CS301 Assignment 4

Ataollah Hosseinzadeh Fard (ID: 28610)

11 December 2022

Algorithm flow:

1) create a matrix that in each sell writes the count of steps needed to reach (1, 1).
2) for the sells that there are weeds in farm, add 1 to same position in matrix.
3) to calculate the path from matrix, recursively from (n, m) start looking for max{leftCell, upCell}, and move to max one in terms of value.
4) in case up and left have same value, look to their parent cells, parent of left is (i, j-2) and parent of up is (i-2, j).
5) if parents of left and up have same value, the move up if i > j else move left.

(a) Recursive Formulation

m is the matrix that has stored values from step 1 and 2 of algorithm.

$$optimalPath \begin{cases} m(i,j) & if\ i = 0\ and\ j = 0 \\ m(i,j) + max\{m(i-1,j), m(i,j-1)\} & else \end{cases}$$

(b) Pseudocode

```
def optimalPath(farm, i, j):
    if I == 0 and j == 0:
        return “(i, j)”
    else if i == 0 and j > 0:
        return optimalPath(farm, i, j - 1) + “(i, j)”
    else If i > 0 and j == 0:
        return optimalPath(farm, i - 1, j) + “(i, j)”
    else:
        if farm[i][j-1] > farm[i-1][j]:
            return optimalPath(farm, i, j - 1) + “(i, j)”
        else if farm[i][j-1] < farm[i-1][j]:
            return optimalPath(farm, i - 1, j) + “(i, j)”
        else:
            upper ← (i > 1) if farm[i-2][j] else -1
            lower ← (j > 1) if farm[i][j-2] else -1
            if upper > lower:
                return optimalPath(farm, i, j - 1) + “(i, j)”
            else if upper < lower:
                return optimalPath(farm, i - 1, j) + “(i, j)”
            else:
                if I > j:
                    return optimalPath(farm, i - 1, j) + “(i, j)”
                else:
                    return optimalPath(farm, i, j - 1) + “(i, j)”
```

```
def getOptimalPath(farm):
        valueMatrix ← []
        for i in range(farm.size):
                for j in range(farm[i].size):
                        valueMatrix[i][j] ← i + j + farm[i][j]
        return optimalPath(valueMatrix, farm.size - 1, farm[0].size – 1)
```

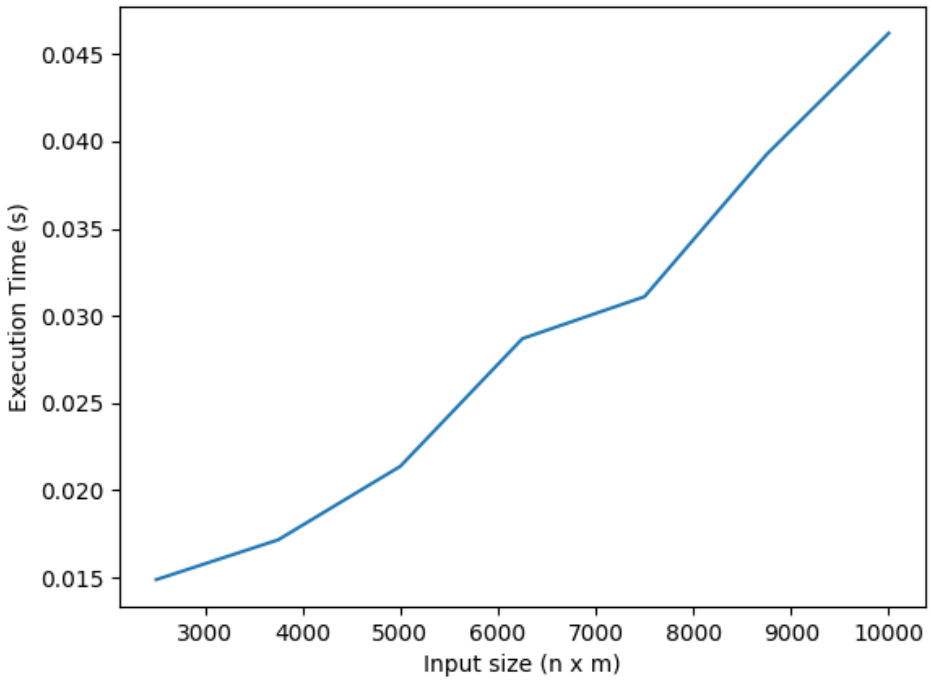
(c) Asymptotic time & space complexity analysis

filling the value matrix will cost O(n*m) since the size of farm is n*m.

