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 (GPA)
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?

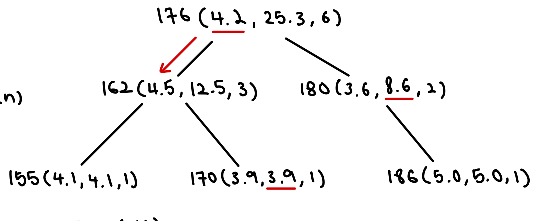
string name
int height
double grade

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.

findAvgGrade(name)
↳ height(name)
↳ find Total Grade(height)
↳ findTotalCount(height)

-height(name)
HashTable O(1)

- findTotalGrade(height)
  ↳ naive: iterate everything O(n)
  ↳ try BST (AVL) O(log n)
  - augment
    - grade
    - totalGrade
    - count



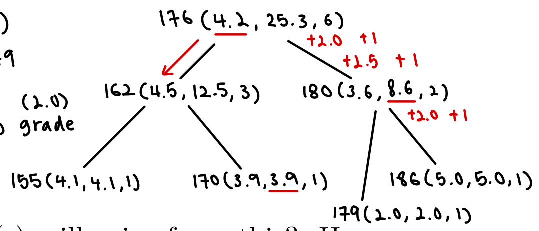
findTotal (162)

```
algo: init totalTaller=0
           count=0
      Traverse → Left
            ↳ totalTaller += right.totalGrade + current.grade
              count += right.count + 1
           End : totalTaller += right.totalGrade
                 count += right.count
```

Insert ("Alpha", 179, 2.0)

HashTable ["Alpha"] = 179

for every node traversed
- increment totalGrade by grade (2.0)
- increment count by 1



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)?

Insert ("Beta", 176, 2.5)

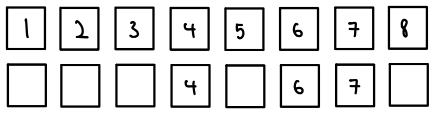
Just insert and combine with the current key, since individual statistic is not important

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:

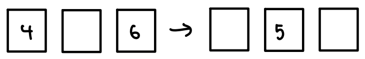
Operation	Behaviour
`query(i)`	Return whether card at index `i` is facing up or down.
`turnOver(i, j)`	Turn over all cards in the subsequence specified by the index range [`i`, `j`].

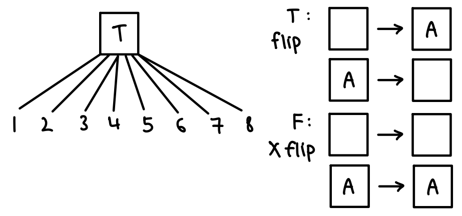
Problem 2.a. Given n cards already laid out on the table, how do you design a DS that implements such an ADT? Can you achieve `turnOver` in $O(\log n)$ time *regardless* of the length of subsequence to be turned over? What a magic trick indeed to be able to achieve that!

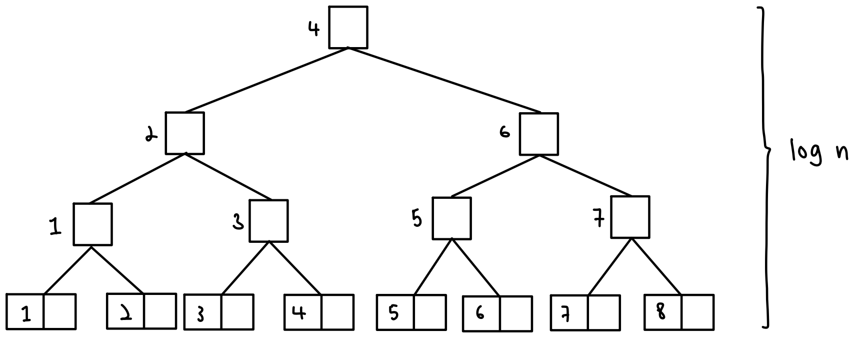


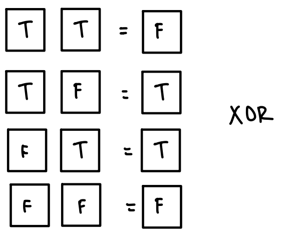
initial position

- query (i): whether card faced up/down
- turn(i, j) : flip i..j





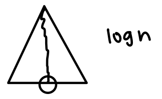




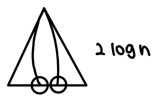
- query (4)
- turn (1, 8) → set root to T
- turn (1, 4) → set left child of parent to T
- turn (4, 7) → turn (1, 3) + turn (1, 7)

turn (i, j) = turn (1, i−1) + turn (1, j)

Bad Cases:
- turn (i, j) → flipping just the one card (leaf)



- turn ($\lfloor \frac{n}{2} \rfloor$, $\lfloor \frac{n}{2} \rfloor + 1$) – e.g. turn (4, 5)



- turn (2, n−1)