CS344: MidTerm 1 (Sample) Solutions

October 26, 2017

1. *(Sorting)*[**10 Pts**] Someone claims she can sort n numbers by dividing into $\sqrt{n}$ subproblems of size $\sqrt{n}$ each, solving each subproblem recursively, and then combining those solutions. How much time is she allowed to use to compose the solutions of these subproblems into the solution for the overall problem, if the entire algorithm takes $O(n \log n)$ time?

   Say $T(n)$ is the worst case running time of her algorithm. We have $T(n) = O(n \log n)$. Further, from the way she solves the problem, we have $T(n) = \sqrt{n}T(\sqrt{n}) + f(n)$, where $f(n)$ is the worst case time to compose the solutions. The problem is to figure out $f(n)$.

   Observe that $f(n) = O(n^2)$ will not work.

   For $f(n) = O(n)$, we have $T(n) = \sqrt{n}T(\sqrt{n}) + O(n)$. Divide both sides by n and setting $H(n) = T(n)/n$, we have $H(n) = H(\sqrt{n}) + O(1)$. Say $n = 2^{2^k}$, we have $k = \log \log n$. Then, $2^{(2^k)/2} = 2^{2^{k-1}}$. Substituting $G(k) = H(2^{2^k})$, we have $G(k) = G(k-1) + O(1)$, solving it we have $G(k) = O(k)$, and hence, $T(n) = O(n \log \log n)$ which is certainly $T(n) = O(n \log n)$. Thus, $f(n) = O(n))$ is a correct answer. Many other answers may be possible including $O(n(\log n)/\log \log n)$ we discussed in the class.

   A better answer will be $f(n) = n \log n$, because then, using the tree method, we have work at the first level is $n \log n$, and the work at the next level is $\sqrt{n} \times \sqrt{n} \log(\sqrt{n}) = n \log(\sqrt{n}) = n(\log n)/2$, and work at the level below is $\sqrt{n} \times n^{1/4} \times n^{1/4} \log(n^{1/4}) = n \log(n^{1/4}) = n(\log n)/4$ and so on. Hence the total work will be $O(n \log n)$ which is $T(n)$.

   Observe that she can *not* have time $f(n) = \omega(n \log n)$.

2. *(Dynamic Programming)* [**10 pts**]

   A tree T consists of a root node and zero or more children, each of which is a tree. The *maximum independent set* of a tree T is a subset I of nodes such that no two of them are children of each other, and such that I is as large as possible. Use dynamic programming[1] solve the maximum independent set problem. What is the best running time you can get?

   We define $IS(v)$ to the size of the largest independent set of the subtree of T rooted at node v. Say the root of T is r and then $IS(r)$ will solve our problem.

   We focus on $IS(v)$.

   - Suppose v is not in the maximum independent set.

     Then the maximum independent set is simply the union of the maximum independent sets of the subtrees of the children of v. That is,

     $$IS(v) = \sum_{u | u \text{ is a child of } v} IS(u)$$

   - Suppose v is in the maximum independent set

[1]See some practice dynamic programming problems at `http://people.cs.clemson.edu/~bcdean/dp_practice/`.

Then the maximum independent set consists of v, plus the union of the maximum independent sets of the subtrees of the grandchildren of v (the children of v can not be in the independent set with v). That is,

$$IS(v) = 1 + \sum_{u|u \text{ is a grandchild of } v} IS(u)$$

Putting the two choices together, we have

$$IS(v) = \max\{ \sum_{u|u \text{ is a child of } v} IS(u), 1 + \sum_{u|u \text{ is a grandchild of } v} IS(u)$$

To complete the solution, we state that $IS(.)$ is computed bottom up in the rooted tree T so the $IS(.)$ we need on the right side is always available as we compute up the tree; we also state the solutions for the base case when the node v has no children or grandchildren.

The running time is $O(nd)$ where n is the number of nodes in T and d is the maximum number of grandchildren of a node in T.

For extra credit, analyze this algorithm more tightly?

3. *(Rapid Answers)* **[5 pts each]**

- $f(n) = n$ and $g(n) = 2^{\log_2 n}$. Is $f = o(g(n))$? True or False? Provide reasons.
  False. $f(n) = n = 2^{\log_2 n}$. Hence, $f(n) = \Theta(f(n))$.
- Given n items hashed into m buckets using universal hashing with chaining, what is the expected number of comparisons done in the case of an unsuccessful search?
  From lecture notes, book and HW.
- Given strings s and t of length n, we can say s is lexicographically smaller than t if $f(s) < f(t)$ where $f(x)$ is the Karp-Rabin fingerprint for any string x. True or False? Provide reasons. We say s is lexicographically smaller than t if s appears before t in the dictionary of strings.
  False. If s is lexicographically smaller than t, then $N(s) < N(t)$ where N is defined as a number with the maximum length of the two strings. However, if we applied $f(x) = N(x) \bmod q$ for suitable q, $f(s)$ may be larger, smaller or equal to $f(t)$.
- To solve string matching problem with pattern $p[1, m]$ and text $t[1, 2m]$, we used prime $q \in [1..m^5]$ to get probability of false match to be at most $1/m$. Suppose we solve k such problems. Give a bound on the probability that we get a false match.
  Probability is at most k/m.