Basic Algorithms (V22.0310); Fall 2005; Yap

STUDY QUESTION (with SOLUTION)

Nov 30, 2005

Instructions

- Please study these questions carefully.
- They are designed to help you prepare for Quiz 2.
- They will form the basis for some questions in Quiz 2 (but I might ask some unrelated questions as well).
- Quiz 2 will be based on Chapter 5 (Divide-and-Conquer) and Chapter 6 (Dynamic Programming).

Question 1

In a recurrence for a function $T(n)$, I want you to think of n as a real number (not only as natural number). I also allow you to choose ANY initial conditions you like. E.g. you may choose $T(n) = 0$ for all $n < 2$. Of course you should choose them so that your final result is as simple as possible.

Solve the following recurrences by Rote Method. Be sure to clearly indicate the 4 steps (Expand, Guess, Verify, Stop).

- (i) $T(n) = 7T(n/7) + n^2$.
- (ii) $T(n) = 100T(n/10) + n$.

Furthermore, please state the solution to (i) and (ii) that the Master Theorem tells you. [This should confirm what your Rote Method tells you.]

SOLUTION:

(i)

$$
\begin{array}{rcll}
T(n) & = & 7T(n/7) + n^2 & (1st\ expansion) \\
 & = & 7[7T(n/49) + (n^2/49)] + n^2 & (2nd\ expansion) \\
 & = & 7^2T(n/7^2) + n^2/7 + n^2 & \\
 & = & 7^2[7T(n/7^3) + (n^2/7^4)] + n^2/7 + n^2 & (3rd\ expansion) \\
 & = & 7^3T(n/7^3) + n^2/7^2 + n^2/7 + n^2 & \\
 & = & \vdots & \\
 & = & 7^iT(n/7^i) + \sum_{j=0}^{i-1} n^j/7^j & (GUESS : ith\ expansion) \\
 & = & 7^i[7T(n/7^{i+1}) + (n^2/7^{2i})] + \sum_{j=0}^{i-1} n^j/7^j & (i+1st\ expansion) \\
 & = & 7^{i+1}T(n/7^{i+1}) + \sum_{j=0}^{i} n^j/7^j & (VERIFIED) \\
 & = & 7^kT(n/7^k) + \sum_{j=0}^{k-1} n^j/7^j & (STOP :\ choose\ k = \lceil \lg n \rceil) \\
 & = & n^2 \sum_{j=0}^{k-1} (1/7^j) & (T(n/7^k) = 0,\ assuming\ T(x) = 0\ for\ x < 1) \\
 & = & \Theta(n^2) & (since\ \sum_{j=0}^{k-1} (1/7^j) = \Theta(1))
\end{array}
$$

The last step uses the geometric series,

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

which is valid for any $|x| < 1$. If we choose $x = 1/7$, then we see that

$$1 \leq \sum_{j=0}^{k-1}(1/7^j) < \sum_{j=0}^{\infty}(1/7^j) = \frac{1}{1-1/7} = 7/6).$$

In other words, $\sum_{j=0}^{k-1}(1/7^j) = \Theta(1)$.

ADDITIONAL REMARKS: The geometric series is basically the ONLY series you need for simple analysis of algorithms. I call it "mother of series" because most simple infinite series can be derived from this one. It should be committed to memory, and the best way to do this is to remember it's simple proof: just verify by direct multiplication!

$$\begin{aligned}(1-x)\sum_{j=0}^{\infty} x^j &= (1-x)(1+x+x^2+x^3+\cdots) \\ &= 1+(x-x)+(x^2-x^2)+(x^3-x^3)+\cdots \\ &= 1+0+0+0+\cdots \\ &= 1.\end{aligned}$$

(ii)

$$\begin{aligned}
T(n) &= 100T(n/10)+n && (1st\ expansion)\\
&= 100[100T(n/10^2)+(n/10)]+n && (2nd\ expansion)\\
&= 100^2T(n/10^2)+10n+n\\
&= 100^2[100T(n/10^3)+(n/10^2)]+10n+n && (3rd\ expansion)\\
&= 100^3T(n/10^3)+100n+10n+n\\
&= \vdots\\
&= 100^iT(n/10^i)+\textstyle\sum_{j=0}^{i-1}10^jn && (GUESS: ith\ expansion)\\
&= 100^i[100T(n/10^{i+1})+n/10^i]+\textstyle\sum_{j=0}^{i-1}10^jn && (i+1st\ expansion)\\
&= 100^{i+1}T(n/10^{i+1})+\textstyle\sum_{j=0}^{i}10^jn && (VERIFIED)\\
&= 100^kT(n/10^k)+\textstyle\sum_{j=0}^{k-1}10^jn && (STOP:\ choose\ k=\lceil\log_{10}n\rceil)\\
&= n\textstyle\sum_{j=0}^{k-1}10^j) && (T(n/10^k)=0,\ assuming\ T(x)=0\ for\ x<1)\\
&= \Theta(n^2) && (since\ \textstyle\sum_{j=0}^{k-1}10^j=\Theta(n))
\end{aligned}$$

Again, the last step requires knowing how to sum a geometric series:

$$S_k = \sum_{i=0}^{k-1} x^i = \frac{x^k-1}{x-1} \qquad (1)$$

where $x > 1$. Using $x = 10$, we get $S_k = (10^k - 1)/(10 - 1) < 10^k < 10n$ where $k = \lceil \log_1 0n \rceil$. Of course $S_k > 10^k \geq n$. Hence $S = \Theta(n)$.

