CSE 101 Class Notes

The Master Theorem, and how (not) to use it

October 25, 2006

Master Theorem (from last time)

For a recurrence
$$T(n) = aT(\frac{n}{b}) + \Theta(n^k)$$
with $b > 1, k \geq 0$, the solution will fall into one of three classes:

1. **Top-heavy:** If the algorithm is top-heavy (i.e. $a/b^k < 1$), then the top line of the table dominates, and $T(n) = O(n^k)$.

2. **Balanced:** If the algorithm is balanced (i.e. $a/b^k = 1$), then the total time is the sum of the times at each level, or $T(n) = \log_b n O(n^k) = O(n^k \log_b n)$

3. **Bottom-heavy:** f the algorithm is bottom-heavy (i.e. $a/b^k > 1$), then the table's height (hence the number of base cases) dominates and $T(n) = O(n^{\log_b a})$.

Example: Binary Search

```
1     Search(v, A[1..n])
2         if n = 0
3             return false
4         else if n = 1
5             return v = A[1]
6         else if v = A[n/2]
7             return true
8         else if v < A[n/2]
9             return Search(v, A[1 .. n/2-1]
10        else
11            return Search(v, A[n/2+1 .. n]
```

Applying the MT, $a = 1$ (one of lines 7 and 9 is executed), $b = 2$, and $k = 0$, so $T(n) = O(n^0 \log n) = O(\log n)$.

Example: Tree Size

```
1     Size : array of sizes
2     Tree_size(t)
3         if t = NIL
4             return 0
5         sl <- size(left(t))
6         sr <- size(right(t))
7         Size[r] <- sl + sr + 1
8         return Size[r]
```

Applying the MT again, we might say that lines 5-6 are called on problems of size $n/2$, so $a = 2, b = 2, k = 0$, we are in case (3), and $T(n) = O(n^{\log 2}) = O(n)$.

But we are cheating here in assuming that the tree is balanced – what if the tree were perfectly unbalanced (i.e. a linked list)? Then $T(n) = T(n-1)+O(1)$. We can't apply the MT here, since $n-1$ is not related to n by any multiplicative factor $1/b$. In this case, we can apply the "stare at it" method to conclude that the algorithm is $\sum_{i=1}^{n} O(1) = O(n)$.

We still need to show that the algorithm is $O(n)$ for *every* tree, not just completely balanced or unbalanced ones. To do so, note that

$$\begin{aligned} T(|x|) &= T(|\text{left}(x)|) + T(|\text{right}(x)|) + O(1) \\ &= T(|\text{left}(x)|) + T(|x| - |\text{left}(x)| - 1) + c \end{aligned}$$

We can now prove by strong induction that $T(n) = O(n)$.

Base Case: $T(0) \leq c'0 + d$.

Inductive Case: Assume that $\forall i < n, T(i) \leq c'i + d$. Then

$$\begin{aligned} T(n) &= T(\text{left}(n)) + T(n - \text{left}(n) - 1) + c \\ &\leq c'\text{left}(n) + d + c'(n - \text{left}(n) - 1) + d + c \\ &= c'n + 2d + c - c' \leq c'n + d \end{aligned}$$

where in the last step we assume that $c' \geq d + c$. Therefore $T(n) = O(n)$.

Example: Tree Isomorphism

Given two trees, we want to find if they are identical up to swapped children.

```
Same_tree(a, b)
    if a = NIL || b = NIL
        return a = b
    else
        return (Same_tree(left(a), left(b)) &
                Same_tree(right(a), right(b)))
        ||     (Same_tree(left(a), right(b)) &
                Same_tree(right(a), left(b)))
```

In the worst case, we have to perform all 4 recursive calls, and

$$T(n_1, n_2) = T(L_1, L_2) + T(R_1, R_2) + T(L_1, R_2) + T(R_1, L_2) + O(1)$$

However, two trees can only be isomorphic if they have equal numbers of nodes, and we can check this by precomputing the number of nodes in each subtree of each tree using `Tree_size` above; this takes $O(n_1 + n_2)$ time. By checking for equal size at line 5 before performing the recursive calls, we can avoid half the recursive calls when a node is unbalanced. This lets us divide the time analysis into two cases:

1. $L_1 = R_1$**:** $T(n) = 2T(L) + 2T(R) = 4T((n-1)/2)$. Applying the MT, $a = 4, b = 2, k = 0$, and $T(n) = O(n^2)$.
2. $L_1 \neq R_1$**:** $T(n) = T(L) + T(R) + O(1)$. We expect this case to be faster since we only perform two recursive calls. If we only executed this second case, then the running time would be linear (see the proof for `Tree_size`).

To show that `Same_tree` is $O(n^2)$, we need to show that any mixture of cases (1) and (2) is $O(n^2)$, i.e. that $T(n) \leq cn^2 + d$ for some constants c and d. Assume that $\forall i < n, T(i) \leq ci^2 + d$. Then

1. $L_1 = R_1$**:**

$$\begin{aligned} T(n) &= 4T(\frac{n-1}{2}) \\ &\leq 4c\left(\frac{n-1}{2}\right)^2 + 2d + c' \\ &\leq c\,(n)^2 - c + 2d + c' \\ &\leq cn^2 + d \end{aligned}$$

2. $L_1 \neq R_1$**:**

$$\begin{aligned} T(n) &= T(L) + T(R) + c' \\ &\leq cL^2 + cR^2 + d + c' \\ &\leq cn^2 - c + 2d \\ &\leq cn^2 + d \end{aligned}$$