CS173 Lecture B, October 20, 2015

Tandy Warnow

October 18, 2015

CS 173, Lecture B
October 20, 2015
Tandy Warnow

Preparation for Examlet #5

The examlet has four problems:

1. (14 pts) One problem with three parts:
   - You give the formal definition (in terms of constants C_1 and C_2) for big-oh
   - You say (without written justification) whether a function is big-oh of another.
   - You differentiate a function of the form $g(n)^{h(n)}$
2. (10 pts) One problem where you prove a function f is big-oh of another function g by finding the constants C_1 and C_2
3. (10 pts) One counting problem
4. (12 pts) One dynamic programming algorithm (that you come up with, but do not need to prove correct)

Review of counting

$C(n, k) = \binom{n}{k}$

- is the number of subsets of size k of an n-element set
- order does not matter
- no repetitions allowed (sets have each element at most once)

The formula is

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Review of counting

$P(n, k)$

- is the number of ways you can select k elements out of an n-element set, where the order matters
- no repetitions allowed
- $P(n, k) = k!\binom{n}{k}$

So the formula is

$$P(n, k) = \frac{n!}{(n-k)!}$$

Techniques

Remember the formula for $\binom{n}{k}$ and then the basic techniques:

- Algorithmic generation (count the leaves of the decision tree) and make sure each item is counted only once
- Algorithmic generation with correction for over-counting
- Count the complement
- Do a case analysis

Review of counting

- $C(n,k) = \binom{n}{k} = \frac{n!}{(n-k)!k!}$
- $P(n,k) = \frac{n!}{(n-k)!}$
- $|\mathbb{P}(X)| = 2^{|X|}$
- Number of functions from X to Y is $|Y|^{|X|}$

Review of dynamic programming

Dynamic programming (DP) is an algorithm design technique best seen as a "bottom-up" algorithm. You will be given a problem and asked to describe a DP algorithm for it.

- State the meaning of the variables you compute
- State the boundary cases (i.e., how to set the variable when it cannot be set recursively)
- State the recursive formula (i.e., how to set the variable using the previous values)
- State the order in which you will compute the values for the variables
- State where the answer will be stored
- Figure out the big-oh running time (and explain why)

Example of DP algorithm

- Input: positive integer n
- Output: TRUE or FALSE based on whether you can make change for n cents using some non-negative number of 3-cent and 5-cent coins.

DP Algorithm:

- Array $M[1...n]$ where $M[i]$ is TRUE if and only if you can make change for i cents using 3-cent and 5-cent coins.
- The answer will be in $M[n]$.
- How do we fill in the array?

Computing $M[1...n]$: Do the recursion first

We have 3-cent and 5-cent coins.
The recursion is simple: $M[i] = M[i-5] \vee M[i-3]$
Note that this recursion cannot be applied to any number i that is not at least 6 (since otherwise $M[i-5]$ is not defined).
Therefore our base cases are $M[1...5]$, set as follows:

- $M[1] := F$
- $M[2] := F$
- $M[3] := T$
- $M[4] := F$
- $M[5] := T$

Computing $M[1...n]$: Figuring out the order of the subproblems

So we have the recursion and the base cases. We now need to set the order in which we fill in the matrix.
Easy!
We compute the matrix values from left-to-right, i.e.,

For $i = 1$ upto n DO
    Compute $M[i]$.

Computing $M[1...n]$: putting it all together

$M[1] := F$

$M[2] := F$

$M[3] := T$

$M[4] := F$

$M[5] := T$

For $i = 6$ upto n DO

$M[i] = M[i-5] \vee M[i-3]$

Return $M[n]$

Running time: easily seen to be constant time to compute $M[i]$ after $M[1...i-1]$ filled in. Hence, $O(n)$.