BINARY SEARCH TREE

CONCEPT

- BST is one kind of BTs basically
  - Left child is always less than the root
  - Right child is always greater than the root

PROBLEM SET

- 220. Contains Duplicate III
- 230. Kth Smallest Element in a BST
- 235. Lowest Common Ancestor of a Binary Search Tree
  - Refer to DFS report slides
- 449. Serialize and Deserialize BST
- 450. Delete Node in a BST

NO.220

CONTAINS DUPLICATE III

PROBLEM DESCRIPTION

- Given an array of integers, find out whether there are two distinct indices i and j in the array such that the absolute difference between nums[i] and nums[j] is at most t and the absolute difference between i and j is at most k.

IDEA

- Review No.217 Contain Duplicate and No.219 Contain Duplicate II
  - Next few slides
- No.220 seems no need to use BST, use bucket algorithm instead
- Time complicity of BST is O(nlogk)
- Time complicity of bucket algorithm is O(n)

SOLUTION

- https://github.com/Brady31027/leetcode/tree/master/220_Contains_Duplicate_III

Should have a BETTER solution!!

NO.217

CONTAINS DUPLICATE

PROBLEM DESCRIPTION

- Given an array of integers, find if the array contains any duplicates. Your function should return true if any value appears at least twice in the array, and it should return false if every element is distinct.

IDEA

- Use set() to remove redundant elements
- Compare the length of set_elements and list_elements

SOLUTION

- https://github.com/Brady31027/leetcode/tree/master/217_Contains_Duplicate

NO.219

CONTAINS DUPLICATE II

PROBLEM DESCRIPTION

- Given an array of integers and an integer k, find out whether there are two distinct indices i and j in the array such that nums[i] = nums[j] and the absolute difference between i and j is at most k

IDEA

- Use hash to remember whether the incoming value already existed hash[value] = index
  - If existed, compare their indices
    - If not in the same sliding widow with length *k*, update hash
- Otherwise, add to hash

SOLUTION

- https://github.com/Brady31027/leetcode/tree/master/219_Contains_Duplicate_II

NO.230

K TH SMALLEST ELEMENT IN A BIT

PROBLEM DESCRIPTION

- Given a binary search tree, write a function kthSmallest to find the kth smallest element in it.

IDEA

- Go straight to the left bottom leaf
  - Maintain the traversed path by a stack
  - No left child? Pop the top from the stack
    - Go to its right sub-tree and traverse to the left bottom leaf

SOLUTION

- https://github.com/Brady31027/leetcode/tree/master/230_Kth_Smallest_Element_in_a_BST

NO.449

SERIALIZE AND DESERIALIZE BST

PROBLEM DESCRIPTION

- Serialization is the process of converting a data structure or object into a sequence of bits so that it can be stored in a file or memory buffer, or transmitted across a network connection link to be reconstructed later in the same or another computer environment.

- Design an algorithm to serialize and deserialize a binary search tree. There is no restriction on how your serialization/deserialization algorithm should work. You just need to ensure that a binary search tree can be serialized to a string and this string can be deserialized to the original tree structure.

- The encoded string should be as compact as possible.

- Note: Do not use class member/global/static variables to store states. Your serialize and deserialize algorithms should be stateless.

IDEA

- Similar to No.297 serialize/deserialize binary tree
- Serialize by using pre-order traversal (dfs)
  - Save nodes' value in a list
  - Join the list to generate a string with separator "(space)"
- Deserialization orders are the same as serialization
  - Go left, then go right
  - Convert serialized string to iterator
  - Use next() to get each node's value and recursively get the next node

SOLUTION

- https://github.com/Brady31027/leetcode/tree/master/449_Serialize_and_Deserialize_BST

