

Binary Search Trees



Fawzi Emad
Chau-Wen Tseng

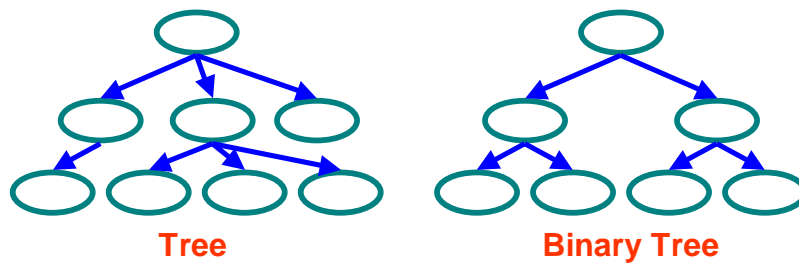
Department of Computer Science
University of Maryland, College Park

Overview

- **Binary trees**
- **Binary search trees**
 - **Find**
 - **Insert**
 - **Delete**

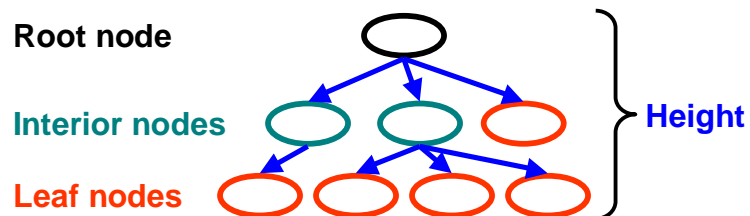
Hierarchical Data Structures

- One-to-many relationship between elements
- Tree
 - Single parent, multiple children
- Binary tree
 - Tree with 0–2 children per node



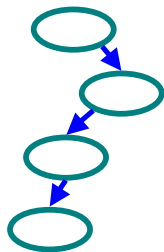
Trees

- Terminology
 - Root \Rightarrow no predecessor
 - Leaf \Rightarrow no successor
 - Interior \Rightarrow non-leaf
 - Height \Rightarrow distance from root to leaf

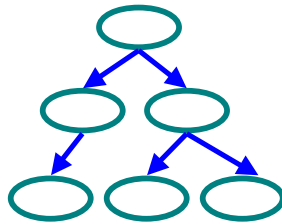


Types of Binary Trees

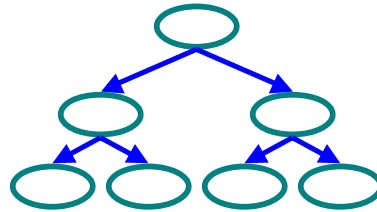
- Degenerate – only one child
- Balanced – mostly two children
- Complete – always two children



Degenerate
binary tree



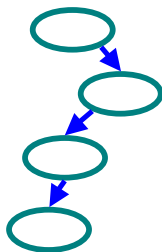
Balanced
binary tree



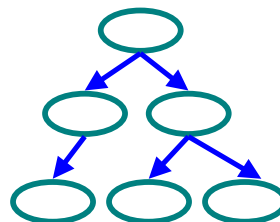
Complete
binary tree

Binary Trees Properties

- Degenerate
 - Height = $O(n)$ for n nodes
 - Similar to linear list
- Balanced
 - Height = $O(\log(n))$ for n nodes
 - Useful for searches



