Foundations

Created	@January 8, 2025 9:18 AM
Class	DS 4300
Reviewed	☐

Searching

- Most common operation
- In SQL, the SELECT is most versatile / complex
- Baseline for efficiency is Linear Search: $\theta(n)$
- Record - a collection of values for attributes - row
- Collections - a set of records of the same entity type, a table
- Search Key - A value for an attribute from the entity type
- Contiguously Allocated List
    - All n*x bytes are allocated as a single chunk of memory
- Linked List
    - Each record needs x bytes + additional space for 1 or 2 memory addresses
    - Individual records are linked together in a type of chain using memory addresses
- Arrays are faster for random access, but slow for insertion anywhere besides end
- Linked Lists are faster for inserting anywhere in the list, but slower for random access
- Arrays
    - fast random access
    - slow for random insertions

- Linked Lists
    - slow for random access
    - fast for random insertions
- Binary Search is faster: $\theta(logn)$
- Database Searching:
    - Searching for specific id = fast
    - Searching for specific specialVal needs linear search of that column
    - Can't store data on disk sorted by both id and specialVal (at the same time) → data duplicated, inefficient space
    - Options:
        1. An array of tuples (specialVal, rowNumber) sorted by specialVal
            - We could use Binary Search to quickly located specialVal
            - But every insert into the table would be slow
        2. A linked list of tuples (specialVal, rowNumber) sorted by specialVal
            a. searching for specialVal would be slow - linear scan required
            b. Inserting would be faster

Binary Search Tree

BST

- Binary Search tree - fast insert and fast search
    - Creating / inserting into BST
    - EX: 23, 17, 20, 42, 31, 50

```
    23
 / \
17   42
```

```
\  / \
20 31  50
```

- Always a reference to a root node
- Tree Traversals:
  - Pre-Order
  - Post Order
  - In Order
  - Level Order: BFS search
    - 23 → 17, 42 → 20, 31, 50
    - Python doesn't have queue: use deque
- Want to minimize number of levels

For Homework:

```
class BinaryTreeNode(self, value, left=None, right=None)
    value int
    left BinaryTreeNode
    right BinaryTreeNode

''' creating 23
root = BinaryTreeNode(23)
root.left = BinaryTreeNode(17)
root.right = BinaryTreeNode(42)
```

