CS2040S: Data Structures and Algorithms

Discussion Group Problems for Week 6

For: February 13–February 17

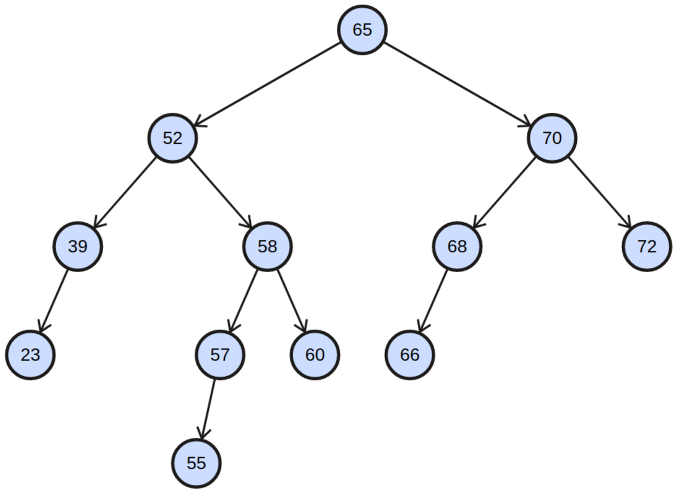
1 Check in and PS4

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.

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

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?

Problem 1.d. Given a pre-order traversal result of a binary search tree T, suggest an algorithm to reconstruct the original tree T.

Problem 2. Iterative BFS and DFS

During lecture, we've learnt how to do tree traversal in various ways. For this question, we'll focus on DFS and BFS. Since you already know how to use DFS and BFS to traverse a tree recursively, can you propose a way using non-recursive DFS and BFS to traverse a tree? Write your answer in the form of pseudocode.

Problem 3. 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 3.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?

Problem 3.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 3.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.)

Problem 4. 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
...
```

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

Your goal is to divide the dataset into "equi-wealth" age ranges. That is, given a parameter k, you should produce k different age ranges $A_1, A_2, ..., A_k$ 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 5. Height of Binary Tree After Subtree Removal Queries

You are given the root of a binary tree with n nodes. Each node is assigned a unique value from 1 to n. You are also given an array of queries of size m.

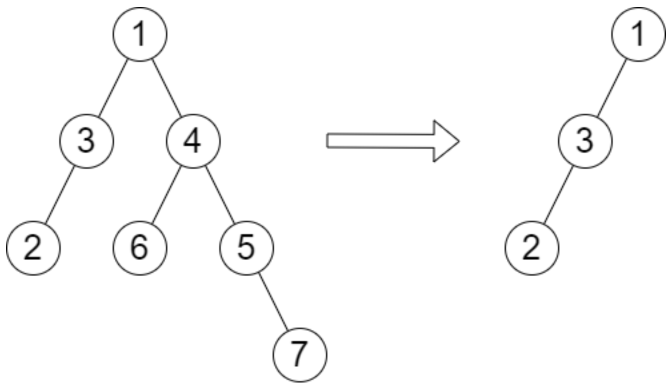
You have to perform m **independent** queries on the tree where in the i-th query you do the following:

1. Remove the subtree rooted at the node with the value $queries[i]$ from the tree. It is guaranteed that $queries[i]$ will not be equal to the value of the root.

2. Return an array answer of size m where $answer[i]$ is the height of the tree after performing the i-th query.

Note that the queries are independent, so the tree returns to its initial state after each query. The height of a tree is the number of edges in the longest simple path from the root to some node in the tree.

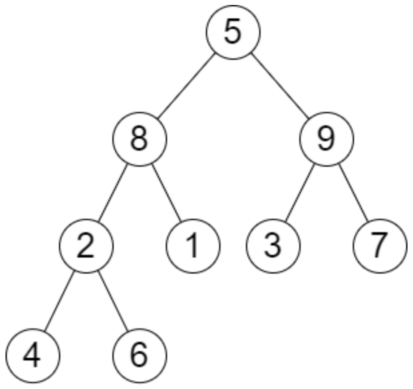
For examples:



Input: Tree data structure as left diagram above, queries = [4]

Output: [2]

Explanation: The diagram above shows the tree after removing the subtree rooted at node with value 4. The height of the tree is 2 (The path 1− > 3− > 2).



Input: root = $[5, 8, 9, 2, 1, 3, 7, 4, 6]$, queries = $[3, 2, 4, 8]$

Output: $[3, 2, 3, 2]$

Explanation: We have the following queries:

1. Removing the subtree rooted at node with value 3. The height of the tree becomes 3 (The path 5− > 8− > 2− > 4).

2. Removing the subtree rooted at node with value 2. The height of the tree becomes 2 (The path 5− > 8− > 1).

3. Removing the subtree rooted at node with value 4. The height of the tree becomes 3 (The path 5− > 8− > 2− > 6).

4. Removing the subtree rooted at node with value 8. The height of the tree becomes 2 (The path 5− > 9− > 3).