Trees

Brian Green [CC BY-SA 2.0 (https://creativecommons.org/licenses/by-sa/2.0)], via Wikimedia Commons

CAC 210
Spring 2018
Amber Wagner

Indiana Jones

- In Indiana Jones and the last crusade, Indiana is looking for his father
- He winds up in a library in Venice
- Let’s say you wanted to find a particular book on a library shelf
- How would you search?

Indiana Jones

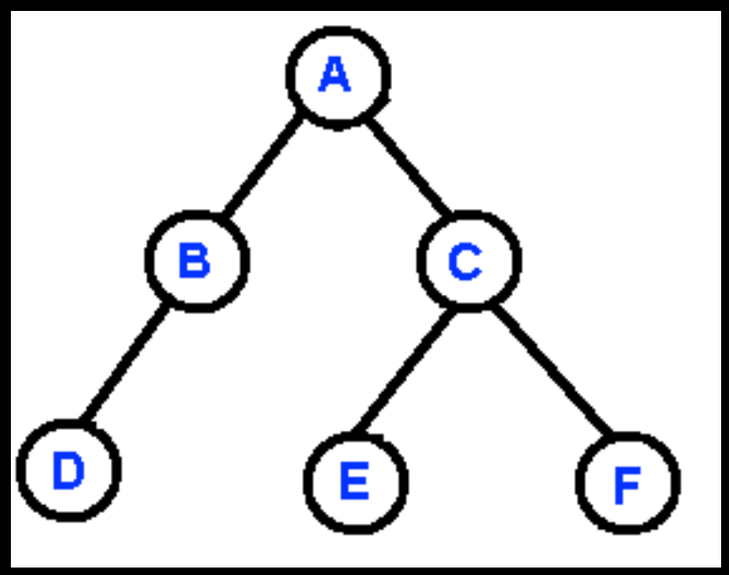
http://filmfisher.com/films/indiana-jones-last-crusade/

“X never, ever marks the spot” - Indiana Jones

- The key to a fast search is to have a structure that allows the search space to be effectively narrowed down quickly
- The smaller the search area, the better

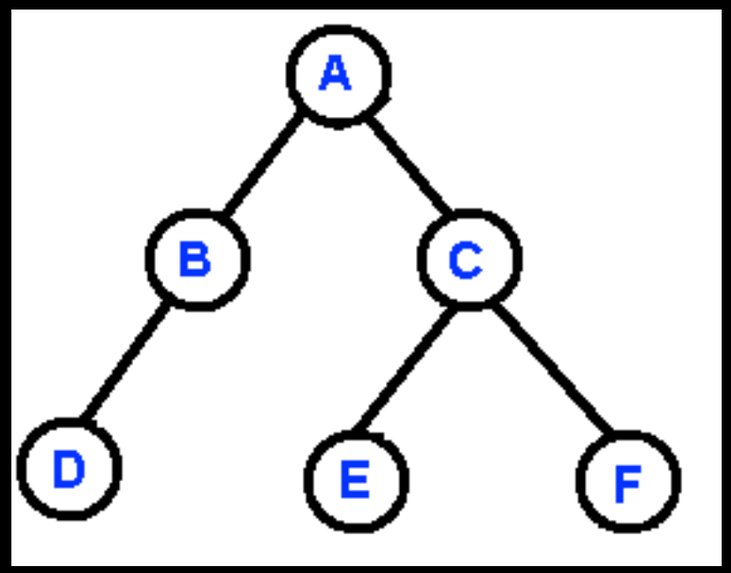
What is a Tree?

- It is a data structure with a root, branches, and leaves
- Each node within the tree is connected by a directed edge from one other node



Terminology

- If one node is connected to another, the top node is the parent, and the connected node is the child
  - A is the root
  - A is B's parent and C's parent
  - B is A's child and D's parent
  - D is considered a leaf because it does not have any children



