Lecture 10: AVL TREES deletion
Review of Removal in BST

Case 2: node $v$ storing entry to be removed has no leaf child
- replace entry at $v$ with entry whose key $k$ preserves binary tree order
  - either largest key entry in left subtree of $v$ or smallest key entry in right subtree of $v$
  - we choose to take the smallest key entry in the right subtree of $v$
- copy entry at node $u$ into node $v$
- remove node $u$ and its leaf left child $l$ with $\text{removeExternal}(l)$
- $w$ is the parent of removed node $u$

Example: remove 3
Removal in AVL Tree

- Start by removing as in BST
- Let $u$ be the removed node
  - may be different from the node which used to hold the removed entry
  - any ancestor of $u$, starting with the parent, $w$, may have an imbalance
- Example: delete 32

Before delete:

```
    44
   /   \
 17    78
 /       \
32 50 88
```

After delete:

```
   1
  /   \n17    78
 /       \
44      3
```

```
   50
  /   \
48 62
```

```
   88
   |
   |
```
Removal in an AVL Tree

- Follow the path up the tree from the parent of the removed node
- If an unbalanced node is encountered, rebalance the tree with TriNodeRestructure
  - unlike insertion, may have to perform several triNodeRestructure operations
Rebalancing After Removal

- Let $z$ be the first unbalanced node
- Height difference between left and right subtrees of $z$ is exactly 2
- One subtree has height $p$, the other $p+2$
  - $u$ was deleted from the smaller subtree
- 2 cases: right subtree is higher or left subtree is higher
- Similar to insertion, should be able to adapt trinode restructuring
  - get $y$ from the higher subtree of $z$ and $x$ from the higher subtree of $y$
- Unlike insertion, cannot say as much about the higher subtree of $z$
Let us expand the higher subtree of $z$

- At least one child of $y$ is of height $p+1$, both could be of height $p+1$.
- If one is of height $p$ and another of height $p+1$, take $x$ from the higher subtree, as before.
- From which tree to get $x$ if both subtrees have height $p+1$?
Rebalancing After Removal

- If both subtrees of $y$ have height $p+1$, take $x$
  - from the right subtree of $y$ if $y$ is the right child of $z$
  - from the left subtree of $y$ if $y$ is the left child of $z$

- Tree may not rebalance properly if this rule is not followed
Rebalancing After Removal
Performing Rebalance Improperly

- Taking $x$ from the “same side” as $y$ when both children of $y$ have the same height $p+1$, we insure that children of $x$ are not separated.
How Many Tribalance Procedures for Removal?

- Consider case when y has one child higher than the other
- Height of z before deletion is \( p+3 \), after trinode restructuring that position (now holding x) has height \( p+2 \)
- Might cause imbalance among ancestors:
  - must continue checking for imbalance
Example of AVL tree Deletion

before delete

after delete, before restructuring

after restructuring
Algorithm AVLtreeDelete(k)

Input: key k

1. \( w = \text{TreeDelete}(k, T.\text{root}) \)  // \( w \) holds the parent of deleted node
2. \( z = w \)
3. while (\( z \neq \text{null} \))  // traverse up the tree, checking for imbalance
   1. \( \text{setHeight}(z) \)
   2. if \( |\text{getHeight}(z.\text{left}) - \text{getHeight}(z.\text{right})| > 1 \) then
      1. \( z = \text{TriNodeRestructure}(\text{tallerChild}(\text{tallerChild}(z)), \text{tallerChild}(z), z) \)
      2. \( \text{setHeight}(z.\text{left}); \text{setHeight}(z.\text{right}); \text{setHeight}(z); \)
      3. \( z = \text{parent}(z) \)

- \( \text{setHeight}(z) = 1 + \max(z.\text{left}.\text{height}, z.\text{right}.\text{height}) \)
- \( \text{tallerChild}(v) \) returns child with larger height
  - if children have equal height, return the left child if \( v \) is a left child, or the right child if \( v \) is the right child
- complexity of \( \text{AVLtreeDelete}(k) \) is \( O(\log n) \)
Another Deletion Example

```
30
   5
  /    
30---20---55
   /     /
10---10  45---60
    1    3  4 1
   /    /    /  
15---25---40---50
    2  1    2   1
   /    /    /    
   1    2    1    1
```

13
Another Deletion Example
Another Deletion Example

```
    30
   /  \
  a    z
 /     \
15     55
 /       /
10      45
 /     /  \
25     50   60
 /     /     /
35     40    1
 /     /     \
T1     T3     T4
1 1 3 2

T2
1
```

Another Deletion Example
AVL Tree Complexity

- A single trinode restructure is $O(1)$.
  - Using a linked-structure binary tree.
- Find is $O(\log n)$.
  - Height of tree is $O(\log n)$.
- Insert is $O(\log n)$.
  - Initial find is $O(\log n)$.
  - Restructuring up the tree, maintaining heights is $O(\log n)$.
- Remove is $O(\log n)$.
  - Initial find is $O(\log n)$.
  - Restructuring up the tree, maintaining heights is $O(\log n)$.
- Ordered dictionary implementation where main operations are as efficient as possible $O(\log n)$ in the worst case.
“Tree” of Trees, so far

- General Trees
  - Arbitrary number of children

- Binary Tree
  - At most 2 children

- Complete Binary Tree
  - Heap-order
  - Each level (except last one) has maximum number of nodes. Nodes as far to the left as possible

- Full (Proper) Binary Tree
  - 0 or 2 children
  - Binary search tree order

- Binary Search Tree

- AVL Tree
  - Height-balance property