Binary Search Tree Notation

- \( T \) - binary search tree
- \( T.\text{root} \) - root node of the tree \( \text{might be NIL} \)
- \( x, y, z \) - nodes in the tree \( \text{NIL} \)
- \( x.\text{key} \) - key to search for \( k \)
- \( x.\text{parent} \) - parent
- \( x.\text{left}, x.\text{right} \) - left and right child
- \( n \) - number of nodes
- \( h \) - height of the tree
Tree Search

Assume x is in a tree that satisfies binary search tree property.

TREE-SEARCH(x, k)

1. if x == NIL or k == x.key  \[\text{initial call Tree-Search(T, root, k)}\]
2. \[\text{if } k \text{ is in the tree, it must be in } x\text{'s subtree}\]
3. return x  \[\text{k is not in the tree or } k = x\text{.key}\]
4. if k < x.key
5. \[\text{return Tree-Search(x.left, k)}\]
6. else return Tree-Search(x.right, k)

ITERATIVE-TREE-SEARCH(x, k)

1. while x \neq NIL and k \neq x.key
2. if k < x.key
3. \[x = x.left\]
4. else \[x = x.right\]
5. return x
**Tree Successor**

**Tree-Successor**($x$)

1. if $x$.right $\neq$ NIL
2. return TREE-MINIMUM($x$.right)
3. $y = x$.p
4. while $y \neq$ NIL and $x = y$.right
5. $x = y$
6. $y = y$.p
7. return $y$

**Theorem 12.2**

We can implement the dynamic-set operations SEARCH, MINIMUM, MAXIMUM, SUCCESSOR, and PREDECESSOR so that each one runs in $O(h)$ time on a binary search tree of height $h$. 

Either one pass down the tree
or one pass up the tree
Initial call \text{Walk} (\text{Tree-Minimum}(T, \text{root}))

\text{Walk}(x)

\text{while } x \neq \text{NIL}

\text{process } x

x = \text{Tree-Successor}(x)

Does an inorder walk,

means process nodes in

sorted order.

This walk algorithm is $O(n)$

by Amortized Analysis.
Tree Insert  

\[ z = \text{node to insert} \quad x = \text{search for } z.\text{key} \]

Assume \( z \) is a new node to insert, \( z.\text{key} \) has been initialized, but \( \exists p = z.\text{left} = z.\text{right} = \text{NIL} \)

**TREE-INSERT(\( T, z \))**

1. \( y = \text{NIL} \)
2. \( x = T.\text{root} \)
3. \( \text{while } x \neq \text{NIL} \)
   - \( y = x \)
   - \( \text{if } z.\text{key} < x.\text{key} \)
     - \( x = x.\text{left} \)
   - \( \text{else } x = x.\text{right} \)
4. \( z.p = y \quad x = \text{NIL} \)
5. \( \text{if } y = \text{NIL} \)
   - \( \text{no iterations of loop, } x = \text{NIL} \text{ initially} \)
6. \( T.\text{root} = z \quad \text{// tree } T \text{ was empty} \)
7. \( \text{elseif } z.\text{key} < y.\text{key} \)
   - \( y.\text{left} = z \)
8. \( \text{else } y.\text{right} = z \)
Tree Deletion Example

![Tree Diagram]
**Tree Delete Illustration 1**

\[ z = \text{node to delete} \]

- \[ q = z.p \]
- \[ l = z.left \]
- \[ r = z.right \]

Transplant \((z, r)\)

Transplant \((z, l)\)

Transplant \((z, NIL)\)

- **BST Property**
- **Examples**
- **Tree Walk**
- **Search**
- **Min and Max**
- **Successor**
- **Insert**
- **Delete 1**
- **Delete 2**
- **Delete 4**
- **Delete 3**
- **Random Trees**

---

**CS 3343 Analysis of Algorithms**

Binary Search Trees – 9
Tree Delete Illustration 2

BST Property
Examples
Tree Walk
Search
Min and Max
Successor
Insert
Delete 1
DELETE 2
Delete 4
Delete 3
Random Trees

\[ y = \text{minimum in } z's \text{ right subtree} \]

\[ x = y.\text{right} \]

\[ \text{Transplant}(z, y) \]

\[ \text{Transplant}(z, y) \]

\[ \text{fix } x.\text{p}, \text{fix } y.\text{right} \]