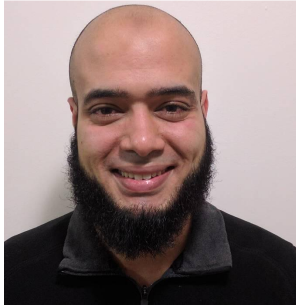
Data Structures

AVL Homework 1

Mostafa S. Ibrahim
Teaching, Training and Coaching for more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)



Problem #1: Lower Bound

- Assume we have a BST of integers, and its inorder traversal is
  - 2, 5, 10, 13, 15, 20, 40, 50, 70
- Implement: `def lower_bound(self, val)`
- Lower-bound finds the first element X, where X >= target
  - In other words, if the target exists within the tree, it will return the 'target' itself
  - If the target itself isn't in the tree, we want to return the smallest value greater than the target (similar to the 'successor' in previous sections)
- Return the value or None if no lower bound
- Input ⇒ output
  - 50 ⇒ 50
  - 51 ⇒ 70
  - 70 ⇒ 70, 71 ⇒ NA, 7 ⇒ 10, 25 ⇒ 40, 60 ⇒ 70

Problem #2: Upper Bound

- Assume we have a BST of integers, and its inorder traversal is
  - 2, 5, 10, 13, 15, 20, 40, 50, 70
- Implement: `def upper_bound(self, val)`
- Upper-bound finds the first element X, where X > target
  - We want the smallest value greater than the target itself (think of it as like finding the successor of the target)
- Return the value or None if no upper bound
- Input ⇒ output
  - 50 ⇒ 70, 51 ⇒ 70, 70 ⇒ NA, 11 ⇒ 13, 20 ⇒ 40, 45 ⇒ 50
- Tip: lower and upper bound are very useful utilities in a BST

Problem #3: Count inversions

- Given an array of **distinct** numbers, count the number of inversions in the array
    - We want the sum of the inversions.
    - For every individual element, how many elements are there **before** it with a **bigger** value?
- E.g. 10, 5, 8, 2, 12, 6
    - 10 ⇒ 0
    - 5 ⇒ 1 {5}
    - 8 ⇒ 1 {8}
    - 2 ⇒ 3 {10, 5, 8} // These 3 numbers are A) Each before 2 B) Each > 2
    - 12 ⇒ 0
    - 6 ⇒ 3 {10, 8, 12}
    - **Total: 8**
- Find an O(nlogn) solution based on the AVL tree
    - You may assume this is the only usage for the tree

Problem #4: Priority Queue

- Priority queue is a queue in which each element has a "**priority**" associated with it. Elements with **higher priority** are always served before those of lower priority
- Assume, in an OS, we have a number of tasks; each with a specific priority (and all are positive values)
  - Assume we enqueued as follows:
  - Enqueue (task_id = 1131, priority = 1)
  - Enqueue (task_id = 3111, priority = 3)
  - Enqueue (task_id = 2211, priority = 2)
  - Enqueue (task_id = 3161, priority = 3)
  - Let’s print tasks in order: 3111 3161 2211 1131
- Implement a priority queue based on the AVL tree code
  - Your tree can't have multiple nodes of the same priority. Strictly 1 node per priority!
  - Enqueue and dequeue should both be O(log n) for time complexity

Problem #4: Priority Queue

```python
tasks = PriorityQueue()

tasks.enqueue(1131, 1)
tasks.enqueue(3111, 3)
tasks.enqueue(2211, 2)
tasks.enqueue(3161, 3)

print(tasks.dequeue())  # 3161
print(tasks.dequeue())  # 3111

tasks.enqueue(1535, 1)
tasks.enqueue(2815, 2)
tasks.enqueue(3845, 3)
tasks.enqueue(3145, 3)

while not tasks.empty():
    print(tasks.dequeue(), end=' ')
# 3145 3845 2815 2211 1535 1131
```

Problem #5: Min nodes from AVL height

- What is the **minimum** number of nodes in an AVL tree of height H?
  - The Sequence is (from height = 0): 1, 2, 4, ?, 12, ?, 54, ?, 143
  - Draw out the trees, and guess the values of some of the question marks
- What is the formula for the sequence?
  - Tip: It is a recursive formula that depends on the last few terms
  - Does this sequence remind you of any similar sequences?
  - Describe an informal mathematical justification for the formula
- Write 2 functions (recursive and iterative versions of the same solution)
  - **def** avl_nodes_rec(height)
  - **def** avl_nodes_iter(height)
- Optional: assume the AVL tree allows |BF| <= k (k = 1 in main version)
  - What is the new recurrence?

Problem #6: AVL Dictionary

- Design an AVL tree of words that can help us know if a word or a prefix exists in our data or not.
  - ABCD prefixes are:
    - “”, A, AB, ABC, ABCD
- What is your time complexity?

```
tree = AVLTree("", True)

tree.insert_string("abcd")
tree.insert_string("xyz")

print(tree.word_exist("abcd"))       # True
print(tree.word_exist("ab"))         # False
print(tree.prefix_exist("abcd"))     # True
print(tree.prefix_exist("ab"))       # True

tree.insert_string("ab")

print(tree.word_exist("ab"))         # True
print(tree.word_exist("cd"))         # False
print(tree.word_exist("abcde"))      # False
```

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."