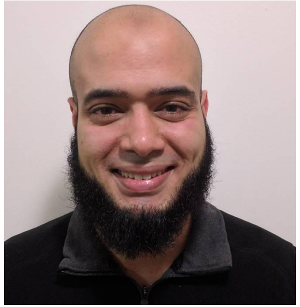
Data Structures

AVL Homework 1

Mostafa S. Ibrahim
Teaching, Training and Coaching since 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 BST, and its inorder traversal is
  - 2, 5, 10, 13, 15, 20, 40, 50, 70
- Implement: pair<bool, int> lower_bound(int target)
- Lower-bound finds the first element X where X >= target
  - In other words, if target exists, it return it
  - Otherwise, the smallest value greater than it (thin like successor)
- The bool is true if found, false otherwise
- Input ⇒ output
  - 50 ⇒ 50, 51 ⇒ 70, 70 ⇒ 70, 71 ⇒ NA, 7 ⇒ 10, 25 ⇒ 40, 60 ⇒ 70

Problem #2: Upper Bound

- Assume we have BST, and its inorder traversal is
  - 2, 5, 10, 13, 15, 20, 40, 50, 70
- Implement: pair<bool, int> lower_bound(int target)
- Upper-bound finds the first element X where X > target
  - The smallest value greater than target (thin like successor of target)
- The bool is true if found, false otherwise
- 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 TO DO

- Given an array of **distinct** numbers, count inversions(array) =
  - Sum of: For every element: how many elements **before** it has **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

Problem #3: Count inversions

- Find O(nlogn) solution based on AVL tree
- You can assume this is the only usage for the tree

```
int main() {
    AVLTree tree;

    //vector<int> v { 10, 5, 8, 2, 12, 6 }; // 8
    vector<int> v { 5, 4, 3, 2, 1};          // 10
    cout << tree.count_inversion(v);
```

Problem #4: Priority Queue

- Priority queue is a queue in which each element has a "**priority**" associated with it. Elements with **high priority** are **served first** before low priority.
- Assume, in an OS, we have tasks each with priority [and positive value]
  - Assume we enqueued as following:
  - 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 avl-tree code
  - Your tree can't have several nodes with same priority. Only 1 node per priority
  - Time complexity should be O(logn) to enqueue or dequeue

Problem #4: Priority Queue

```
PriorityQueue tasks;

tasks.enqueue(1131, 1);
tasks.enqueue(3111, 3);
tasks.enqueue(2211, 2);
tasks.enqueue(3161, 3);
tasks.enqueue(7761, 7);

cout << tasks.dequeue() << "\n";     // 7761
cout << tasks.dequeue() << "\n";     // 3161

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

// 3145 3845 3111 2815 2211 1535 1131
while (!tasks.isempty())
    cout << tasks.dequeue() << " "<<flush;
```

- Observe
  - Value 3161 is added before 3111
  - Value 3161 is printed after 3111
  - This is valid: the constraint is tasks with higher priority are printed first
  - If same priority = print in any order

Problem #5: Min nodes from AVL height TO DO

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

Problem #6: AVL Dictionary TO DO

- Design an AVL tree of words that can help us know if a word or a prefix exists in our data or not.
- What is your time complexity?

```
AVLTree tree;

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

cout<<tree.word_exist("abcd")<<"\n";      // 1
cout<<tree.word_exist("ab")<<"\n";        // 0
cout<<tree.prefix_exist("ab")<<"\n";      // 1

tree.insert_string("ab");

cout<<tree.word_exist("ab")<<"\n";        // 1
cout<<tree.word_exist("cd")<<"\n";        // 0
cout<<tree.word_exist("abcde")<<"\n";     // 0
```

Problem #7: AVL using 1 class TO DO

- The lecture provided the code based on class + struct
- In this one, you will rewrite the whole code using one class only
  - We know its limitations and issues
- Converting the code is classical, but you will face a new problem as the root itself is changing due to the balance operation
- You need to detect the problem and properly solving it
- Note: this is not AVL problem
  It is a consequence of the data-design

```
class AVLTree {
private:
	int data { };
	int height { };
	AVLTree* left { };
	AVLTree* right { };
```

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."