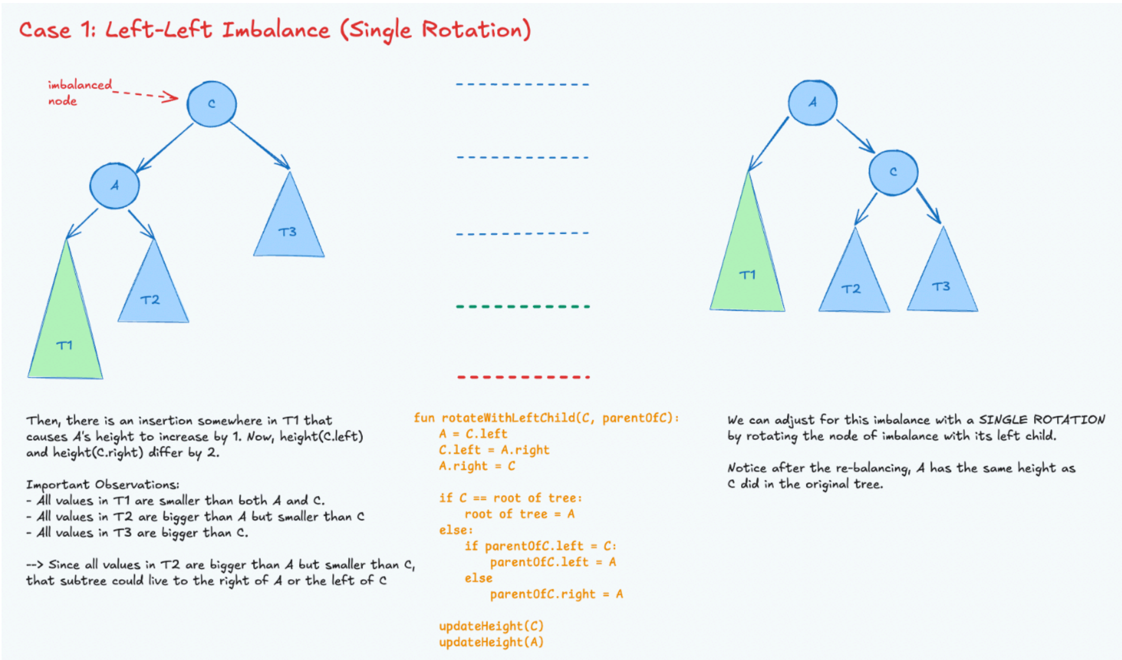
AVL Tree

- Approximately balanced binary search tree
- AVL Property Balancing by = $|h(LST) - h(RST)| \leq 1$
- Self balancing

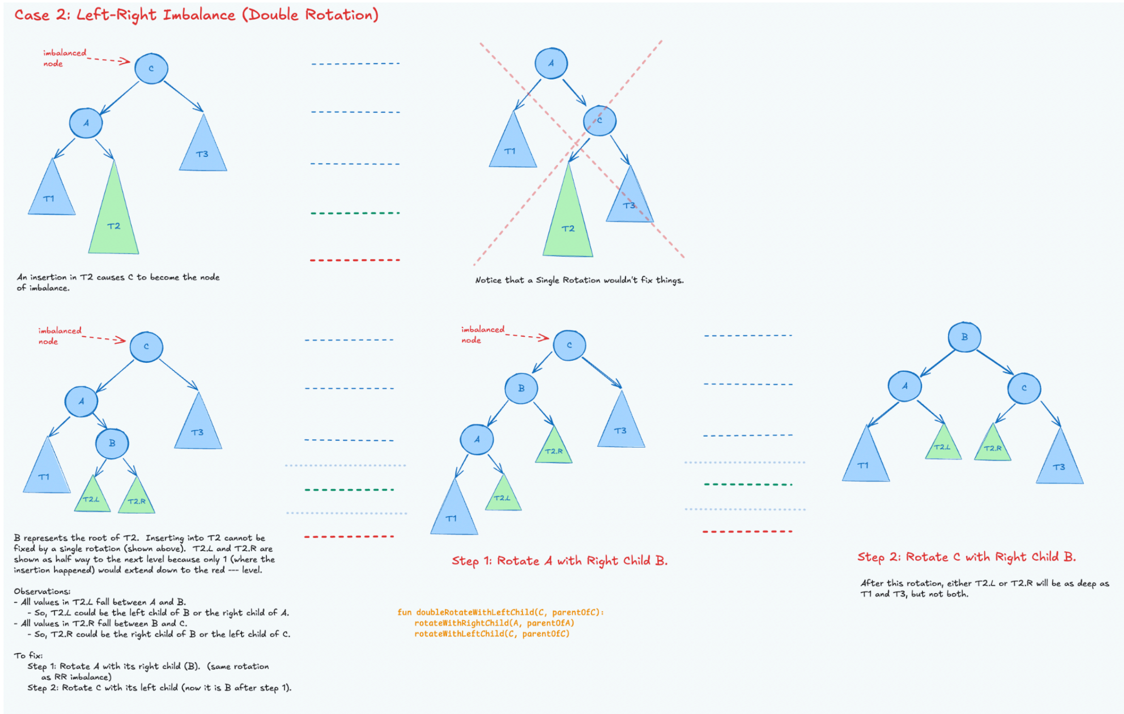
```
      40
    /    \
   20     60
  /  \      \
 10   30     80

 Insert 50

     40
    /    \
   20     60
  /  \    /  \
 10  30  50  80

 Insert 5 then 7

      40
     /    \
    20     60
   /  \    /  \
  10  30  50  80
  /
 5
  \
   7

Unbalanced: node of imbalance is 5 - 7 (denoted by alpha)
To balance:

      40
      /   \
     20    60
    /  \   /  \
   7  30  50  80
  / \
```

```
5 10
```

