HW #2 - Problem 2

Ryan St.Pierre (ras70)

September 20, 2017

Problem 2A

Let's imagine we are trying to calculate the largest possible value of the path from (1,1) once we are already at location (n,m). We would need to add the value of our current location to the maximum value of the largest value from the path if we came from the left and the largest value from the path if we came from below (since these are the only valid ways we could have gotten to (n,m)). An analysis of the sub-problems, transition function, and base cases are given below:

Sub-problems

Given the explanation above we need the answers to the problems where we came from the left and if we came from below. In other words we need the answer to two sub-problems,

1. *(left.)* Need the largest maximum path for the grid constructed from rows 1 to n and columns 1 to $m-1$. This is the original grid **without** the last column.

2. *down.* Need the largest maximum path for the grid constructed from rows 1 to $n-1$ and columns 1 to m. This is the original grid **without** the first row.

Transition function

Let $a[i][j]$ be the largest possible path value from coordinate (1,1) to (i,j). Given this the transfer function can be defined as,

$$a[i,j] = v[i,j] + max(a[i-1][j], a[i][j-1])$$

Note: the transition function above has potential index out of bounds errors. To avoid these we need to define the base cases in clever ways. This is done in the next section.

Base case

The primary base case is given by:

```
a[1,1] = v[1,1]
```

In words this means the longest path between (1,1) and (1,1) is simply the value of $v[1][1]$ (the value of the tile at (1,1)). This is intuitive - since there is no path between (1,1) and (1,1) the largest value must simply be the value at that location.

We also to define a base case for when the index being accessed is out of bounds. This is done below:

```
2. a[i,0] = 0 for 1<=i<=n

3. a[0,j] = 0 for 1<=j<=m

(a[0,0] can also be set for consistency but it is not strictly necessary)
```

These base cases essentially ensure a path from out of bounds is never chosen. This can be achieved by replacing the zero with any negative number.

Problem 2B

Algorithm

Once the proper transfer function has been established (as was done in Problem 2A) it can be used to define the proper algorithm. The algorithm can be summarized in the following steps:

1. Set $a[1][1] = v[1][1]$ *(base case)*. Also set the value of $a[i][j]$ for when either i or j is equal to zero and the other is from 1 to its upper bound.
2. Calculate $a[i][j]$ for all values i: 1 to n and j: 1 to m. This defines the largest possible paths from location (1,1) to any other location in the grid.
3. Return $a[n][m]$, as this corresponds to the largest value of the path from (1,1) to (n,m).

For further clarity please reference the pseudo-code given below.

Visualizing the DP Table

The DP table can be visualized as a 2 dimensional grid. If we allow i to be represented by the columns in the table and j by the rows in the table then the columns range from 0 to n and the rows range from 0 to m (a further description of these bounds and how they were chosen is given below). The

first cell of the DP table (1,1) is equal to $v[1][1]$, per the base case. The first row and column of the DP table, corresponding to zero indices are equal to zero.

In this representation any cell in the DP table (i,j) corresponds to the largest path between location (1,1) and (i,j) in the grid from the problem. Any cell (i,j) is calculated by taking the max of the cell directly to its left and above and adding that to the value $v[i][j]$

The solution to this problem corresponds to the cell in the DP table (n,m), which defines the largest value of the path from (1,1) to (n,m).

Ordering/Bounds

Transfer function (ignoring conditional statement):

$$v[i,j] + max(a[i-1][j], a[i][j-1])$$

From the transfer function it is clear that the pair (i, j) is dependent on $(i-1, j)$ and $(i, j-1)$. In the DP table this can be interpreted as values to one to the left and one above of the value being calculated.

The bounds for the i and j are given as $[0, n]$ and $[0, m]$ respectively. An ordering that ensures $a[i, j]$ has the values necessary to be calculated is given below (ignoring the base case with zero indices):

```
i: 1 -> n
j: 1 -> m
```

Pseudo-code

```
computeLargestPath(n, m, v) {

    a[1][1] = v[1][1]

    for i = 1 to n {
        a[i,0] = 0
    }

    for j = 1 to m {
        a[0,j] = 0
    }
```

```
    for i = 1 to n {
        for j = 1 to m {
            // Transfer function
            a[i, j] = v[i,j] + max(a[i-1, j], a[i, j-1])
        }
    }
    return a[n][m]
}
```

Running time

From the pseudo-code above the algorithm has two nested for loops ranging from 1 to n and from 1 to m. The work within these nested for loops is constant (simple comparisons). Thus, the running time of this algorithm can be given by $O(nm)$. This is the desired running time given in the problem statement.

There also exists two *for* loops for the base case. Thus, the true running time is $O(nm + m + n)$. However, this is bounded by $O(nm)$.

Proof of correctness

Proof by induction:

Let $(i, j) < (i', j')$ if $i < i'$ or ($i = i'$ and $j < j'$). Also let a[i,j] be the value of the transfer function a for values i and j.

Induction Hypothesis: The algorithm correctly computes a[i,j] for all values of $a(i, j)$ where $(i, j) < (i', j')$.

Base case:

n=1,m=1 / i=1,j=1: The algorithm returns v[1,1]. This is correct because it is the largest path from (1,1) to (1,1) which is just the value at (1,1).

i=0,$1 < j < m$*:* For this case the path is out of bounds and no path can be made. Thus, zero is returned.

$1 < i < n$*,j=0:* For this case the path is out of bounds and no path can be made. Thus, zero is returned.