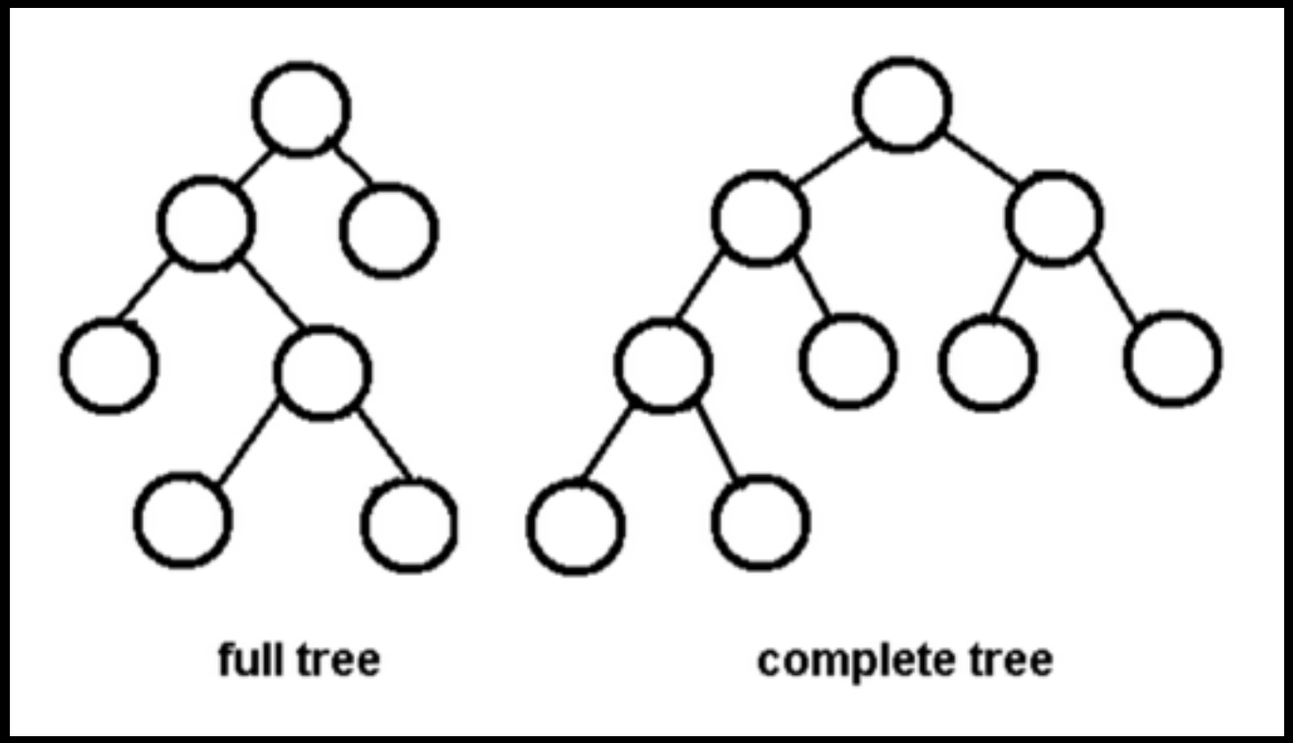
Tree Metadata

- The depth of a node is the number of edges from the root to the node
- The height of a node is the number of edges from the node to the deepest leaf
- The height of a tree is the same as the height of the root

More Terminology

- A full binary tree is a binary tree in which each node has exactly zero or two children

- A complete binary tree is a binary tree, which is completely filled, with the exception of the bottom level



Why Trees?

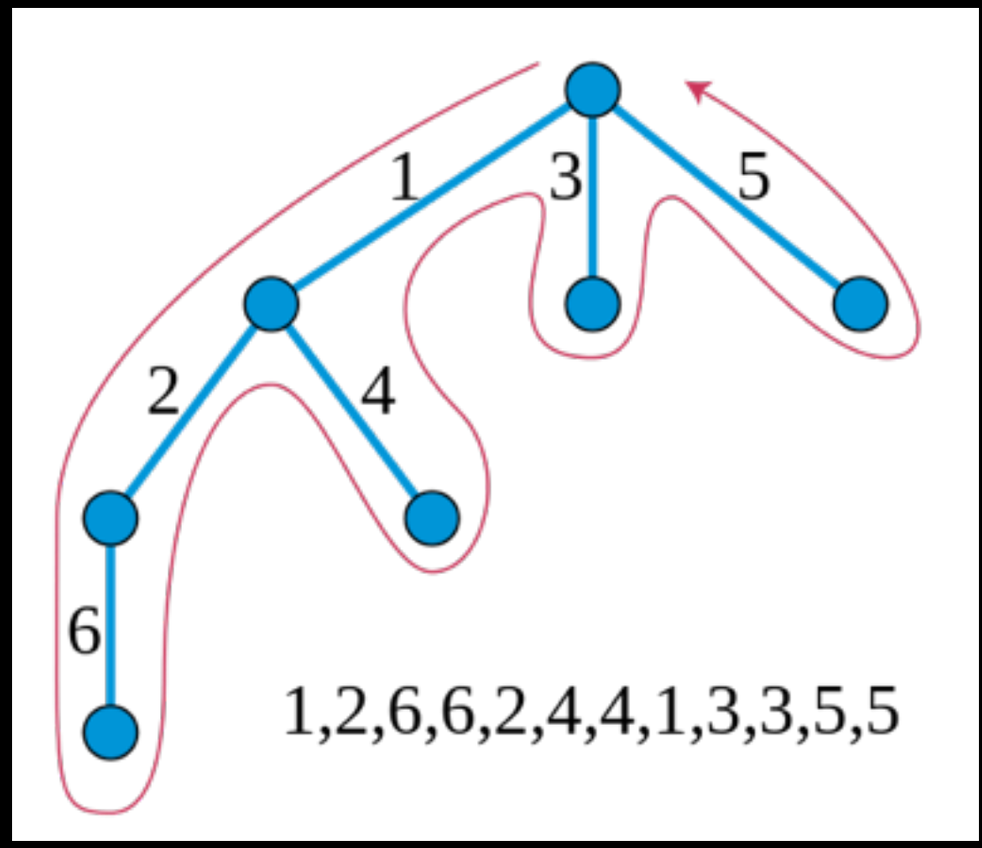
- Trees reflect structural relationships in the data
  - Consider a family tree
- Trees are used to represent hierarchies
- Trees provide an efficient method for inserting and searching
- Trees are very flexible data structures - subtrees can be moved with minimum effort

