CS7800: Advanced Algorithms. Fall 2016
Homework 5

Instructor: Jonathan Ullman, TA: Mehraneh Liaee

Due Wed, Dec 7 at 11:59pm
(Email to `m.liaee2050+CS7800@gmail.com`)

- Note the unusual deadline. I am trying to strike a balance between giving more time due to Thanksgiving break and giving us time to grade before the end of the semester.
- You must type your solutions using LaTeX. Please submit both the source and PDF files using the naming conventions `lastname_hw5.tex` and `lastname_hw5.pdf`.
- **Please put your name somewhere on the first page of your submission.**
- Strive for clarity and conciseness in your solutions, emphasizing the main ideas over low-level details. I recommend looking at the introduction in Jeff Erickson's textbook for advice on writing up solutions to algorithms problems.
- Do not share written solutions, and remember to cite all collaborators and sources of ideas. Sharing written solutions, and getting solutions from outside sources such as the Web or students not enrolled in the class is strictly forbidden.

Problem 1 (Probability Practice, 25 points)**.**

You are filming an epic fight sequence on top of a train with n cars. The cars are numbered $1, 2, \ldots, n$, with car 1 in the front, and car n in the rear. In each shot, you move backwards one car with probability 1/2 and forward one car with probability 1/2. The moves between each scene are independent. If you go forward from car 1 you "fall off the front" and if you go backwards from car n you "fall off the back." The process continues until you fall off one end of the train.

(a) Suppose you start on car 1. Prove that the probability you fall off the front of the train is exactly $\frac{n}{n+1}$.

(b) Suppose you start on car k. Prove that the probability you fall off the front of the train is exactly $\frac{n-k+1}{n+1}$.

(c) Suppose you start on car 1. Prove that the expected number of shots before you fall off one end of the train is exactly n.

(d) Suppose you start on car $\lceil n/2 \rceil$. Prove that the expected number of shots before you fall off one end of the train is cn^2 for some constant c.

Problem 2 (The Game Show, 25 points)**.**

You have been selected to appear on the game show *BigMoneySuitcaseThrow!* There are n suitcases, containing distinct enormous sums of money $p_1, \ldots, p_n$. The values $p_1, \ldots, p_n$ are different in every episode, and you have absolutely no idea what they are. The suitcases are then violently thrown at you in a *uniformly random order*. You catch each suitcase, open it, and then have to make a snap decision: either take the money in that suitcase and end the game, or throw the suitcase into the *GiantPitOfRegrets* and keep going on the remaining suitcases.

You come up with the following type of strategy: Throw out the first t suitcases for some $1 \le t < n$. Let m be the largest amount you saw in the first t suitcases. Among the remaining $n-t$ suitcases, take the first one you see whose value is larger than m. Let $p_{\max} = \max\{p_1, \ldots, p_n\}$ be the largest amount of money you could win.

(a) What is the probability that you select the suitcase with $p_{\max}$ as a function of t and n?[1]

(b) Find a choice of t such that you select the suitcase with $p_{\max}$ with probability at least 1/8.[2]

Problem 3 (Generalized Contention Resolution, 25 points)**.**

Consider the following variant of the contention resolution problem we studied in class: There are n processes and 1 resource. Time proceeds in rounds $t = 1, 2, \ldots,$ and in each round each process can try to access the resource or not. We are given an unweighted, undirected, *conflict graph* $G = (V, E)$ where $(i, j) \in E$ denotes that if process i and process j try to access the resource in the same round, they both fail. (The case we considered in class is where G is the complete graph on n nodes.) Assume that every node in G has exactly d neighbors. Or, equivalently, that every process i has exactly d conflicts.

[1] Note that the only random outcome is the order in which the suitcases are thrown at you. The numbers $p_1, \ldots, p_n$ are not random, and your strategy does not make any random decisions.

[2] You can actually do better than 1/8. For fun, try to find the highest possible probability.

(a) Suppose that in every round, every process tries to access the resource with probability p, independent of all other processes and rounds. What is the probability that it succeeds in a given round? Find the value of p that maximizes the probability that a processor succeeds in terms of d and n.

(b) Using the optimal choice of p from part **(a)**, what is the expected number of processes that successfully access the resource in each round? What is the expected number of processes that successfully access the resource after t rounds?

(c) Again, using the optimal choice of p from part **(a)**, compute a number T such that with probability at least $1 - \delta$ every process succeeds in accessing the resource after T rounds.

Problem 4 (Sampling in Data Streams, 25 points)**.**

Imagine you are Amazon and you would like to monitor your transactions to find items that have suddenly become popular. Fortunately for you, you do millions of transactions every day. Unfortunately, with that many transactions, even storing all your transactions in one place is impossible.

A *data stream* is a long sequence of items $x_1, x_2, \ldots$ that you can only read once. A data stream algorithm must process each item very quickly, and with too little memory to store the stream of items, or even a significant fraction of the items. Streaming algorithms typically look like this:

```
ProcessStream(stream S):
  Repeat:
    $x_i$ = next item in the stream
    [do something really simple and fast with $x_i$]
  Until the stream stops
  Output [something interesting]
```

Figure 1: A caricature of a streaming algorithm.

(a) One interesting thing to do with a data stream is to sample a uniformly random item. If you knew that the stream had length n, you could simply choose a number i randomly from $\{1, \ldots, n\}$ and then output the item x_i when it arrives. Design an algorithm that chooses a uniformly random sample from the stream *without knowing the length of the stream in advance.* Your algorithm should use $O(1)$ time and $O(1)$ space to process each item (assuming each stream element is $O(1)$ size).[3]

(b) Assume that the items in the stream are numbers, and we want to find the *median* of the items in the stream. Being the median is a very global property, so it is difficult to find the median without storing a lot of the stream. Using the algorithm from **(a)**, we can obtain a set of m independent and uniformly random samples from the stream (with replacement). Suppose we take the median x^* of these m samples. Prove that there exists a constant m_0 such that if $m \geq m_0$, then with probability at least $\frac{99}{100}$, x^* is an $\frac{1}{100}$-*approximate median* of the items in the data stream. By $\frac{1}{100}$-approximate median, we mean that if the stream has length n, then there are at least $\frac{49n}{100}$ items in the stream smaller than x^* and at least $\frac{49n}{100}$ items in the stream larger than x^*. You may assume all items in the stream are distinct.

[3]Hint: store a single item that will be your sample, after each item arrives, decide whether to keep the same sample or throw it out and replace it with the item that just arrived.