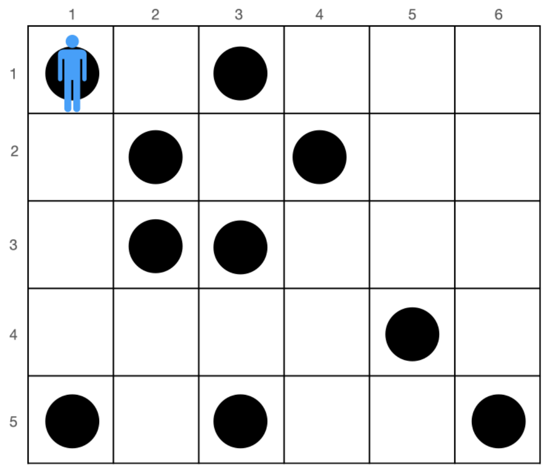
CS301 A4

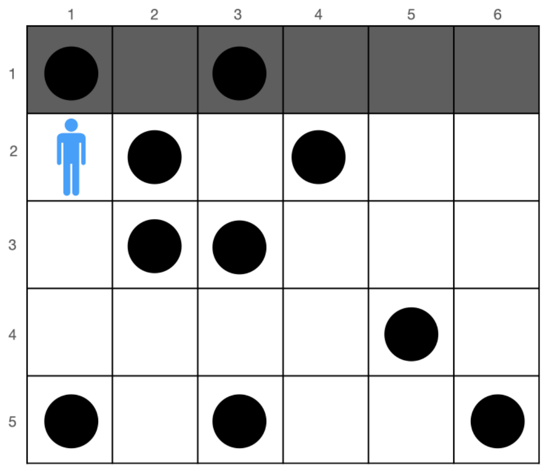
Gizem Topsakal

December 11, 2022

1 Recursive formulation

To find this algorithm's recursive formulation, I first looked at the optimal substructure. Since the robot can only move either right or down, this makes our matrix get smaller and smaller at each step. For example when the robot is on the (1,1) cell, if it goes to (2,1) it cannot go back to the first row anymore. After that, if robot goes to it's right cell, now it cannot go back to first column. This goes on and divides our problem into sub-problems. To make my statement more clear, I added two images describing the path that robot cannot go.

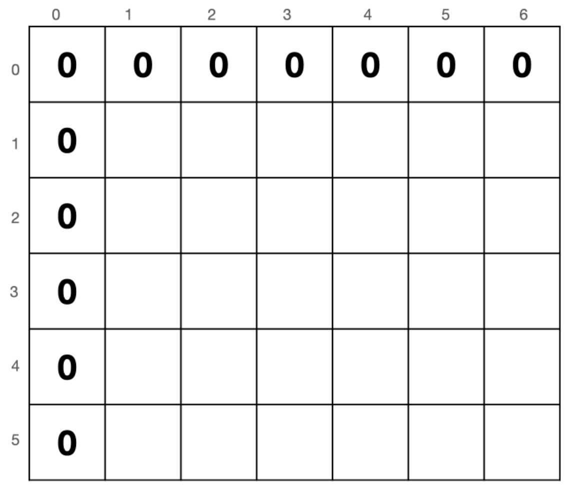




This means, we can divide our problem into smaller sub-problems, so optimal substructure is obtained.

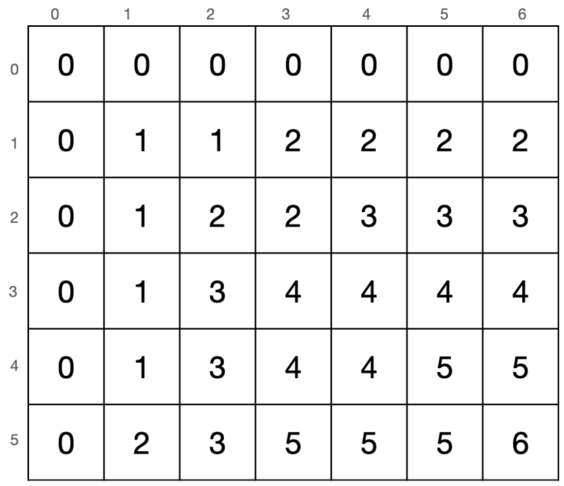
Since we are trying to find paths to collect the maximum number of weed, it is obvious that we are going to travel some cells more than once since we will try different paths. I also drew an example tree in section 3, it can be seen that we are visiting the cells we already visited, so overlapping sub-problems property is also obtained.

Now to generate a recursive formulation, since we have overlapping sub-problems, to decrease the time complexity, I used memoization. I was inspired by LCS dynamic programming algorithm, so I tried to adapt it's mechanism. First I created a base case, when m or n is 0 I made everything 0. So this action generated a safe area for my memoization matrix as I showed below.



