DAVP Week 4 and Week 5

Topics are:

1. **Tree Data Structure**
2. **balanced search trees or segment trees**
3. **You can try some advanced ones now**
4. **Red Black Trees Application**

Some guidelines:

1. We would like to have questions for all sorts of people. So it would be a good idea to have questions of each category, at least 9 questions.
    a. Easy - 3
    b. Medium - 3
    c. Hard - 3
2. We can discuss the questions with prof. diwan if we are able to complete it before sunday evening.

Easy

1) **Level-order traversal reconstruction of a BST:** Given a sequence of keys design a linear-time algorithm to determine whether it is the level-order traversal of some BST (and construct the BST itself).

2) **Constant memory:** Describe how to perform an inorder tree traversal with constant extra memory (e.g., no function call stack).

Answer: on the way down the tree, make the child node point back to the parent (and reverse it on the way up the tree). This traversal is called morris traversal.

3) **Isomorphic binary trees:** Two binary trees are said to be isomorphic if one can be obtained from the other by permuting the left and right subtrees of some nodes in the tree. Any number of nodes at any level can have their children swapped. Write a function to test whether two binary trees are isomorphic. Try to make it as efficient as possible. It is possible to do this in O(n) time.

Answer Yes. Traverse both the trees simultaneously.

Medium

1) **Construct all possible BSTs for keys 1 to N:** First find the count of all the possible BSTs and then construction of all possible BSTs.

   Answer: The count is of the form nth catalan number

2) **K'th smallest element in BST using O(1) Extra Space and O(n) time:** Given a Binary Search Tree (BST) and a positive integer k, find the k'th smallest element in the Binary Search Tree. If modification is allowed to the BST then the answer could be found using O(n) extra space and O(logn) time. How?

   Answer: O(n) space & time: inorder traversal retrieves elements of tree in the sorted order
   O(1) extra space and O(n) time is achieved by **morris traversal**.
   O(n) extra space and O(logn) time is achieved by storing the count of left subtree and right subtree nodes at the parent node.
   So combining morris traversal and the ability to have O(n) extra space initially the kth smallest element can be found using O(1) extra space during the run of algo and O(logn) time.

3) **Swapped nodes:** Given a BST in which two keys in two nodes have been swapped, find the two keys and correct them.

   Answer: Consider the inorder traversal a[] of the BST. There are two cases to consider. Suppose there is only one index p such that a[p] > a[p+1]. Then swap the keys a[p] and a[p+1]. Otherwise, there are two indices p and q such a[p] > a[p+1] and a[q] > a[q+1]. Let's assume p < q. Then, swap the keys a[p] and a[q+1].

4) Let p_1,p_2,...,p_n be a permutation of n. An inversion pair is a pair (i,j) such that i < j but p_i > p_j. Describe an O(nlog n) time algorithm using **trees** to find the number of inversion pairs in a given permutation.

   Answer: Divide and conquer uses O(nlogn) time but we need a tree algorithm.
   Here is one possible solution with variation of binary tree. It adds a field called rightSubTreeSize to each tree node. Keep on inserting number into binary tree in the order they appear in the array. If number goes lhs of node the inversion count for that element would be (1 + rightSubTreeSize). Since all those elements are greater than current element and they would have appeared earlier in the array. If element goes to rhs of a node, just increase its rightSubTreeSize. Traverse the tree and find the sum of the inversion counts for each node.

   Suppose q is another permutation. A common inversion pair is a pair (i,j) which is an

inversion pair in both. Can you find the number of common inversion pairs in O(nlog n) time using any method?

Hard

1) A binary tree can be used to represent a boolean function in the following way. Each leaf node stores a value 1 or 0. Ever\y internal node stores a variable. Given the values of the variables, the function is evaluated by traversing a path from the root to a leaf. If the variables in a node is true go left else go right. The value in the leaf node reached is the value of the function.
   Write functions for performing all the basic boolean operations on functions using this representation. Also, write functions to convert between this representation and the sum of products form.

   Answer: Binary tree decision diagrams

2) Given a binary search tree containing the values 1 to n, write a function to compute the number of ways in which the values can be inserted into an empty binary search tree, to get the given tree. Only standard insertion is used without any balancing. For which binary tree is this value maximized?

   Answer:

```
      z
    /   \
   L     R
```

   The root z has to be inserted first. Suppose that S_L is a sequence of elements that, if inserted into a BST, forms the BST L. Similarly, let S_R be a sequence of elements that if inserted into a BST form the BST R. If we consider any possible interleaving of the sequences S_L and S_R, we will end up with a sequence of elements that, if inserted into a BST containing just z, will build up the tree.

   # ways = (# interleaves of S_L and S_R) × (# possible S_Ls) × (# possible S_Rs)

3) The level of a node in a binary tree is the length of the path from the root node to that node. Root is at level 1, its children at level 2 etc. Suppose you are given numbers n_1, n_2,..., n_k. Write down a necessary and sufficient condition for the existence of a binary tree with exactly n_i leaf nodes at level i. If the condition is satisfied, show how you can construct a binary tree with the required property.

Extra Questions tutorial:

1. Consider a data structure in which we want to maintain a sequence of n integer values, assumed to be 0 initially. The operations to be performed are set(i,j,x) : sets the values of a_i, a_{i+1},...,a_j to x. find(i) : returns the value of a_i. Show how these operations

can be performed in O(log n) time.

2. Let a_1,a_2,....,a_n be a sequence of integers. Each a_i has an associated weight w_i, which is again an integer. Both the a_i's as well as the w_i's may be positive or negative. It is required to find a substring of the sequence of a's, say a_i, a_{i+1}, .., a_j such that the sum of the a_i's in the substring is > a given number M, and the sum of weights of a_i's in the substring is as small as possible. If no substring has sum > M, return the empty string. Describe an O(nlog n) time algorithm for this problem.

3. A binary search tree and a circular doubly linked list are conceptually built from the same type of nodes - a data field and two references to other nodes. Given a binary search tree, rearrange the references so that it becomes a circular doubly-linked list (in sorted order).*Hint*: create a circularly linked list A from the left subtree, a circularly linked list B from the right subtree, and make the root a one node circularly linked list. Them merge the three lists.

   Answer: The three circular linked lists can be made in O(n) time, and can be merged in O(1) time.

About red-black trees from wikipedia:

A **red–black tree** is a binary search tree with an extra bit of data per node, its color, which can be either red or black.[2] The extra bit of storage ensures an approximately balanced tree by constraining how nodes are colored from any path from the root to the leaf.[2] Thus, it is a data structure which is a type of self-balancing binary search tree.

Balance is preserved by painting each node of the tree with one of two colors (typically called 'red' and 'black') in a way that satisfies certain properties, which collectively constrain how unbalanced the tree can become in the worst case. When the tree is modified, the new tree is subsequently rearranged and repainted to restore the coloring properties. The properties are designed in such a way that this rearranging and recoloring can be performed efficiently.

The balancing of the tree is not perfect but it is good enough to allow it to guarantee searching in $O(\log n)$ time, where n is the total number of elements in the tree. The insertion and deletion operations, along with the tree rearrangement and recoloring, are also performed in $O(\log n)$ time.[3]

Tracking the color of each node requires only 1 bit of information per node because there are only two colors. The tree does not contain any other data specific to its being a red–black tree so its memory footprint is almost identical to a classic (uncolored) binary search tree.

PS: So learn insertions, deletions, balancing rotations and coloring in a red-black tree using examples. All the above BST problems are in a way red-black tree applications too.