

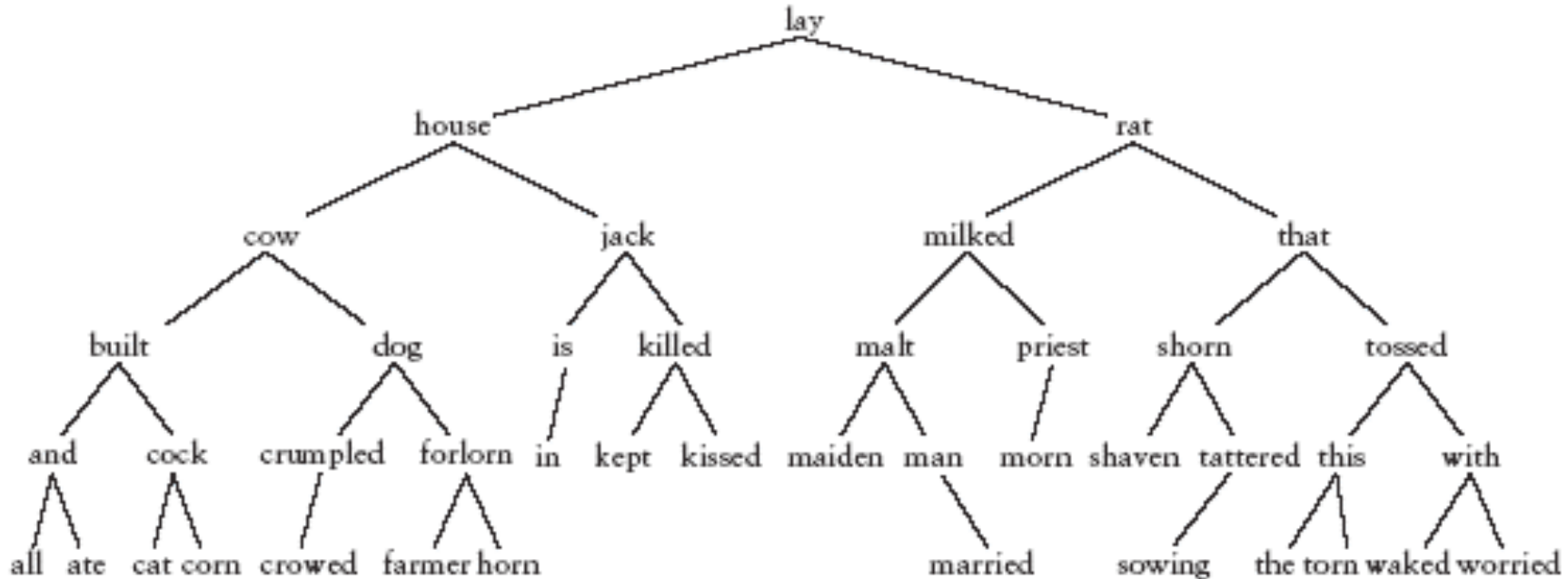
Binary Search Trees
CMSC 132
Chapter 8.1

Nelson Padua-Perez
Bill Pugh

Binary Search Tree

FIGURE 8.13

Binary Search Tree Containing All of the Words from "The House That Jack Built"



Building maps with binary search trees

- Books, etc., often talk about each node of a binary tree just containing a key
- Search trees are often used to implement maps
 - each non-empty node contains a key, a value, and left and right subtrees
- What the `?!?!` is the generic type
`<K extends Comparable<K>>`
- Denotes any type `K` that can be compared to `K`'s
 - e.g., `Strings` can be `compareTo`'d `Strings`, but `Strings` cannot be `compareTo`'d `Integer`

Searching a binary tree for a value X

- Is the tree empty?
 - if so, then X is not present
- If the key at the root of the tree:
 - equal to X
 - X is present
 - greater than X
 - check to see if X present in left subtree
 - less than X
 - check to see if X present in right subtree

Inserting X into a binary tree (Listing 8.4)

- Define the insertion function to return a new tree
- If tree is empty, return a new tree containing just X
- If the key at the root of the tree:
 - equal to X
 - X is already represent present, just return existing tree
 - greater than X
 - replace left subtree with result of inserting X into left subtree
 - return existing tree
 - less than X
 - replace right subtree with result of inserting X into right subtree
 - return existing tree

