CS2040S: Data Structures and Algorithms

Discussion Group Problems for Week 6

For: February 14–February 18

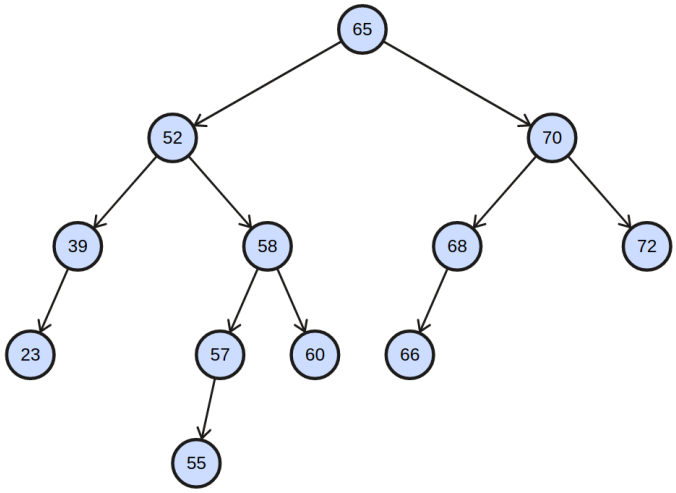
1 Check in and PS4

Discuss questions, if you have any, with the tutor and the rest of the class, about the material and content so far.

2 Problems

Problem 1. Trees Review

The diagram below depicts a BST.



Problem 1.a. Trace the deletion of the node with the key 70.

1.find the successor of the key 70, which is 72
2.replace 70 with 72
3.let the left sub-tree of 72 point to 68

Problem 1.b. Identify the roots of all maximally imbalanced subtrees in the original tree. A maximal imbalanced tree is one with the minimum possible number of nodes given its height h.

All of them

Problem 1.c. During lectures, we've learnt that we need to store and maintain height information for each AVL tree node to determine if there is a need to rebalance the AVL tree during

insertion and deletion. However, if we store height as an `int`, each tree node now requires 32 extra bits. Can you think of a way to reduce the extra space required for each node to 2 bits instead?

11 -> equal
10 -> left heavy
01 -> right heavy
00 -> leaf

if u is left heavy, and its left child v change from equal to left / right heavy, then u need to be rebalanced by right rotation

Problem 2. Chicken Rice

Imagine you are the judge of a chicken rice competition. You have in front of you n plates of chicken rice. Your goal is to identify which plate of chicken rice is best.

Problem 2.a. A simple algorithm:

- Put the first plate on your table.
- Go through all the remaining plates. For each plate, taste the chicken rice on the plate, taste the chicken rice on the table, decide which is better. If the new plate is better than the one on your table, replace the plate on your table with the new plate.
- When you are done, the plate on your table is the winner!

Assume each plate begins containing n bites of chicken rice. When you are done, in the worst-case, how much chicken rice is left on the winning plate?

n^2 - 2n + 2

Problem 2.b. Oh no! We want to make sure that there is as much chicken rice left on the winning plate as possible (so you can take it home and give it to all your friends). Design an algorithm to maximize the amount of remaining chicken rice on the winning plate, once you have completed the testing/tasting process. How much chicken rice is left on the winning plate? How much chicken rice have you had to consume in total? (Give a tight asymptotic bound!)

Problem 2.c. Now I do not want to find the best chicken rice, but (for some perverse reason) I want to find the median chicken rice. Again, design an algorithm to maximize the amount of remaining chicken rice on the median plate, once you have completed the testing/tasting process. How much chicken rice is left on the median plate? How much chicken rice have you had to consume in total? (Give a tight asymptotic bound. If your algorithm is randomized, give your answers in expectation.)

use AVL tree and store the weight of every sub tree

Problem 3. Economic Research

You are an economist doing a research on the wealth of different generations in Singapore. You have a huge (anonymised) dataset that consists of ages and wealth, for example, it looks something like:

```
1  24  150,000
2  32   42,000
3  18    1,000
4  78  151,000
5  60  109,000
...
```

1. Build a segment tree, each node is an interval of age.
2. Sort the dataset based on age.
3. Use two pointers, fix the left pointer, move right pointer one step at a time until the sum of [l, r] is sum / k. Then move left pointer to the right pointer

That is, each row consists of a unique identifier, an age, and a number that represents their amount of wealth.

Your goal is divide the dataset into "equi-wealth" age ranges. That is, given a parameter k, you should produce k different lists as output $A_1, A_2, ..., A_k$ with the following properties:

1. All the ages of people in set A_j should be less than or equal to the ages of people in A_{j+1}. That is, each set should be a subset of the original dataset containing a contiguous age range.

2. The sum of wealth in each set should be (roughly) the same (tolerating rounding errors if k does not divide the total wealth, or exact equality is not attainable).

In the example above, if taking the first five rows and $k = 3$, you might output `(3,1),(2,5),(4)`, where the age ranges are $[0, 30), [30, 70), [70, \infty)$ respectively, with the same total wealth of $151,000$.

Notice this means that the age ranges are not (necessarily) of the same size. There are no other restrictions on the output list. You should assume that the given k is relatively small, e.g., 9 or 10, while the dataset is very large, e.g., the population of Singapore. Also note that the dataset is unsorted.

Design the most efficient algorithm you can to solve this problem, and analyse its time complexity.

Problem 4. Order Maintenance

Design a data structure for Order Maintenance. The goal here is to maintain a total order over some arbitrary objects. The data structure supports two operations:

- `InsertBefore(A, B)`: insert B immediately before A.
- `InsertAfter(A, B)`: insert B immediately after A.
- `IsAfter(A, B)`: is B after A in the total order?

Notice that the insert operation `InsertAfter(A, B)` adds B *immediately* after A, while the query operation `IsAfter(A, B)` asks whether B is *anywhere* after A in the total order. The expected complexity of each operation is $O(\log n)$, where n is the number of items in the data structure.

Construct a balanced BST. InsertBefore(A, B) is to insert B as A's left child.InsertAfter(A, B) is to insert B as A's right child. We keep track of the insertion so for each node we will have a insertion path, for example {left, right, left, left}. We use 0 to represent left and 1 to represent right. So for IsAfter(A, B) we just need to

For this question, you may assume that when `IsAfter(A, B)` is called, you're given the nodes corresponding to A and B. Also, if you are using balanced trees in your solution, you don't need to handle the rebalancing yourself and you can assume that it works magically!

Problem 5. (Optional) Ancestor Queries

Our job is now to *simulate* a binary tree. Each node has zero, one, or two children, and the tree is of height h. Unfortunately, it is not a balanced tree. By preprocessing the binary tree, design and implement an auxiliary data structure to support the following operations efficiently:

- `InsertLeft(x, y)`: insert y as a left child of x in the binary tree.
- `InsertRight(x, y)`: insert y as a right child of x in the binary tree.
- `IsAncestor(x, y)`: is x an ancestor of y in the binary tree that contains them?

Hint: Think about how you answer the `IsAncestor(x, y)` query without the extra data structure. What would the cost of that operation be? How can you improve this?