



A Standardized Approach to Solving Coding Problems

```
object to mirror  
mirror_mod.mirror_object =  
operation == "MIRROR_X":  
mirror_mod.use_x = True  
mirror_mod.use_y = False  
mirror_mod.use_z = False  
operation == "MIRROR_Y":  
mirror_mod.use_x = False  
mirror_mod.use_y = True  
mirror_mod.use_z = False  
operation == "MIRROR_Z":  
mirror_mod.use_x = False  
mirror_mod.use_y = False  
mirror_mod.use_z = True  
selection at the end  
obj.select= 1  
obj.select=1  
context.scene.objects.active  
"Selected" + str(modifier)  
mirror_ob.select = 0  
= bpy.context.selected_object  
data.objects[one.name].select  
print("please select exactly  
-- OPERATOR CLASSES --  
types.Operator):  
X mirror to the selected  
object.mirror_mirror_x"  
error X"  
is not
```

Step 1: Read
the question
thoroughly

Identify the desired outcome
to prevent solving a different
problem due to skimming

Identify the constraints to
prevent a workable approach
that is invalid for problem

Step 2: Identify the pattern to the solution

Using your "hands" (or pencil and paper), solve the problem as you would if no program were involved. Can you identify a pattern to such solution? Does it fit within the specified performance constraint?


The majority of your time should be spent on this step here. If you are already proficient with programming (as well as the needed data structures / algorithms), you can pinpoint the design of the solution without having to write a single piece of code! **Skipping this step to the coding portion can lead you down a rabbit hole due to no clear thought process first and foremost. NEVER CODE WHILE THINKING**

Step 3:
Identify any
relevant
data
structures
or
algorithms
(or both!)

FOCUS ON THE DESIRED
BEHAVIORS OFFERED BY
SUCH DS&A AND HOW THEY
RELATE TO THE PATTERN AT
HAND



IDENTIFY OBVIOUS
IMPROVEMENTS TO
PERFORMANCE IF POSSIBLE
(ASSUMING YOU'RE ALREADY
WITHIN THE PERFORMANCE
CONSTRAINT)



Step 4:
Implement
the code!

This is where we translate
our pseudocode into
actual code

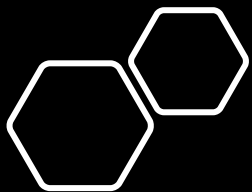
Assuming your original
design was complete, you
should be done!

Application of Concepts

To provide a better perspective on this process, we will be working through an actual problem with a low acceptance rate