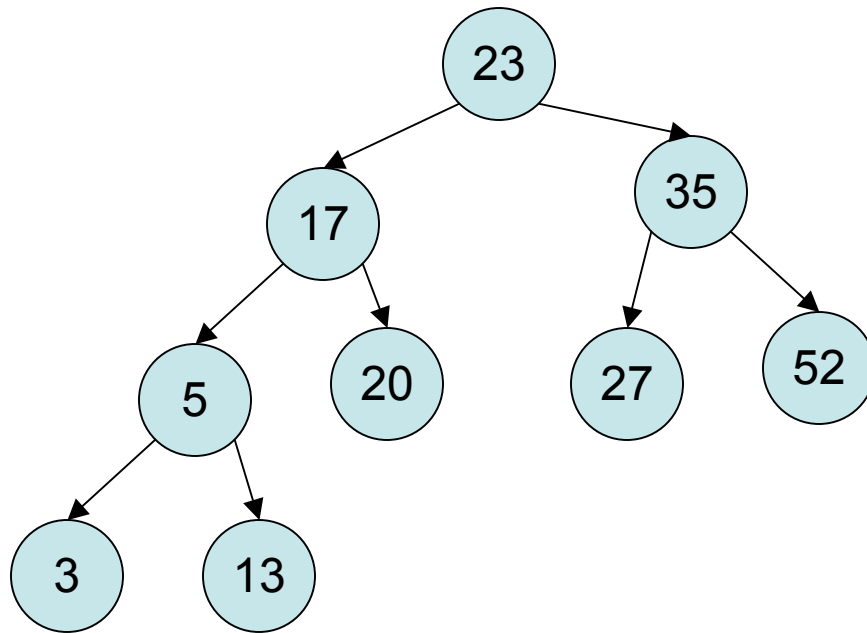
Deleting X into a binary tree (Listing 8.4)

- Define the deletion function to return a new tree
- If tree is empty, X isn't in tree, return empty tree
- If the key at the root of the tree:
 - greater than X
 - replace left subtree with result of deleting X from left subtree
 - return existing tree
 - less than X
 - replace right subtree with result of deleting X from right subtree
 - return existing tree
 - equal to X
 - hard/tricky case

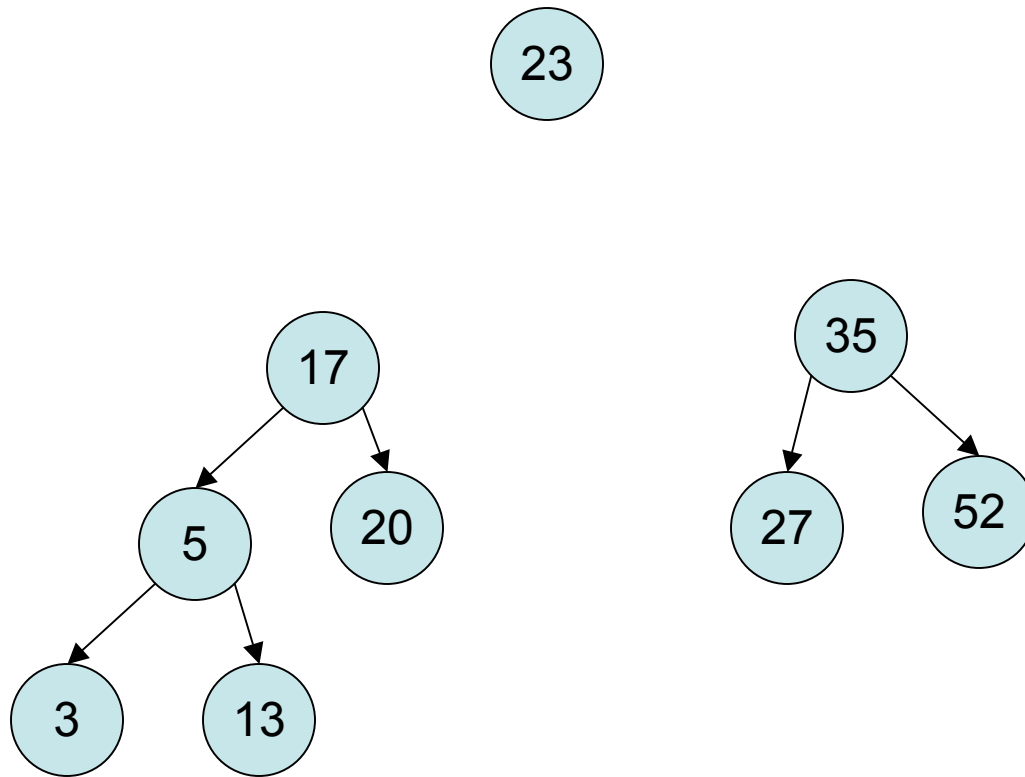
Deleting the root of a binary search tree

- If one of the subtrees is empty can just return the other subtree
- Otherwise:
 - find the maximum element from the left subtree
 - delete it from the left subtree
 - make new tree, with the maximum element from the left subtree