Traversing a Tree

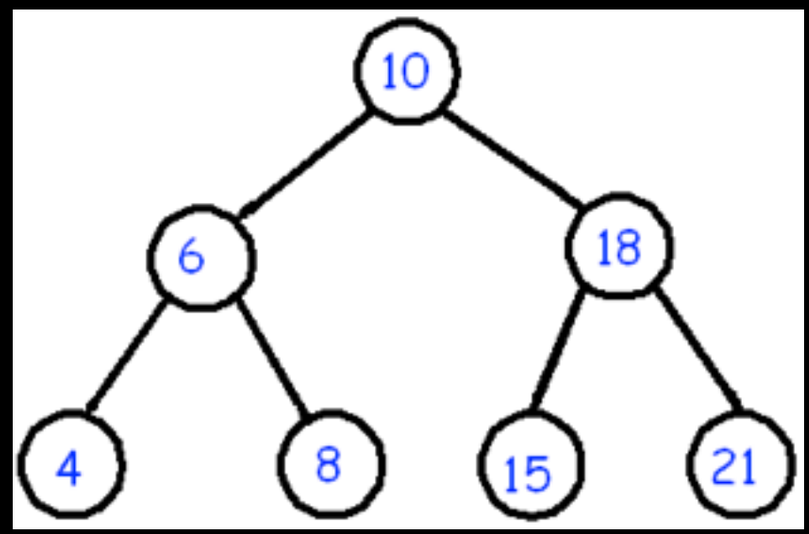
- Breadth-first traversal
    - Starts at Level 0 and moves down to the bottom of the tree level-by-level, left to right
- Depth-first traversal
    1. Pre-order traversal: parent, left, right
    2. In-order traversal: left child, parent, right child
    3. Post-order traversal: left child, right child, parent

Euler Tour

- Named after Leonhard Euler
- Method in graph theory for representing trees
- Directed graph (set of vertices connected by edges with an associated direction) containing two directed edges for each edge in the tree
- Allows for efficient, parallel computation of solution to common problems



Binary Search Tree



- Special kind of tree where the nodes are ordered in the following way:
  - each node has one piece of data
  - the data in the left subtree are less than the data in the parent (L < P)
  - the data in the right subtree are greater than the data in the parent (P < R)
  - duplicate data are not allowed