ADDITIONAL REMARKS: In this case, since $x > 1$, we cannot replace the FINITE geometric series by the INFINITE (mother) series of the part(i). The finite geometric series is somewhat more complicated than the infinite (mother) geometric series above. Nevertheless, you could derive this from the mother series. If S_∞ is the infinite series, then you notice that

$$S_k = S_\infty - x^k S_\infty.$$

Now you can plug in $S_\infty = 1/(1 - x)$ and solve for S_k to get the formula (**??**). Again, we strongly recommend committing to memory (**??**) (perhaps through this proof).

Question 2

Real Induction is a method to confirm a guessed bound on a recurrence function. Consider $T(n)$ in part (ii) of Question 1. Suppose that you guessed that $T(n) = \Theta(n^2)$. You would then reduce this to two separate statements to prove:

- (a) $T(n) \geq K_1 n^2$, for some $K_1 > 0$
- (b) $T(n) \leq K_2 n^2$, for some $K_2 > 0$

Please prove (a) and (b). BUT you may have to modify the form of your initial guess in (a) or (b) in order to carry out your induction.

If you prefer (or in hindsight), you may be able to directly choose specific constants (e.g., pick $K_1 = 7$, or whatever works).

SOLUTION:

(a) In induction, you may assume that the statement $P(n)$ that you want to prove is true for smaller values of n. In the present case, the statement $P(n)$ is "$T(n) \geq K_1 n^2$" for some $K_1 > 0$.

Now, it is clear that you have no hope of reaching the desired conclusion in this way. As hinted, you need to modify the guess. The trick is to add a lower order term (so it does not affect our main conclusion that $T(n) = O(n^2)$). Let us try to prove the modified statement $P(n) \equiv T(n) \leq K_1 n^2 - Cn$ (for some C to be specified):

$$\begin{array}{rcll} T(n) & = & 100T(n/10) + n & \\ & \leq & 100[K_1(n/10)^2 - Cn] + n & (\textit{by induction hypothesis}) \\ & = & K_1 n^2 - 100Cn + n & \\ & = & K_1 n^2 - 100n(C - 1/100) & \\ & \leq & K_1 n^2 - Cn & (\textit{provided } C \geq 101/100). \end{array}$$

Thus we have shown what we need: we must choose K_1 and C so that they satisfy the initial conditions (but we normally do not specify these).

(b) This is simpler, as there is no need to modify our original hypothesis.

Question 3

Consider the farthest pair problem: given a set $S = \{p_1, \ldots, p_n\}$ of points, compute the maximum distance $d(p_i, p_j)$ between any pair of points.
(a) Does the ideas of the closest pair problem apply to this setting?
(b) Give a simple algorithm to solve this problem. You must write a program that is very close to a proper Java Program. My reason for this is not to test your Java ability, but to force you to reduce all operations to simple primitives.
(c) Analyze its running time.

SOLUTION:

(a) Apparently not. For if you recursively split the set S into two halfs S_0, S_1 by a vertical L, and discovered that the maximum pairwise distance on each side is δ, you cannot conclude that the pairs from $S_0 \times S_1$ to be searched must lie OUTSIDE the strip of width δ centered about L. Why? It seems that you still have to check $\Omega(n^2)$ pairs. Since the original problem could be solved in $O(n^2)$ time, it seems that you have not really reduced the problem.

(b) Assume that $dist(p, q)$ computes the distance between two points p and q. This is easily implemented and takes $O(1)$ time.

```
FARTHEST PAIR(S)
    INPUT: S = S[0..n-1] is an array of points
    OUTPUT: D, the distance between the farthest pair
    D = 0
    for (i = 0; i < n-1; i++)
        for (j = i+1; i < n-1; j++)
            D = max(D, dist(S[i], S[j]))
    Return(D)
```

(c) The double for-loop takes $\Theta(n^2)$ time.

Question 4

You need to be able to do hand simulations of Dynamic Programming problems. You need to set up the appropriate matrices and fill in the entries by hand.

