CS4800: Algorithms & Data
Jonathan Ullman

Lecture 10:

- Dynamic Programming Wrap-up
- Midterm Review

Feb 9, 2018

Notes for Hackerrank Problems

- More public test cases
- List of common issues

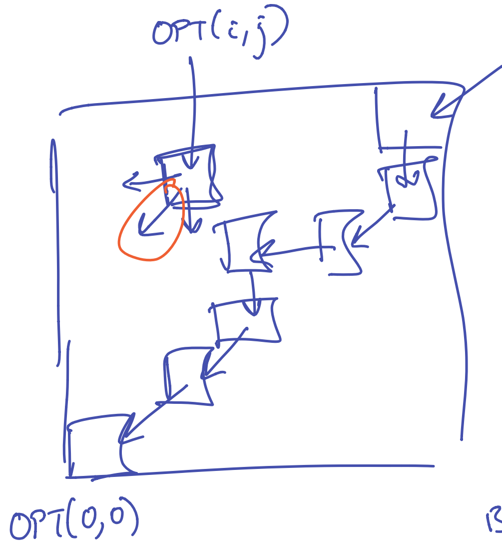
Midterm I Review

Edit Distance / Optimal Alignment

x: peas

y: east

p	e	a	s	—
—	e	a	s	t



OPT(i,j) is the cost of aligning $x_1 \ldots x_i$ w/ $y_1 \ldots y_j$

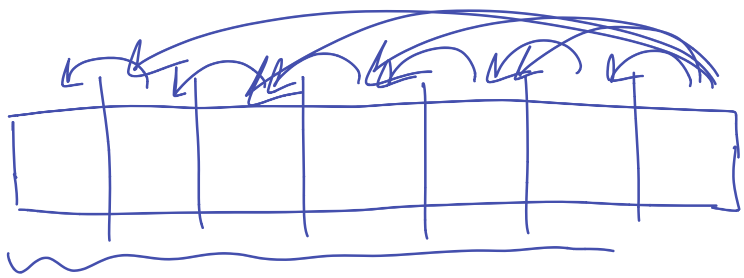
- Each alignment is $n+m$ letters

$n = m$ $\qquad n \cdot n \cdot (2n) = 2n^3$

Analyzing Runtime of DP Algs

- (Optional: Sorting the input) $O(n \log n)$
  - You can use Mergesort w/o proof
  - Feel free to "relabel" inputs into sorted order

- Filling the dynamic programming table
  - $\leq$ (# of subproblems) $\times$ (max time to fill each subproblem)



$$(n-1) \cdot O(1) + O(n) = O(n)$$

- Backtrack through the table to find the solution
  - Usually is dominated by the time to fill the table.

Running Time Analysis of Memoization

(Top-Down)

- Let M be a table of solutions to subproblems
- Initialize M to "empty"

Alg(i):
    If M[i] not empty : return M[i]
    Else:
        M[i] ← [something recursive]   ≤ k calls [depends on problem]
        return M[i]

Every time we fill one entry of M[i] we make ≤ k recursive calls.   Total # of calls ≤ k · (# of entries of M)

Running time: (# of calls) + (# of entries of M) · (time to fill one entry)

Topics: Induction

- Proof by Induction:
  - Mathematical formulae, e.g. $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$
  - Spot the bug
  - Solutions to recurrences
  - Correctness of divide-and-conquer algorithms

- Good way to study:
  - Lehman-Leighton-Meyer, *Mathematics for CS*
  - Review D&C algorithms in Kleinberg-Tardos
  - HW2.3

Example Question: Induction

Question: Consider the recurrence $T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{3n}{4}\right) + 2n$

Use the ~~substitution method~~ to show that $T(n)=\Omega(n \log n)$

Solution: We will prove $T(n) \geq cn \log n$ for some *c*. We first prove the induction step.

$$T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{3n}{4}\right) + 2n \geq \frac{cn}{4} \log \frac{n}{4} + \frac{3cn}{4} \log \left(\frac{3n}{4}\right) + 2n$$

After some math (omitted), we get that $T(n) \geq cn \log n$ is true whenever $c \leq \frac{2}{2-\left(\frac{3}{4}\right) \log 3}$. In particular, we can set c=1

Topics: Asymptotics

- Asymptotic Notation
  - $o, O, \omega, \Omega, \Theta$
  - Relationships between common function types

- Good way to study:
  - Kleinberg-Tardos Chapter 2

Topics: Asymptotics

Notation	… means …	Think…	E.g.
$f(n)=O(n)$	$\exists c > 0, n_0 > 0, \forall n \geq n_0$: $0 \leq f(n) \leq cg(n)$	Upper bound "$\leq$"	$100n^2 = O(n^3)$
$f(n)=\Omega(g(n))$	$\exists c > 0, n_0 > 0, \forall n \geq n_0$: $0 \leq cg(n) \leq f(n)$	Lower bound "$\geq$"	$2^n = \Omega(n^{100})$
$f(n)=\Theta(g(n))$	$f(n) \in O\big(g(n)\big)$ **and** $f(n) \in \Omega(g(n))$	Tight bound "$=$"	$\log(n!) = \Theta(n \log n)$
$f(n)=o(g(n))$	$\forall c > 0, \exists n_0 > 0, \forall n \geq n_0$: $0 \leq f(n) < cg(n)$	"$<$"	$n^2 = o(2^n)$
$f(n)=\omega(g(n))$	$\forall c > 0, \exists n_0 > 0, \forall n \geq n_0$: $0 \leq cg(n) < f(n)$	"$>$"	$n^2 = \omega(\log n)$

Topics: Asymptotics

For all real $a > 0$, m, and n, we have the following identities:

$$\begin{aligned} a^0 &= 1 , \\ a^1 &= a , \\ a^{-1} &= 1/a , \\ (a^m)^n &= a^{mn} , \\ (a^m)^n &= (a^n)^m , \\ a^m a^n &= a^{m+n} . \end{aligned}$$

For all real $a > 0$, $b > 0$, $c > 0$, and n,

$$\begin{aligned} a &= b^{\log_b a} , \\ \log_c(ab) &= \log_c a + \log_c b , \\ \log_b a^n &= n \log_b a , \\ \log_b a &= \frac{\log_c a}{\log_c b} , \\ \log_b(1/a) &= -\log_b a , \\ \log_b a &= \frac{1}{\log_a b} , \\ a^{\log_b c} &= c^{\log_b a} , \end{aligned}$$

where, in each equation above, logarithm bases are not 1.

