CS2040S: Data Structures and Algorithms

Recitation 4

Goals:

- Recognize tree-related problems
- Learn how tree search can efficiently support various user-defined operations
- Appreciate the data-summarization ability granted by augmenting data structures

Problem 1. (Heights and Grades)

Suppose you are given a set of students with heights and grades as follows:

Name	Height (cm)	Grade (CAP)
Charles	176	4.2
Bob	162	4.5
Mary	180	3.6
John	155	4.1
Wick	186	5.0
Alice	170	3.9

Your goal is to implement an Abstract Data Type (ADT) to efficiently answer the question: "What is the average grade of all students taller than _____?". For instance, the average grade of all students taller than John is $(4.2 + 4.5 + 3.6 + 5.0 + 3.9)/5 = 4.24$.

More specifically, the ADT specifications are as follows:

Operation	Behaviour
`insert(name, height, grade)`	Inserts student into the dataset.
`findAverageGrade(name)`	Returns the average grade among all the students that are taller than the given student.

Problem 1.a. How do you capture the information of each student? What should the data type for each of their attributes be?

Problem 1.b. How do you design a Data Structure (DS) that serves as an efficient implementation of the given ADT? You may assume that name and height are unique.

Problem 1.c. What if `height` is now not unique? What issue(s) will arise from this? How might you modify your solution in the previous part to resolve the issue(s)?

Problem 2. (A Game of Cards)

Suppose you have a deck of n cards and they are spread out in front of you on the table from left to right with each card indexed from i to n. Each card can either be facing up or down. We are tasked to implement an ADT for a magic trick with the following specification:

Operation	Behaviour
`query(i)`	Return whether card at index `i` is facing up or down.
`turnOver(i, j)`	Turn over all cards in the subsequence specified by the index range [`i`, `j`].

Problem 2.a. Given n cards already laid out on the table, how do you design a DS that implements such an ADT? Can you achieve `turnOver` in $O(\log n)$ time *regardless* of the length of subsequence to be turned over? What a magic trick indeed to be able to achieve that!

Problem 3. (Genome Sequence Reconciliation)

Suppose you now work for an international bioinformatics organization which maintains a huge open-access DNA databank (e.g. GenBank). In the bid to find a potential cure for the COVID-19, research labs around the world are racing to sequence a critical strain of the virus. The trouble is, due to experimental noise present in the sequencing process as well as the different methodologies and machinery used, there is bound to be small discrepancies across different sequences reported. Each genome sequence for a particular coronavirus strain is presented as a set of records with each record containing a sequence partition of 60 characters, such as follows:

Starting index	Sequence partition
001	`aaaggtttat accttcccag gtaacaaacc aaccaacttt cgatctcttg tagatctgtt`
061	`ctctaaacga actttaaaat ctgtgtggct gtcactcggc tgcatgctta gtgcactcac`
121	`gcagtataat taataactaa ttactgtcgt tgacaggaca cgagtaactc gtctatcttc`
181	`tgcaggctgc ttacggtttc gtccgtgttg cagccgatca tcagcacatc taggtttcgt`
241	`ccgggtgtga ccgaaaggta agatggagag ccttgtccct ggtttcaacg agaaaacaca`
...	...

Table 1: Source: Severe acute respiratory syndrome coronavirus 2 2019-nCoV/Japan/KY/V-029/2020 RNA, complete genome

Your task is therefore to aid the scientific community by consolidating the (slightly) different sequences reported into a single canonical sequence. In order to achieve that, you'll need to first detect the inconsistencies between any 2 reported sequences by identifying their records that differ from one another. As you would know, genome sequences are *incredibly* long and you have many such pairs of record sets to compare, so you need to devise a really efficient method to help you!

Problem 3.a. Given 2 sets of n records each, design a tree-based DS with operations allowing you to quickly determine which are the inconsistent records.