I typed "ocruent" to Google and it asked if I meant "current". Compute the alignment distance between the two strings. Assume that the gap parameter is $\delta = 3$ and the cost of a mismatch is given by

$$\alpha_{x,y} = \begin{cases} 0 & \text{if } x = y, \\ 1 & \text{if } x, y \text{ are both consonants,} \\ 1 & \text{if } x, y \text{ are both vowels,} \\ 2 & \text{else.} \end{cases}$$

SOLUTION:

Let X, Y be the two strings and $OPT(i, j)$ denote minimum cost to align X_i and Y_j where X_i is the prefix of X of length i. Assume $|X| = m, |Y| = n$. Then $OPT(i, j) = 3(i + j)$ if $ij = 0$. Otherwise,

$$OPT(i,j) = \min\{\alpha(x_i, y_j)+OPT(X_{i-1}, Y_{j-1}), \quad 3+OPT(X_{i-1}, Y_j), \quad 3+OPT(X_i, Y_{j-1})\}.$$

We now use this rule to fill in the following array:

	ϵ	c	u	r	r	e	n	t
ϵ	0	3	6	9	12	15	18	21
o	3	2	4	7	10	13	16	19
c	6	3	4	5	8	11	14	17
r	9	6	5	4	5	8	11	14
u	12	9	6	7	6	6	9	12
e	15	12	9	8	9	6	8	11
n	18	15	12	11	10	9	6	9
t	21	18	15	14	13	12	9	6

So the alignment cost for the 2 strings is 6.

Food for Thought: In filling in this table for some other X and Y, suppose you see two consecutive entries in a row with values 123 and 119. Is something wrong?

Question 5

Longest path in an ordered graph problem (p.314 of text).

SOLUTION:

(a) You can get a counter example from Figure 6.29 by deleting one of the edges, and modifying another one.

(b) Let G_i be the subgraph of the input graph induced by the first i vertices. So G_n is the original graph. If $OPT(i)$ is the length of the longest path in G_i, then $OPT(i) = 1 + \max\{OPT(j) : (v_j, v_n) \in E\}$. What is the running time for this solution?

Question 6

Not all inversions are equal! The book discussed the problem of counting the number of inversions in a permutation. E.g., The sequence $S_0 = (1, 2, 3, 4)$ has no inversions, but sequence $S_1 = (2, 1, 4, 3)$ has two inversions, namely the pairs $\{1, 2\}$ and $\{3, 4\}$. Now, the sequence $S_2 = (2, 3, 1, 4)$ also has two inversions, namely the pairs $\{1, 2\}$ and $\{1, 3\}$.

I propose to distinguish between the quality of the inversions of S_1 and S_2. The inversions $\{1, 2\}$ and $\{3, 4\}$ in S_1 are said to have weight of 1 each, so the **weighted inversion** of S_1 is $W(S_1) = 2 = 1+1$. But for S_2, the inversion $\{1, 2\}$ has weight 2 while inversion $\{1, 3\}$ has weight 1. So the weighted inversion is

$W(S_2) = 3 = 2+1$. Thus the "weight" measures how far apart the two numbers are.

In general, let $W(S)$ denote the weighted inversion of an arbitrary sequence $S = (a_1, \ldots, a_n)$. A pair $\{a_i, a_j\}$ is an **inversion** if $i < j$ and $a_i > a_j$. The weight of this inversion is $j - i$. Hence $W(S)$ is the sum over the weights of all inversions.

- (a) Design an algorithm to compute $W(S)$. I.e., describe the ideas leading to your algorithm and explain any data structures, etc.
- (b) Write Java-like code to implement (a).
- (c) Analyze its running time.

SOLUTION:

(a) As before, we do divide and conquer. We modify the Merge-and-Count(A, B) subroutine in Chapter 5 (p.224): Assume that $A[1..n], B[1..n]$ are arrays in sorted (non-decreasing) order: $A[i] \leq A[i+1]$ and $B[i] \leq B[i+1]$.

We also maintain arrays $APOS[1..n]$ and $BPOS[1..n]$ telling us the original positions of the elements in A and B. I.e., $A[i]$ was originally the $APOS[i]$-th element in A.

Merge-and-Count(A, B), we again have a counter C initialized to zero. Firse, we compute the cumulative distance D of all the elements in A from its final position n. More precisely, let

$$D = \sum_{k=1}^{n} (n - APOS[k]).$$

Now, whenever we output the ith element of A, we decrement D using the rule

$$D- = (n - APOS[i]).$$

Whenever we output the jth element of B, we increment C using the rule

$$C+ = D + (n - i + 1) \times BPOS[j].$$

It is easy to see that C is keeping track of the total weight of inversions.

(b) Keep the Sort-and-Count algorithm in page 225. We just have to re-implement Merge-and-Count as described in (a).

(c) COMPLEXITY: The Merge-and-Count routine is $O(n)$ as before. Hence the recurrence is $T(n) = 2T(n/2) + n$ with solution $\Theta(n \log n)$.