2020/04/22 - AVL Trees

21 Tháng Tư 2020 1:48 SA

Synopsis

- Final lab!
- Unfortunately, due on Apr 25 at 6 AM... yikes!

Getting Started

- Run the following commands:

```
mkdir obj bin
cp -r ~jplank/cs140/Labs/LabB/include .
cp -r ~jplank/cs140/Labs/LabB/src .
cp ~jplank/cs140/Labs/LabB/makefile .
```

- Read the lecture notes for AVL. You must be familiar with BST and AVL before attempting this!
- If you didn't do Lab A, do it or sit in on lab stream on Wednesday where I show solution. AVLs are based on BST (just balanced) so you must understand them.

Submission Command

- tar -cvf labB.tar src/avltree_lab.cpp

SUGGESTED ORDER

- Like Lab A, follow the gradescript for proper implementation steps.
  - Height — No recursion needed. It's a one-liner.
  - Ordered_keys — Straight from Lab A.
    - make_key_vector — Straight from Lab A.
  - imbalance
  - fix_height
  - rotate
  - fix_imbalance

  (imbalance, fix_height, rotate, fix_imbalance — Optional non-class helper functions described in lab writeup.)
  - insert & delete
  - operator=
    - recursive_postorder_copy

  (operator=, recursive_postorder_copy — Similar to last lab)

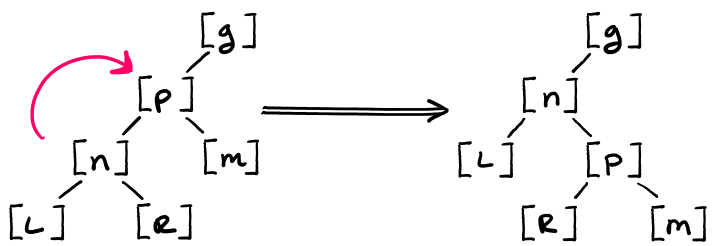
SOME HELP: ROTATE

- Consider rotating on a node "n".
- Have FIVE more pointers:
  - p - parent of "n".
  - g - grandparent of "n". Ok if sentinel.
  - m - middle. p's other child. Ok if sentinel.
  - L - left child of "n". Ok if sentinel.
  - R - right child of "n". Ok if sentinel.

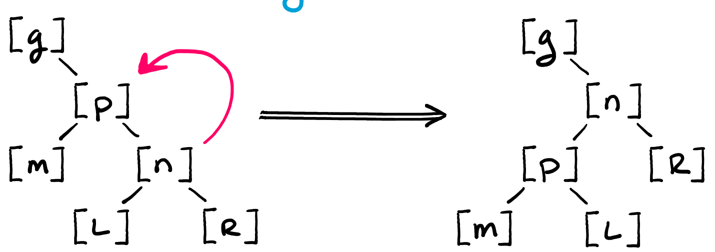
- I know, Dr. Plank suggests only n, g, p, m. But hear me out... It'll save a LOT of trouble.

- Determine which way to rotate:
  - if p -> left is n, rotate right.
  - if p -> right is n, rotate left.

- Behold the following diagrams for rotation:
  - If "n" is to left of p: RIGHT ROTATE



  - If "n" is to right of p: LEFT ROTATE



- Update g's pointers too. If g -> left was p, replace with n. If not it's to the right. Replace that instead.

- I wrote an additional helper function called "node_quickset":

```
void node_quickset ( AVLNode * n,
                     AVLNode * l,
                     AVLNode * r,
                     AVLNode * p );
```

Sets n -> left, n -> right, and n -> parent if l, r, and p aren't NULL respectively.

Sounds useless, but it makes rotate even easier.

if we rotate LEFT:

```
node_quickset(n, ..., ..., ...)
node_quickset(p, ..., ..., ...)
node_quickset(l, ..., ..., ...)
```

if we rotate RIGHT:

```
node_quickset(n, ..., ..., ...)
node_quickset(p, ..., ..., ...)
node_quickset(r, ..., ..., ...)
```

Makes you do less confusing pointer stuff and skip straight to solving the problem. It's optional, of course.

SOME HELP: imbalance & fix_height

- Can be one-liners.
- Imbalance: Height of left + right is off by more than 1.

  $|LEFT_H - RIGHT_H| > 1$

- fix_height: Height is largest of two children + 1.

  $max(LEFT_H, RIGHT_H) + 1$

SOME HELP: fix_imbalance

- You need to identify two cases:
  - straight (zig-zig)
  - jagged (zig-zag)

- In a nutshell ...

zig-zig =

LEFT-LEFT RIGHT-RIGHT

zig-zag =

LEFT-RIGHT RIGHT-LEFT

- Determining nodes below "n":

- make two pointers of AVLNode: p (parent), c (child)
- p is one of n's children. c is one of p's.
- choose based on height for both.
  - choose greatest height.
    - for p, a tie is impossible. You wouldn't call this function otherwise.
    - for c, if tie, force same direction as p. (e.g. if p = n->left, c = p->left).

- Rotation

- If zig-zig, perform "rotate" on p.
- If zig-zag, perform "rotate" on c ... twice.
- You may have to manually fix the heights of n, p, and c.

INSERTION

- Resort to lecture notes.

- From inserted node n, go up until n == sentinel and:
  - store height in variable. Then fix_height it.
    - Check if height changed. Break if not.

  - Check for imbalance. If there is one, fix_imbalance and then break.

- Do not modify Dr. Plank's code unless you feel that you must. These actions are appended at the end.

DELETION

- Simpler than the lecture notes may imply.
  - From n = parent, go up until n = sentinel and:
    - Fix the height
    - If there is an imbalance, fix it.

- Again... append at end of function. Don't modify the given code if you don't need to.

OPERATOR = AND RECURSIVE_POSTORDER_COPY

- Similar to last lab.

- Start recursion at t.sentinal->right.

- Go left, go right, make new node. (Post-order)

- Be sure to set height of the new node to n's.

- Finishing this should knock out gradescript 51-100.