Problem Set 5

This problem set is due at **11:59 pm** on **Wednesday, October 21, 2015**. The exercises are **optional**, and should not be submitted.

Exercise 5-1: Do Exercise 31.8-1 in CLRS on page 975.

Exercise 5-2: Do Exercise 31.8-2 in CLRS on page 975.

Exercise 5-3: Do Problem 31-4 in CLRS on page 982.

Problem 5-1: Fun Factoring

Let N be a positive integer. Recall that Z_N^* is the set of integers $1 \leq a < N$ such that $gcd(a, N) = 1$. We say that x is a square root of a mod N if it satisfies:

$$a \equiv x^2 \pmod{N}$$

We call those elements $a \in Z_n^*$ *quadratic residues* mod N if they have square roots mod N.

For example, for $N = 5$, $Z_5^* = \{1, 2, 3, 4, 5\}$ and the quadratic residues mod 5 are $\{1, 4\}$. Furthermore, the square roots of 4 mod 5 are $\{2, 3\}$, since $2^2 \equiv 3^2 \equiv 4 \pmod{5}$ and the square roots of 1 mod 5 are $\{1, 4\}$. More generally, if N is prime, then every quadratic residue mod N has exactly two square roots.

In contrast, if $N = pq$ is a composite number product of two primes p and q, then every quadratic residue mod N has exactly four distinct square roots. For example, for $N = 15$, $Z_{15}^* = \{1, 2, 4, 7, 8, 11, 13, 14\}$ and the quadratic residues are $\{1, 4\}$. Furthermore, the square roots of 1 mod 15 are $\{1, 4, 11, 14\}$, since $(\pm 4)^2 \equiv 1 \pmod{15}$ and $(\pm 1)^2 \equiv 1 \pmod{15}$ and the square roots of 4 mod 15 are $\{2, 13, 7, 8\}$.

(a) Let p be prime. Show that for any quadratic residue $a \in Z_p^*$, there are exactly two solutions of the equation $a \equiv x^2 \pmod{p}$.

(b) Assume that you are given a magic-box deterministic polynomial time $SQRT(a, N)$ algorithm that outputs one of the square roots of a modulo N when a is a quadratic residue and N is a product of two distinct primes. Design a Las Vegas algorithm that computes the factorization of N when N is a product of two primes, which can make calls to the magic box algorithm $SQRT(\cdot, \cdot)$. How many calls does your algorithm make to $SQRT$, analyze its expected running time. What does this tell you about the relative difficulty of computing square roots modulo N and factoring N? [Hint: Use the fact that a has 4 distinct square roots. You can draw inspiration from the modular square root theorem taught in lecture]

(c) Now consider general composite N. When N is odd and not a perfect power, then every quadratic residue mod N has at least four square roots. You are now given a deterministic $SQRT(a, N)$ algorithm for computing a square root of a mod N for general N. Extend your algorithm from part (b) to factor N. You may assume you have access to a deterministic primality test. [Hint: Don't forget to check for prime powers.]

Problem 5-2: Fun with the SkipList

In this problem you will augment the SkipList to obtain a structure with faster INSERT and SUCCESSOR running times. Additionally, this new structure has particularly useful characteristics for concurrency.

You might find it helpful to recall the trie structure - see problem set 2, problem 2.

(a) Let's start with a top-truncated SkipList of height $h = \log\log u$, where u is the size of the universe (all allowable keys). As in the usual SkipList, a node is promoted to a higher level with probability $p = 1/2$, but the height is capped at $h = \log\log u$.

For parts (i)-(iv) just state the answer.

i. What is the probability that a node will raise to the top level of this SkipList?

ii. In expectation, what is the number of keys in the structure in between two top-level nodes?

iii. With n keys inserted, how many nodes, in expectation, will the top level of this truncated SkipList have?

iv. With n keys inserted, what would be the expected running time of FIND? How about INSERT?

