxFarm(i-1, j-1) to farm[i-1][j-1] (0 if there is no weed, 1 if there is weed)

else if i = 1 assign maxFarm(i-1, j-1) to MW(i,j-1) (if cell is on the upper corner just take the max of left cell) and return maxFarm(i-1,j-1)

else if j = 1, assign maxFarm(i-1, j-1) to MW(i-1,j) (if cell is on the left corner just take the max of upper cell) and return maxFarm(i-1,j-1)

else assign maxFarm(i-1,j-1) to max(MW(i-1,j), MW(i, j-1)) + farm[i-1][j-1] (0 if there is no weed, 1 if there is weed) and return maxFarm(i-1,j-1)

Then, PF (path finder) is called with the given values i,j.

PF():

Create a list called pathList for tuples which is initially empty and after that it stores the path by storing all the related cells.

If the cell given matrix is emty give an error else continue with next steps.

Start from the indexes i, j
while i and j not equal to 1:
In each step select the max(MW(i,j-1), MW(i-1,j)) (select only options if the cell is on the upper part or the left-most part) and add the corresponding i, j values to the path list as a tuple.

Assign i and j to the corresponding values to ensure iteration.

After the iteration, print the pathList in a reversed order to start from (1,1) and end in (i,j).

Small note: i and j values in the questions are corresponds to i-1 and j-1 values in the farm matrix due to the fact that indexes of matrices starts from 0 instead of 1. The functions MW and PF takes farm, and maxFarm matrices to avoid bad looks they did not shown in each step.

(c)

Asymptotic Time and Space Complexity Analysis of the Algorithm

Asymptotic Time Complexity:

This algorithm first creates an mxn size matrix called maxFarm $\Theta(mn)$
Then for each uniqe sub-problem:
This algorithm makes a max comparison between left and up cells $\Theta(2)$ and 1 assignment operator to update the value on that cell $\Theta(1)$, therefore $\Theta(1)$ asymptotically
Since there are mn uniqe subproblems and $\Theta(1)$ operations, time comlexity for the above process is $\Theta(mn)$.

Then pathFinder function starts from last cell and by left or up moves go to the first cell, total m+n-1 moves:

In each move it compares two values $\Theta(2)$ and adds 1 path to the pathList $\Theta(1)$ as well as since maxFarm matrix is ready all calls of the MW() functions just takes $\Theta(1)$, therefore $\Theta(1)$ asymptotically
Pathfinder runs in $\Theta(m+n)$ which is smaller than the $\Theta(mn)$
Therefore, from the equation $\Theta(mn)+\Theta(mn)+\Theta(m+n)$ asymptotic time complexity of the algorithm is $\Theta(mn)$.

Asymptotic Space Complexity:

The algorithm uses an mxn matrix for the given values. $\Theta(mn)$

The algorithm uses an mxn matrix for the max weed values. $\Theta(mn)$

The algorithm uses an m+n array for the pathList. $\Theta(m+n)$

Therefore, from the equation $\Theta(mn)+\Theta(mn)+\Theta(m+n)$ the asymptotic space complexity is $\Theta(mn)$.

(d)
All the codes and bencmark suites are in the python file.
Functional Testing:

Both black-box testing and white box testing are implemented. In the black-box testing empty cases, 1xn, nx1, matrix consists of all 1's and matrix consists of all

0's are tested. In white-box testing the matrix in the homework document and 2 more random matrixes are tested.

Both black-box and white-box testing are successfull and the algorithm returns a correct path.

Performance Testing:

In performance testing 5 random matrices are created with different sizes that are 2x2, 4x4, 8x8, 16x16, and 32x32. Then all these matrices are runned in the function, and their running times plotted to the graph.

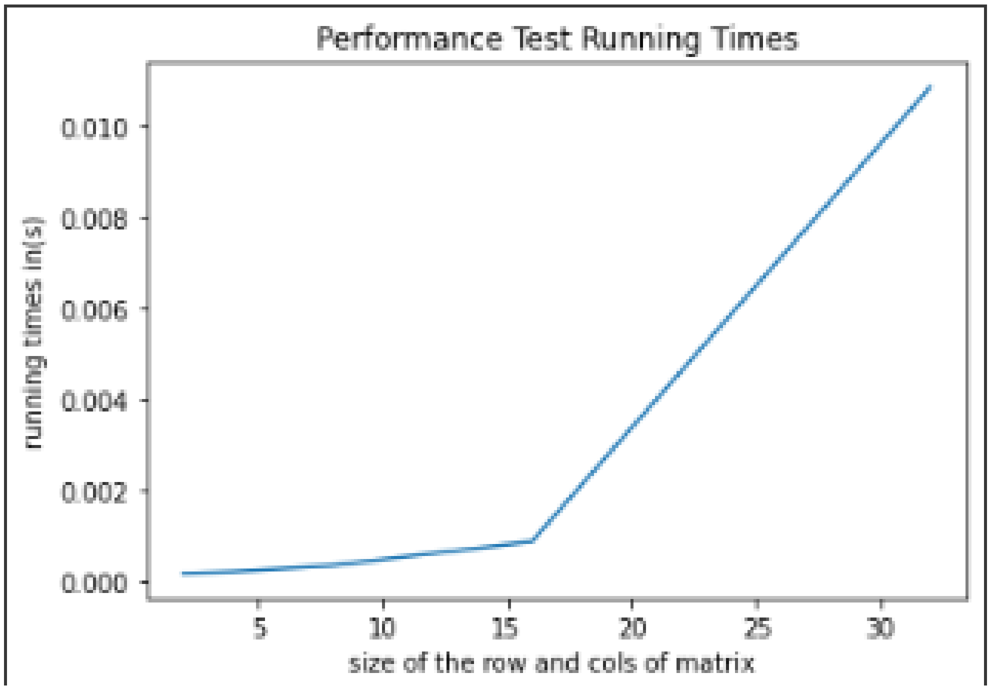

Figure 2: Performance Testing

As it can be seen in the graph, the running times of the different sizes of matrices(inputs) growing like quadratic, Therefore, the performance test also verified the founded asymptotic time complexity of the algorithm which is $\Theta(mn)$.

This algorithm is a good algorithm to solve agricultural robot problem in terms of scalibility when it compared to the Brute Force algorithm in which the overlapping sub-problems will computed again and again.