CS2040S: Data Structures and Algorithms

Recitation 5

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.

- height tree → tree ordered by height
- traverse the whole tree

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 $[\texttt{i}, \texttt{j}]$.

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. (Drug Discovery)

In the bid to find a potential cure for the COVID-19, research labs around the world are racing to study the effects different drugs have on the coronavirus. Viruses respond to drugs via *mutations*. A mutation occurs when there are changes (often small subsequences) along the original genome sequence. Suppose you now work for an international bioinformatics organization which maintains a huge open-access and canonical DNA databank (e.g. GenBank). Your organization is supporting the drug discovery effort by not only releasing the canonical sequence of the virus to labs all over the world, but also delivering it in a special format that would efficiently facilitate the comparisons between different mutations. In this way, all a research lab needs to do is to first download a local copy of the canonical unmutated sequence (in the special format), update it with the mutations from their experimental results, then quickly compare it with the mutated sequence from another lab which ran a different drug experiment. The mutational differences and similarities of the virus in response to different drugs reveal important drug properties to scientists, which in turn help them formulate more effective drugs.

In its raw format (i.e. before special formatting), a virus genome sequence is presented as a list of records where each record contains a subsequence of 60 characters with each character representing a nitrogenous base (e.g. Adenine, Guanine, Cytosine, Thymine). This is illustrated in Table 1 below.

Record#	Subsequence
1	AAAGGTTTAT ACCTTCCCAG GTAACAAACC AACCAACTTT CGATCTCTTG TAGATCTGTT
2	CTCTAAACGA ACTTTAAAAT CTGTGTGGCT GTCACTCGGC TGCATGCTTA GTGCACTCAC
3	GCAGTATAAT TAATAACTAA TTACTGTCGT TGACAGGACA CGAGTAACTC GTCTATCTTC
4	TGCAGGCTGC TTACGGTTTC GTCCGTGTTG CAGCCGATCA TCAGCACATC TAGGTTTCGT
5	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

As the chief of data in the organization, you are tasked to design the special format which would make comparisons between 2 genome sequences efficient. Having been a former student of CS2040S, you know that the "special format" necessarily entails a clever DS that is well-suited for the comparison operation.

Problem 3.a. Design a tree-based DS that can capture a list of n records (i.e a genome sequence) such that when given the tree for another list (i.e a mutated sequence), you can *efficiently* (read: better than $O(n)$ time) determine which are their records that differ from one another.