Problem Set 3

All parts are due on November 1, 2016 at 11:59PM. Please download the .zip archive for this problem set. Remember, your goal is to communicate. Full credit will be given only to a correct solution which is described clearly. Convoluted and obtuse descriptions might receive low marks, even when they are correct. Also, aim for concise solutions, as it will save you time spent on write-ups, and also help you conceptualize the key idea of the problem.

Part A

Problem 3-1. [20 points] **The Future Campus of MIT**

The year is 26100, and humans have achieved (1) individual flight and (2) access to unlimited dimensions of space. MIT buildings now form a grid in d-dimensional space, and are numbered with d-digit decimal numbers, with each digit representing location along one dimension. There could be at most one building with a given number, but not all numbers are occupied by buildings.

An i-***artery*** is a maximal set of buildings that differ in the ith digit, but share all $d-1$ other digits. In other words, an artery is the set of buildings lying on a line parallel to a coordinate axis. For example, building 25628 is between building 25627 and building 25629 along a 0-artery (where digit 0 is the "ones" digit) in 5-dimensional MIT.

The MIT students of the future are still smart and refer to the buildings by their numbers, but for some reason the building directory given to you has the buildings sorted by name (an irrelevant feature of buildings).

(a) [10 points] Give an efficient algorithm to sort the buildings by number. The algorithm should run in $O(nd)$ time for n buildings.

(b) [10 points] You want to traverse a particular i-artery of MIT in its entirety, given by coordinates $x_0, x_1, \ldots, x_{i-1}, x_{i+1}, x_{i+2}, \ldots, x_{d-1}$, in increasing order by ith digit. Given the sorted-by-building directory from part (a), give an $O(d \log n)$-time algorithm to efficiently extract an in-order list of buildings you will pass through.

Problem 3-2. [25 points] **Finding a Place in the Search Engine Space!**

Elexa, the internet ranking site, has gotten word of the up-and-coming 6006LE, so they decided to publish a new dataset: the amount of internet traffic received by each of the top n sites (n is even) in the past year. As it turns out, the amounts of traffic received for the top n sites are all distinct.

Unfortunately (or fortunately), Estoy, the Elexa engineer responsible for the data, recently fell in love with binary search trees, after encountering them in 6.006. He randomly partitions the data into two disjoint datasets, each containing the amount of traffic for $\frac{n}{2}$ of the sites, and stores each set in a BST. In each BST, a node's key stores the amount of traffic for one site.

As an algorithmically efficient traffic analyst, your goal is to be able to quickly find the kth largest site among these two binary search trees.

(a) [10 points] To start, imagine Estoy gave you two sorted arrays, instead of two BSTs. Give an $O(\log^2 n)$-time algorithm to determine the kth largest number among these two arrays sorted in ascending order. Justify your time complexity.

(b) [5 points] Next, describe how to modify the AVL-INSERT algorithm to maintain, at each node x, the number of nodes in the subtree rooted at x. Note that this modification should be such that the AVL-INSERT algorithm should run in $O(\log n)$-time. (And suppose that Elexa implements these modifications.)

(c) [10 points] Finally, give an $O(\log^2 n)$-time algorithm to find the kth-largest data point across these two size-augmented AVL trees.

Problem 3-3. [25 points] **Chicken Farming**

Following the success of 6006LE, you decide to follow your dreams and open a carbon-neutral, cruelty-free, cage-free, fence-free chicken farm. You order m chickens to be delivered to your house in the hilly countryside, and go to work (to solve Problem 3-2 on your recent 6.006 problem set). The problem takes longer than you expected, and by the time you get back home that evening the chickens should have already been delivered . . . but they're nowhere to be seen!

As you realize that a "fence-free chicken farm" may have been a bad idea, you wander up to a nearby hill and realize where all the chickens have gone. The m chickens (lovingly named $c_1, c_2, \ldots, c_m$) are all roosting in the $|V|$ fields near your house, given by the set V. There are dirt paths between some pairs of the fields, given by the set E, and every dirt path is the same length. There can be multiple chickens (or none) nesting in a single field $v \in V$. You are at the starting field $s \in V$, and you can't see from here where each chicken is. . . so it looks like you'll have to go searching.

(a) [10 points] You decide you'd like to map out where the chickens have settled, and how far you need to walk to collect the eggs from each chicken's nest. Give an algorithm for constructing a list of entries $L = \{(c, v, d) \mid c \in c_1, c_2, \ldots, c_m\}$ denoting that chicken c is located at field v, and that the shortest path from s to v is of length d. Your algorithm should run in $O(V + E + m)$ time.

(b) [5 points] Now that you've catalogued every chicken, you'd like to start farming by collecting eggs from 5 different chickens. You can assume that you can carry infinitely many eggs at the same time. Define a "k-collection path" to be a path starting from s, passing through some number of fields, collecting eggs from the k chickens you pass, and finally returning to s. You pick 5 chickens from your catalogue that have the shortest distance d, and you claim that the shortest possible 5-collection path contains these 5 chickens. Are you correct? If yes, provide a proof. If not, provide a counterexample.

