Ryan Young

CSE 3500

11/11/2019

Homework 8: At the Bird Market

1. We denote OPT(I, g) the total number of ways to fill I cages without adjacent males, if the last bird has gender g.
   a. Express the solution to the original problem using the optimal solutions to the subproblems.
      The optimal solution to the subproblems:

      We know that if the last gender g is male then the prior bird must be female, however if that bird's gender is female then the prior bird can be either male or female. This uncertainty is where the complexity of this problem comes from. So, each time there is a female we can think of it as a fork in a tree or a branch where it can go back to being male or it can have another female and create a whole new line of possibilities. So, to solve a subproblem we must solve this male to male, female to female and male to female relationship. I wrote pseudo code below that will solve each subproblem and then add to a sum variable for each different possible ordering of the birds.

```
Sub-problem-solution(n, g): // takes in number of cages and last birds' gender
    Int Sum = 1
    If n == 0: // check if the number of cages is 0
        Return sum

    If g is male: // will not add a new ordering because we know the next gender
        n-= 1 // subtract from the number of cages

    Else: //female case, adds a branch/ fork because we don't know the next gender
        Sum += 1 // add the current uncertainty
        n-=1 // subtract from the number of cages
        Sub-problem-solution(n, male) + sub-problem-solution(n, female) //
        with the uncertainty that follows
```

   b. If the last bird is male, the next bird must be female. If the last bird is female, then the next bird can be either male or female. Express this as a recurrence relation. Let g be a 0 if the gender is male indicating there is only on recursive call (female option) and g be 1 if the gender is female (male or female options). Then each call the next level down is n-1 because we are filling one cage at a time with a bird based on that gender option. So, when the gender is male it will add g and when the gender is female it will go all the way down calculating the recurrence.

$$C(n, g) <= g * (C(n-1, male) + (C(n-1), female)) + g$$

   c. Design a Dynamic Programming Algorithm that solves the problem.

Since this algorithm will build from the top down, we want to return the sum when we no longer have any cages left. Then we check

```
M=[] // array to hold values
Dynamic_Programming_Solution(n, g):
    If m != NULL:
        Return m[g][n]
    Else if n == 1:
        Return M[g][n] = 1
    Else:
        If g = Male:
            M[g][n] = Dynamic_Programming_Solution(n-1, female)
        Else:
        M[g][n] = Dynamic_Programming_Solution(n-1, male) + Dynamic_Programming_Solution(n-1, female)
        End if
    End if
Return M[g][n]
End func
```

d. The amount of space used to solve the problem can be reduced drastically. Redesign your algorithm without the 2d array.
Using a one d array

```
Dynamic_Programming_Solution(n, g):

        If m != NULL:
            Return m[n]
        Else if n == 1:
            Return M[n] = 1
        Else:
            If g = Male:
                Dynamic_Programming_Solution(n-1, female)
            Else:
                M[n] = Dynamic_Programming_Solution(n-1, male) +
                Dynamic_Programming_Solution(n-1, female)
            End if
        End if

    Return M[n]

    end func
```