Deletion from a Binary Search Tree

- Can you write an algorithm that can delete any node from a BST?
Deletion Strategy

- To solve hard data structure problems, we often divide the problem into cases.
  - Like proof by cases

- What is the simplest deletion scenario?
Deletion: Case 1

- Case 1: Target node is a leaf.
- Solution: Simply delete it. Nobody will miss it.
Deletion: Case 2

- Case 2: The target node has just one child.
- Solution: Parent adopts its grandchild ("splice")
Deletion: Case 3

- Case 3: The target node has two children
- Solution: Swap with predecessor, then delete predecessor

Is it possible that the predecessor might be one of the 2 children?
Deletion: Case 3

Consider an in-order traversal of the nodes near the target node.

In-Order Traversal (excerpt):
Deletion: Case 3

Swap target and predecessor. (This puts the whole tree out of order.)
Swap target and predecessor. (This puts the whole tree out of order.)

Now delete the (moved) target. The order of the tree is restored.
Deletion: Case 3

- What if the predecessor is not a leaf???
- Do the same...but treat this (recursively) as Case 2.
Case 3 Recursion

• In Class:

  Prove that the recursive deletion (in Case 3) is never another Case 3 deletion.

  Prove that the recursive deletion (in Case 3) always happens in a descendant of the original node-to-delete.
Cost of Delete

- $O(h)$ to search
- $O(h)$ to find successor
- $O(1)$ to move successor
- $O(1)$ to delete successor

$O(h)$ total
Rotations

- Basic BST insertion is sometimes inadequate.
  - Height of tree depends on the order of insertions, which could be bad.
  - (Reflect: what is “bad,” and why?)

- We can adjust the height using “rotations”: change the links (the topology), preserving the order of the keys.

- Very useful for rebalancing a tree
These trees are “equivalent”

... in the sense that they have the same keys, and the same in-order traversal. All we did was change the links (the *topology*).
Rotations

Right Rotation in General

Assume that C, E are nodes.

Subtrees a, d, f can be any size: empty, just leaves, large trees, or whatever.
Rotations

An In-Order traversal of either tree:
An In-Order traversal of either tree:

\[
\begin{align*}
\ &a \quad C \quad d \quad E \quad f \\
\end{align*}
\]
An In-Order traversal of either tree:

```
  a     C     d     E      f
  |     |     |     |      
 a   C   f   a   C   E
 /     |     |     |    |
E     d   f   d   f
```

left rotation at C transforms it to

TURN LEFT
Rotations

An In-Order traversal of either tree:

```
   a      C      d      E      f
```

Trees are equivalent.
Keep in mind:
This rotation probably happens deep within a larger tree.
Using Rotations, we can convert a tall tree into an equivalent one.
Using Rotations, we can convert a tall tree into an equivalent one.
Rotations

Using Rotations, we can convert a tall tree into an shorter one that is equivalent.
Rotations

Using Rotations, we can convert a tall tree into a shorter one that is equivalent.
Using Rotations, we can convert a tall tree into an shorter one that is equivalent.
Using Rotations, we can convert a tall tree into a shorter one that is equivalent.
Topic 07: BSTs (Overview)

- Structure
- Basic Operations
- Deletion
- Rotations
- Balanced Trees
Balanced Trees

• All of the dictionary operations in binary search trees are $O(h)$, where $h$ is the height.

• For speed, we want small height.
  – *Minimum* height sounds nice, but that could be too extreme a requirement.

• So we prefer a tree with $n$ nodes to have height $\log n$ or less– thus all of the operations would be $O(\log n)$.

• But worst case height is $O(n)$.
Balanced Trees

• A variety of systems try to keep the tree balanced
  – Typically, we control the height; prove $h = O(\log n)$
  – A few more complex designs
Balanced Trees

Typical Concept
Start with a simple, small tree.
Balanced Trees

Typical Concept
Allow insertions to grow the tree.
Balanced Trees

Typical Concept

If one subtree gets much larger than the other, then rotate to rebalance.
Balanced Trees

Typical Concept
If one subtree gets much larger than the other, then rotate to rebalance.
Balanced Trees

Typical Concept

Usually, the balancing happens throughout the tree, at multiple nodes.

This subtree needs rebalancing.