Degenerate
binary tree



Balanced
binary tree

Binary Search Trees

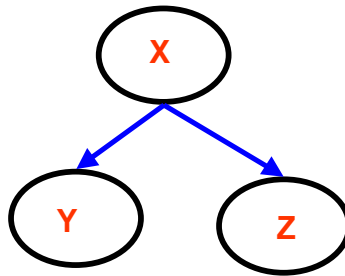
■ Key property

■ Value at node

- Smaller values in left subtree
- Larger values in right subtree

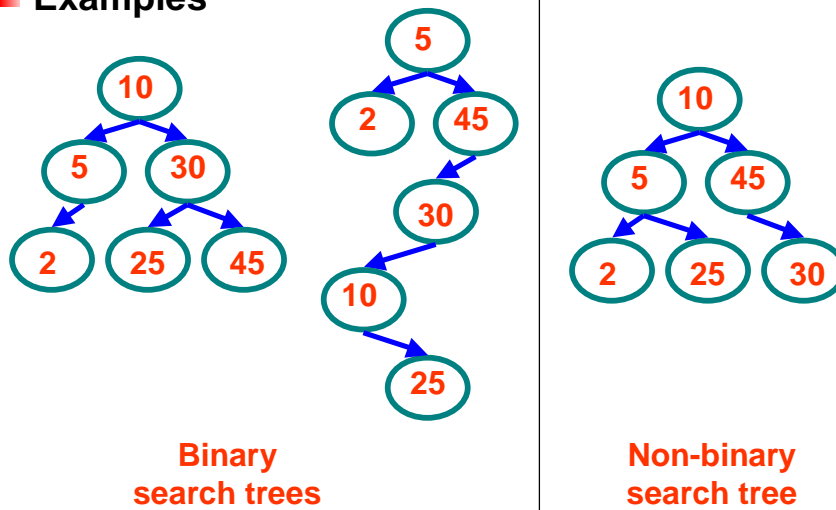
■ Example

- $X > Y$
- $X < Z$



Binary Search Trees

■ Examples



Binary Tree Implementation

```
Class Node {  
    Value data;  
    Node left, right; // null if empty  
  
    void insert ( Value data1 ) { ... }  
    void delete ( Value data2 ) { ... }  
    Node find ( Value data3 ) { ... }  
    ...  
}
```

Polymorphic Binary Tree Implement.

```
Interface Node {  
    Node insert ( Value data1 ) { ... }  
}  
Class EmptyNode {  
    Node insert ( Value data1 ) { ... }  
}  
Class NonEmptyNode {  
    Value data;  
    Node left, right; // Either Empty or NonEmpty  
    void insert ( Value data1 ) { ... }  
}
```

Iterative Search of Binary Tree

```
Node Find( Node n, Value key) {
    while (n != null) {
        if (n.data == key)    // Found it
            return n;
        if (n.data > key)    // In left subtree
            n = n.left;
        else                  // In right subtree
            n = n.right;
    }
    return null;
}
Find( root );
```

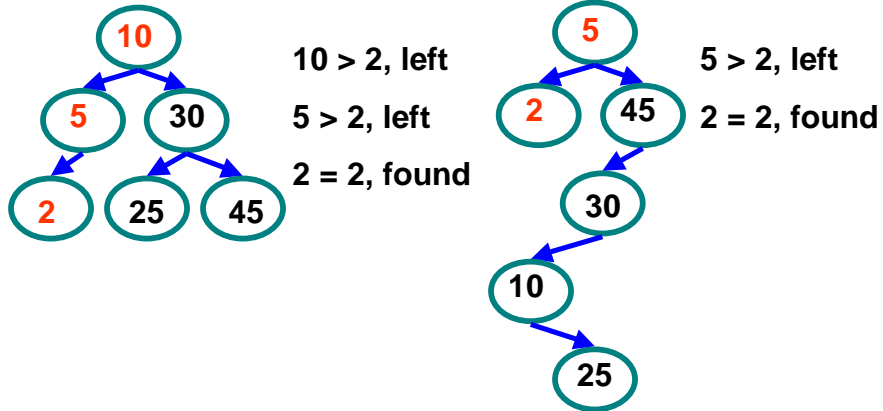
Recursive Search of Binary Tree

```
Node Find( Node n, Value key) {
    if (n == null)          // Not found
        return( n );
    else if (n.data == key) // Found it
        return( n );
    else if (n.data > key)  // In left subtree
        return Find( n.left );
    else                    // In right subtree
        return Find( n.right );
}

Find( root );
```

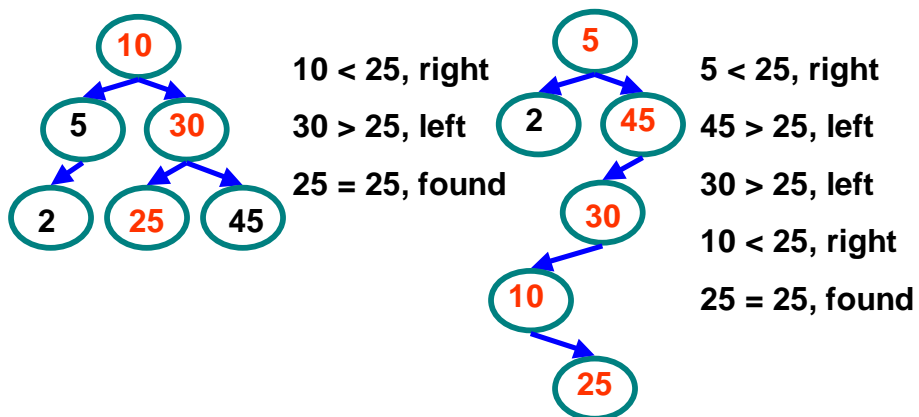
Example Binary Searches

■ Find (2)



