AVL Tree

Wednesday, January 08, 2025 9:22 AM

Goal is to minimize the height of the tree, which is dictated solely by the order that the values are inserted

- For example 10, 15, 20, 30, 40 inserted in order would have 5 levels – the max
- But it's also very difficult to maintain the order of a balanced tree

AVL Tree - Approximately balanced binary search tree

- Maintains balance factor
- AVL Balance Property: | h(LST) - h(RST) | <= 1
- Self-balancing (with algos)

If you insert 5 then 7 to 10 you get

```
  10
5
  7
```

Which makes 10 a node of imbalance (alpha)

We would want to restructure with 7, the middle value, as the root here

```
  7
5  10
```

4 Cases of Imbalance

1. LL
   - Inserting into the left subtree of the left child of the node of imbalance
2. LR
   - Inserting into the left subtree of the right child of the node
3. RL
   - Inserting into the right subtree of the left child of the node
4. RR
   - Inserting into the right subtree of the right child of the node

Basically just two cases with mirrored directions

- Rebalancing Case 1 (LL):

```
          C
      A      T3   ...
   T1  T2      ...
 ^              ...
```

  Inserting LL into t1 causes C to become imbalanced

  Single Rotation
  - Everything under T1 is < A and < C
  - Everything under T2 is > A and < C
  - Everything under T3 is > C
  - So T2 can be pointed to by the left of C and that can be pointed right of A
  - *Created a temp variable to store C's left child (A) so you don't lose it

```
   A
 T1    C
     T2   T3
```

- Rebalancing Case 2 (LR):

```
      C
   A        t3     ...
T1   T2              ...
      ^              ...
```

  Double Rotation

```
       C
    A         t3     ...
T1      B               ...
    T2L  T2R              ...
```

  Convert this to a Case 1 then apply that rotation

```
          C
```

```
        B          T3
    A       T2R
T1  T2L
```

Now perform Single Rotation

```
          B
      A          C
  T1  T2L    T2R  T3
```

Coding a RR switch
Give the root and child a reference (A and C)
C.left = A
A.right = T2

RL (Case 3)
A = root
C = root.right
B = C.left
T1 = A.left
T3 = C.right
T2L = B.left
T2R = B.right

A.left = T1
A.right = B
C.left = T2R
B.right = C

B.left = A
A.right = T2L
B.right = C