CS2040S: Data Structures and Algorithms

Tutorial Problems for Week 6

For: Sep 20, 2019

Problem 1. Harld-Rock Cafe

You are dining at the hottest new restaurant in Singapore: *Harld-Rock Cafe*. As you walk in the door, a bell goes off, and the owner of the restaurant comes out to announce that you are the 100th customer and have won a special deal.

Their menu consists of four components: Appetizers, Soups, Mains, and Desserts. Each component of the menu contains a long and daunting list of items. If you can choose one item from each section that add up to SGD 100, then you can eat for free! Quickly, go at it! Come up with the fastest algorithm you can.

Problem 2. Tree to Tape

Naruto was exploring the office and found a big box of tape storage! Before the advent of hard disks and solid-state drives (SSDs), data used to be stored on magnetic tape. It might surprise you to learn that tape is still being used today; compared to other data storage technologies, tape is cheap and can last decades.

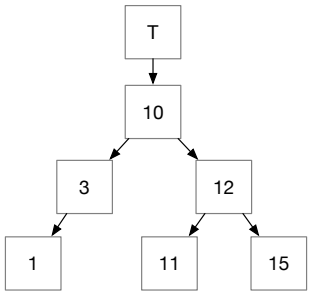
For this problem, we want to save binary search trees (BSTs) to tape for long-term storage. But one limitation of tape drives is their *sequential access* nature; sequential data transfer is fast, but random access is very slow. So, we want to convert a given BST to a *sorted doubly linked list*, which is more easily written to (and read from) the tape drive. Assume you have a standard BST with the following node structure:

Figure 1: A Magnetic Tape Drive. (Image Credit: Wikipedia)

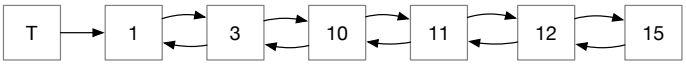
```
class Node {
    Node left
    Node right

    int key
    Object value
}
```

As a specific example, given a tree T:



Your method should create the following sorted doubly linked list:



You can reuse the nodes in the BST by using `left` as the `prev` reference and `right` as the `next` reference in the doubly linked list. Recall that in a doubly linked list, `prev` points to the previous node, and `next` points to the next node.

Problem 2.a. Describe an efficient method that converts a Binary Search Tree (BST) to a doubly-linked list. Be brief but precise. For this problem, assume that the keys are *unique*. Explain the time and space complexity of your algorithm.

Now that we have managed to store our BST to tape, we need a way to convert the stored doubly linked list back to a BST. One trivial way might be just to use the linked list as a lop-sided BST. But this is inefficient since many operations will take $O(n)$ time. Instead, we want to convert the doubly linked list to a *height-balanced* BST.

Problem 2.b. What is a an efficient way to convert the data back from a doubly linked list to a *height-balanced* BST? Provide a description for your algorithm. Be precise, but pseudocode or Java is not necessary unless it helps you to explain. What is the worst-case time and space complexity of your method in terms of the number of nodes n?

Hint: *It is possible to solve this problem in $O(n)$ time and use less than $O(n)$ space.*