Anand Mahadevan - SID 862132182

CS141 Homework 3

due Saturday, August 28th 11:59 PM

Problem 1.

$$p_{i,j} = \begin{cases} \textit{base case} & ; \textit{base case condition} \\ \textit{recurrence case} & ; \textit{otherwise} \end{cases}$$

Let $A[i, j]$ represent the number of pounds of cherries in plot in row i and column j The cherry field has m rows and n columns.

1. (5 points) Write down the base case and base case condition of the recurrence relation.

   Base Case: 0
   Base Case Condition: $i > m$ *or* $j < 1$ *or* $j > n$

2. (5 points) Write down the recurrence case of the recurrence relation.

   Recurrence Case: $A[i, j] + max\{p_{i+1,\ j-1}\ , \quad p_{i+1,\ j}\ , \quad p_{i+1,\ j+1}\}$

3. (5 points) Which value of the recurrence relation represents the final answer of the problem? In other words, which entry in the table (i, j) will contain the final answer.

   The answer of this problem will be the max element located in row 1 of my table.

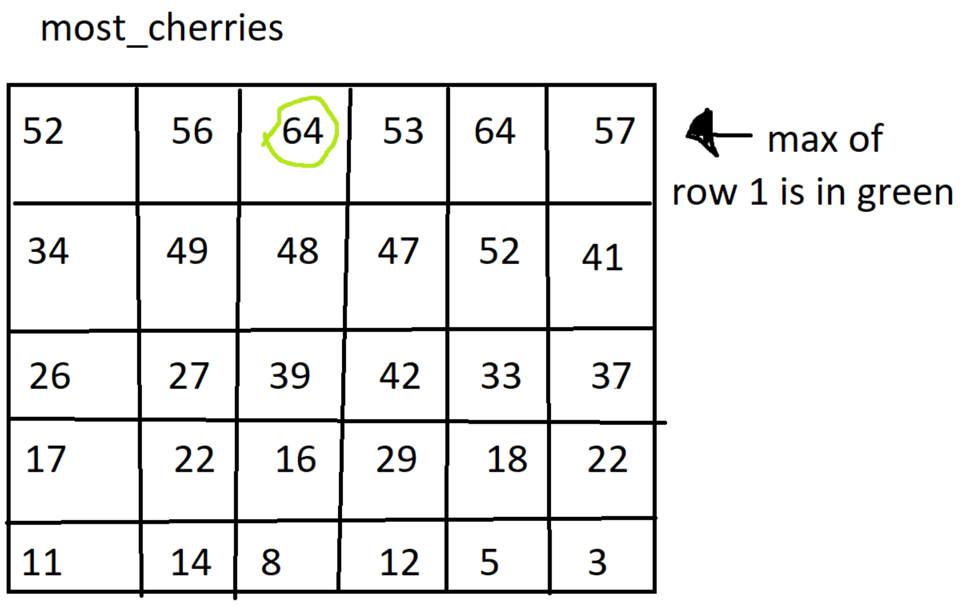
4. (5 points) Write down a bottom-up iterative algorithm based on your recurrence relation that finds the maximum amount of cherries to pick. You do not have to produce the path in this part.

   Strategy: Fill m by n array "most_cherries" with max cherries able to be picked from mth row to 1st row using the recurrence relation, and iterate through each column in each row. Then choose largest number of cherries in row 1 of "most_cherries". (A is the cherry field which is also m x n)

```
MAXCHERRY(A):
    most_cherries = [][] // if invalid location is accessed such as (-1,-1), it will be 0
    for row = A.rows; row >= 1; row − = 1 :
        for column = 1; column <= A.columns; column + = 1 :
            if row == A.rows : // last row means nothing to look below
                most_cherries[row][column] = A[row][column]
            else:
                most_cherries[row][column] = A[row][column]+ max{most_cherries[row+1][column-1],
                  most_cherries[row+1][column],most_cherries[row+1][column+1]}
    return max( most_cherries[1] ) // max element in first row of most_cherries 2D array − > O(n)
```

5. (5 point) What is running time of your algorithm?
   $O(m * n)$

6. (5 points) Use your bottom-up algorithm to answer the problem given above and highlight the optimal answer. Keep in mind that, like all optimization problems, there might be more than one optimal answer but they should all have the same final value.



most_cherries

52	56	64	53	64	57
34	49	48	47	52	41
26	27	39	42	33	37
17	22	16	29	18	22
11	14	8	12	5	3

max of row 1 is in green

7. (5 points) (Bonus) Update your algorithm to produce the optimal path as a sequence of plot locations, in addition to the final value.

   Strategy: Fill m by n array "most_cherries" with max cherries able to be picked from mth row to 1st row using the recurrence relation, and iterate through each column in each row. *Then add the sequence of plots below the current (i,j) position, including the current plot location first.* Then choose largest number of cherries in row 1 of "most_cherries" *and the sequence associated.* (A is the cherry field which is also m x n)

```
MAXCHERRY(A):
    most_cherries = [][] // if invalid location is accessed such as (-1,-1), it will be 0
    for row = A.rows; row >= 1; row − = 1 :
        for column = 1; column <= A.columns; column + = 1 :
            if row == A.rows : // last row means nothing to look below
                most_cherries[row][column] = (A[row][column], [(row, column)])
            else:
                below_cherries = [most_cherries[row+1][column-1][1],
                  most_cherries[row+1][column][1],most_cherries[row+1][column+1][1]]
                most_cherries[row][column][1] = max( below_cherries )
                index_max = below_cherries.index(most_cherries[row][column][1])
                match index_max :
                    case 1 :
                      most_cherries[row][column][2] =
                        [(row,column)].append(most_cherries[row+1][column-1][2]
                    case 2 :
                      most_cherries[row][column][2] =
                        [(row,column)].append(most_cherries[row+1][column][2]
                    case 3 :
                      most_cherries[row][column][2] =
                        [(row,column)].append(most_cherries[row+1][column+1][2]

    max_cherries = max( most_cherries[1][][1]) // max cherries in first row of most_cherries 2D array
    idx_max = most_cherries[1].index(max_cherries) // column location of max cherries
    return (max_cherries, most_cherries[1][idx_max][2])
```