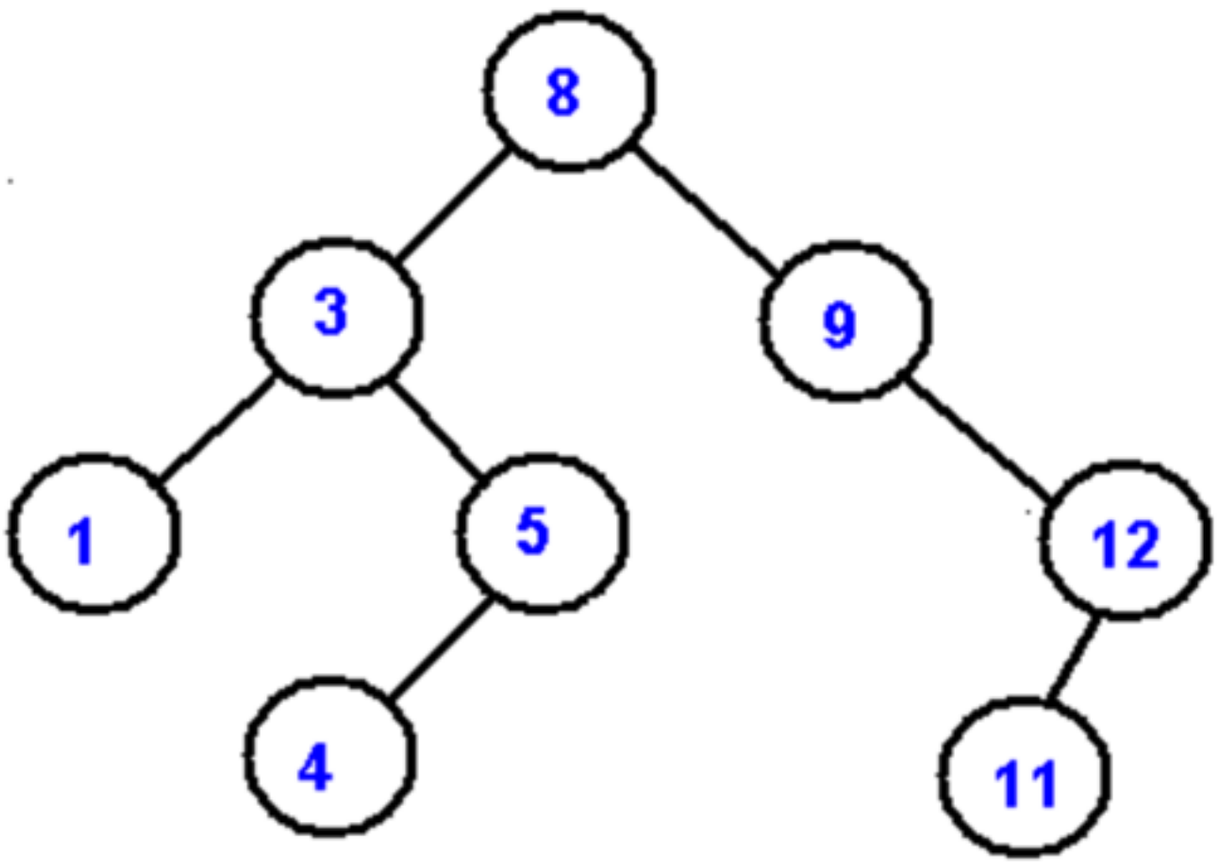
Implementation

- BST node will have a pointer to a left child and a right child
- BST node should also have a variable to store the data (key)

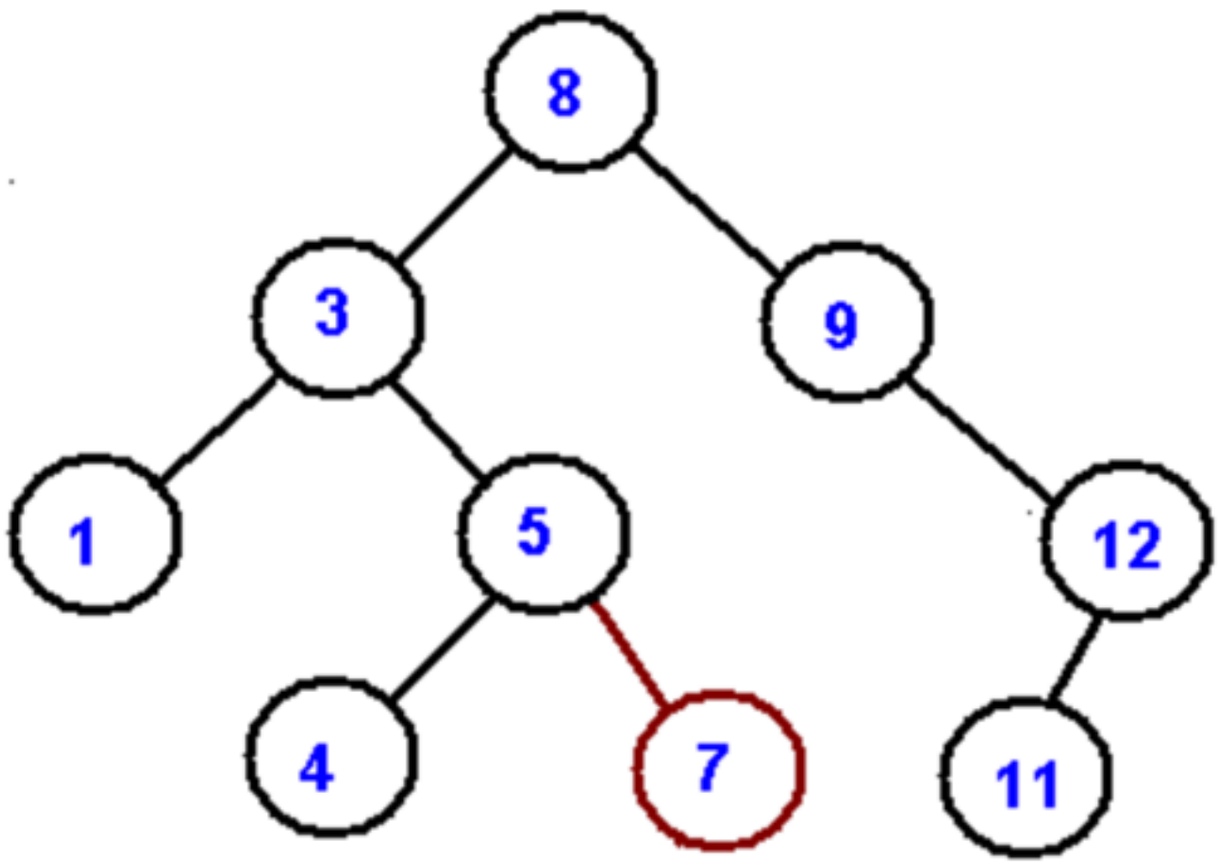
Inserting an Element

- Start at the root, and recursively navigate the three searching for a location in the BST to insert the new node
- If the element is already in the tree, we're done (no duplicates)
- New node will always replace a NULL reference

Inserting an Element



before insertion



after insertion

Inserting an Element

```
BSTInsert(tree, node) {
 if (tree->root is null)
    tree->root = node
    node->left = null
    node->right = null
 else
    cur = tree->root
    while (cur is not null)
      if (node->key < cur->key)
         if (cur->left is null)
           cur->left = node
           cur = null
         else
           cur = cur->left
      else
         if (cur->right is null)
           cur->right = node
           cur = null
         else
           cur = cur->right
    node->left = null
    node->right = null
}
```

Searching

- Always starts at the root
- If the data is less than the current node, go left; otherwise, go right
- Then what?

