# CS135 T08

More practice with BT & BSTs

# Agenda

- Recap of BT & BSTs data definition
- BT-Search Trace
- BST Dictionaries
- BSTDict add/remove

# BT Data Definition

```
;; A Binary Tree (BT) is one of:
;; * empty
;; * a Node

(define-struct node (key left right))
;; A Node is a (make-node Nat BT BT)
```

# BT Search

In class you covered how to search a BT. Let's Review!

```
;; (search k tree) produces true if k is in tree; false otherwise.
;; search: Nat BT → Bool
(define (search-bt k tree)
  (cond [(empty? tree) false]
        [else (or (= k (node-key tree))
                  (search-bt k (node-left tree))
                  (search-bt k (node-right tree)))]))
```

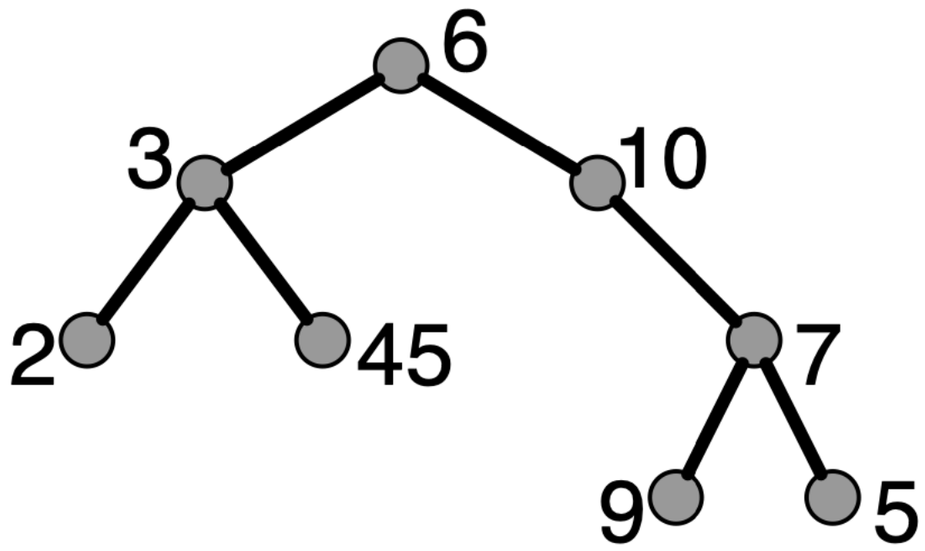
# BT Path

We also covered how to find the path to a certain node.

```
;; search-bt-path: Nat BT → (anyof false (listof (anyof 'right 'left)))
(define (search-bt-path k tree)
  (cond [(empty? tree) false]
        [(= k (node-key tree)) empty]
        [(list? (search-bt-path k (node-left tree)))
         (cons 'left (search-bt-path k (node-left tree)))]
        [(list? (search-bt-path k (node-right tree)))
         (cons 'right (search-bt-path k (node-right tree)))]
        [else false]))
```

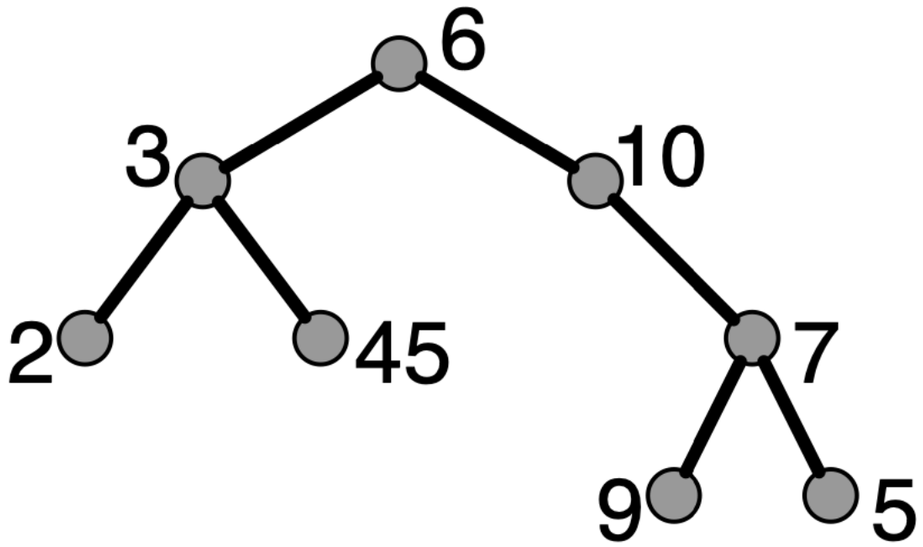
# Now let's trace it...

```
(define test-tree
(make-node 6 (make-node 3 ...)
                  (make-node 10 ...)))

(search-bt-path 0 test-tree)
(search-bt-path 9 test-tree)
(search-bt-path 3 test-tree)
```



# Now let's trace it…

```
(define test-tree
(make-node 6 (make-node 3 ...)
                  (make-node 10 ...)))

(search-bt-path 0 test-tree)
(search-bt-path 9 test-tree)
(search-bt-path 3 test-tree)
```




We're calling search-bt-path twice on the left and the right…
That's not good, but how do we fix it?

# Improved BT Path

```
;; search-bt-path: Nat BT → (anyof false (listof (anyof 'right 'left)))
(define (search-bt-path-v2 k tree)
  (cond [(empty? tree) false]
        [(= k (node-key tree)) empty]
        [else (choose-path-v2 (search-bt-path-v2 k (node-left tree))
                              (search-bt-path-v2 k (node-right tree)))]))

(define (choose-path-v2 left-path right-path)
  (cond [(list? left-path) (cons 'left left-path)]
        [(list? right-path) (cons 'right right-path)]
        [else false]))
```

# BST Data Definition

```
;; A Binary Search Tree (BST) is one of:
;; * empty
;; * a Node

(define-struct node (key left right))
;; A Node is a (make-node Nat BST BST)
;; Requires: key > every key in left BST
;; key < every key in right BST
```

# BST Dictionaries

```
(define-struct node (key val left right))
;; A binary search tree dictionary (BSTD) is either:
;; * empty
;; * (make-node Nat Str BSTD BSTD)
;; Requires: key > every key in left BSTD
;; key < every key in right BSTD
```

# Adding to a BSTD

Write a function BSTD-add that consumes Nat key, Str val and a BSTD and produces a new BSTD with new key added with the corresponding value and updates its value if the key already exists in the tree

# Removing from a BSTD

Write a function BSTD-remove that consumes a Nat key and a BSTD and produces a new BSTD with the key and its value removed and the same tree if the key doesn’t exist