CS:5350 Homework 3, Fall 2019

Due in class on Thu, Nov 7th

Notes: (a) It is possible that solutions to some of these problems are available to you via textbooks, on-line lecture notes, etc. If you use any such sources even partially, please acknowledge these in your homework fully *and* present your solutions in your own words. You will benefit most from the homework, if you seriously attempt each problem on your own first, before seeking other sources. (b) It is okay to form groups of **three** in solving and submitting homework solutions. (The syllabus says groups of at most two are allowed, but given the size of the class and the fact that the class has no TA, I am modifying this requirement to allow groups of three.) But, my advice from (a) still applies: you will benefit most from the homework, if you seriously attempt each problem on your own first, before seeking help from your group partner(s). (c) Unless you have a documented accomodation, no late submissions are permitted. You will receive no points for your submission if your submission is not turned in at the beginning of class on the due date. (d) Your submissions will be evaluated on correctness *and* clarity. Correctness is of course crucial, but how clearly you communicate your ideas is also quite important.

1. *Time division multiple access (TDMA)* is a method for different nodes in a distributed system to access a shared physical medium. For example, for nodes in a wireless network, their *radio frequency channel* is the shared medium that they all need to access to be able to communicate. TDMA is also used in other settings such as bus networks, where different nodes need to access a shared bus in order to communicate. In TDMA, time is partitioned into time slots and each node has its own time slot during which it uses the shared medium.

   Consider the situation in which there are n identical[1] nodes in a wireless network, each of which needs to send a message to a base station. The base station is within the transmission range of the wireless nodes, but it is possible that not all pairs of wireless nodes are in each others' transmission ranges. The wireless nodes have access to a single radio frequency channel and therefore if two or more of these nodes transmit their message at the same time, the base station hears a "collision," but does not receive any of the transmitted messages. More precisely, a base station has the ability to distinguish among three situations in a time slot: (i) it hears no transmission, (ii) it hears a "collision", and (iii) it hears a message. The base station can also transmit, but it also uses the same radio frequency channel as the wireless nodes. Therefore, if the base station is transmitting at the same time as a wireless node, it will hear a collision.

   Now notice that n nodes have to send messages to the base station and no two nodes can transmit at the same time. So at least n time slots are needed for all the messages to reach the base station. The problem is how should the wireless nodes and the base station coordinate transmissions so that it does not take too many time slots for all messages to reach the base station.

   (a) Use ideas you have learned from our discussion of Luby's MIS algorithm to design a randomized algorithm that uses $O(n)$ times slots in expectation to ensure that the messages from all n wireless nodes reach the base station. You can assume that nodes initially know n.

   (b) Prove that your algorithm runs in expected $O(n)$ rounds.

2. Consider the execution of Luby's MIS algorithm on a cycle C of length n.

   (a) Consider an arbitrary vertex v that is active at the start of some iteration i. Prove that

$$\text{Prob}[v \text{ joins the MIS in iteration } i] \geq c$$

[1]Here "identical" means that nodes do not have IDs.

for some constant $c > 0$.

(b) Consider a path P of vertices all of which are active at the start of some iteration i. Using the result in (i), show that the probability that all vertices in P are active after iteration i is at most $1/2^{c'|P|}$ for some constant $c' > 0$.
Note: Be careful; if u and v are neighbors then the event that u stays active after iteration i is not independent of the event that v stays active after iteration i. But, you can pick vertices that are far enough from each other in P, you can still use independence.

(c) Consider a path P of length $\lceil\sqrt{\log n}\rceil$ in C. Show that for a sufficiently large constant C, the probability that all vertices in P are active after $C\sqrt{\log n}$ iterations is at most $1/n^2$.

(d) Modify Luby's MIS algorithm so that terminates in $O(\sqrt{\log n})$ rounds with high probability on cycles of length n. Use the result in (c) to guide your modification. Show that your algorithm is correct and that it has the desired running time.

3. A *graph coloring* is an assignment of colors to vertices of a graph so that no two adjacent vertices have the same color.

Here is a simple "Luby-like" randomized, distributed algorithm to color the vertices of a graph $G = (V, E)$ using 3Δ colors, where Δ is the maximum degree of a vertex in G. Initially, let $C_v = \{1, 2, \ldots, 3 \cdot \Delta\}$ be the set of colors available for each vertex v. Every vertex is initially colorless. In each iteration, each colorless vertex v in G picks a color from C_v uniformly at random. Each vertex v that has just picked a color then checks if any neighbor has chosen the same color. If there is a neighbor who has chosen the same color as v, then v "backs off" i.e., reverts back to being colorless. Otherwise, v makes its color choice permanent. Finally, for each colorless v, we delete from C_v all colors that became permanent at a neighbor of v. The algorithm then proceeds to the next iteration.

(a) Show that in any iteration, the probability that a colorless vertex v gets colored in that iteration is at least a constant.

(b) Let x denote the number of colorless vertices at the start of an interation. Using the concepts of indicator random variables and linearity of expectation, show that in expectation at most x/c vertices, for constant $c > 1$, of the graph remain colorless after the iteration.

4. As mentioned in class, there is a simple randomized Las Vegas algorithm that solves the SELECTION problem. This is very similar to the randomized quicksort algorithm discussed in class. Here is an informal description of this algorithm. An element in a list is said to have *rank* k if it is the k-th smallest element in the list. Suppose we want to find an element of rank k in the given list L (of n distinct elements). We pick a *pivot* index $p \in [1 \ldots n]$ uniformly at random and construct sublists $L_1 = \{u \in L \mid u < L[p]\}$ and $L_2 = \{v \in L \mid v > L[p]\}$. If $L[p]$ has rank k, we are done. Otherwise, we recurse on one of L_1 *or* L_2 looking for an element of appropriate rank.

(a) State this algorithm precisely in pseudocode.

(b) Show that the expected running time of the algorithm is $O(n)$ by mimicking the approach used for analyzing randomized quicksort in class. Specifically, let X_{ij} denote the number of times y_i and y_j are compared by the algorithm. (Recall the definitions of y_i and y_j from lecture notes.) Let $X = \sum_i \sum_j X_{ij}$ denote the total number of comparisons made by the algorithm. Show that $E[X] = O(n)$.

5. Consider a modification of Karger's mincut algorithm in which in each iteration we pick a *pair* of vertices u and v uniformly at random and contract this pair. Whether u and v are

connected by an edge is not relevant to this choice. It turns out that this modified version of Karger's algorithm is pretty bad.

To see this consider the following graph. Let n be an even, positive integer. Construct an n-vertex graph by taking two size-$n/2$ cliques C_1 and C_2 and adding one edge between them. The edge we add between C_1 and C_2 connects an arbirarily chosen pair of vertices.

(a) What is the probability that Karger's (original) mincut algorithm performs a "bad" contract in the first iteration of the algorithm?

(b) What is the probability that Karger's (modified) mincut algorithm performs a "bad" contract in the first iteration of the algorithm?

(c) Starting with your calculation in (b), show that the probability that modified Karger's mincut algorithm will find the mincut in the given graph is at most $1/2^{c \cdot n}$ for some constant $c > 0$.

(d) As we have seen in several examples, repeating a randomized algorithm a bunch of times and returning the best answer found is a simple way of amplifying the probability of correctness. Why isn't this a useful option for modified Karger's mincut algorithm?