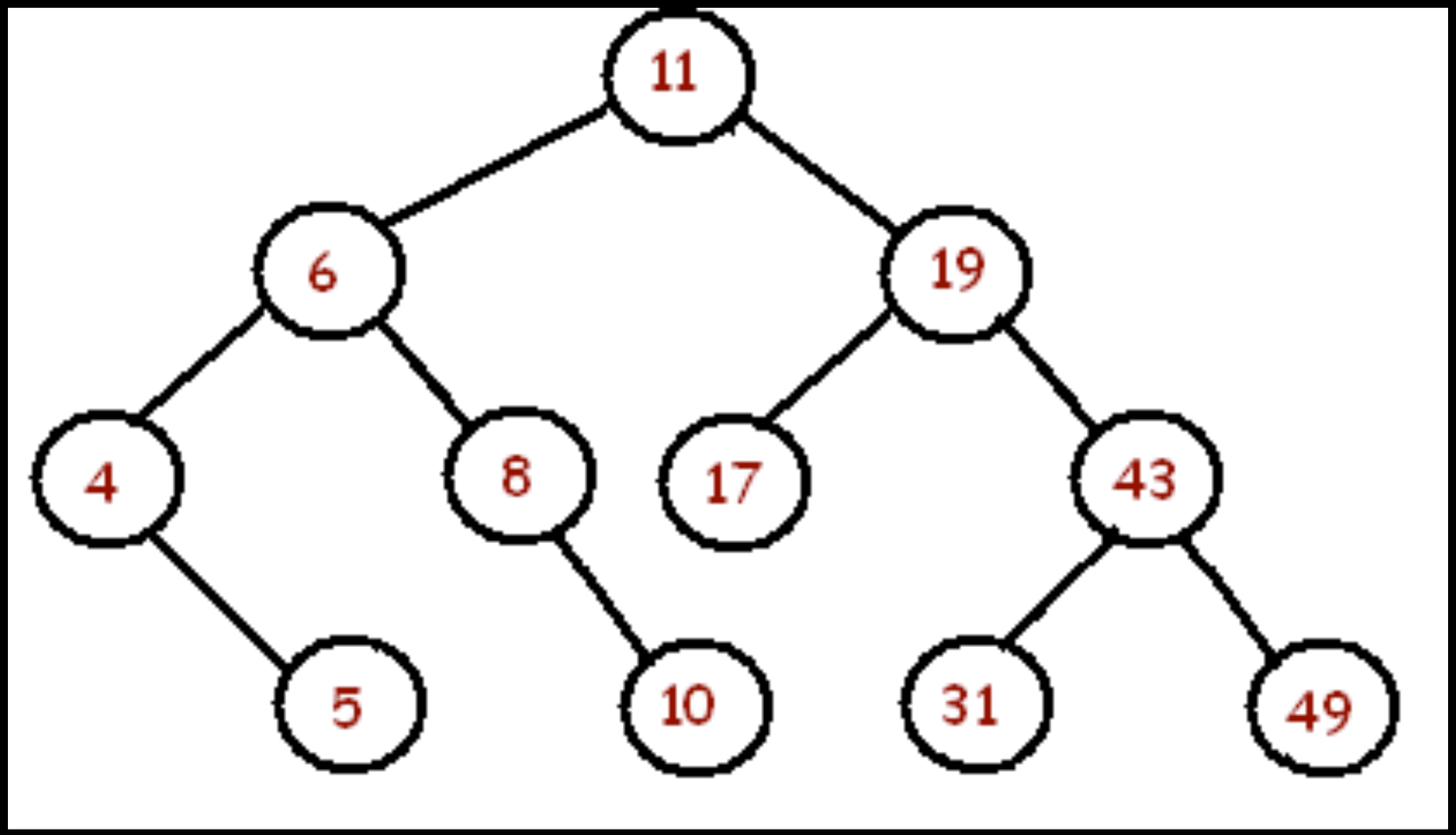
Searching

- Always starts at the root
- If the data is less than the current node, go left; otherwise, go right
- Then what? Recursively continue
- Runtime: O(h) where h is the height of the tree
    - Since s binary search tree with n nodes has a minimum of O(log n) levels, it takes at least O(log n) comparisons to find a particular node
    - Can degenerate into a linked list giving O(n) runtime complexity

Searching

```
BSTSearch(tree, key) {
  cur = tree->root
  while (cur is not null)
    if (key == cur->key)
      return cur // Found
    else if (key < cur->key)
      cur = cur->left
    else
      cur = cur->right
  return null // Not found
}
```

