6.006 Recitation 3

Note Title 2/11/2009

* Maximum Contiguous subset sum
* Master Theorem
* Lecture 3 / Recitation 2, Python Class Review
* Binary Search Trees - Search, Insert, Delete

Master Theorem

$T(n) = aT(n/b) + f(n)$

Cases

1) $f(n) = O(n^{\log_b a - \varepsilon})$, $\varepsilon > 0$ $\rightarrow$ $\Theta(n^{\log_b a})$
2) $f(n) = \Theta(n^{\log_b a})$ $\rightarrow$ $\Theta(n^{\log_b a} \log n)$
3) $f(n) = \Omega(n^{\log_b a + \varepsilon})$, $\varepsilon > 0$ $\rightarrow$ $\Theta(f(n))$ $af(n/b) \leq cf(n)$ $\exists c < 1, n \geq n_0$

$T(n) = 9T(n/3) + n$ $\Theta(n^2)$

$T(n) = T(2n/3) + 1$ $\Theta(\log n)$

$T(n) = 2T(n/2) + n$ $\Theta(n \log n)$

$T(n) = 3T(n/2) + n$ $\Theta(n^{\log_2 3})$

$T(n) = 3T(n/4) + n \log n$ $\Theta(n \log n)$

$T(n) = 2T(n/2) + n \log n$ (Master Theorem not directly applicable)

Classes In Python

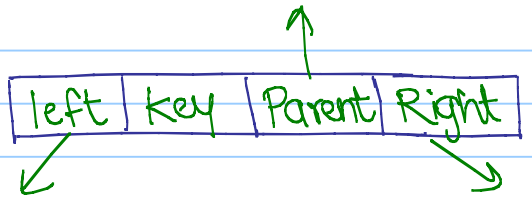
```
class Counter(object):
    num_counters = 0
    def __init__(self, start=0):
        self.count = start
        Counter.num_counters += 1
        self.id = Counter.num_counters
    def increment(self):
        self.count += 1
        return self.count
    def __repr__(self):
        return "<Counter #%d: %d>" % (self.id, self.count)

class UpDownCounter(Counter):
    def decrement(self):
        self.count -= 1
        return self.count
```

```
C = Counter(2)
C.increment()      3
C.increment()      4
D = UpDownCounter(5)
D.increment()      6
D.decrement()      5
```

