CS2040S: Data Structures and Algorithms

Discussion Group Problems for Week 6

For: February 17–February 21

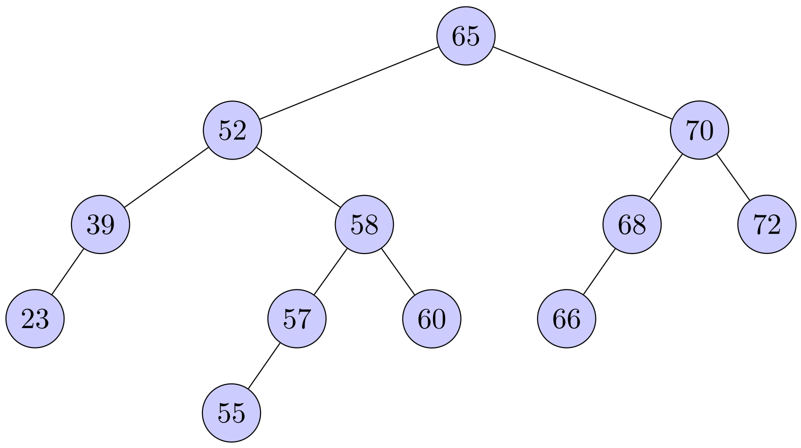
1 Check in and PS4

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. Suppose now that the BST is an AVL Tree. Is the resulting tree balanced? If not, balance the tree.

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

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

Problem 2. Iterative In-Order Tree Traversal

During lecture, we've learnt how to do tree traversal in various ways. For this question, we'll focus on In-Order Traversal. Since you already know how to use In-Order Traversal to traverse a tree recursively, can you propose a way using non-recursive In-Order Traversal 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. However, you have a very poor taste memory! You can only compare the plates of chicken rice pairwise and you forget the taste of both plates immediately after comparing them.

Problem 3.a.   A simple algorithm:

- Put the first plate on your table.
- Go through the remaining plates. Take a bite out of the chicken rice on the table and the chicken rice on the new plate to compare them. If the new plate is better than the one on the 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 contains n bites of chicken rice in the beginning. 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. Unification of Valeria

The kingdom of Valeria is divided into numerous factions, each controlling a land ID that determine their territory. However, after years of conflict and political turmoil, the High Council has decided that the factions must be merged into K dominions to restore stability.

You are a member of the High Council and have been tasked with merging the factions. You have been given the following dataset:

```
1  3  150,000
2  4  42,000
3  1  1,000
4  8  151,000
5  7  109,000
...
```

Each row of the dataset consists of a faction's unique identifier, a unique land ID and a number that represents their power level.

Your goal is to divide these factions into k dominions such that each dominion roughly has the same total power level to make sure no dominion can dominate. Furthermore, we want the factions within a dominion to fall under one contiguous range of land IDs so they are neighbouring. This range cannot overlap with another group's range.

That is given a parameter k, you need to output a list of k sets of unique identifiers $(A_1, A_2, \ldots, A_k)$, such that each set has the following properties:

1. All the land IDs in set A_j should be less than or equal to the land ID of faction in A_{j+1}.
2. The sum of power levels in each set should be (roughly) the same (tolerating rounding errors if k does not divide the total power level, 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 land ID ranges are $[0, 4), [4, 8), [8, \infty)$ respectively, with the same total power level of $151,000$.

Notice this means that the land ID ranges are not (necessarily) of the same size. Some factions have merged with others factions during previous conflicts due to which some land IDs could be missing. These IDs don't have to be accounted for in the range. There are no other restrictions on the output list. You should assume that the given k will be relatively small, e.g., 9 or 10, but the dataset can still be very large, e.g., every small faction in Valeria. Also note that the dataset is unsorted.

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

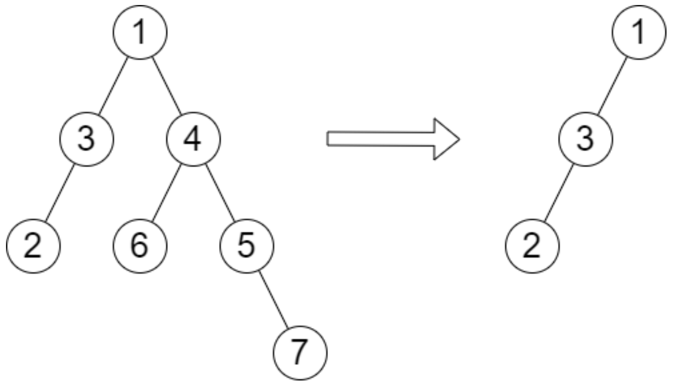
Problem 5. (Optional) Height of Binary Tree After Subtree Removal Queries

You are given the root of a binary tree with n nodes where each node is assigned a unique value from 1 to n. You are also given an array of queries of size m, $queries$, where in the i-th query you do the following: 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.

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 a tree with n nodes, design the most efficient algorithm you can to run all m queries and output an array of size m, $answers$, where $answers[i]$ is the height of the tree after performing the i-th query.

Example 1:

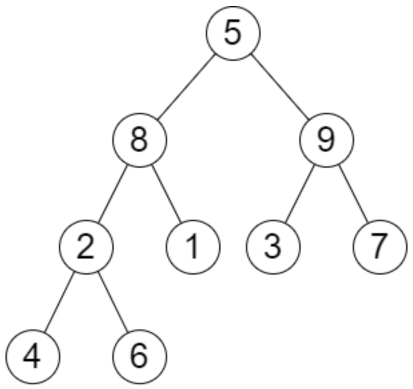


Input: Tree data structure as left diagram above, queries = [4]

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 $1- > 3- > 2$).

Example 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 $5- > 9- > 3$).