Topics: Asymptotics

- **Polynomials.** $a_0 + a_1 n + \ldots + a_d \boxed{n^d}$ is $\Theta(n^d)$ if $a_d > 0$.

  $\boxed{n^3} + \cancel{100n^2 + 210n}$

- **Logarithms.** $\log_a n = \Theta(\log_b n)$ for all constants $a, b > 0$.

  can avoid specifying the base

  log grows slower than every polynomial

  For every $x > 0$, $\log n = O(n^x)$.

- **Exponentials.** For all $r > 1$ and all $d > 0$, $n^d = O(r^n)$.

  Every exponential grows faster than every polynomial

- **Factorial.**

  $\log(n!)$

  $\cancel{\log(n)!} = \Theta(n \log n)$

  factorial grows faster than every exponential

Example Question: Asymptotics

$$2n^2 = o(n^3)$$

Question: Give a value of n_0 which proves the above statement.

Solution: Recall definition $\approx \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$

$o(g(n)) = \{ f(n) :$ for any constant $c > 0$, there is a constant $n_0 > 0$ such that $0 \le f(n) < cg(n)$ for all $n \ge n_0 \}$

Apply definition: $0 \le 2n^2 < cn^3$

Solve for n: $n > \frac{2}{c}$ $\quad \forall c$ if $n \geq \frac{2}{c}+1$, then $2n^2 < c \cdot n^3$

Let $n_0 = \frac{2}{c} + 1$

Example Question: Asymptotics

True or False?

1) $n^2 – 5n – 100 = O(n)$
2) $n^3 + 10n^2 + 125 = \omega(n)$
3) $n^2 + O(n) = O(n^2)$
4) $2^{n+1} = O(2^n)$
5) $2^{5n} = O(2^n)$
6) $\log(n^2) = O(\log(n))$

Topics: Recurrences

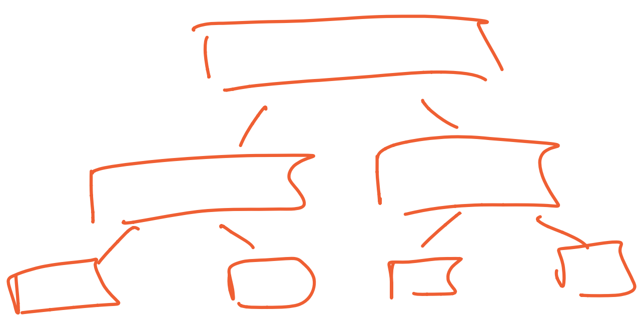
- Recurrences
  - Representing running time by a recurrence
  - Solving common recurrences
  - Master Theorem

e.g. My alg makes 2 recursive calls on inputs of size $n/3$ and does n^2 work

$$T(n) = 2T\left(\frac{n}{3}\right) + n^2$$

- Good way to study:
  - Erickson book
  - Kleinberg-Tardos D&C Chapter

- Master Theorem
- Recursion Tree



Example Question: Recurrences

Consider the recurrence $T(n) = 5T(n/3) + n$

a) Draw the recursion tree (at least two levels)
b) What is the depth of the tree?
c) What is the amount of work done at level i?
d) What is the total amount of work done?

Know your geometric series

$$\sum_{i=0}^{n} x^i = \frac{x^{n+1} - 1}{x - 1} = \frac{1 - x^{n+1}}{1 - x}$$

Topics: Divide-and-Conquer

- Divide-and-Conquer
  - Writing pseudocode
  - Proving correctness by induction
  - Analyzing running time via recurrences

- Good way to study:
  - Example problems from Kleinberg-Tardos or Erickson
  - HW 2.2, 2.3, 3.1
  - Practice, practice, practice!

Good Review Problems for D+C:

① There are n items. In $O(1)$ time you check if item i equals item j. Suppose $> \frac{n}{2}$ of the items are the same. Design an $O(n \log n)$ D+C algorithm to ind one such item.

- $O(n \log n)$ algorithm to decide if $> \frac{n}{2}$ items are the same.

② n items, each item is either good or bad. There is a function HELP(i,j) s.t.

HELP(i,j) returns either {same, different}

i	j	HELP(i,j)
good	good	same
good	bad	different
bad	good	different
bad	bad	same

HELP$(i,j) = O(1)$ time

Problem: Suppose $> \frac{n}{2}$ items are good. Find all the good items in $O(n)$.

Topics: Dynamic Programming

- Dynamic Programming
  - Identify sub-problems
  - Write a recurrence, $OPT(n) = \max\{v_n + OPT(n-6), OPT(n-1)\}$
  - Fill the dynamic programming table
  - Find the optimal solution
  - Analyze running time

- Good way to study:
  - Example problems from Kleinberg-Tardos or Erickson
  - HW 4.1, 4.2 (solutions posted Saturday morning)
  - Practice, practice, practice!