Induction Step:

When computing $a[i', j']$ by induction hypothesis $a[i'-1][j']$ and $a[i'][j'-1]$ have been computed correctly. The algorithm considers the value of the

current location ($v[i'][j']$) and the optimal value of the two possible paths to the current location, $a[i'-1][j']$ and $a[i'][j'-1]$. Therefore: value at $a[i'-1][j'-1]$ is also correct.

Thus, the a[n,m] by definition is the maximum possible value for the path from (1,1) to (n,m)

Problem 2C

Algorithm Modification

The approach for this new problem, with the addition of new moves, is similar to that of Problem 2A and 2B. If we imagine we are at location (n,m) of the grid, the solution to the problem can be reduced to the summing of the value of tile (n,m) and the largest value from all the possible paths to (n,m). However, in Problem 2C there are 4 locations that could lead to a move to (n,m), not 2. Thus, these 4 sub-problems need to be added to the transfer function. The modified transfer function is given below:

$$a[i,j] = v[i,j] + max(a[i-1][j], a[i][j-1], a[i-2][j+1], a[i+1][j-2])$$

The visualization of the DP table, algorithmic steps taken, and base case are all the same for this problem (in comparison to Problem 2B). The steps of the algorithm and base case have been included below for reference.

1. Set $a[1][1] = v[1][1]$ *(base case).*
2. Calculate $a[i][j]$ for all values i: 0 to n and j: 0 to m. This defines the largest possible paths from location (1,1) to any other location in the grid.
3. Return $a[n][m]$, as this corresponds to the largest value of the path from (1,1) to (n,m).

The base cases are given by:

```
a[1,1] = v[1,1]
```

```
2. a[i,0] = 0 for 1<=i<=n

3. a[0,j] = 0 for 1<=j<=m

(a[0,0] can also be set for consistency but it is not strictly necessary)
```

We also need to add a check now for when i and j are negative because these can now be encountered with the minus 2 cases. These should be handled such that they are not considered in the maximum

Ordering/Bounds

The bounds for the i and j are still given as $[0, n]$ and $[0, m]$. However, the ordering needs to be changed. The ordering of the modified algorithm is the **most** important difference between Problem 2C and 2B.

From the transfer function it is clear that the pair (i, j) is dependent on $(i-1, j)$, $(i, j-1)$, $(i-2, j+1)$, and $(i+1, j-2)$. It is clear that simply linearly iterating through i and j from 1 to their respective upper bounds will **not** work. For example, if i and j are increased linearly and $n = 4$ and $m = 5$, the calculation of (3,3) will require the solution to (1,4), which has not yet been computed.

This ordering problem can be solved by iterating through i and j in a diagonal fashion. Thus, the ordering is the following:

(0,0), (1,0), (0,1), (2,0), (1,1), (0,2), (3,0), (2,1), (1,2), (0,3)...

By careful inspection of this ordering it is clear that the ordering is dependent on the **sum** of i and j. In other words the ordering of i and j can be summarized as follows:

```
The pair (i,j) is said to come after the pair (i',j')
in the ordering if i+j > i'+j'
```

The relative ordering of pairs (i, j) and (i', j') where $i + j > i' + j'$ is not crucial to the success of the algorithm. Thus, the relative ordering of such points can be chosen in any fashion desired.

Pseudo-code

```
computeLargestPath(n, m, v) {

    a[1][1] = v[1][1]

    for i = 1 to n {
```

```
        a[i,0] = 0
    }

    for j = 1 to m {
        a[0,j] = 0
    }

    int dim = n+m;
    for( int k = 1 ; k < dim ; k++ ) {
        for( int i = 1 ; i <= k ; i++ ) {
            int j = k - i;
            if( i <= n && j <= m && j>-1 && i>-1 ) {
                a[i][j] = TRANSFER FUNCTION*
            }
        }
    }

    return a[n][m]

}

* The transfer function has been excluded from this pseudo-code for the sake of cl
```

Running time

Although it is not as clear from the pseudo-code above the running time for the modified algorithm is still given by $O(nm)$. Even though the algorithm does not loop through its two iterating variables (i and j) linearly it still must touch all the elements of an n by m grid, thus the running time is $O(nm)$.

Proof of correctness

Proof by induction:

Important: Let $(i, j) < (i', j')$ if $i + j < i' + j'$. Also let a[i,j] be the value of the transfer function a for values i and j.

Induction Hypothesis: The algorithm correctly computes a[i,j] for all values of $a(i, j)$ where $(i, j) < (i', j')$.

Base case:

n=1,m=1: The algorithm returns v[1,1]. This is correct because it is the largest path from (1,1) to (1,1) which is just the value at (1,1).

Induction Step:

When computing $a[i', j']$ by induction hypothesis $a[i' - 1][j']$, $a[i'][j' - 1]$, $a[i' - 2][j' + 1]$, and $a[i' + 1][j' - 2]$ have been computed correctly. This falls directly from the ordering (shown below):

$$\begin{gathered} i' - 1 + j' = i' + j' - 1 < i' + j' \\ i' + j' - 1 < i' + j' \\ i' + 1 + j' - 2 = i' + j' - 1 < i' + j' \\ i' - 2 + j' + 2 = i' + j' - 1 < i' + j' \end{gathered}$$

The algorithm considers the value of the current location ($v[i']$[j']) and the optimal value of the four possible paths to the current location, $a[i'-1][j']$ and $a[i'][j' - 1]$. Therefor: value at $a[i' - 1][j' - 1]$ is also correct.

Thus, the a[n,m] by definition is the maximum possible value for the path from (1,1) to (n,m)