**Binary Search Tree (BST) Cheat Sheet**

This cheat sheet contains the most essential Binary Search Tree (BST) operations and code templates in Python. – MD ARAFAT KOYES

It also includes possible exam questions that are commonly asked during DSA exams.

**1. BST Class Structure**

class BST: class Node: # Node's constructor with data and left, right pointers def \_\_init\_\_(self, value=None, left=None, right=None):

self.value = value self.left = left self.right = right

# BST's constructor with an empty root def \_\_init\_\_(self): self.root = None **2. Insert a Node (Iterative)**

def insert(self, data): if self.root is None:

self.root = BST.Node(data) # Create root if empty return curr = self.root while True:

if data < curr.value: if curr.left is None:

curr.left = BST.Node(data) # Insert on left break else:

curr = curr.left else: if curr.right is None:

curr.right = BST.Node(data) # Insert on right break else: curr = curr.right **3. Search for a Value (Iterative)**

def search(self, data): curr = self.root while curr is not None: if data < curr.value:

curr = curr.left # Go left if smaller elif data > curr.value:

curr = curr.right # Go right if larger else:

return curr # Return node if found return None # Return None if not found **4. Inorder Traversal (Recursive)**

def print\_inorder(self, subtree): if subtree is not None:

self.print\_inorder(subtree.left) # Visit left subtree print(subtree.value, end=" ") # Print current node self.print\_inorder(subtree.right) # Visit right subtree

**5. Find Minimum and Maximum Value**

def find\_min(self): curr = self.root while curr.left is not None:

curr = curr.left return curr.value def find\_max(self): curr = self.root while curr.right is not None:

curr = curr.right return curr.value **6. Delete a Node from BST**

def delete\_node(self, subtree, data): if subtree is None: return subtree # Node not found

if data < subtree.value:

subtree.left = self.delete\_node(subtree.left, data) # Go left elif data > subtree.value:

subtree.right = self.delete\_node(subtree.right, data) # Go right else: # Node found - 3 possible cases: if subtree.left is None: # Case 1: No left child return subtree.right elif subtree.right is None: # Case 2: No right child return subtree.left else: # Case 3: Two children - find inorder successor successor = self.get\_min(subtree.right) # Leftmost of right subtree subtree.value = successor.value subtree.right = self.delete\_node(subtree.right, successor.value) return subtree

def get\_min(self, subtree):

curr = subtree while curr.left is not None:

curr = curr.left return curr

1. **Find Inorder Successor**

def inorder\_successor(self, value):

curr = self.root successor = None

while curr is not None: if value < curr.value: successor = curr # Possible successor if we go left curr = curr.left elif value > curr.value: curr = curr.right else: if curr.right is not None: successor = curr.right while successor.left is not None: successor = successor.left break return successor

1. **Print Values Between Two Ranges**

def print\_between(self, min, max): def inorder\_print(subtree): if subtree is None:

return if min < subtree.value:

inorder\_print(subtree.left) # Visit left if possible if min <= subtree.value <= max:

print(subtree.value, end=" ") # Print in range if max > subtree.value: inorder\_print(subtree.right) # Visit right if possible

inorder\_print(self.root) print("") # Print newline after finishing

**Important Questions for Exam**

1. Write a function to insert a value in a BST (both recursive and iterative).
2. Implement a function to search for a value in a BST.
3. Perform inorder, preorder, and postorder traversals.
4. Find the minimum and maximum value in a BST.
5. Write a function to delete a node from a BST.
6. Find the inorder successor of a given node.
7. Print all values between two given values in a BST.

**Answers to Important Questions**

**1. Write a function to insert a value in a BST (both recursive and iterative).**

# Iterative Insert def insert(self, data): if self.root is None:

self.root = BST.Node(data) return curr = self.root while True: if data < curr.value:

if curr.left is None:

curr.left = BST.Node(data) break else:

curr = curr.left else: if curr.right is None:

curr.right = BST.Node(data) break else: curr = curr.right

# Recursive Insert def ins(self, data, subtree): if subtree is None:

return BST.Node(data) elif data < subtree.value:

subtree.left = self.ins(data, subtree.left) else:

subtree.right = self.ins(data, subtree.right) return subtree

def recursive\_insert(self, data): self.root = self.ins(data, self.root)

**2. Implement a function to search for a value in a BST.**

# Iterative Search def search(self, data): curr = self.root while curr is not None: if data < curr.value: curr = curr.left elif data > curr.value: curr = curr.right else: return curr return None

# Recursive Search def r\_search(self, data, subtree): if subtree is None: return None elif data < subtree.value:

return self.r\_search(data, subtree.left) elif data > subtree.value:

return self.r\_search(data, subtree.right) else: return subtree

def search\_recursive(self, data): return self.r\_search(data, self.root)

**3. Perform inorder, preorder, and postorder traversals.**

# Inorder Traversal

def print\_inorder(self, subtree): if subtree is not None:

self.print\_inorder(subtree.left) print(subtree.value, end=" ") self.print\_inorder(subtree.right)

# Preorder Traversal def print\_preorder(self, subtree): if subtree is not None:

print(subtree.value, end=" ") self.print\_preorder(subtree.left) self.print\_preorder(subtree.right)

# Postorder Traversal def print\_postorder(self, subtree): if subtree is not None:

self.print\_postorder(subtree.left) self.print\_postorder(subtree.right) print(subtree.value, end=" ")

**4. Find the minimum and maximum value in a BST.**

# Find Minimum def find\_min(self): curr = self.root while curr.left is not None:

curr = curr.left return curr.value

# Find Maximum def find\_max(self): curr = self.root while curr.right is not None:

curr = curr.right

return curr.value

**5. Write a function to delete a node from a BST.**

# Delete a Node from BST def delete\_node(self, subtree, data): if subtree is None: return subtree

if data < subtree.value:

subtree.left = self.delete\_node(subtree.left, data) elif data > subtree.value:

subtree.right = self.delete\_node(subtree.right, data) else: if subtree.left is None: return subtree.right elif subtree.right is None: return subtree.left else:

successor = self.get\_min(subtree.right) subtree.value = successor.value subtree.right = self.delete\_node(subtree.right, successor.value) return subtree

# Get Minimum Node for Inorder Successor def get\_min(self, subtree):

curr = subtree while curr.left is not None:

curr = curr.left return curr

1. **Find the inorder successor of a given node.**

# Inorder Successor def inorder\_successor(self, value): curr = self.root successor = None

while curr is not None: if value < curr.value: successor = curr curr = curr.left elif value > curr.value: curr = curr.right else: if curr.right is not None: successor = curr.right while successor.left is not None: successor = successor.left break return successor

1. **Print all values between two given values in a BST.**

# Print Values Between Min and Max def print\_between(self, min, max): def inorder\_print(subtree): if subtree is None:

return if min < subtree.value:

inorder\_print(subtree.left) if min <= subtree.value <= max: print(subtree.value, end=" ") if max > subtree.value: inorder\_print(subtree.right)

inorder\_print(self.root) print("")