Assignment 3

Joe Puccio

November 4, 2014

Collaborators: Sana Imam, Ryan Allan.

15.1-2

Consider the case where $n = 4$, and $p_1 = 1$, $p_2 = 5$, $p_3 = 8$, and $p_4 = 9$. Well, if this greedy algorithm were to be applied, it would examine each of the densities, $p_1 = 1$, $p_2 = 2.5$, $p_3 = 2.67$, and $p_4 = 2.25$, and choose to split at position 3, thus leaving the algorithm to repeat on size $n = 1$ which would halt immediately. The resulting revenue would be $8 + 1 = 9$. However, we examine that if we had instead split at position 2, the resulting revenue would have been $5 + 5 = 10$. Therefore, this greedy algorithm does not always determine the optimal way to cut the rods to maximize revenue.

15.1-3

The following bottom-up algorithm solves the modified problem containing penalties, of size c, for each cut.

Require: recall[0...n] be a new array that stores the established maximums. That is, if recall[n] is defined, then it $recall$[n] = $p[n]$.

```
1:  recall[0] = 0
2:  for j = 1 → n do
3:      q = −∞
4:      for i = 1 → j do
5:          if j-i=0 then
6:              q = max(q, p[i] + recall[j − i])
7:          else
8:              q = max(q, p[i] + recall[j − i] − c)
9:          end if
10:         recall[j] = q
11:     end for
12: end for
13: return recall[n]
```

.

3.

We achieve the following values of m (may be off by 1 on some values due to varying initial conditions): [1, 1, 1, 2, 2, 3, 4, 6, 7, 11, 15, 22, 31, 45, 60, 87, 118, 171, 233, 334, 458, 655, 904, 1287, 1781, 2535]. So ultimately, the answer is that numbers as large as 2535 may be passed between the two spies given that they have only 26 stones to throw.

The following algorithm was used to produce this list:

```
#!/usr/bin/python
import sys
#collaborators: Sana Imam, Ryan Allan

currentRow = []
overallOutput = []

for n in range(0,27):
        nextRow = []
        if n==0:
                nextRow.append(1)
        else:
                m=1 #start at the first column
                while True:

                        #calculate valueToAdd
                        valueToAdd = 0
                        iterationsRequired = len(currentRow)
                        for i in range(0,iterationsRequired):
                        #essentially go from the
                        #left (col 1) through the nonzero values
                                if currentRow[i]>=m:
                                        valueToAdd+=1

                        #add the value directly above to get info
                        #from all rows above currentRow
                        #(this is the dynamic programming bit)

                        if valueToAdd!=0:
                                        if m-1>=len(currentRow):
                                                pass
                                        else:
                                                valueToAdd+=currentRow[m-1]
                        #print valueToAdd
                        m+=1
                        if valueToAdd == 0:
                                break
                        else:
                                nextRow.append(valueToAdd)
        #print nextRow
        #we've constructed the row, now let's get our value of interest from that row
        for indexOfRow in range(0,len(nextRow)):
                if nextRow[indexOfRow]<indexOfRow:
                        overallOutput.append(indexOfRow)
                        break

        currentRow=nextRow
        print n
        print overallOutput

print overallOutput
```