Problem Set 3

This problem set is due at 11:59pm on Monday, March 11, 2013.

Both exercises and problems should be solved, but only the problems should be turned in. Exercises are intended to help you master the course material. Even though you should not turn in the exercise solutions, you are responsible for material covered by the exercises.

Mark the top of each sheet with the following: (1) your name, (2) the name of your recitation instructor, and the time your recitation section meets, (3) the question number, (4) the names of any people you worked with on the problem, or “Collaborators: none” if you solved the problem completely alone.

Each problem must be turned in separately to stellar.

You will often be called upon to “give an algorithm” to solve a certain problem. Your write-up should take the form of a short essay. A topic paragraph or sentence should summarize the problem you are solving and what your results are. The body of your essay should provide the following:

1. A description of the algorithm in English and, if helpful, pseudocode.
2. A proof of the algorithm’s correctness.
3. An analysis of the algorithm’s running time.
4. If appropriate, a worked example or diagram to show more precisely how your algorithm works.

Remember, your goal is to communicate. Graders will be instructed to take off points for convoluted and obtuse descriptions.

Exercise 3-1. Augment the skip-list data structure to support finding the kth order statistic (i.e., the kth smallest number) in \(O(\log n)\) time with high probability.

Exercise 3-2. Show how to implement Successor\( (x)\) in a skip list using \(O(\log n)\) time with high probability.

Exercise 3-3. Which of the following Markov chains are connected? Which are aperiodic?

- A king on an 8 \(\times\) 8 chessboard making a random, legal move every turn.
- A bishop on (black squares of) an 8 \(\times\) 8 chessboard making a random, legal move every turn.
- A bishop on (black squares of) a 2 \(\times\) 2 chessboard making a random, legal move every turn.
Problem 3-1. Lazy Hashing

Lazy Luke only attended enough 6.046 classes to learn the basics of hash functions, but didn’t learn how to handle collisions. Unfortunately, his new job requires him to implement a real hash table (of size $m$). Conveniently he is also guaranteed all keys to be inserted will be unique.

Luke informed his employer that he doesn’t intend to do any messy analysis, and rather than firing him, they generously agreed to grant him access to their private collection of uniform hash functions $F$. The set $F$ has the property that, for all $a, b$ and $x_1, x_2, \ldots, x_a$, if $f_1, f_2, \ldots, f_b$ are sampled uniformly at random from $F$, then the random variables $f_i(x_j)$ are i.i.d. in $\{0, 1, \ldots, m-1\}$.

(a) Luke spends a few minutes during his lunch break coming up with an idea, and decides to just put all collisions into a separate linked list. Specifically, he samples one function $f$ from $F$. He has one array of size $m$ called `main` and one linked list (initially empty) called `extra`. He then implements the following methods:

- **INSERT($x$)**: If `main[$f(x)$]` is empty, set `main[$f(x)$] = x`. Otherwise, append $x$ to `extra`.
- **CONTAINS($x$)**: If `main[$f(x)$] = x`, return true. Otherwise, return `extra.CONTAINS(x)`.

Observe that INSERT runs in time $O(1)$ always, but that CONTAINS may have to scan an entire linked list. Come up with a worst-case upper bound (worst case with respect to the inserts and the query) on the expected running time (expectation over the random choice of $f$) of CONTAINS after $n \leq m$ distinct inserts have been made.

(b) Show that, if $n \geq 50\sqrt{m \log m}$, then the same bound holds with probability at least $1 - \frac{1}{m^{100}}$ (with respect to the random choice of $f$). Specifically, if $g(n)$ is an upper bound from part (a) on the number of elements in `extra`, show that, for any sequence of $n$ distinct INSERTs followed by a CONTAINS query, with probability at least $1 - \frac{1}{m^{100}}$ (over the choice of $f$) the actual length of `extra` during CONTAINS is at most $\frac{3}{2}g(n)$.

**Hint:** Try to apply a Chernoff bound. The random variables you used in part (a) might not be independent, so you first have to relate them to random variables that are. Don’t be intimidated by the $50\sqrt{m \log m}$, it will pop out at the end.

(c) Unhappy with the performance of Luke’s first solution, his employers, rather than fire him, decide to give him more space, allowing him to store $m\ell$ elements instead of just $m$. During his second lunch break of the day, Luke decides to use the extra space for separate hash tables. Specifically, his new idea is the following. Sample $\ell$ functions $f_1, f_2, \ldots, f_\ell$ from $F$. Store $\ell$ arrays of size $m$ called `main_1, main_2, \ldots, main_\ell` and one linked list called `extra`. He then implements the following methods:

- **INSERT($x$)**: Set $i = 1$. If `main_i[f_i(x)]` is empty, set `main_i[f_i(x)] = x`. Otherwise, if $i < \ell$, set $i = i + 1$ and repeat. If $i = \ell$, add $x$ to `extra`.
- **CONTAINS($x$)**: Set $i = 1$. If `main_i[f_i(x)] = x`, return true. Otherwise, if $i < \ell$, set $i = i + 1$ and repeat. If $i = \ell$ return `extra.CONTAINS(x)`.
Observe that `INSERT` now takes time $O(\ell)$, but that `CONTAINS` may still have to scan an entire linked list. Show that, when $\ell > \log \log m$, for any sequence of $n \leq m$ distinct `INSERTs` followed by a `CONTAINS` query, with probability at least $1 - 1/m^{99}$, the new actual runtime of `CONTAINS` is $O(\sqrt{m \log m})$.

(d) Show that, for $\ell \geq 201$, $n = O(\sqrt{m \log m})$, and any sequence of $n$ distinct `INSERTs` followed by a `CONTAINS` query, with probability at least $1 - O(1/m^{99})$, the new actual runtime of `CONTAINS` is $O(1)$.

*Hint:* Don’t use a Chernoff bound for this part!

(e) Show that, when $\ell = \log \log m + 202$, for any sequence of $n \leq m$ distinct `INSERTs` followed by a `CONTAINS` query, with probability at least $1 - O(1/m^{98})$, the new actual runtime of `CONTAINS` is $O(1)$.

### Problem 3-2. Shuffling Cards

In this problem, we will use the theory of Markov chains to prove that a card shuffling algorithm achieves its goal of returning a uniformly random arrangement of the cards.

Consider the following algorithm for shuffling a deck of $k$ cards:

```plaintext
SHUFFLE-DECK(deck, n)
1   for i ← 1 to n
2       insert the top card of deck in a uniformly random position in the deck
3   return deck
```

Note that $k$ is the number of cards in the deck, while $n$ is the number of iterations of the shuffling loop. At every iteration of the loop, there are exactly $k$ possible positions in which the top card can be inserted.

For parts (a) and (b), assume that $k = 3$.

(a) Model the evolution of the entire configuration of the deck as a Markov chain. Draw a weighted directed graph whose six nodes represent states, whose edges represent transitions, and whose edge weights represent probabilities. Give the transition matrix that correctly describes the shuffling process.

(b) Find a stationary distribution, and show that it is the unique stationary distribution of this Markov chain.

*Hint:* Notice something special about the column sums of your transition matrix in part (a).

For part (c), do not assume that $k = 3$.

(c) Conclude that the algorithm achieves its goal: argue that, as $n \to \infty$, `SHUFFLE-DECK(deck, n)` produces a uniformly random permutation of the cards in `deck`.