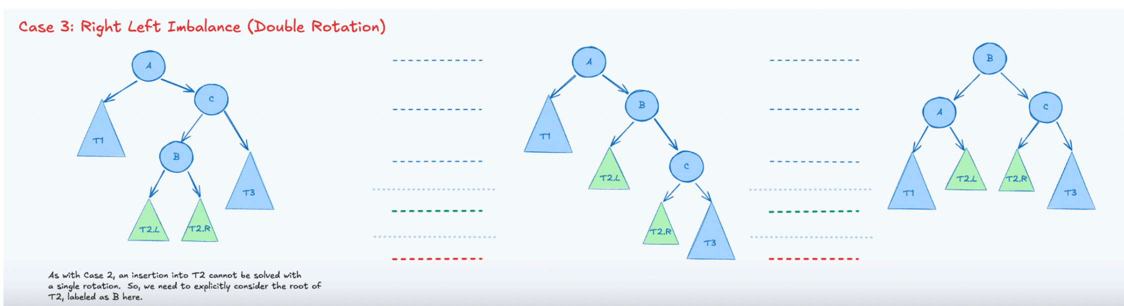
- Four Cases of Imbalance:
    1. Left Left Insertion (LL)



    2. Left Right Insertion (LR)

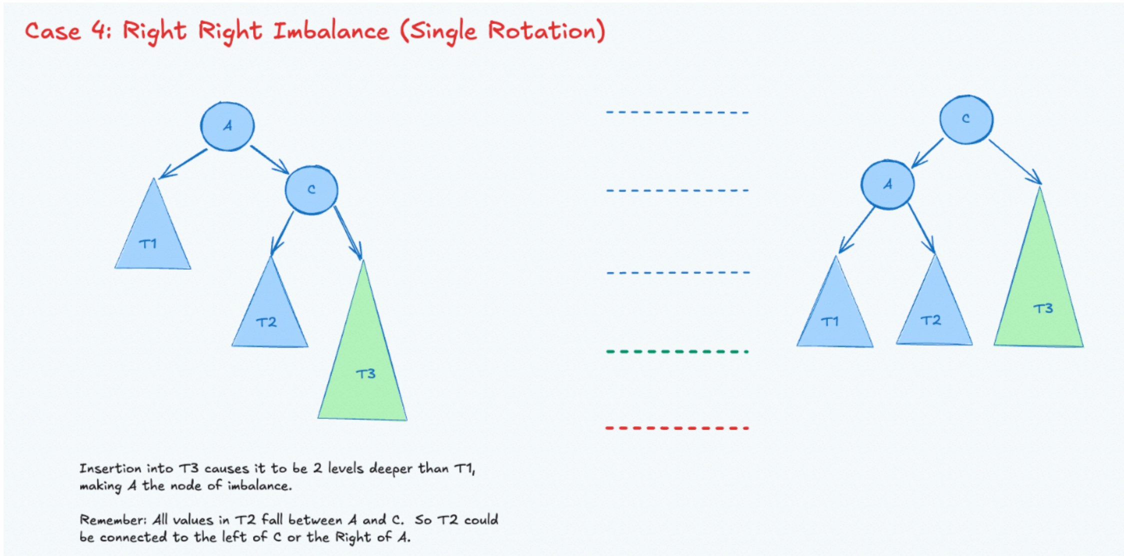


3. Right Left Insertion (RL)



- Rotate B with C, rotate A with B

4. Right Right Insertion (RR)



- Reassign c.left to A, then a.right to T2

- Case 1 & 4 are mirrors; 2 & 3 are also mirrors

-