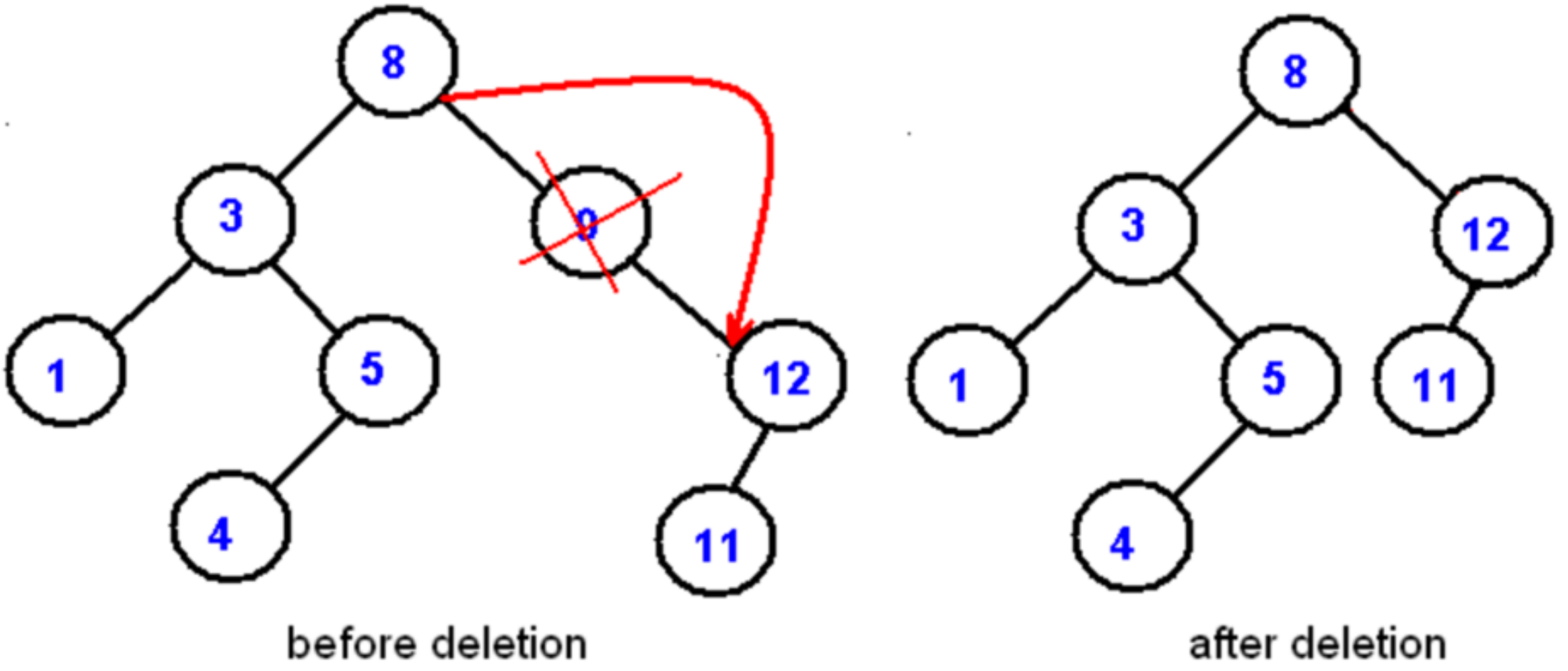
Searching



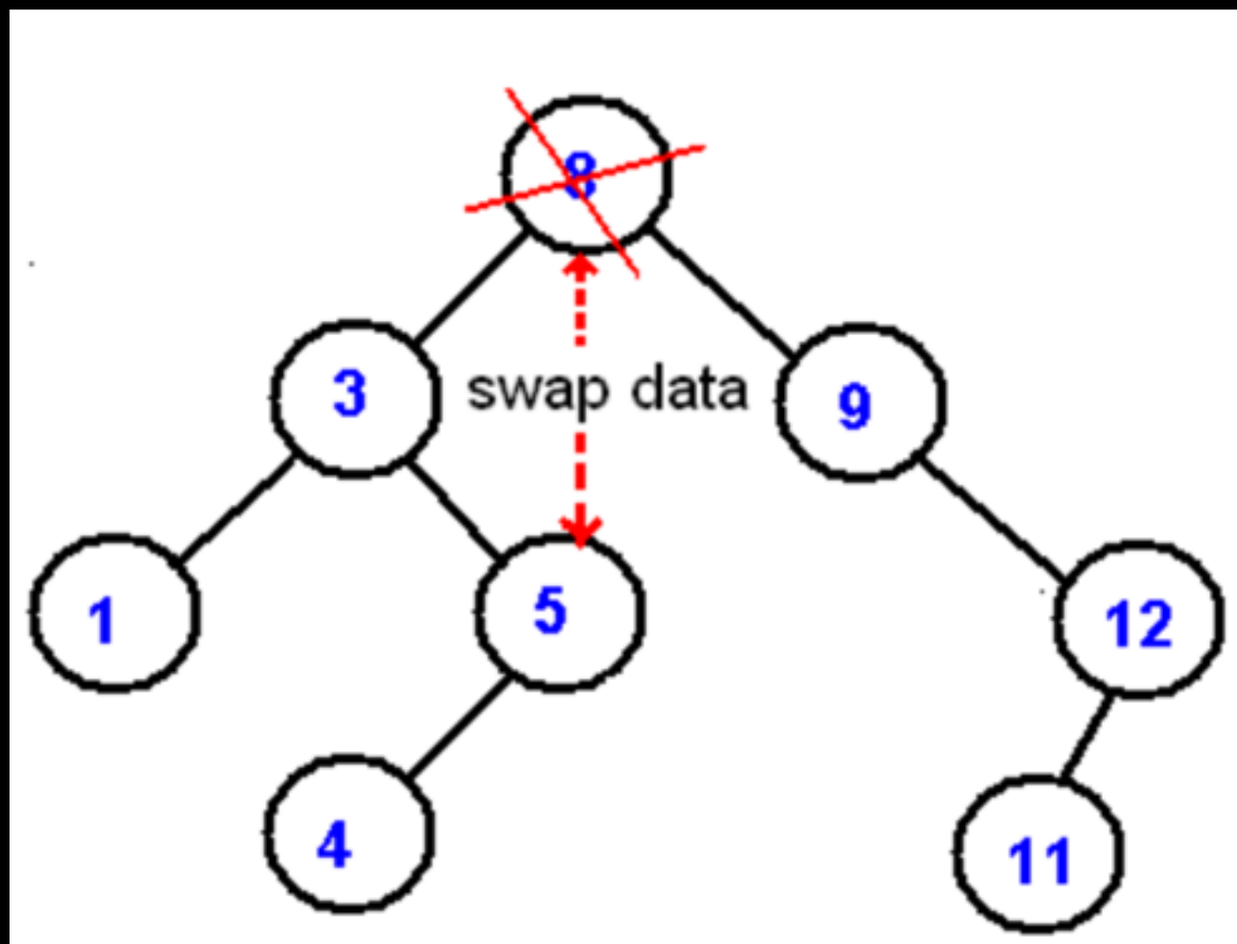
Removing an Element

- Scenarios to consider…node to delete
  - is not in a tree
  - is a leaf
  - has only one child
  - has two children
- Similar to a linked list, you have to be sure to preserve the links

