

CS375 Weekly Assignment 6

Due April 7th 10PM ET

A general note for CS375: When writing up your homework, please write neatly and explain your answers clearly, giving all details needed to make your answers easy to understand. Graders may not award full credit to incomplete or illegible solutions.

You are encouraged to discuss the problems with your classmates, **but your submission should be your own**. When writing your solutions up, you should not be looking at any other student's submission.

Clear communication **is the point**, on every assignment. In general in CS375, unless explicitly specified otherwise, answers should be accompanied by explanations. Answers without explanations may not receive full credit. Please feel free to ask me any questions about explanations that might come up!

Necessary acknowledgements: many of these questions are reproductions or adaptations of questions of Eric Aaron's devising. Others are from our course texts as listed on the syllabus.

1. The following few questions concern the rod-cutting problem from class:

Rod Cutting

Input: A length n rod and an array $p[0, \dots, n]$ where $p[i]$ denotes the sell-price of a length i rod.

Output: A collection S of positive integers such that $\sum_{s \in S} s = n$ and $\sum_{s \in S} p[s]$ is maximized.

- (a) First, an algorithm that was proposed by a student last semester:

```
def OPT(n):
    if n <= 0:
        return []

    output = []

    i = index less than or equal to n that maximizes p[i] / i
    # so i maximizes price-per-unit-length

    remaining = n

    while remaining >= i:
        add i to output
        remaining -= i

    # here + is using list concatenation
    return output + OPT(remaining)
```

Show that this algorithm is wrong.

Solution: Consider the input $p = [0, 1, 8, 10]$ (so $p[0] = 0$, $p[1] = 1$, ...). Then by the algorithm above $\text{OPT}(n = 3) = 8 + 1 = 9$, since the best value per unit length is for a length 2 rod (which gets a value of 4 per unit), leaving a unit length left over. However, the optimal solution would just be to sell the length 3 rod without cutting it, getting a value of $10 > 9$.

- (b) Let's change the problem a little: suppose we are only allowed to make at most k cuts for some integer $k \geq 0$. Develop a $\Theta(n^2k)$ algorithm to determine the best way to cut a length n rod using **at most** k cuts.

Trace through your algorithm on the input $p = [0, 8, 18, 26, 26, 36, 41, 49, 55, 63, 68, 77, 79, 84]$ and $k = 3$.

Solution: First, let

$\text{OPT}(i, j)$ = the best value we can obtain from a length i rod using at most j cuts.

To determine a recursive formula for this, note that

- $\text{OPT}(0, j) = 0$ for all j , because if I don't have any rod left then I can't get any value from it
- $\text{OPT}(i, 0) = p[i]$ for all i , because if I can't make any cuts then I have to sell the rod at whatever length it is
- for $i > 0$ and $j > 0$: the optimal solution either doesn't make a cut (getting value $p[i]$), or it needs to further reduce it down by some number of cuts. Let's say one of the pieces the optimal solution breaks the length i rod into has length ℓ : then the rest of the $i - \ell$ length has been broken into at most j pieces (by making at most $j - 1$ cuts) **and is hence the most value I can get from a length $i - \ell$ rod with at most $j - 1$ cuts.**

Hence, putting the above cases together, we get

$$\text{OPT}(i, j) = \begin{cases} 0 & \text{if } i = 0 \\ p[i] & \text{else if } j = 0 \\ \max \begin{cases} p[i] \\ \max_{0 < \ell < i} p[\ell] + \text{OPT}(i - \ell, j - 1) \end{cases} & \text{else} \end{cases}$$

Let's write some pseudocode to calculate this value, bottom-up:

```

let A be a new  $(n + 1) \times (k + 1)$  array
for j in range(k + 1):
    A[0, j] = 0
for i in range(n + 1):
    for j in range(k + 1):
        A[i, j] = p[i]

        if j > 0:
            for l in range(1, k + 1):
                A[i, j] = max(A[i, j], p[l] + A[i - l, j - 1])
// At this point, A[n, k] holds the optimal value

i = n
j = k
output = []
while i > 0:
    if A[i, j] == p[i]:
        add i to output
        i = 0
    else:
        l = 1
        while A[i, j] != p[l] + A[i - l, j - 1]:
            l++

        add l to output
        i = i - l
        j -= 1
    // note that there must be some valid value
    // of l by construction of our algorithm
return output

```

Note that the runtime of the above code is $\Theta(n^2k)$ for the construction of A : the table is of size $\Theta(nk)$ and the entries on row i takes $\Theta(i)$ time to compute (due to the for loop), making the total time $\sum_{i=0}^n \sum_{j=0}^k \Theta(i) = (k+1) \sum_{i=0}^n \Theta(i) = (k+1)\Theta(n^2)$. The trace-back of the output takes $\Omega(1)$ and $O(n^2)$ time, giving us $\Theta(n^2k)$ time overall.

General problem solutions for the big list coming next week