CS5330: Randomized Algorithms

Problem Set 3

Due: February 12, 6:30pm

Instructions. The *exercises* at the beginning of the problem set do not have to be submitted—though you may. They mainly cover topics related to Cuckoo hashing (which we covered in tutorial) and hash functions (which we will be talking about next week). There are four problems, related to different balls-and-bins process, the Coupon Collector problem, etc.

- Please submit the problem set on IVLE in the appropriate folder. (Typing the solution using latex is recommended.) If you want to do the problem set by hand, please submit it at the beginning of class.
- Start each problem on a separate page.
- If you submit the problem set on paper, make sure your name is on each sheet of paper (and legible).
- If you submit the problem set on paper, staple the pages together.

Collaboration Policy. The submitted solution must be your own unique work. You may discuss your high-level approach and strategy with others, but you must then: (i) destroy any notes; (ii) spend 30 minutes on facebook or some other non-technical activity; (iii) write up the solution on your own; (iv) list all your collaborators. Similarly, you may use the internet to learn basic material, but do not search for answers to the problem set questions. You may not use any solutions that you find elsewhere, e.g. on the internet. Any similarity to other students' submissions will be treated as cheating.

Exercises and Review

Exercise 1. Prove the following facts regarding Cuckoo Hashing. (Note: we have already covered these in tutorial; the goal here is to carefully write down a proof.) Assume throughout that $m = 4n$.

- Given a Cuckoo Graph containing n edges, show that any path of length p in the Cuckoo Graph generated by an insert operation that terminates has a simple subpath (with no repeated nodes) of length at least $p/3$.
- Given a Cuckoo Graph containing n edges, show that for all nodes i and j, for all ℓ, the probability that the graph contains a path of length exactly ℓ from i to j is at most $\frac{1}{4^\ell}\frac{1}{m}$.
- Prove that the expected cost of an insert operation is $O(1)$ (as long as the Cuckoo Hash Table contains at most n items).
- Given a Cuckoo Graph containing n edges, prove that the probability it contains a cycle is at most $1/2$.
- Whenever an insert does not terminate (i.e., ends in an infinite loop), you need to create a new Cuckoo Hash Table, choose new hash functions, and reinsert all n items. (If your insert operation continues for $2m$ steps, you can assume you have entered an infinite loop!). Assume that you continue this process, rehashing as necessary, until you successfully insert all n items. Prove that the expected total cost of this process is $O(n)$ (including the cost of the repeated rehashing).

Exercise 2. Consider a version of Cuckoo hashing that only uses one array A, but has two hash functions f and g. Each element x is inserted at either $A[f(x)]$ or $A[g(x)]$. As before during an insert, if a space is occupied, then the old item is kicked out and moved to its new location. Can you modify the existing analysis to work in this setting?

Exercise 3. Think about the variant where you have one array A and k hash functions $f_1, f_2, \ldots, f_k$. Each element x is inserted at some $A[f_j(x)]$, for some j. How would you design an insert algorithm for this variant? How do you decide where to move an item when it is evicted? What if you are allowed to store ℓ items in each slot in the array? What are the trade-offs involved?

Exercise 4. Define the following three random variables:

- A has value 100 with probability $1/2$ and value 200 with probability $1/2$.
- B has value 100 with probability $1/3$, value 120 with probability $1/6$, and value 200 with probability $1/2$.
- C has value 1000 with probability $1/4$ and value 0 otherwise.

Show that B stochastically dominates A, that B does *not* stochastically dominate C, and that C does not stochastically dominate B. (That is, B and C are incomparible.)

Problem 1. [Deploying a Sensor Network.]

Imagine you are deploying a sensor network in a flat, square field that is 1km by 1km. Each sensor has a range $r < 1km$, meaning that it can record all events that occur within distance r of the sensor. The sensors are deployed randomly in the field (e.g., imagine they are dropped from an airplane).

How many sensors should be deployed to ensure that with probability at least $(1 - \epsilon)$, every event in the field can be monitored, i.e., every point in the field is within range of at least one sensor. Assume ϵ is a fixed constant error parameter < 1. (*Hint:* a sensor in a square with side-length $r/\sqrt{2}$ can reach every point in the square.)

Problem 2. [Random Graphs.]
Assume you build a random graph $G(n, m)$ with n vertices and m edges where the edges are chosen uniformly at random from the set of all possible edges. (That is, the graph includes a random subset of the $\binom{n}{2}$ possible edges.) Via the principle of deferred decisions, we can imagine that you construct the graph as follows:

Repeat until the graph is connected:

- Choose a node u at random.
- Choose a node v at random.
- If (u, v) is not an edge in the graph, add edge (u, v) to the graph.

What is the expected number of edges you have to sample before the graph G becomes connected? Argue that from this you can derive a size m so that a random graph with m edges has at least a probability $1/2$ of being connected. (*Hint:* think about the graph in terms of its connected components, and apply the Coupon Collector's technique.)

Problem 3. Group Assignment.

Professor Unfriendly wants to group students in his class in such a way that no pair of friends are in the same group. Luckily, Professor Unfriendly has access to the Facebook friend graph, and can tell who are friends. He has access to a graph $G = (V, E)$ where V is the set of n students in the class and each edge $e \in E$ indicates a pair of friends. The maximum degree of the graph is Δ, and Professor Unfriendly wants to create at most $T = 2\Delta$ groups.