	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0						
2	0						
3	0						
4	0						
5	0						

As a next step, I iterated through the matrix recursively but if there is weed on the cell I added 1, so that I can keep the weed count and find the maximum weed that can be collected at the end. After I applied the algorithm to the example in the homework document, the last version of the memoization matrix is below.

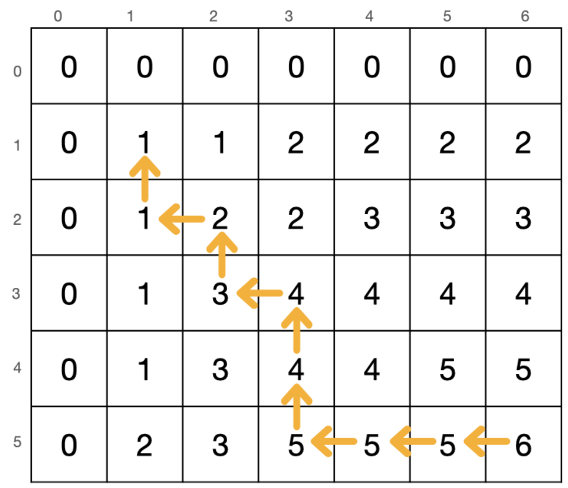


	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	1	2	2	2	2
2	0	1	2	2	3	3	3
3	0	1	3	4	4	4	4
4	0	1	3	4	4	5	5
5	0	2	3	5	5	5	6

To summarize my recursive formula has became this at the end:

$$memo[m][n] = \begin{cases} 0 & \text{if m = 0 or n = 0} \\ 1 + max(memo[(m-1,n)],(memo[(m,n-1)]) & \text{if (m,n) in weed} \\ max(memo[(m-1,n)],(memo[(m,n-1)]) & \text{otherwise} \end{cases}$$

For the path finding part, I created another function called optimalPath. In this function I took the memoization matrix as my parameter, and I tried to find the path by going from bottom to up. Since in the memorization matrix, I had a safe area where everything was 0, I added an if statement where I check if m and n is bigger than to 0. And two other if statements where n is equal to 0, m is bigger than 1, then m is equal to 0, n is bigger than 1 to check the first row and first column. In the algorithm, starting from the last cell, I checked every cell's top and left cell recursively, and whichever is the maximum, I called the recursive function for them and added the current cell I am on into an array. If the top and left cells are equal, I went to the left cell. At the end, since I started from the bottom of the memoization matrix, path was backwards, so I reversed the array. This way, I made my way from the last cell to the first, following the weeds I collected and find an optimal path. Visualization of my algorithm can be seen below.



	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	1	2	2	2	2
2	0	1	2	2	3	3	3
3	0	1	3	4	4	4	4
4	0	1	3	4	4	5	5
5	0	2	3	5	5	5	6

My recursive formulation of optimalPath has became this at the end:

$$optimalPath(m, n, memo) = \begin{cases} optimalPath(m, n-1, memo) & \text{if memo[m][n-1]>=memo[m-1][n]} \\ optimalPath(m-1, n, memo) & \text{otherwise} \end{cases}$$

2 Pseudocode

Algorithm to find the maximum number of weeds to be collected:

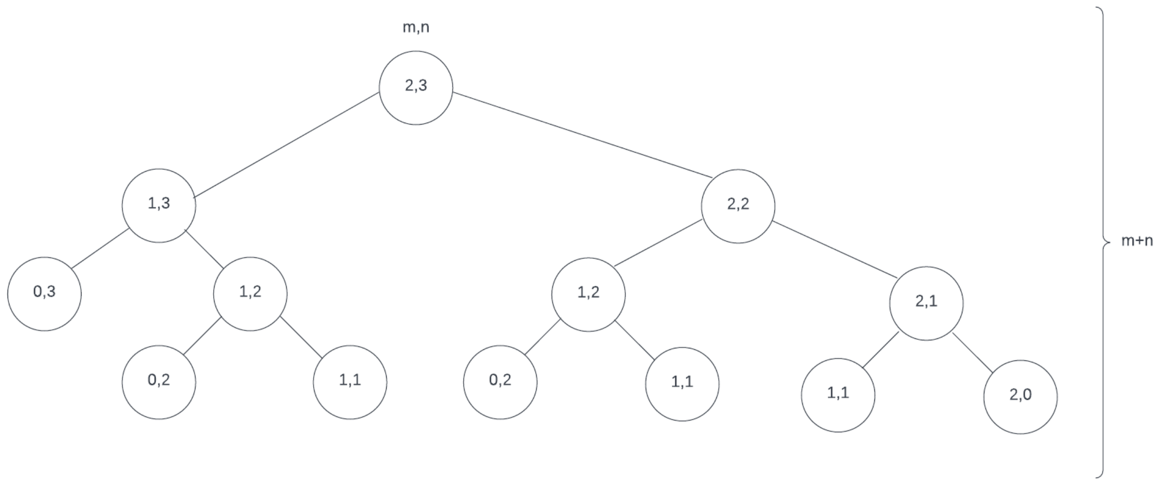
```
agriculturalRobot(m,n,weed,memo)
  if (m,n) in memo then
      return memo[m][n]
  end if
  if n==0 or m==0 then
      memo[m][n] = 0
  end if
  if (m,n) in weed then
      memo[m][n] = 1 + max(agriculturalRobot(m1,n,weed,memo),agriculturalRobot(m,n-
  1,weed,memo))
  else
      memo[m][n] = max(agriculturalRobot(m-1,n,weed,memo),agriculturalRobot(m,n-1,weed,memo))
  end if
  return memo[m][n]
```

