CSCE629 Analysis of Algorithms

Homework 12

Texas A&M U, Fall 2019 *11/12/19*
Lecturer: Fang Song *Due: 10am, 12/02/19*

Instructions.

- Typeset your submission by LaTeX, and submit in PDF format. Your solutions will be graded on *correctness* and *clarity*. You should only submit work that you believe to be correct, and you will get significantly more partial credit if you clearly identify the gap(s) in your solution. You may opt for the "I'll take 15%" option.
- You may collaborate with others on this problem set. However, you must **write up your own solutions** and **list your collaborators and any external sources** for each problem. Be ready to explain your solutions orally to a course staff if asked.
- For problems that require you to provide an algorithm, you must give a precise description of the algorithm, together with a proof of correctness and an analysis of its running time. You may use algorithms from class as subroutines. You may also use any facts that we proved in class or from the book.
- **If you describe a Greedy algorithm, you will get no credit without a formal proof of correctness, even if your algorithm is correct.**

This assignment contains 5 questions, 4 pages for the total of 38 points and 3 bonus points. A random subset of the problems will be graded.

Exercises. Do not turn in.

1. (Hat-check) Each of n customers gives a hat to a hat-check person at a restaurant. The hat-check person gives the hats back to the customers in a uniformly random order. What is the expected number of customers who get back their own hat?
2. (Streaks) Suppose you flip a fair coin n times. What is the longest streak of consecutive heads that you expect to see?

Problems to turn in.

1. (Happy holidays) This time of year is here, you known, holidays, and everyone is facing various (joyful) challenges.
   (a) (10 points) For Kevin (character in movie "Home Alone"), he is handed n pieces of candy with weights $W[1, \dots, n]$ (in ounces) that he needs to load into boxes. The goal is to load the candy into as many boxes as possible, so that each box contains at least L ounces of candy. Help Kevin solve this by describing an efficient 2-approximation algorithm for this problem. Prove that the approximation ratio of your algorithm is 2. [Hint: First consider the case where every piece of candy weighs less than L ounces.]

   (b) (10 points) For Juliet, this means that she needs to fulfill Romeo's wishlist - n Nintendo® Switch games. She happened to notice that in a personal care and beauty store AROHPES, her favorite lip balm includes a 99% discount code for one of the n games Romeo wants. However, the code is hidden at the bottom of the package, which Juliet cannot tell before purchasing and opening it, and the code comes in random so that it is equally likely to be good for any of the n games. She intends to purchase sufficient lip balms, which she likes and needs anyways, so she gets discount codes for all n games. How many lip balms does she need to buy *in expectation* before getting a code for each game?

   (c) (3 points (bonus)) Continuing from Part (b), it's 7 days till Thanksgiving now. Due to popularity, AROHPES limits the number of this particular lip balms one can purchase in a day to 6. She convinced Romeo to reduce his demand to 10 games. Can Juliet succeed in collecting all discount codes by Thanksgiving? [Hint: look up Markov's inequality]

2. (3-Coloring) Suppose you are given a graph $G = (V, E)$, and we want to color each node with one of three colors, even if we aren't necessarily able to give different colors to every pair of adjacent nodes. We say an edge (u, v) is satisfied if the colors assigned to u and v are different.

   Consider a coloring scheme that maximizes the number of satisfied edges, and let c^* denote this number. Give a poly-time algorithm that produces a coloring that satisfies at least $\frac{2}{3}c^*$ edges. If you want to use an randomized algorithm, the *expected* number of edges it satisfies should be at least $\frac{2}{3}c^*$.

3. (Errors in randomized algorithms) Suppose you want to write a computer program C to compute a Boolean function $f : \{0,1\}^n \to \{0,1\}$, mapping n bits to 1 bit. If C is a deterministic algorithm, then "C successfully computes f" has a clear meaning that that $C(x) = f(x)$ for all inputs $x \in \{0,1\}^n$. But what if C is a randomized algorithm?

   (a) (8 points) The best thing is if C is a *zero-error* algorithm with failure probability p. Namely

   - on every input x, the output of $C(x)$ is either $f(x)$ or $\perp$ (denoting failure).
   - on every input x we have $\Pr[C(x) = \perp] \le p$ (NB. the probability is only over the internal randomness of C, not the random choice of x.).

   i) If you have a zero-error algorithm C for f with failure probability 90%, show how to convert it to a zero-error algorithm C' with failure probability at most 2^{-500}. The "slowdown" should only be a factor of a few thousand.

   ii) Alternatively, show how to convert C to an algorithm C'' for f which: (i) always outputs the correct answer, meaning $C''(x) = f(x)$ for all x; (ii) has expected running time only a few powers of 2 worse than that of C. (Hint: look up the mean of a geometric random variable.)

   (b) (5 points) The second best thing is if C is a one-sided error algorithm for f, with failure probability p. There are two kinds of such algorithms, "no-false-positives" and "no-false-negatives". For simplicity, let's just consider "no false-negatives" (the other case is symmetric);

   - on every input x, the output $C(x)$ is either 0 or 1;
   - on every input x such that $f(x) = 1$, the output $C(x)$ is also 1;
   - on every input x such that $f(x) = 0$, we have $\Pr[C(x) = 1] \le p$.

   Show how to convert a no-false-negatives algorithm C for f with failure probability 90% to another no-false-negatives algorithm C' for f with failure probability at most 2^{-500}. The "slowdown" should only be a factor of a few thousand.

   (c) (5 points) The third possibility (which is rare in practice) is if C is a two-sided error algorithm for f, with failure probability p. Namely,

   - on every input x, the output $C(x)$ is either 0 or 1.
   - on every input x, we have $\Pr[C(x) \neq f(x)] \le p$.

   If you have a two-sided error algorithm C for f with failure probability 40%, show how to convert it to a two-sided error algorithm C' for f with failure probability at most 2^{-500}. The "slowdown" should only be a factor of a few dozen thousand. (Hint: look up the Chernoff bound.)