ECE 590.01
C++ Programming, Data structures, and Algorithms

Balanced Trees
Admin

- Reading
  - Chapter 4 + 12.2
  - Starting Chapter 5 for next time is a good idea

- Midterm soon
  - March 1\textsuperscript{st} in recitation
  - Page of notes
  - Be prepared: know how to code
  - Fair game:
    - Anything through lecture on 2/18 (before now)
    - Anything from Ch 1.1 to 4.3 in your book.
    - Coding question: may be something you haven’t seen before
      - Goal is to be able to come up with it, not regurgitate it
What have we been talking about?

- What did we talk about last time?
What have we been talking about?

- What did we talk about last time?
  - Binary Search Trees
  - ...and more binary search trees

- Idea: $O(\log N)$ access
  - “We hope”
A bad BST

- Can end up with degenerate BSTs
  - O(N) access time
    - What’s the point?
    - Example: add 1, 2, 3, 4, 5.
  - How likely are bad cases to come up? It depends...
So what do we do about it?

- Two approaches (know both)
  - AVL (Adelson-Velskii and Landis) trees
    - Book: 4.4
  - Red-black trees
    - Book: 12.2

- Will not guarantee “perfect tree” (very hard)
  - But will guarantee $O(\lg N)$
  - Generally $\sim 2^* \lg N$ maximum height
AVL trees

- AVL trees work on principle of balance
  - Height of two sub-trees cannot differ by more than 1

\[
\text{height}(\text{node}) = \begin{cases} 
0 & \text{if node is NULL} \\
1 + \max(\text{height}(\text{node} \rightarrow \text{left}), \text{height}(\text{node} \rightarrow \text{right})) & \text{otherwise}
\end{cases}
\]

- Note: your book defines \( \text{height} (\text{NULL}) = -1 \)
  - Doesn’t matter too much
  - I think its silly

- Insert (or delete) would violate this principle?
  - Rotate tree to fix
AVL trees

- Add 1 to empty tree
  - Heights are shown next to nodes
    - Green = OK
    - Red = violating AVL rule
  - All good so far
AVL trees

- Add 2 to tree
  - Everything is still fine
    - Children of 1 have heights 1 and 0, differ by 1
AVL trees

- Add 3 to tree
  - Now we have a problem at 1
    - Right child: height = 2
    - Left child: height = 0
    - Difference: 2
AVL trees

- Fix with single right rotation
  - \( v = \text{violated node} \)
  - \( r = v->\text{right} \)
AVL trees

- Fix with single rotation: violated node = \(v\)
  \[v\rightarrow\text{right} = r\rightarrow\text{left};\]

- How do we know this respects the BST rules?
AVL trees

- Fix with single rotation: violated node = v

  v->right = r->left;
  r->left = v;

How do we know this respects the BST rules?
AVL trees

- Fix with single rotation: violated node = v
  
v->right = r->left;
  r->left = v;
  r is now the new root of this sub tree
  (return it up in the recursion etc)
AVL trees

- Resulting tree respects AVL rules
  - Now let’s add 4 and see what happens
AVL trees

- Adding 4 works fin: no re-balance needed
  - Height differences at most 1 everywhere
- Add 5?
AVL trees

- Adding 5
  - Looks like two violations of rules
    - First one (coming up): at 3 (2 vs 0)
    - Second one at 4 (3 vs 1)
  - Reality: one violation (at 3)
    - Fix it, and everything is fine
AVL trees

- Single rotate at 3
  - 4 is the new root of that subtree
AVL trees

- Single rotate at 3
  - 4 is the new root of that subtree
  - Which means new right child of 2
    - Now 2’s children are balanced: 2 vs 1
AVL trees

More generally
  • Start with something like this
    • Adding to the right side of r and increasing its height
AVL trees

- More generally
  - Start with something like this
    - Adding to the right side of \( r \) and increasing its height
    - This causes the violation
AVL trees

- More generally
  - Start with something like this
    - Adding to the right side of $r$ and increasing its height
    - This causes the violation
  - Rotating fixes the violation
    - And reduces the height of the sub-tree back to $N+2$
    - Its original height, so no further violations
AVL trees

- Mirror image case for left
  - E.g., if we added 5, 4, 3, 2, 1
AVL trees

- But what if we add to the right-side of the left
  - (or the left side of the right)