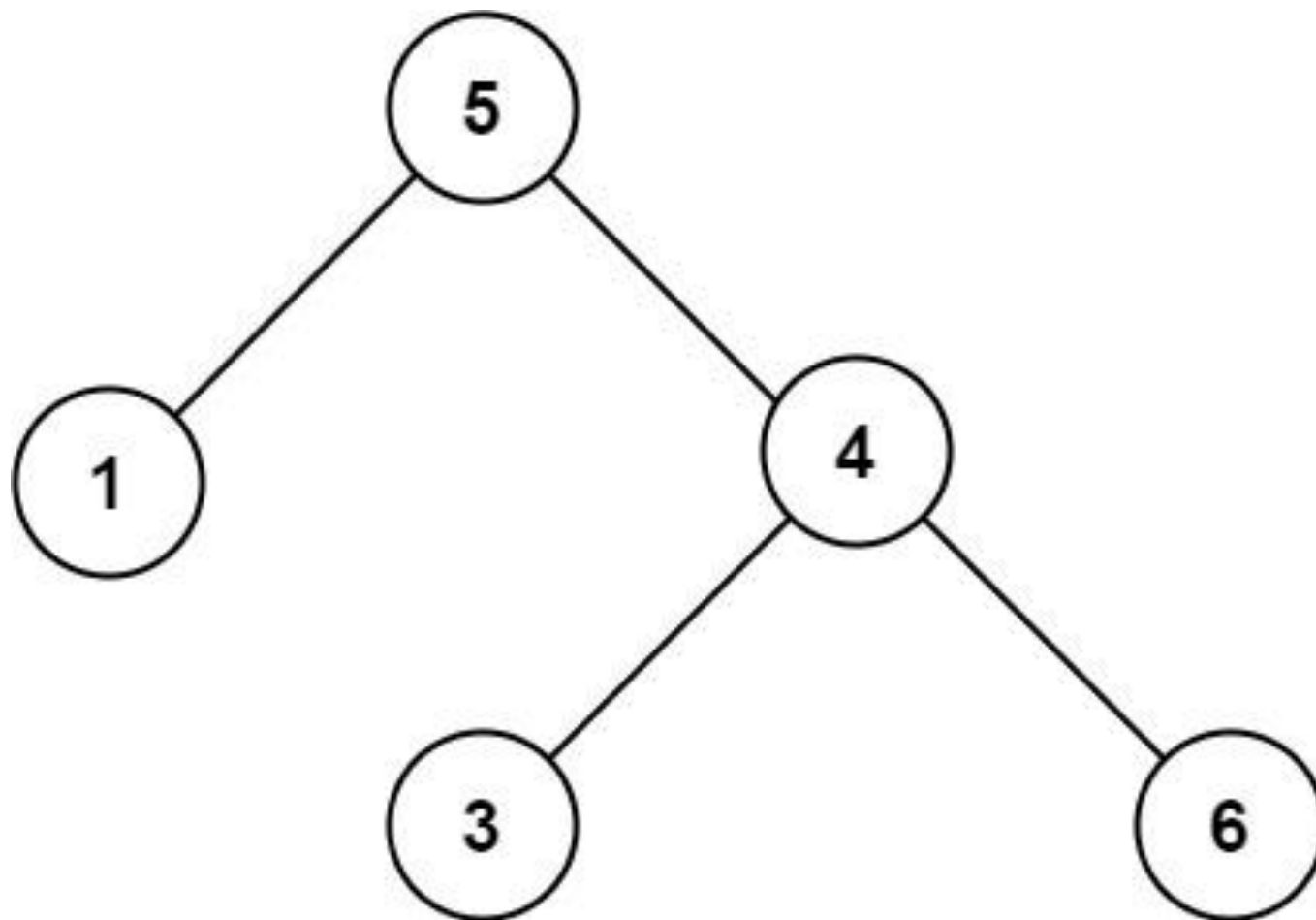
Focus on how each of the steps relate to the problem

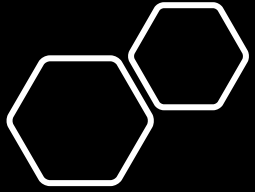
Consider how this process relates to your own experiences



Validate Binary Search Tree (98)

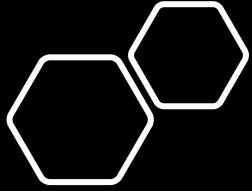
- Given the root of a binary tree, determine if it is a valid binary search tree (BST).
- Accepted: 29%
- Problem Source:
<https://leetcode.com/problems/validate-binary-search-tree/>





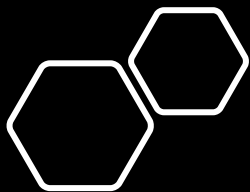
Step 1a:
Break the
problem up
into bullet
points

- Validate binary search tree
- Left subtree keys must be less than the current key
- Right subtree keys must be greater than the current key
- Subtrees must be binary search trees.