Example Binary Searches

■ Find (25)



Binary Search Properties

- **Time of search**
 - Proportional to height of tree
 - **Balanced binary tree**
 - $O(\log(n))$ time
 - **Degenerate tree**
 - $O(n)$ time
 - Like searching linked list / unsorted array
- **Requires**
 - Ability to **compare** key values

Binary Search Tree Construction

- **How to build & maintain binary trees?**
 - **Insertion**
 - **Deletion**
- **Maintain key property (invariant)**
 - **Smaller values in left subtree**
 - **Larger values in right subtree**

Binary Search Tree – Insertion

■ Algorithm

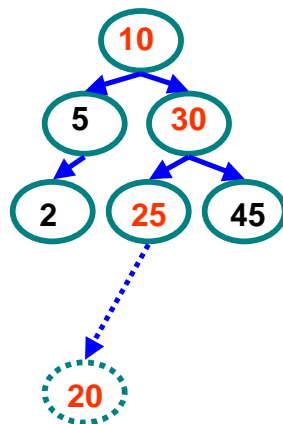
1. Perform search for value X
2. Search will end at node Y (if X not in tree)
3. If $X < Y$, insert new leaf X as new left subtree for Y
4. If $X > Y$, insert new leaf X as new right subtree for Y

■ Observations

- $O(\log(n))$ operation for balanced tree
- Insertions may unbalance tree

Example Insertion

■ Insert (20)



10 < 20, right

30 > 20, left

25 > 20, left

Insert 20 on left

Binary Search Tree – Deletion

■ Algorithm

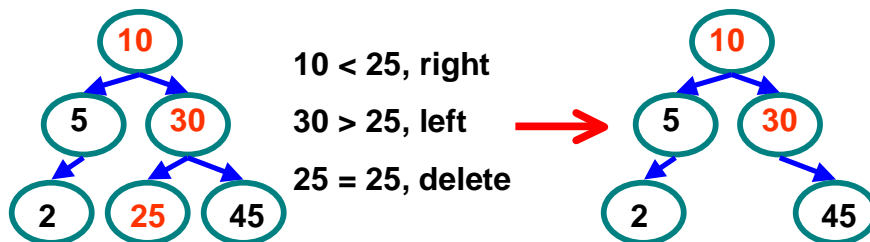
1. Perform search for value X
2. If X is a leaf, delete X
3. Else // must delete internal node
 - a) Replace with largest value Y on left subtree
OR smallest value Z on right subtree
 - b) Delete replacement value (Y or Z) from subtree

■ Observation

- $O(\log(n))$ operation for balanced tree
- Deletions may unbalance tree

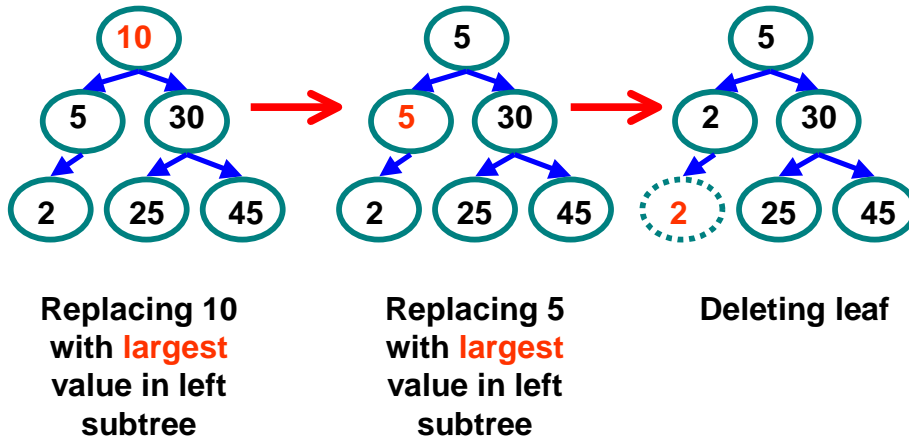
Example Deletion (Leaf)

■ Delete (25)



Example Deletion (Internal Node)

■ Delete (10)



Example Deletion (Internal Node)

■ Delete (10)