Algorithm to find an optimal path to collect the maximum number of weeds:

```
optimalPath(m,n,memo,path)
  if m > 0 and n > 0 then
      if memo[m][n-1]>=memo[m-1][n] then
          path.append((m,n))
          optimalPath(m,n-1,memo,path)
      else
          path.append((m,n))
          optimalPath(m-1,n,memo,path)
      end if
  else if n = 0 and m > 1 then
      path.append((m-1,n+1))
      optimalPath(m-1,n,memo,path)
  else if m = 0 and n > 1 then
      path.append((m+1,n-1))
      optimalPath(m,n-1,memo,path)
  end if
```

3 Asymptotic time and space complexity analysis

To investigate asymptotic time complexity, I am going to use m = 2 and n = 3 matrix and crate a tree of paths I can go.



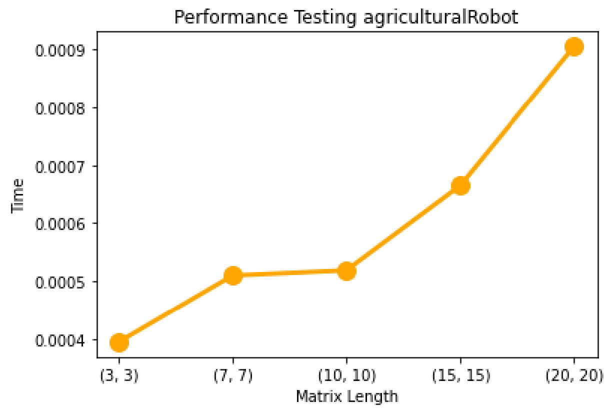
This tree represents if I were to use a brute-force recursive algorithm, I should have travel around this tree. From one node to the next, I can either decrease the m by one or I can decrease the n by one. Overall, from root to some base case (leaf), is going to have the distance m + n. As we can see from the tree, number of levels are increasing in a two times two pattern. Since we know that there is m + n levels, time complexity will be $\theta(2^{(m+n)})$. And the space complexity will be the height of the tree which is the number of levels, so it is $\theta(m + n)$.

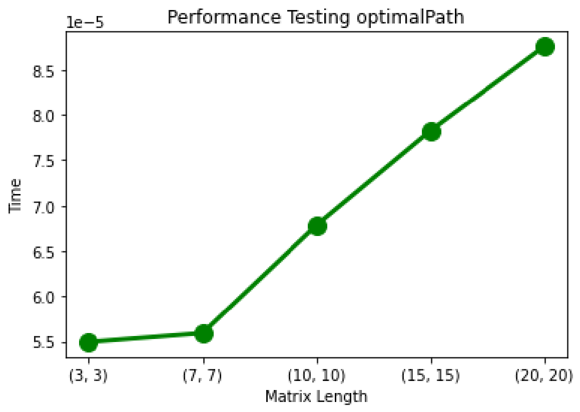
The algorithm I used was a memoization algorithm. In a brute-force algorithm, I am traveling through all the nodes but in a memoization algorithm, since I save the nodes I have traveled, I do not have to travel repeated nodes again. So the time complexity will be the number of unique nodes which is same as the number of cells in my memoization matrix. That will give us $\theta(m * n)$ time. Space complexity will remain $\theta(m + n)$ since it depends on the input size.

In the optimalPath algorithm, all I do is just come back from the last cell to the first by following the path I found in the agriculturalRobot algorithm. So, since I am only following one path, time will be the path size. And while following the path I cannot go diagonal, I can only move left or up, the path size will be m + n. So this makes our time complexity $\theta(m + n)$. Space will be the number of cells I went through while following the path so it is also $\theta(m + n)$.

4 Experimental evaluations

To evaluate experimental results I generated 5 matrices. Their sizes are (3,3), (7,7), (10,10), (15,15), (20,20). Weed lists were randomly generated. I calculated their times separately for each function and plot them in a graph.





The results were not far from what I expected. optimalPath results were closer to what I imagined, since the time should be $\theta(m+n)$. , it looks more like a linear function. And proportions are okay, especially, the (10,10), (15,15), and (20,20) matrices are really convenient. In agriculturalRobot the graph kept changing but I didn't really understand why it was unstable. The results I expected were more exponential, since time is $\theta(m*n)$. But what I got was more sharped-edged results every time.