NO.450

DELETE NODE IN A BST

PROBLEM DESCRIPTION

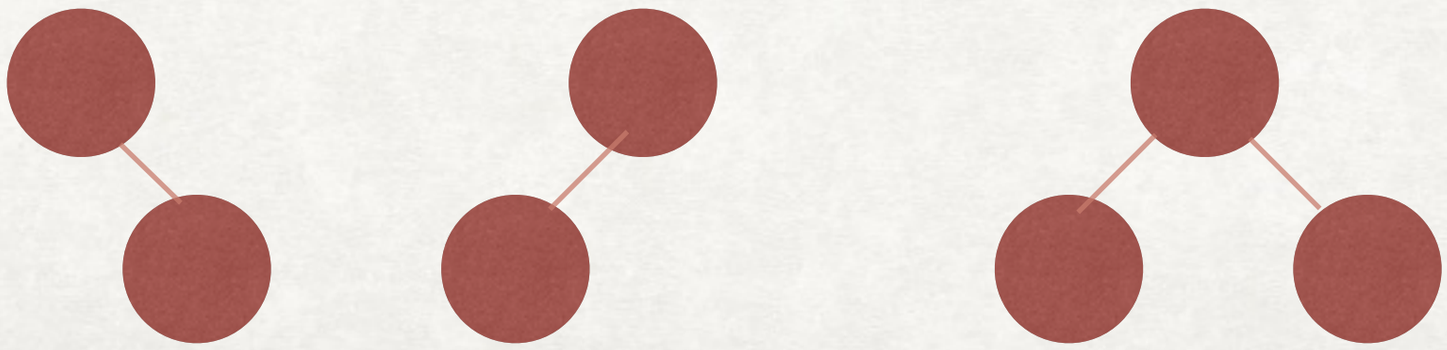
- Given a root node reference of a BST and a key, delete the node with the given key in the BST. Return the root node reference (possibly updated) of the BST. Basically, the deletion can be divided into two stages:

- Search for a node to remove.

- If the node is found, delete the node.

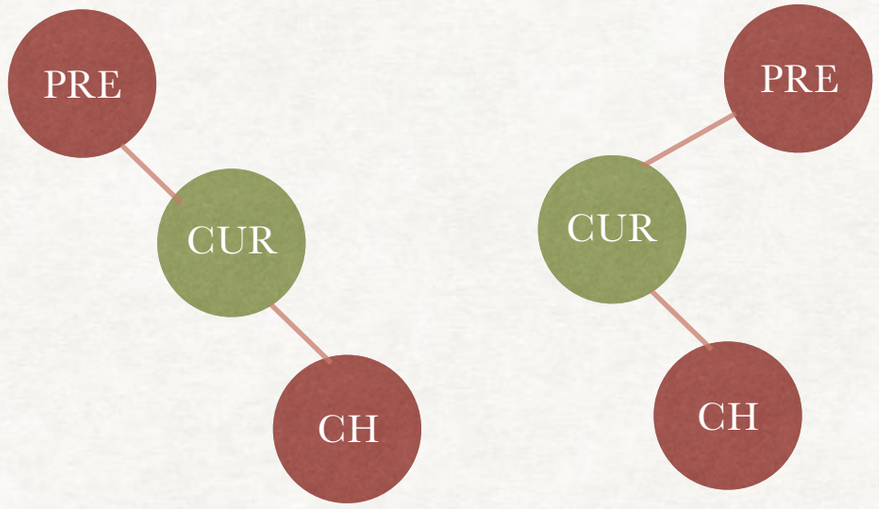
IDEA

- Similar to DFS, recursively traverse the tree until we find the node needed to be deleted
- Three cases if we found the node needed to be deleted
  - Node has no left child
  - Node has no right child
  - Node has left child and right child

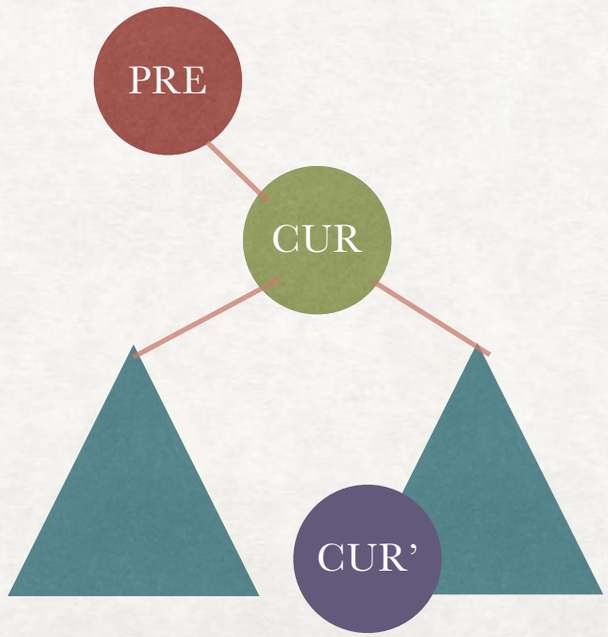


IDEA

- Node has no left child
  - pre.right = ch
  - pre.left = ch



- Node has no right child, similar to “no left child” case
  - pre.right = ch or pre.left = ch
- Node has left child and right child
  - replace cur with cur’



SOLUTION

- https://github.com/Brady31027/leetcode/tree/master/450_Delete_Node_in_a_BST