Binary Search Trees

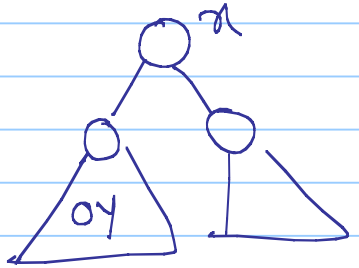
Linked data structure of nodes

node n

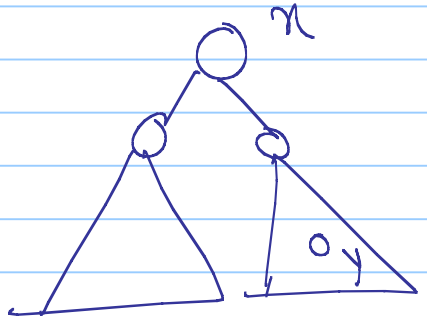


The invariant needed to be satisfied
$n \rightarrow$ node in BST

If y : node in left-subtree of $n \rightarrow key(y) \leq key(n)$

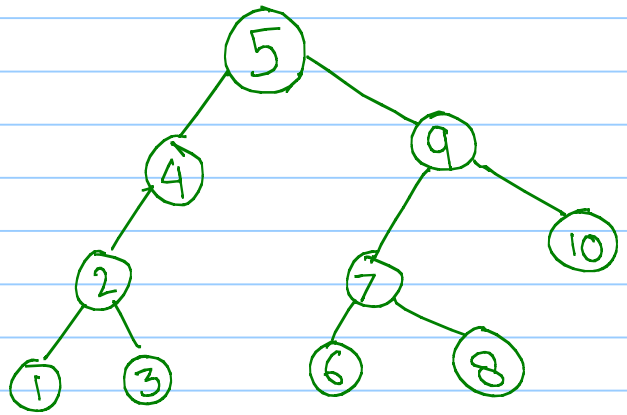


If y : node in right-subtree of $n \rightarrow key(y) \geq key(n)$

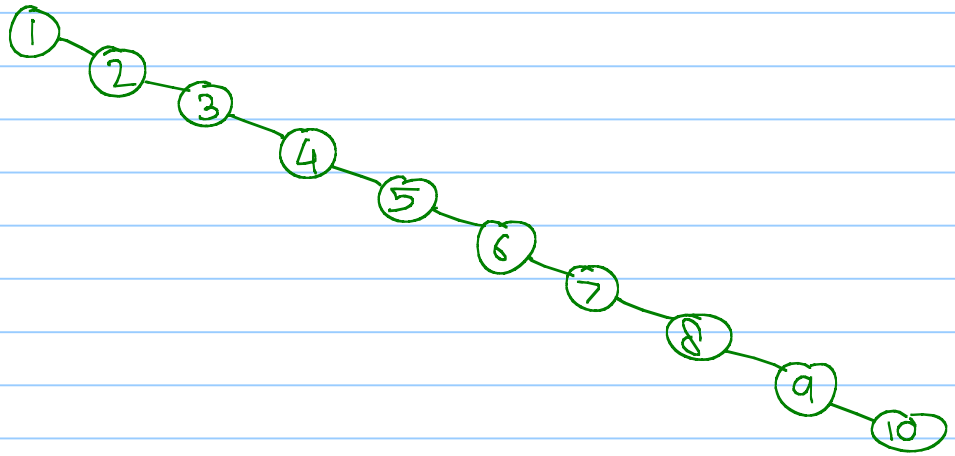


Examples

5 9 4 7 8 2 10 1 6 3



1 2 3 4 5 6 7 8 9 10



10 1 9 2 8 3 7 4 6 5

5 4 3 2 1 6 7 8 9 10

Inorder walk

```
INORDER-TREE-WALK(x)
 If x ≠ NIL
     then INORDER-TREE-WALK(left[x])
          print key[x]
          INORDER-TREE-WALK(right[x])
```

$O(n)$ — n: number of nodes in the tree

TREE-SEARCH

```
TREE-SEARCH(x,k)
   If x = NIL or key[x] = k
      return x
   If k < key[x]
      return TREE-SEARCH(left[x],k)
   else
      return TREE-SEARCH(right[x],k)
```

```
ITERATIVE TREE-SEARCH
 while x ≠ NIL and k ≠ key[x]
     if k < key[x]
        x = left[x]
     else
        x = right[x]
 return x
```

$O(h)$, h: height of the tree

TREE-MINIMUM

```
TREE-MINIMUM(x)
    while left[x] ≠ NIL
        x = left[x]
    return x
```

TREE-MAXIMUM

```
TREE-MAXIMUM(x)
    while right[x] ≠ NIL
        x = right[x]
    return x
```

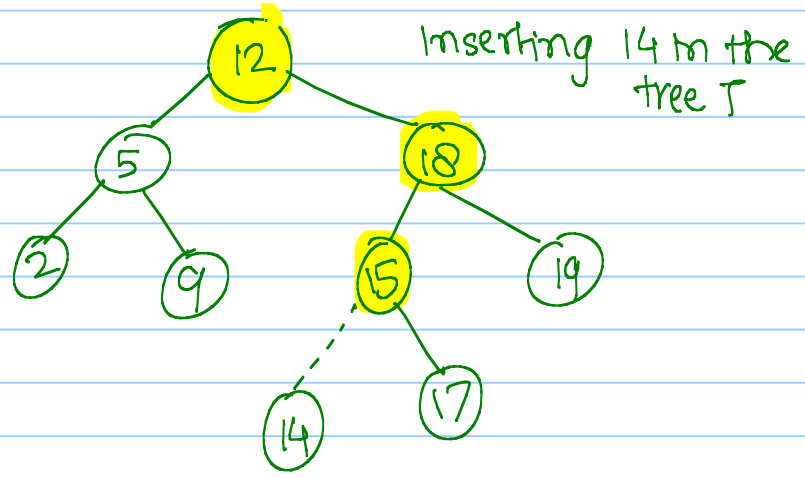
TREE-SUCCESSOR

```
TREE-SUCCESSOR(x)
    If right[x] ≠ NIL
      return TREE-MINIMUM(right[x])
    y = p[x]
    while y ≠ NIL and x = right[y]
        x = y
        y = p[y]
    return y
```

TREE-INSERT

```
TREE-INSERT (T, z)
    y = NIL
    x = root(T)
    While x ≠ NIL
        y = x
        if key[z] < key[x]
            x = left[x]
        else
            x = right[x]
    p[z] = y
    If y = NIL
        root[T] = z
    else if key[z] < key[y]
            left[y] = z
        else
            right[y] = z
```

Find the correct location to insert z



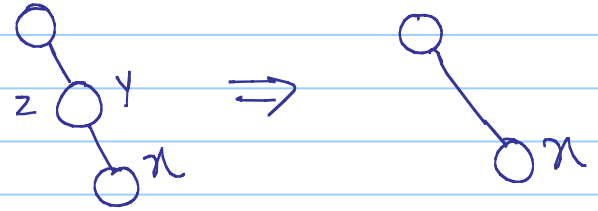
Inserting 14 in the tree T

TREE-DELETE

```
TREE-DELETE (T,z)
 If left[z] =NIL or right[z] = NIL
     y=z
 else
     y = TREE-SUCCESSOR(z)
 If left[y] ≠ NIL
     x = left[y]
 else
     x= right[y]
 If x≠NIL
     p[x]= p[y]
 If p[y] = NIL
    root [T]=x
 else if  y = left [p[y]]
          left[p[y]]=x
     else right[p[y]]=x
 If y≠z  key[z]= key[y]
 return y
```

a) If z has no children, delete it

b) if z has 1 child, splice out z



c) if z has both children, find its successor, splice it and copy key value