**Search:** Given a set of \( n \) entries each associated with key \( x \) and value \( v_i \),
- Insert \( (x, v) \) in dict. (No duplicates)
- Delete \( x \) from dict. (Error if \( x \) not there)
- Find \( (K, x) \) - returns a reference to associated value \( v_i \), or null if not there.

**Binary Search Trees**
- Basic definitions
- Finding keys

**Efficiency:** Depends on tree's height
- Balanced: \( O(\log n) \)
- Unbalanced: \( O(n) \)

**Example:**
- Initial call: \( \text{find}(5, \text{root}) \)
- \( \text{find}(x, p) \), \( p.\text{find}(x) \)

**Idea:** Store entries in binary tree sorted (inorder traversal) by key
- Hashing

```
public
   find(x, root) {
      if (p == null) return null
      else if (x < p.key) help(p.left)
      else if (x > p.key) help(p.right)
      else return p.value
   }
```

- Can we achieve \( O(\log n) \) time for all ops?
- *Binary Search Trees*
BSTNode insert (Key x, Value v, BSTNode p)

if (p == null)
    → p = new BSTNode (x, v)
else if (x < p.key)
    p.left = insert (x, v, p.left)
else if (x > p.key)
    p.right = insert (x, v, p.right)
else throw exception → Duplicate!

return p

Why did we do: p.left = insert (x, v, p.left)?

Time: $O(\text{height})$

$p1$ : insert (14) → return p

$p1$.left = insert (14, v, p1.left)

$p2$ = new BSTNode

return p2

Insert (Key x, Value v)
- find x in tree
- if found ⇒ error! duplicate key
- else: create new node where we "fell out"

Replacement Node?

Inorder successor

3. $\cdot$ has two children

Find replacement node

1. Copy to $\cdot$, and then delete $\cdot$

3 cases:
1. $\cdot$ is a leaf
    return null
2. $\cdot$ has single child
    return "non-null" child

Binary Search Trees II
- insertion
- deletion

Delete (Key x)
- find x
- if not found ⇒ error
- else: remove this node & restore BST structure

How?
Java Implementation:
- Parameterize Key + Value types:
  extends Comparable
  class BinSearchTree<K,V>:
- BSTNode - inner class
- Private data: BSTNode root
- insert, delete, find: local
- provide publicEns
  insert, delete, find

But height can vary from \(O(\log n)\) to \(O(n)\)...

Expected case is good

Thm: If \(n\) keys are inserted in random order, expected height is \(O(\log n)\).

Analysis:
All operations (find, insert, delete) run in \(O(h)\) time, where \(h = \text{tree's height}\)
Announcements: Thu 2/16

- **Homework 1**
  - Due: Tue, Feb 21, start of class
    → No late submissions

- **Programming Assignment 1**
  - Leftist Heaps
  - Almost ready
  - Due, Wed, Mar 1

Amortized cost - $\tau$

1. Runs
2. $+1$ cost per op
   \[ \Rightarrow \tau - 1 \text{ tokens} \]
   
   Total: \[ \frac{m-1}{m} = \tau \]
   
   $\left( \tau - 1 \right)$ (ops until expansion) \[ \geq \text{(cost of expansion)} \]

3. Solve for $\tau$

   4. root
      
      \[ \cdots \]
      
      \[ \rightarrow \]
      
      \[ \rightarrow \]
public class BSTree<Key extends Comparable<Value> { 

private BSTNode { 
  Key key;
  Value value;
  BSTNode left, right;
} 

private insert helper 
protected find helper 
delete helper 

public insert(Key x, Value v) 
  T insert(x, v, root) 
}