(c) [10 points] After a while, you get tired of getting up every morning to wander through the fields gathering eggs. You decide to give up on your dreams and finally build a fenced enclosure for the chickens in field s. Now you need to bring the chickens back to the enclosure, but every time you go to a field with chickens in it, all the chickens run away from you. Luckily:

- Chickens run only on the existing dirt paths E between the fields.
- Every path $e \in E$ is sloped, and the chickens only run downhill on each path (meaning that every path is unidirectional for chickens).
- Field s is reachable via some directed (chicken-traversable) path from every field.
- No chicken will run to any field that you have visited before (as you leave your scent behind).
- You now have a helicopter, so assume that you instantaneously transport yourself between any two fields.

Give an algorithm to compute a sequence of fields in V that you should visit to get all the chickens to run into your fenced enclosure s. Your algorithm should run in $O(V + E)$ time.

(Note: there are no cycles in the chickens' potential paths because, as in reality, there are no downhill cycles.)

(d) [1 points] (Extra Credit) Can you visit the fields in such an order that you would force a chicken to violate its own behavioral rules? (Of course, as this is a cruelty-free chicken farm, you would never actually do such a thing.)

Part B

Problem 3-4. [30 points] **Rolling in the Deep with Corrupted Strings**

6006LE believes in giving its interns high impact projects. For your first project, you've been tasked with working on the 6006LE core search functionality. Specifically, you need to implement various forms of a FIND($pattern$, $document$) function, that searches for a given string $pattern$ inside of a larger $document$ string. Both the $pattern$ and the $document$ strings are written in an alphabet A with a letters.

Your colleague suggested that you should use rolling hashing, and gave you a helper library (`rolling_hash.py`) that you can import and use. Specifically, you will be using a rolling hash function H that maps strings to the range $[0, 1, \ldots, m-1]$, and is given by:

$$H(c_1c_2c_3\cdots c_k) = \left(c_1a^{k-1} + c_2a^{k-2} + c_3a^{k-3} + \cdots + c_ka^0\right) \mod m.$$

Here, $c_i \in A$ for $i = 1, 2, \ldots, k$ are characters from the alphabet A. If you have computed the hash of a string $c_1c_2c_3\cdots c_k$, and you want to compute the hash of $c_2c_3\cdots c_kc_{k+1}$ from some $c_{k+1} \in A$, you can "roll the hash forward" as follows:

$$\begin{aligned} H(c_2c_3\cdots c_kc_{k+1}) &= c_2a^{k-1} + c_3a^{k-2} + \cdots + c_ka^1 + c_{k+1}a^0 \mod m \\ &= \left(c_1a^{k-1} + c_2a^{k-2} + c_3a^{k-3} + \cdots + c_ka^0 - c_1a^{k-1}\right) \cdot a + c_{k+1} \mod m \\ &= \left(H\left(c_1c_2c_3\cdots c_k\right) - c_1a^{k-1}\right) \cdot a + c_{k+1} \mod m \end{aligned}$$

(a) [5 points] Implement the function ROLLFORWARD that rolls the hash forward: skipping a character from the start of the rolling hash, followed by appending a new character to the end of the rolling hash. Look at the `rolling_hash` library that has been given to you; you will use and modify some of the fields inside the `rolling_hash` instance that is passed to ROLLFORWARD. Your implementation should run in $O(1)$ time.

(b) [10 points] Now, complete the implementation of EXACTSEARCH, which determines if the $document$ string contains the specified $pattern$ string. Your implementation should run in $O(n)$ time, where n is the length of the document.

(*Hint:* You can use part (a).)

(c) [15 points] After finishing your intern project, you realize that you still have plenty of time left until the end your internship! 6006LE is worried that hackers might be corrupting documents by changing some of the document characters. They know which characters could have been changed, but they're not sure what those characters are supposed to be. However, 6006LE still wants to implement a document search.

To approach this, you'd like to write a function called CORRUPTEDSEARCH($pattern$, $document$). In this case, any character of the document could be '?', which means that the character could have been corrupted, and it can match with any character from the alphabet. For example, the *pattern* `abc` matches the *document* `acc?bcc?`, if the first '?' is an 'a.' Notice that the alphabet A does not contain the character '?', which is only used to indicate a corrupted character.

Assume that the document has length n, and the pattern has length k. Using the idea of a rolling hash, design an algorithm to solve CORRUPTEDSEARCH in $O(2^k + n)$ time. As usual, you should justify the correctness and runtime of your algorithm. In addition, discuss *approximately* under what conditions this algorithm performs better than the naive algorithm of iterating through every length-k substring, and checking if any of them match the pattern string.

You do **not** need to implement this function; it suffices to include your answer in your written portion. Note that your implementation must work if the range of the hash function m satisfies $m = \Theta(n)$.