CS 261 Recitation 6: AVL Tree

In order to get credit for the recitation, you need to be checked off by the end of recitation. For non-zero recitations, you can earn a maximum of 3 points for recitation work completed outside of recitation time, but you must finish this recitation before the next recitation. For extenuating circumstance, contact your recitation TAs and Instructor.

Group work, and individual work are highlighted

Recitation 7 Grade Breakdown:

- Part 1: AVL Tree Operations 4 pts
- Part 2: Implement an AVL tree (insertion) 6 pts

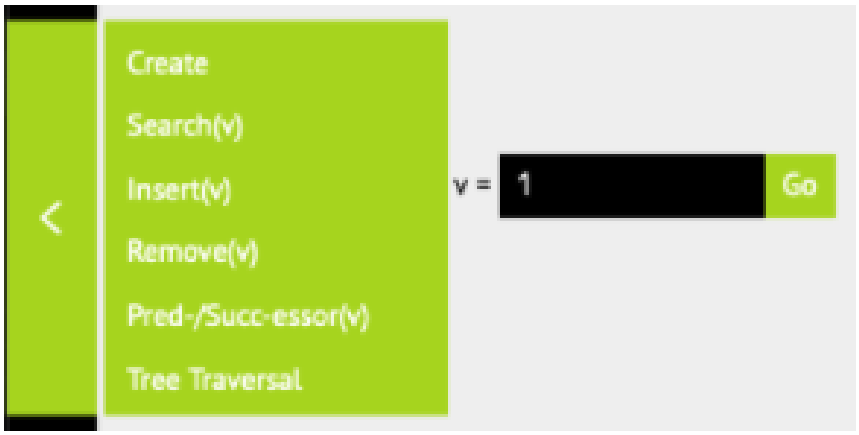
Part 1: AVL Tree Operations

Step 0:

Take a minute to familiarize yourself with the way the BST/AVL tree implementation there works:

https://visualgo.net/en/bst

There are a couple things to note about this page. First, you can use the box at the lower left of the page to select and execute operations on a BST or AVL tree. For example, to insert a value into the current tree (which you should see visualized on the main part of the page), you can select the "Insert" operation, enter the key to be inserted and click "Go":



As the operation runs, you will see the site step through a pseudocode implementation of the operation in the box at the lower right of the page:

Each step of the operation will also be animated in the main visualization of the tree, and at the end, the tree will be modified with the result of the operation.

Finally, note that you can switch between a regular BST implementation and an AVL tree implementation in the navigation bar at the top of the page:

Play around a bit with the operations of both tree implementations so you understand how the site works.

(2 pts) Step 1:

Create a new, empty AVL tree, and run the following sequence of operations in that tree.

```
insert(64)
insert(96)
insert(32)
insert(16)
insert(80)
insert(24)
insert(48)
insert(8)
insert(40)
insert(88)
```

Make note of how the tree changes as it's being built and what it looks like when all the operations are complete. Pay particular attention to the number of times a rotation (single/double) must be performed during the sequence of operations.

Take a screenshot of the final tree for later.

How many rotations were performed by the AVL tree implementation?

(2 pts) Step 2:

insert the same data as above into an AVL tree but using a different ordering of the operations. Specifically, starting from an empty tree, run the following sequence of operations in an AVL tree:

```
insert(8)
insert(16)
insert(24)
insert(32)
insert(40)
insert(48)
insert(64)
insert(80)
insert(88)
insert(96)
```

Again, make note of how the tree changes as it's being built and what it looks like when all the operations are complete. Pay particular attention to the number of times a rotation (single/double) must be performed during the sequence of operations.

Take a screenshot of the final tree.

How many rotations were performed by the AVL tree implementation? How did the finished trees you constructed here differ from the finished trees you constructed above in step #1?

(6 pts) Part 2: Implement AVL tree

Download and unzip the start code: (wget command is recommended)

https://classes.engr.oregonstate.edu/eecs/summer2022/cs261-001/recitations/rec6.zip

In avl.c, modify the program so that the AVL binary search tree maintains height-balance after each insertion. In particular, you need to implement/modify the following three functions:

- `rightRotate(),`
- `leftRotate()`
- `_avl_subtree_insert()`

Feel free to add any helper functions if needed (i.e., `rebalance()`).

Make sure you get checked off by showing them the output of your program, your report, and your group work before the end of your recitation section.

For backup purposes, please submit your work for this recitation (including all documents/text files for group work, and programs) to TEACH.