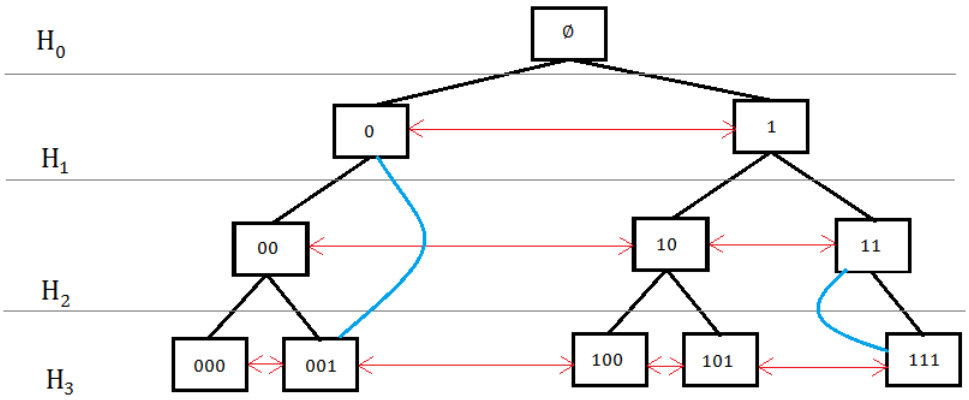
(b) Recall that in the original trie structure, operations INSERT, FIND and SUCCESSOR all take $O(\log u)$ time. Recall that by augmenting the trie with a hash structure at each level (as well as minimum and maximum of a subtree in each node), we managed to speed up SUCCESSOR to $O(\log\log u)$ time. Here we're going to perform a slightly different augmentation.

As before, let's augment each level of the trie with a structure that provides INSERT, FIND and REMOVE in $O(1)$ time (for example, a hash structure), for all binary strings at that level. Since binary strings at k^{th} level have length k, each level has at most 2^k different strings. For example, you could query $H_2(01)$ or $H_3(101)$ in $O(1)$.

Additionally, let's have level of nodes form a (sorted) doubly-linked list: each node stores two pointers: one to its predecessor, the other to its successor on that level.

Lastly, recall that an inner node in the trie has either 1 or 2 children, but never 0. Let's augment inner-nodes (non-leaves) of the structure with a pointer p: if the node does not have the left child, p will point to the *left-most* leaf-node (minimum key) in its *right child*'s subtree. If the node does not have the right child, p will point to the *right-most* leaf-node (maximum key) in its *left child*'s subtree.

See the figure for an example (red lines represent the linked lists, blue lines pointers p):



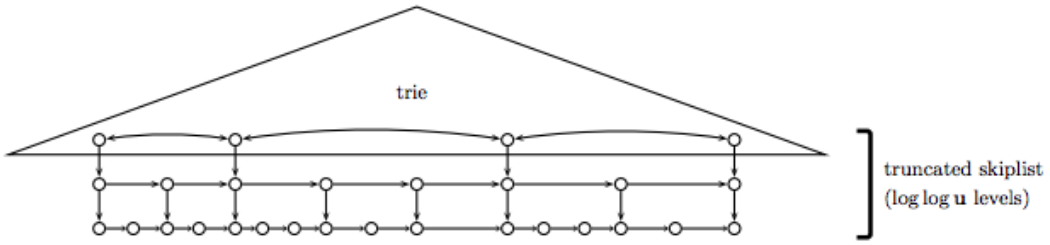
i. You're given a pointer to a key x that'salready present in this structure. What's the running time of SUCCESSOR(x) operation using this pointer? You only need to give a one sentence justification.

ii. **Optional** (do not submit): The optimal INSERT operation has running time $\Theta(\log u)$. Argue why we can't do any better.

iii. Describe the $O(\log \log u)$ SUCCESSOR operation (note that unlike [b-i], you do not have a pointer to work with here).

(c) Notice that while the SkipList in a) is shallow and takes $O(n)$ space, the operations in [a-iv] are comparatively slow. On the other hand, SUCCESSOR in the trie in b) is fast, but the space complexity is $O(n \log u)$, so there's a trade off between speed and memory complexity.

Naturally, we would like to take as little space and be fast for all operations! Let's combine these two structures together. Specifically, let's put the trie *on top* of the SkipList, as in this figure:



The trie on top will contain only the keys from the *top* level of the SkipList. In other words, to insert a key x to this structure, we first insert x to the bottom SkipList, and if x gets promoted to the top level, it is also inserted to the top level of the trie.

i. How many keys will be stored in the top-level trie?

ii. Describe the $O(\log \log u)$-time algorithm for FIND(x) in this combined structure.

iii. Recall that INSERT into the top-level trie still takes $O(\log u)$ time. However, not every key inserted to the combined structure ends up in the top-level trie, and

the bottom level SkipList is shallow - only $O(\log \log u)$ levels. Describe the $O(\log \log u)$ algorithm for INSERT in this combined structure.