CSE 331: Introduction to Algorithm Analysis and Design Fall 2012

HOMEWORK 9

Due Friday, November 30, 2012 by 1:15pm in class

Please submit each problem separately, i.e. each problem should begin on a new page and only the pages for one problem should be stapled together. Failure to do so might result in some problem(s) not being graded.
For general homework policies and our suggestions, please see the policy document. **Please make sure you follow the collaboration policy.**
Do not turn in Q 0 and 1(c).

---

0. (**DO NOT turn this problem in**) We will consider the problem of exponentiating an integer to another. In particular, for non-negative integers a and n, define POWER(a,n) be the number a^n. (For this problem assume that you can multiply two integers in $O(1)$ time.)

   (a) Present a naive algorithm that given non-negative integers a and n computes POWER(a,n) in time $O(n)$. (For this part, there is no need to prove correctness of the naive algorithm.)

   (b) Present a divide and conquer algorithm that given non-negative integers a and n computes POWER(a,n) in $O(\log n)$ time.

   Note: To get full credit you must present a recursive divide and conquer algorithm and then analyze its running time by solving a recurrence relation. (Justify the correctness of your algorithm– a formal correctness proof is not required.)

   Hint: The following mathematical identity could be useful– for any real numbers b, c and d: $b^{c+d} = b^c \cdot b^d$.

1. ($30 + 10 = 40$ points) In this problem, we will look at a query system that is motivated by applications in wireless networks. If you are not interested in the application details, just skip the next paragraph and head straight to the formal description of the problem.

   In particular, consider the scenario where there is a central node such that all the other sensor nodes can communicate directly with the central node. Each sensor node has a bit of information (e.g. "Is the temperature at my location > 70 degrees?") The central node wants to compute some aggregate function over these bits: e.g. are there at least two sensor nodes with temperature greater than 70 degrees? The central node can "poll" multiple sensor nodes at once to see if their bits are one. Each sensor node replies with a positive back if it is polled and its bit is one. Else it remains silent. Now the central node can easily detect whether at least one of the sensor nodes it polled had its bit as one by just checking if some sensor node responded or not. Due to the nature of the wireless medium, it is very hard to count the number of responses (due to collision) but it is easy to check if at least one sensor node responded by just checking for "silence." Now for computing any function, we want to minimize the number of polls as each poll needs a transmission, which in turn lower the battery life. The problem below talks about this scenario but only for "threshold" functions.

   In this problem the input are n bits $x_1, \ldots, x_n$. However, you can access the input using the following kind of queries. A *query* is a subset $S \subseteq \{1, \ldots, n\}$. The *answer* to a query S is 0 if $\sum_{i \in S} x_i = 0$ and

is 1 otherwise (i.e. if $\sum_{i \in S} x_i \geq 1$). Note that *you* have the full freedom to pick the query. So e.g. you can query all the bits one by one and have the full knowledge of all the bits $x_1, \ldots, x_n$. However, this means you will have to make n queries, which is a lot. Your goal will be compute certain function using *as few queries as possible*.

In this problem, we will look at the t-threshold function f_t for some integer $1 \leq t \leq n$. In particular, $f_t(x_1, \ldots, x_n) = 0$ if $\sum_{i=1}^{n} x_n < t$ and $f_t(x_1, \ldots, x_n) = 1$ otherwise. For example, let us consider the the 3-threshold function for $n = 5$. Note that $f_3(1,0,0,1,0) = 0$ and $f_3(1,0,0,1,1) = 1$.

Note that for any input $x_1, \ldots, x_n$ and any $0 \leq t \leq n$, one can compute $f_t(x_1, \ldots, x_n)$ in n queries. (In this case we will query $\{i\}$ for every $1 \leq i \leq n$ and assign $x_i = 1$ if and only if the answer to $\{i\}$ is 1. After all the n queries are done, we can decide $f_t(x_1, \ldots, x_n)$ depending on whether $\sum_{i=1}^{n} x_i$ (which we can now calculate as we know all the values of x_i) is $< t$ or not.)

In this problem, you will design algorithms that decide the threshold functions using less than n queries.

(a) Here is an algorithm to compute the f_1 function with *one* query. Query the set $\{1, \ldots, n\}$ and declare $f_1(x_1, \ldots, x_n)$ to be 1 if and only if the answer to the query is 1.

Design a divide and conquer algorithm that can decide the function f_2 on *any* input $x_1, \ldots, x_n$ using only $O(\log n)$ **queries**. (You cannot do asymptotically better– see part (c).)

(b) For any $1 \leq k \leq n$, design an algorithm that can decide the function f_k on *any* input $x_1, \ldots, x_n$ using only $O\left(k \cdot \log\left(\frac{n}{k}\right)\right)$ queries.

(*Note:* If you solve this part correctly, then you do not need to write up the solution for the previous part.)

(c) (**DO NOT turn this part in**) Show that **any** algorithm that can decide the function f_2 for *every* input $x_1, \ldots, x_n$ needs $\Omega(\log n)$ queries.

2. (45 points) Exercise 1 in Chapter 5.

Hint: Try to find two “distinguished" elements (one in each of the two lists). Try to see if you can rule out having to process some parts of the input depending on which of the distinguished elements is bigger.

Side Note: I recently found out this was an interview question for one of the big companies.

3. (15 points) Given a directed graph $G = (V, E)$, a vertex $s \in V$ is called a *sink* if there are incoming edges from every other vertex to s but no outgoing edge from s, i.e. $|\{(u, s) \in E\}| = |V| - 1$ and $|\{(s, u) \in E\}| = 0$.

Present an $O(n)$ time algorithm to find out if G has a sink and if so, to output it. (Recall that $n = |V|$). Your algorithm is given G in its adjacency matrix format.

(*Note:* The solution we have in mind is something like a divide and conquer algorithm. However, you do not have to present an algorithm based on the divide and conquer paradigm: as long as your algorithm is correct and runs in time $O(n)$, you're good.)