The professor runs the following algorithm:

Repeat until every student is assigned a group:

- Iterate through all the students in order:
  1. Choose a group p uniformly at random from $[1, T]$.
  2. If the current student is not yet assigned to a group, and if none of his/her friends are in group p, then assign the student to group p.
  3. Otherwise, skip the current student and continue with the next one.

Your goal in this problem is to analyze the running time of this algorithm. To do that, we will count the number of times we execute the inner loop (i.e., choosing a group and trying to assign it).

Let X_j be an indicator random variable defined by the jth iteration of the inner loop: if the student in the jth iteration is assigned a group, then $X_j = 1$; otherwise $X_j = 0$. As soon as $\sum X_j = n$, we know that every student has been assigned a group.

Beware, though, that it is not easy to compute $\Pr[X_j = 1]$, since it depends on the outcome of all the previous random choices. (For example, for the first student to be assigned a group, the probabiity is 1, i.e., $\Pr[X_1] = 1$. For the last student to be assigned a group, the probability is lower especially if they have a lot of friends!) So the X_j are not independent!

Problem 3.a. Define a new (independent) set of random variables $Y_1, Y_2, \ldots$ so that X_j stochastically dominates Y_j. (We will use the Y's to bound the running time, so we want them to be less likely to occur than the X's.)

Give the definition of Y_j and prove carefully (using the definition of stochastic domination) that X_j dominates Y_j.

Problem 3.b. Let $t = 2n \log n$, and define $Y = \sum_{j=1}^{t} Y_j$. Show that $Y \geq n$ with probability at least $1 - 1/n$. (If you cannot prove this, then you might want to revisit your definition of Y_j.)

Problem 3.c. Now conclude the proof by showing that, with high probability, the group assignment algorithm completes within time $O(n \log n)$.

You may assume the following fact:

- We define the term *unconditionally sequentially dominates* as follows: A sequence of random variables $(X_1, \ldots, X_n)$ *unconditionally sequentially dominates* another sequence of random variables $(Y_1, ..., Y_n)$ if for each j, $(X_j |$ arbitrary $X_1, ..., X_{j-1})$ stochastically dominates Y_j, i.e., if each X_j stochastically dominates Y_j, regardless of the outcome of all the previous $X_{j-1}, X_{j-2}, \ldots$ (i.e., unconditionally).
- If $X_1, ..., X_n$ are an arbitrary set of (discrete) random variables, and $Y_1, ..., Y_n$ are independent (discrete) random variables, if $(X_1, ..., X_n)$ unconditionally sequentially dominates $(Y_1, ..., Y_n)$ then $\sum(X_j)$ stochastically dominates $\sum(Y_j)$.

Problem 4. Contention Resolution.

One of the major problems in distributed and parallel systems is contention resolution: there are a collection of agents that want to coordinate access to a shared resource. For example, the resource may be an ethernet connection or a wireless channel (where only one device can broadcast a message at a time). Or the shared resource may be a lock that multiple concurrent threads are trying to access in order to update a data structure.

A typical approach is to use a randomized strategy: when a device wants the resource, it randomly chooses one of the next T timeslots at random and tries to claim the resource in that randomly chosen slot. If it fails, it waits until all T slots elapse, and repeats the procedure. (The T slots are often referred to as the "window" and in a backoff protocol, the size of the window may be adjusted dynamically.)

Here we model this strategy as a simple balls and bins problem. Imagine you have n balls (which represents requests to use the shared resource) and b bins (which represent timeslots in the window). We play the following game:

Repeat until all the balls have been removed:

- Place each remaining ball in a bin chosen uniformly at random.
- Every ball that lands in a bin by itself (with no other balls) is removed. (Since this ball/request was the only one in the bin/timeslot, it can safely access the resource during that timeslot.)
- All the balls that lands in a bin containing more than one ball are collected and advance to the next round.

Intuitively, as long as b is sufficiently bigger than n, then in each round of the game we remove many of the balls and make progress toward completing the game. Our goal in this problem is to calculate the expected number of rounds until all the balls have been removed.

Assume throughout this problem that the initial number of balls $n \leq b/8$. Let n_j be the number of balls that remain after round j.

Continued on the next page.

Problem 4.a. Show that:

$$\mathrm{E}\left[n_j \mid n_{j-1} \leq x\right] \leq \frac{2x^2}{b} \ .$$

(*Hint:* remember that if $x \leq 1$, then $e^{-2} \leq (1-1/x)^x \leq e^{-1}$; identically, for $x \geq 1$, $e^{-2x} \leq (1-x) \leq e^{-x}$.)

Problem 4.b. In the previous part, you showed that in expectation the number of balls decreases rapidly in each round. We now want to calculate the probability that we make progress. (Remember, in some rounds we may do better than the expectation; in some rounds we may do worse!)

Let I_j be the event that the number of remaining balls is at most $n/2^{2^j}$ at the end of a round. What is the probability that event I_0 does not occur in a round (if it has not previously occurred)? Show that I_0 occurs with probability at least $1/2$ (if it hasn't occurred previously).

Problem 4.c. Given that event I_{j-1} has already occurred (and I_j has not already occurred), what is the probability of I_j not occuring in a round? Show that I_j occurs with probability at least $1/4$ (if it hasn't occurred previously, and I_{j-1} has occurred previously).

Problem 4.d. What is the expected number of rounds until the game is over? (Hint: define X_j to be the number of rounds after I_{j-1} until I_j occurs.)