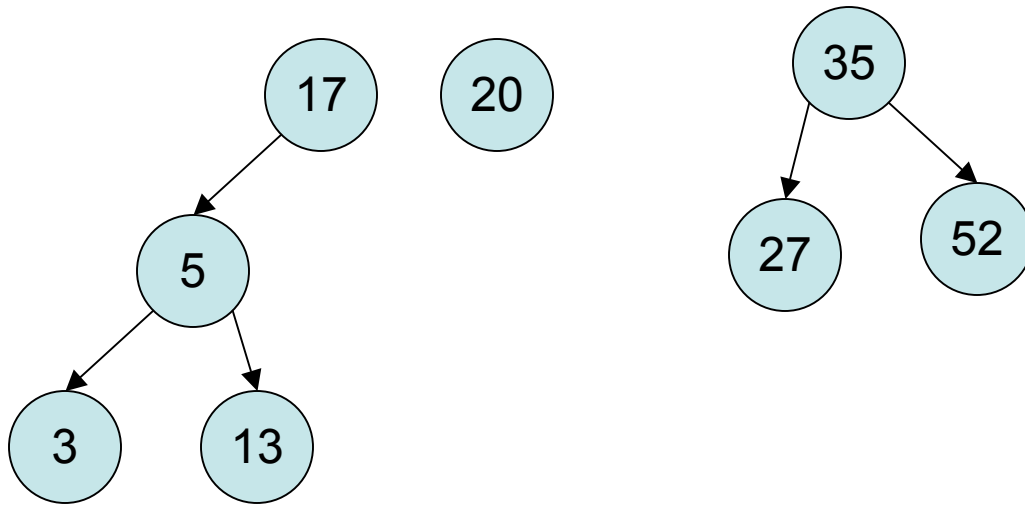
Deleting 23



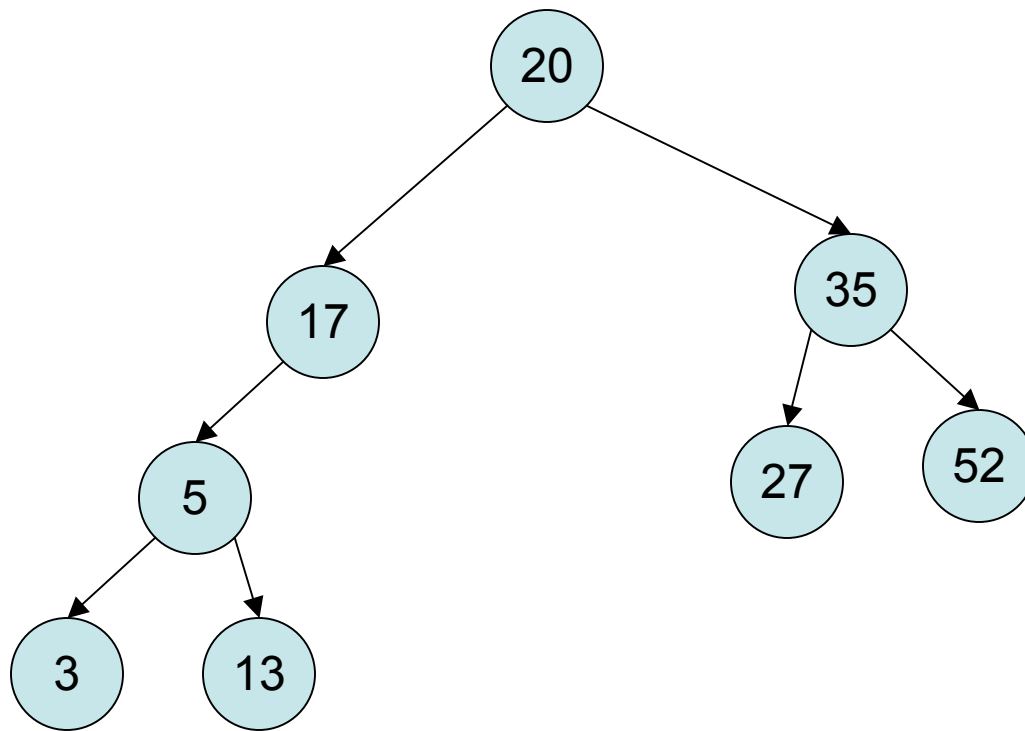
Deleting 23: consider subtrees



Deleting maximum element from left subtree



Form new tree, from resulting left subtree
and original right subtree



Binary search tree project

- Project 6 has been posted to your linuxlab repository
- Recursive, polymorphic binary search tree
 - used to implement a map
- What do we mean by polymorphic?
 - implement two subtypes of Tree: EmptyTree and NonEmptyTree
 - We use EmptyTree, rather than null to represent the empty tree
 - Invoke methods on tree nodes, get empty or nonempty functionality

Recursive, polymorphic linked lists

- Give you code for linked lists that uses the same concept
- Keys are kept in sorted order
- Have an interface `List`, and subtypes `EmptyList` and `NonEmptyList`

compareTo

- I always have to look up which way this works
- Invoke `a.compareTo(b)`
- if `a.compareTo(b) = 0`, then `a == b`
- if `a.compareTo(b) < 0`, then `a < b`
- if `a.compareTo(b) > 0`, then `a > b`