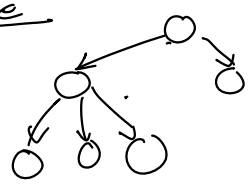
Dictionaries — new ADT

allows us to associate "keys" with "values"
dictionary is a set of key-value pairs where each key appears uniquely (at most once)

V get(K key)
void insert(K key, V value)
void update(K key, V value)
V remove(K key)

Implementation of Dictionary using Binary Search Tree



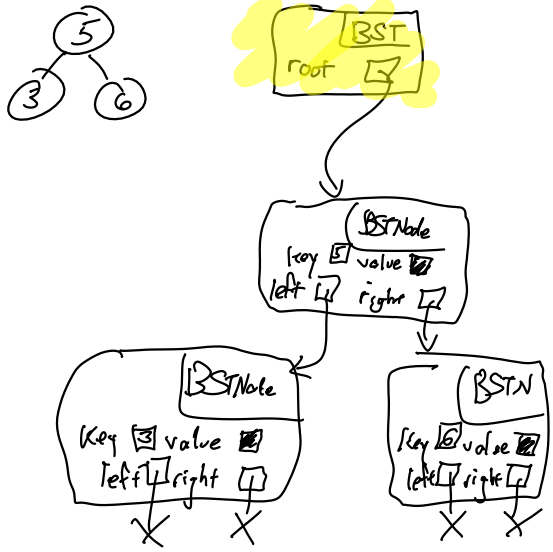
what is a tree? collection of connected, acyclic nodes with a "root" node which has no parent and is an ancestor to all other nodes; all other nodes have 1 parent

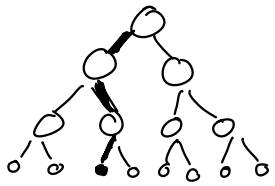
what is a binary tree? a tree where each node has at most 1 left child & 1 right child

what is binary search tree (BST)? a binary tree s.t. for every node, all left descendents have lesser key and all right descendents have greater key

Key must be totally ordered (comparable)

```
Method get(K key):
    Return getInSubtree(key, this->root)
End Method
Method getInSubtree(K key, Node* node):
    If node == nullptr:
        Throw runtime error
    Else:
        If node->key == key:
            Return node->value
        Else If node->key < key:
            Return getInSubtree(key, node->right)
        Else:
            Return getInSubtree(key, node->left)
        EndIf
    EndIf
EndMethod
```





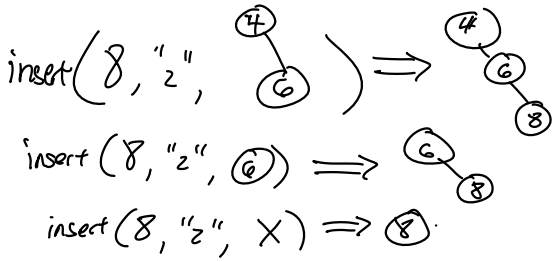
$O(\log_2 n)$

Exercise: implement void insert(K key, V value) RECURSIVELY

```
Method insert(K key, V value):
  this->root<-insertInSubtree(key, value, this->root)
End Method
Method insertInSubtree(K key, V value, Node* node):   <- returns replacement for node
    If node == nullptr:
        Return new Node(key, value, nullptr, nullptr)
    Else:
        If node->key == key:
            ||
            ~
        Else If node->key < key:
            node->right<-insertInSubtree(key, value, node->right)
            Return node
        Else
            node->left<-insertInSubtree(key, value, node->left)
            Return node
        End If
    End If
End Method
```

insert(5, "hello", X) ⟹ 5

insert(8, "z", 6 ...) ⟹



```
Method remove(K key):
    this->root ← removeInSubtree(key, this->root)
End Method
Method removeInSubtree(key, node):
    If node == nullptr:
    Else:
        If node->key < key:
            node->right ← removeInSubtree(key, node->right)
            Return node
        Else node->key > key:
            node->left ← removeInSubtree(key, node->left)
            Return node
        Else:
            If node is a leaf:
                Return nullptr
            Else If node has one child:
                Return that child
            Else:
                smallkey ← findMinKey(node->right)
                smallvalue ← getInSubtree(smallkey, node->right)
                node->right ← removeInSubtree(smallkey, node->right)
                node->key ← smallkey
                node->value ← smallvalue
                Return node
            End If
        End If
End Method
```