CSCI-GA.1170-001/002 Fundamental Algorithms	November 19, 2015

Solutions to Problem 3 of Homework 8 (8 (+7) Points)

Name: GOWTHAM GOLI (N17656180) *Due: Tuesday, November 17*

Imagine a unary alphabet with a single letter x. A (valid) *bracketing* B is a string over three symbols $x, (,)$ defined recursively as follows: (1) a single letter x is a bracketing, and (2) for any $k \geq 2$, if $B_1, \ldots, B_k$ are (valid) bracketings, then so is $B = (B_1 B_2 \ldots B_k)$. A bracketing B is called *binary* if rule (2) can only be applied with $k = 2$. Then the length n of B is the number of x's it has (i.e., one ignores the parenthesis).

For example, there are 11 possible bracketings of length $n = 4$: $(xxxx)$, $((xx)xx)$, $((xxx)x)$, $(x(xxx))$, $(x(xx)x)$, $(xx(xx))$, $((xx)(xx))$, $(x(x(xx)))$, $((x(xx))x)$, $(x((xx)x))$, $(((xx)x)x)$, of which *only the last five* are binary.

(a) (4 points) Let $b(n)$ denote the number of binary bracketings of length n. Show that $b(n)$ is given by the following recurrence:

$$b(n) = \sum_{i=1}^{n-1} b(i)b(n-i) \ .$$

Solution:

Divide the given string into two partitions. The length of first partition being i and the second partition being $n - i$. Clearly i can vary from 1 to $n - i$. Therefore the number of binary bracketings in the first partition is $b(i)$ and the second partition is $b(n - i)$

If the length is 1 then there is only one valid bracketing possible i.e x. Therefore $b(1) = 1$

Let α be one of binary bracketing in the first partition out of the $b(i)$ possibilities. We can choose any of the $b(n - i)$ bracketings from the second partition along with α to get a binary bracketing of length n. Therefore using this specific partition, the number of binary bracketings of length n will be $b(i)b(n - i)$. Also i can vary from 1 to $n - 1$. Thus $b(n)$ evaluates to

$$b(n) = \sum_{i=1}^{n-1} b(i)b(n-i)$$

□

(b) (4 points) Use the result from part (a) to give a dynamic programming algorithm to compute $b(n)$ given n as input. What is the running time of your algorithm? Assume that multiplication of two integers takes time $O(1)$.

Solution:

```
Algorithm: b(n)
b[1] ← 1
for i ← 2 to n do
    for j ← 1 to i-1 do
        b[i] ← b[j]b[i − j]
    end
end
Return b[n]
```

Algorithm 2: Dynamic Programming Algorithm to calculate the number of binary bracketings in $O(n^2)$ time

From the above two *for* loops, it is clear that the above algorithm takes $1 + 2 + \ldots + n - 1 = O(n^2)$ time. $\square$

(c) (7 points (**Extra credit**)) Generalize part (a) and (b) by giving a similar recurrence(with proof) as part (a) to find the total number $f(n)$ of bracketings of length n, and then give a dynamic programming algorithm to compute $f(n)$ and analyze its running time.

Solution:

Given a string, $(x \ldots x)$ of length n

- If the length of the string is 1, then there is only one possible bracketing (x). Therefore $f(1) = 1$
- Otherwise split the string into some p number of partitions such that $(x \ldots x) = (\underbrace{(x \ldots x)}_{\text{i}}\underbrace{(x \ldots x)}_{\text{j}}\underbrace{(x \ldots x)}_{\text{k}} \ldots\ldots \underbrace{(x \ldots x)}_{\text{p}})$ where $(1 \le i \le n), (1 \le j \le n-i), (1 \le k \le n-i-j), \ldots, (1 \le p \le n-i-j-\ldots-p-1)$ then the number of bracketings in this case will be

$$f(n) = \sum_{i=1}^{n} f(i) \sum_{j=1}^{n-1} f(j) \sum_{k=1}^{n-i-j} f(k) \ldots\ldots$$

$\square$