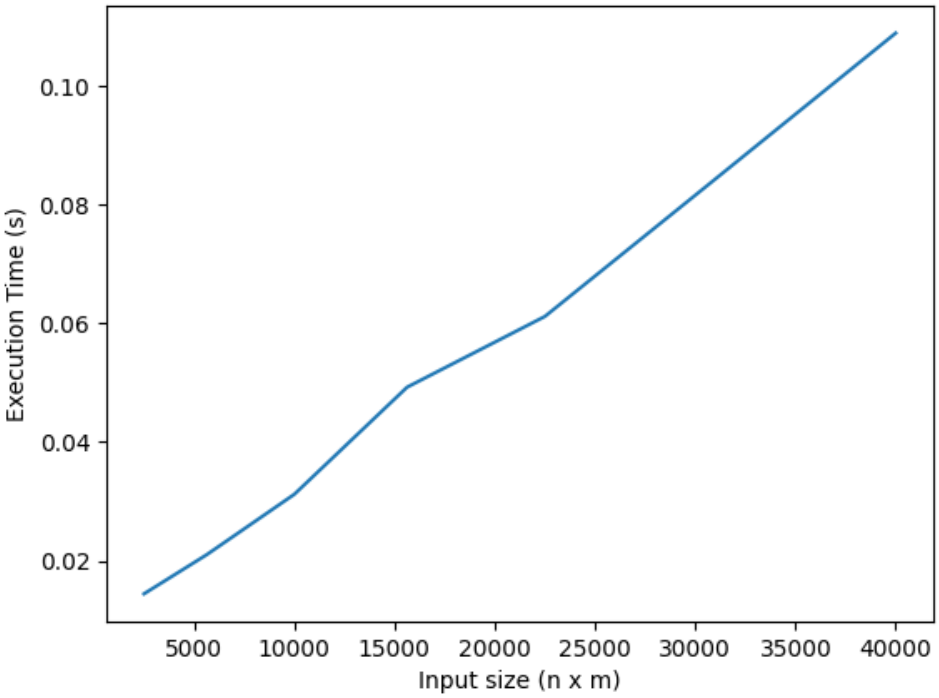
finding path will cost O(n+m) because in finding path we either go up or left, so keep moving in route towards (1, 1) and we do not increase the steps in middle, and we know that (n, m) is n + m steps far from (1, 1) hence finding path costs O(n + m). In total, my time complexity is O(n*m + n+m) and if we make it simpler the total time complexity is O(n*m).

(d) Result and conclusion

I assessed the program with different sizes of farms and different cases, and farms were randomly generated, and the following graphs show the results. However, if I rerun the graph will change due to having random farms. I general, as O(n*m) is depends of growth of n and m, if both increases it will result as quadratic growth, but if only one of them increase and other stay constant O(n), then we will encounter a linear graph O(n^2). On contrary, it is very hard to obtain graphs that shows difference of cases, because having randomly generated matrices, causing dense/sparse matrices, if there are very few weeds in matrix it is considered sparse and if there are more weed cells that empty cells then that matrix is dense. in each trial this density changes and decided randomly so if there are dense and sparce matrices, they cause balance in graphs and both graph look like linear but theoretically, the one that has m as constant should be the only linear among results.



in this case, m was fixed to 50, and n started from 50 and increased until 200, each step n is increased by 25. Each size evaluated on 100 different matrices.



in this case, both n and m started from 50 and increased until 200, each step n and m are increased by 25. Each size evaluated on 100 different matrices.