Dictionary:
- `insert(Key x, Value v)`
  - Insert `(x, v)` in dict. (No duplicates)
- `delete(Key x)`
  - Delete `x` from dict. (Error if `x` not there)
- `find(Key x)`
  - Returns a reference to associated value `v`, or null if not there.

Search:
- Given a set of `n` entries each associated with key `x`:
  - Store for quick access & updates
  - Ordered: Assume that keys are totally ordered: `<`, `>`, `==`

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

Sequential Allocation?
- Store in array sorted by key
  - Find: `O(log n)` by binary search
  - Insert/Delete: `O(n)` time

Can we achieve `O(log n)` time for all ops?

Binary Search Trees I
- Basic definitions
- Finding keys

Find: How to find a key in the tree?
- Start at root `p=root`
- if `(x < p.key)` search left
- if `(x > p.key)` search right
- if `(x == p.key)` found it!
- if `(p == null)` not there!

Example:
- `find(5)`
- `find(14)`

Value: `find(Key x, BSTNode p)`
- if `(p == null)` return null
- else if `(x < p.key)` return `find(x, p.left)`
- else if `(x > p.key)` return `find(x, p.right)`
- else return `p.value`
Insert (Key x, Value v)
- find x in tree
- if found ⇒ error! duplicate key
- else: create new node where we “fell out”

Replacement Node?
θ

Inorder successor

Find replacement node
θ, copy to θ, and then delete θ

3 cases:
1. θ is a leaf
2. θ has single child
3. θ has two children

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

Why did we do:
p.left = insert(x, v, p.left)?
BSTNode delete (Key x, BSTNode p)
if (p == null) error! Key not found
else
  if (x < p.key)
    p.left = delete (x, p.left)
  else if (x > p.key)
    p.right = delete (x, p.right)
  else if (either p.left or p.right null)
    if (p.left == null)
      return p.right
    if (p.right == null)
      return p.left
  else
    r = findReplacement (p)
    copy r's contents to p
    p.right = delete (r.key, p.right)
  return p

Find Replacement Node
BSTNode find Replacement (BSTNode p)
BSTNode r = p.right
while (r.left != null)
  r = r.left
return r

Binary Search Trees III
- deletion
- analysis
- Java

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

Analysis:
All operations (find, insert, delete) run in O(h) time, where h = tree's height.

Java Implementation:
- Parameterize Key + Value types: extends Comparable
  class BinSearchTree(K, V),
- BSTNode - inner class
- Private data: BSTNode root
- insert, delete, find: local
- provide public ens
  insert, delete, find

But height can vary from O(logn) to O(n)...