Prepared by Kaveh Kevin Issapour <kki2101@columbia.edu> and Linan Qiu <lq2137@columbia.edu>

AVL Trees

BSTs are not the best. There is no way to ensure a $O(\log N)$ runtime for search in the tree because it may get skewed to one side. In the worst case, the tree would not branch, and would thus have a height of N; of course, in the best case, the height would be $\log N$.

How do we ensure that we get $O(\log N)$ all the time? We have to balance the tree between additions / removals.

An AVL tree is a binary search tree in which the following condition holds after each operation: **For each node, the heights of the left and right sub-trees differ by at most 1.**

To maintain this balance, after each insertion, we find the lowest node that violates the balance condition (if any such node does); then we perform a rotation to re-balance the tree.

Single Rotation

Left Rotation

```
a
 \
  b
   \
    c
```

We must perform a rotation here, rooted at `a`. To achieve this, the following should be done: `b` will be the new root The left child of `b` becomes `a`s right child (in this case, it is null) `a` becomes the left child of `b`

```
  b
 / \
a   c
```

Right Rotation

```
    c
   /
```

```
  b
 /
a
```

We must perform a rotation here, rooted at a. To achieve this, the following should be done:

`b` will be the new root The right child of `b` becomes `c`s left child (in this case, it is null) `c` becomes the right child of `b`

```
  b
 / \
a   c
```

**The important thing when choosing rotations

Double Rotation