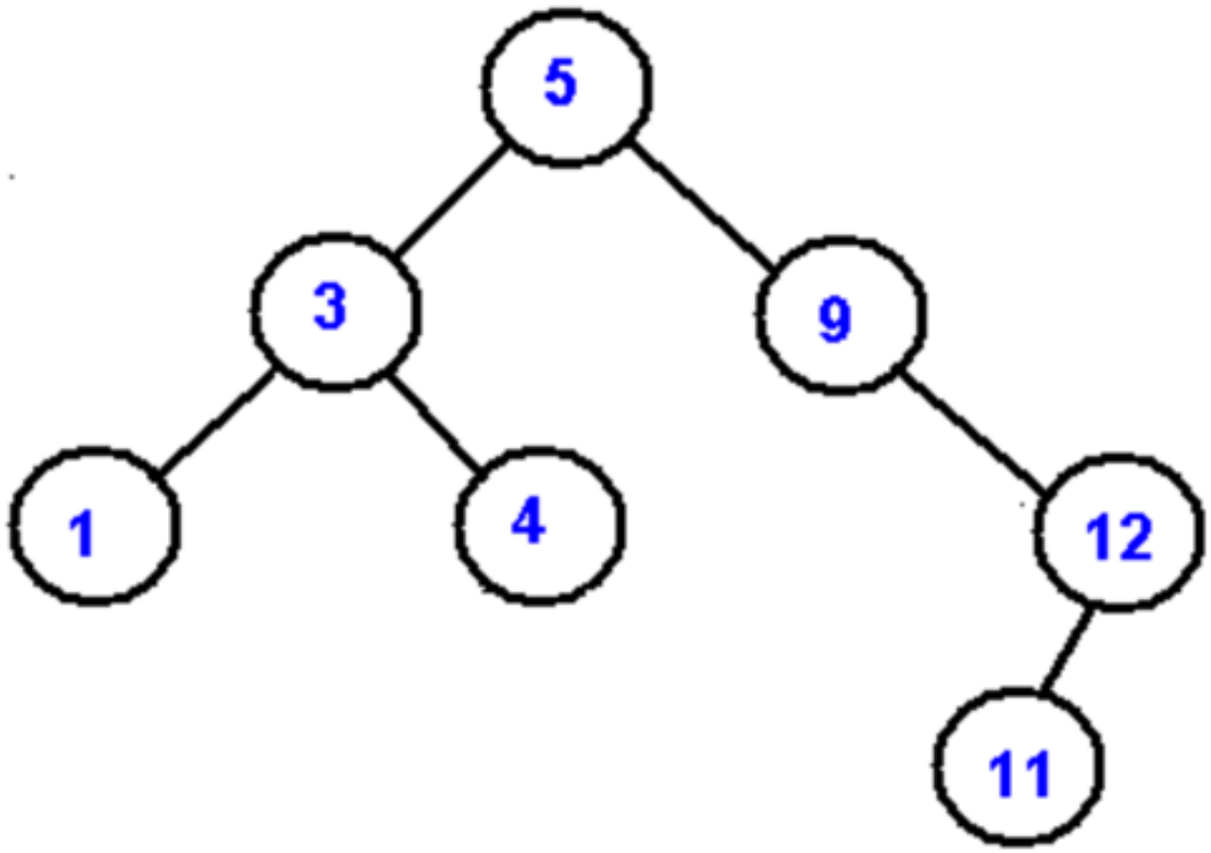
Removing an Element



Removing an Element



before deletion



after deletion

Removing an Element

```
BSTRemove(tree, key) {
   par = null
   cur = tree->root
   while (cur is not null) // Search for node
      if (cur->key == key) // Node found
         if (!cur->left && !cur->right)          // Remove leaf
            if (!par) // Node is root
               tree->root = null;
            else if (par->left == cur)
               par->left = null
            else
               par->right = null
         else if (cur->left && !cur->right)      // Remove node with only left child
            if (!par) // Node is root
               tree->root = cur->left
            else if (par->left == cur)
               par->left = cur->left
            else
               par->right = cur->left
         else if (!cur->left && cur->right)      // Remove node with only right child
            if (!par) // Node is root
               tree->root = cur->right
            else if (par->left == cur)
               par->left = cur->right
            else
               par->right = cur->right
         else                                    // Remove node with two children
            // Find successor (leftmost child of right subtree)
            suc = cur->right
            while (suc->left is not null)
               suc = suc->left
            cur = Copy suc                        // Copy successor to current node
            BSTRemove(cur->right, suc->key)      // Remove successor from right subtree