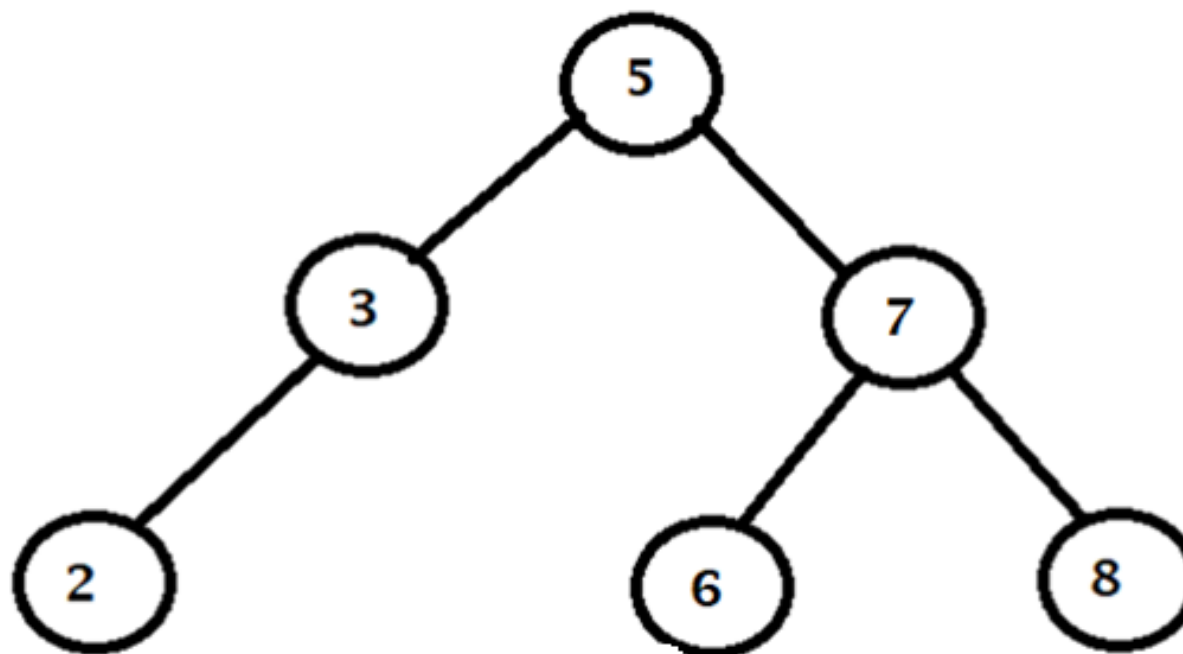
Step 1b:
Identify any
constraints
within the
problem

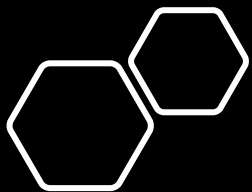
- There will be a minimum of 1 node with a max of 10^4 nodes
- We will have a lowest value of - (2^{31}) and a max value of $(2^{31}) - 1$ for our node values [Note: These are the min/max values for a 32-bit integer type]



Step 2: Identify the patterns to the solution

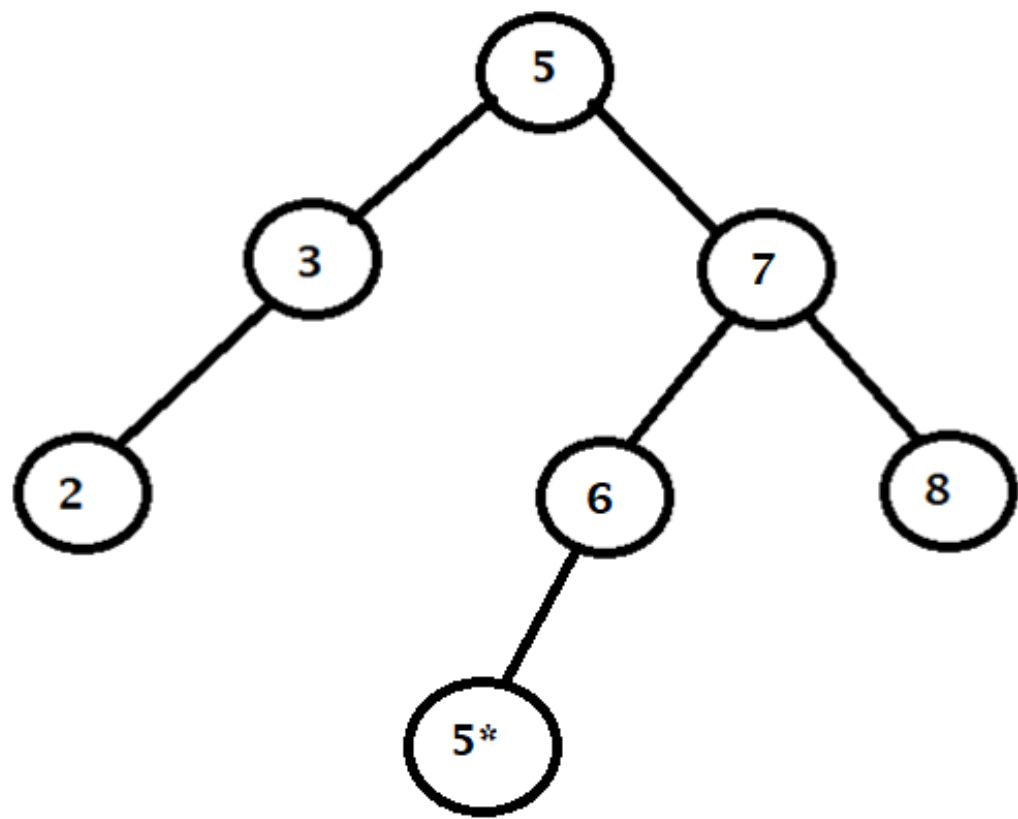
- While we can quickly see that this is a valid BST, **our programs need to be told explicitly what we're doing implicitly.**
- 6 is valid since it is greater than 5 (a valid node) and less than 7 (a valid node)
- 2 is valid since it less than 3 (a valid node), which is less than 5 (a valid node). The same pattern occurs for 8.
- Notice that 2 does not have a minimum bound (nothing to its left) and 8 does not have a maximum bound (nothing to its right)