- Now doing a single rotation doesn’t fix it
  - Just puts the problem on the other side!
AVL trees

- For these cases need double rotation
  - First, “zoom in” on l’s right sub-tree

- Note that $l < r < v$
• For these cases need double rotation
  \[ l\rightarrow\text{right} = r\rightarrow\text{left}; \]
  \[ v\rightarrow\text{left} = r\rightarrow\text{right}; \]
  • L and v both have height N+1 now and respect the AVL balance
  • What do we do with r?
AVL trees

- For these cases need double rotation
  
  \[ r->left = l; \]
  \[ r->right = v; \]
  
  - \( R \) becomes the new root of this sub-tree
    - And is now only height \( N+2 \) (original height of sub tree)
AVL double rotate

Concrete example: add 3, 1, 2
AVL tree implementation

• Implementation
  • Add field to nodes for “height”
  • Write a recursive add
  • On the “way back out” (as recursions return)
    • Check balance
      • Rotate as needed
    • Update height
    • Return correct subtree up recursion

• Code: in your book
AVL deletion

- Deletion from AVL tree
  - Start with basic BST delete algorithm (recursive)
    - On the way back up
      - Calculate balance
      - Rotate as needed
        - Same rotations
      - Update heights
    - Unlike add, multiple rotations may be required
AVL deletion

Suppose we start with this AVL tree
  • Convince yourself it’s a valid AVL tree before we proceed
AVL deletion

- Now delete 35
  - We would recurse down to find 35 as usual
  - Then delete it
  - Then re-balance on the way back up
AVL deletion

- Now delete 35
  - We would recurse down to find 35 as usual
  - Then delete it
  - Then re-balance on the way back up
**AVL deletion**

- Single rotation fixes the problem with 10-20-30
  - It’s the same problem as if we had just added 10....
  - But since we are removing (and the tree is getting shorter) we can create a problem at the next level
AVL deletion

- Rotate at 40 to fix this
  - Similar to if we added 100
AVL deletion

- By the time we reach the root, everything is fixed
  - Note: may need double rotates too (imagine if left sub-tree of 60 were bigger than right sub-tree of 60)
Another approach: red-black trees

- Can also balance with red-black trees
  - Used in Linux Kernel

- Four rules
  1. Every node is either red or black
Another approach: red-black trees

• Can also balance with red-black trees
  • Used in Linux Kernel

• Four rules
  1. Every node is either red or black
  2. The root is black
Another approach: red-black trees

- Can also balance with red-black trees
  - Used in Linux Kernel

- Four rules
  1. Every node is either red or black
  2. The root is black
  3. If a node is red, its children must be black
Another approach: red-black trees

- Can also balance with red-black trees
  - Used in Linux Kernel

- Four rules
  1. Every node is either red or black
  2. The root is black
  3. If a node is red, its children must be black
  4. Every path from root->NULL must have the same number of black nodes.
Another approach: red-black trees

- Four rules
  1. Every node is either red or black
  2. The root is black
  3. If a node is red, its children must be black
  4. Every path from any given node->NULL must have the same number of black nodes.

1. Why do these rules work?
Another approach: red-black trees

- Four rules
  1. Every node is either red or black
  2. The root is black
  3. If a node is red, its children must be black
  4. Every path from root->NULL must have the same number of black nodes.

- Why do these rules work?
  - Height down one path at most 2x height down another path
    - All black path: $N$ nodes
    - Alternate red/black path: $2N$ nodes
      - $2N+1$ nodes on longer path? +1 is...
        - Red? 2 red nodes in a row
        - Black? $N+1$ black nodes on this path
A red-black tree

- Figure 12.9 in your book
- Note: not a valid AVL tree
• Convince ourselves that it meets all the rules
  0. It follows the rules of a BST
Red-black tree

- Convince ourselves that it meets all the rules
  1. Every node is either red or black
Red-black tree

- Convince ourselves that it meets all the rules
  2. The root is black
Red-black tree

- Convince ourselves that it meets all the rules
  3. If a node is red, its children must be black
Red-black tree

- Convince ourselves that it meets all the rules
  4. Every path from any given node -> NULL has the same number of black nodes
Red-black tree

- Adding:
  - On the way down, black node with 2 red children?
    - Flip black->red, red children to black
Red-black tree