         return // Node found and removed
      else if (cur->key < key) // Search right
         par = cur
         cur = cur->right
      else                        // Search left
         par = cur
         cur = cur->left
   return // Node not found
}
```

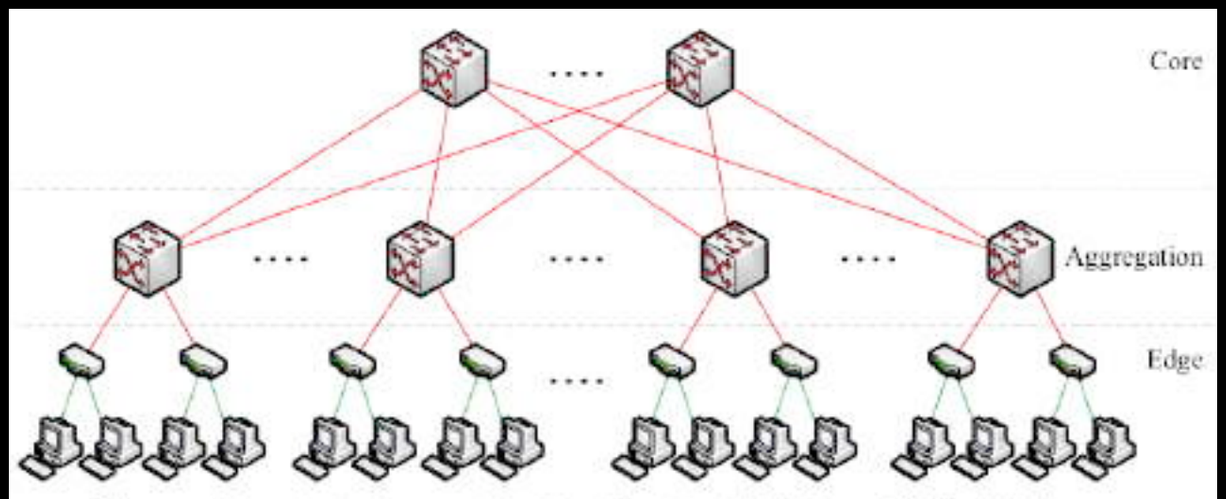
Traversing a BST

```
BSTPrintInorder(node) {
   if (node is null)
       return

   BSTPrintInorder(node->left)
   Print node
   BSTPrintInorder(node->right)
}
```

Applications

- Situation where searching is necessary and there is a lot of data entering and leaving the set
- Routing trees for network traffic



Common data center interconnect topology. Host to switch links are GigE and links between switches are 10 GigE.

https://storagemojo.com/2008/08/24/fat-trees-and-skinny-switches/

Worksheet

References

- https://www.cs.cmu.edu/~adamchik/15-121/lectures/Trees/trees.html

- https://en.wikipedia.org/wiki/Euler_tour_technique

- ZyBooks text

Project #2

- Due April 15
- Using MIDIFile, you will populate a BST with 50 random notes.
- I will provide you with notes on how to install the library in Cloud9 in the assignment.
- You'll need to create a BST class
- Main task: Build sound files with the notes after executing pre-order, in-order, and post-order traversals (3 sound files).

Next Class

- Work on your project
- Read about Heaps and Balanced Trees
- Remaining Semester:
  - Exam #1 - in class April 18
  - Graphs
  - B-Trees
  - Algorithms: Grover's algorithm, divide and conquer algorithms, greedy programming, dynamic programming
  - Final project (final exam)