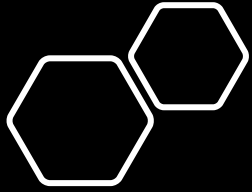




Step 2 (Continued)

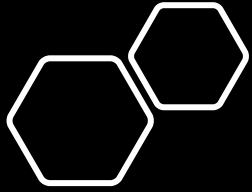
- We now utilize an invalid BST in order to gain additional perspective
- 5^* is less than 6 (valid node), but it is not greater than 5 (valid node).
- Note: Although we are dealing with a potentially large tree in our LC problem, we would be able to discern whether any particular node is valid if given it visually since we are implicitly looking at what the local minimum and maximum values are.
- 5^* has a local minimum of 5 and a local maximum of 6. Since 5^* is not greater than the local minimum, we know it to be false.





Step 3: Identify Relevant Data Structures/Algorithms

- We need to store data for the node, the local minimum, and the local maximum
- Since we need to iterate through each node while passing its local data (min/max bounds), recursion will be heavily suited for this problem
- We identify 2 cases where we will end the recursive search:
 - We have reached null (e.g. `8.right == null`) and return true
 - We have reached a node that violates the local bounds and return false (e.g. `5* <= 5`)
- Since the entire tree, including subtrees, must be valid, we will use an AND operation when testing the truth values:
 - `return recursive(L) AND recursive(R)`
- In order to handle nonexistent local bounds, using NULL can be an easy identifier



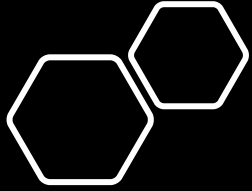
Step 4: Pseudocode

Algorithm isValid(Node, Min, Max)

Input: current node, local min, local max

Output: truth value of validity of BST

1. if Node is null, return true
2. else if (Min is !null AND Node.val <= Min
OR Max is !null AND Node.val >= Max),
return false
3. return isValid(Node.left, Min, Node.val)
AND isValid(Node.right, Node.val, Max)



Step 4: Code [Final]

```
1. public boolean isValidBST(TreeNode root) {  
2.     return processTree(root, null, null);  
3. }  
4.  
5. public boolean processTree(TreeNode node, Integer min, Integer max) {  
6.     if(node == null) return true;  
7.     else if (min != null && node.val <= min || max != null && node.val >= max) return false;  
8.     return processTree(node.left, min, node.val) && processTree(node.right, node.val, max);  
9. }
```

Final Thoughts

- Applying this process helps simplify complex problems as well as producing better code similar to an outline for a paper
- Understanding this process does not replace the fundamentals of Data Structures & Algorithms
- Quality reads on Data Structures & Algorithms:
 - Algorithms 4th Edition
 - Introduction to Algorithms (commonly known as CLRS)
 - Concrete Mathematics
 - The Art of Computer Programming