- Example:
  - Add 86
Red-black tree

- Example:
  - Add 86
Red-black tree

- Example:
  - Add 87
Red-black tree

- Current node is black, both children are red
  - Re-color them
Red-black tree

- Current node is black, both children are red
  - Re-color them
  - Then keep going
Red-black tree

- Keep going
  - Found place to add (left of 90 is NULL)
Red-black tree

- Add 87 on left of 90
  - Color it red
  - Why?
Red-black tree

- Add 87 on left of 90
  - Color it red
  - Why?
    - Adding a red node doesn’t change the black node count on a path
Red-black tree

- Ok, sounds great, but...
Red-black tree

- Ok, sounds great, but...
  - Don’t we have to rotate things sometimes?
  - Sure, now add 88
Red-black tree

- Re-color here
Red-black tree

- Re-color here
Red-black tree

- Re-color here
Red-black tree

- Re-color here
Red-black tree

- Found place to add
  - Want to add 88 as red child of 87
  - This violates rule 3
    - Red nodes must have black children
Red-black rotations

- Problem: B has red child C (just added C)
  - Triangles indicate arbitrary sub trees again
Red-black rotations

• Problem: B has red child C (just added C)
  • Triangles indicate arbitrary sub trees again
• Rotation very similar to AVL
  • But requires re-coloring nodes
Red-black rotations

- New difficulty?
  - What if A’s right sub-tree starts with a red node?
  - Now A is red... and has a red child
    - 🙁
Red-black rotations

- Fortunately for us, we made this impossible
  - Remember the re-coloring?
    - A is a black node with 2 red children...
    - So we color flip before proceeding
Red-black rotations

- Fortunately for us, we made this impossible
  - Remember the re-coloring?
    - A is a black node with 2 red children...
    - So we color flip before proceeding
Red-black rotations

- Fortunately for us, we made this impossible
  - Remember the re-coloring?
    - A is a black node with 2 red children...
    - So we color flip before proceeding
  - In fact, if that happened, we don’t even need to rotate now
Red-black tree

- Back to our example
  - This is where we added 88
  - Rotation case
Red-black tree

- Back to our example
  - This is where we added 88
  - Rotation case
Red-black tree

- May need to rotate on the way down as we re-color too
  - At a black node with 2 red children: rules say re-color
Red-black tree

- May need to rotate on the way down as we re-color too
  - At a black node with 2 red children: rules say re-color
  - Now have a violation of red child rule
Red-black tree

- May need to rotate on the way down as we re-color too
  - At a black node with 2 red children: rules say re-color
  - Now have a violation of red child rule
  - Fix by rotating (which involves re-coloring)
Red-black tree

- How do we know we aren’t breaking any rules now?
Red-black tree

• Can’t have a red node at the top of the blue triangle:
  • Would have red node with red child
Red-black tree

- Even though a red node at the top of grey was possible
  - We made it impossible during our traversal
  - Would have flipped 15->red, and 30 & X -> black
Red-black tree

- Note: I drew the heights this way somewhat on purpose
  - Idea of red-black: balance on the way down
    - Going to add to right? Want it to be shorter than left when we get there!
Red-black tree

• Note: I drew the heights this way somewhat on purpose
  • Idea of red-black: balance on the way down
    • Going to add to right? Want it to be shorter than left when we get there!
Why red-black?

- Advantage of red-black (vs AVL)
  - Can do all balancing on the way down for insert
  - Makes iterative algorithm easier (for people scared of recursion)

- Disadvantage of red-black (vs AVL)
  - Slightly worse guarantee on height
Red-black delete

- Red-black deletion:
  - Remove red node? Easy
  - Remove black node? Now 1 short...

- Amazing insight by Dr. Matt Might (Prof at U. of Utah)
  - Add temporary colors
    - Double black
    - Negative black
  - Allows you to keep invariants the whole time
    - Just have to “fix” colors your aren’t allowed to have
  - [http://matt.might.net/articles/red-black-delete/](http://matt.might.net/articles/red-black-delete/)

- Side note: Matt’s blog is highly recommended reading
  - Programming, grad school, HOWTOs, Productivity...
  - Interest in PhD program? Highly suggest his articles on it
Wrap-up

• That concludes trees
  • Covered a lot here
  • Make sure you absorb it all

• Monday: start hash tables
  • Reading: Chapter 5