cse 101 5-5-20

Part: ext 1 more day

Dictionary ADT

finite

state: a set of Pairs (key, value)

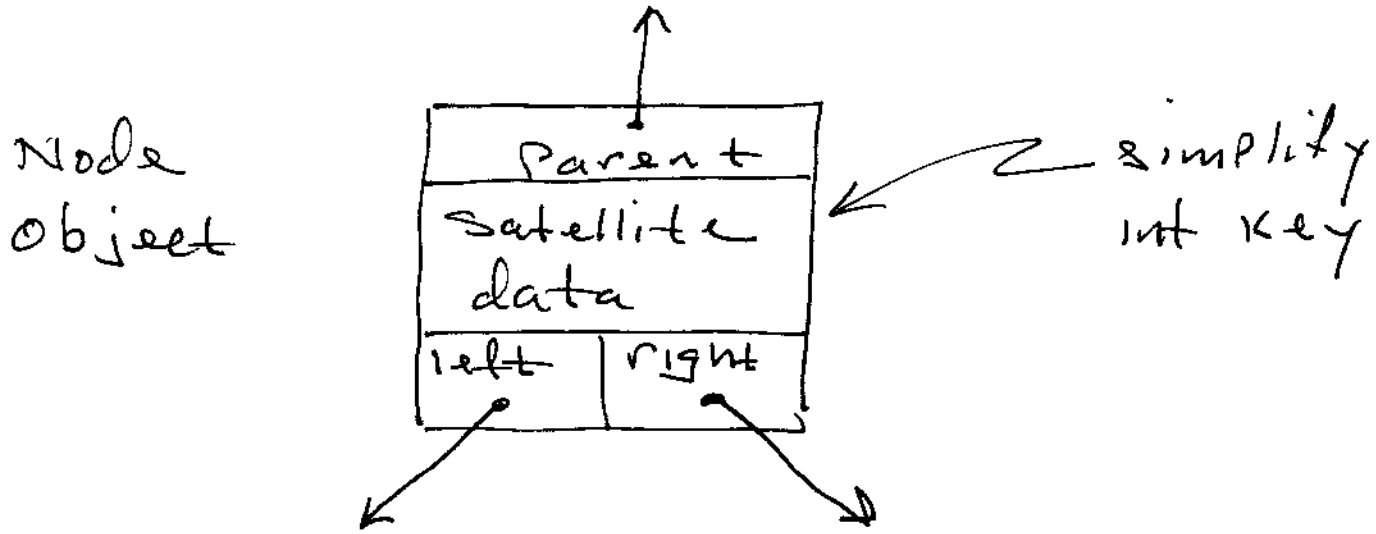
$\{(k_1, v_1), (k_2, v_2), \ldots\}$

ops:

- lookup(key)
- insert(key, value)
- delete(key)

Binary Search Tree (BST)

generalization of linked list.



(see chapter 12 in CLRS)

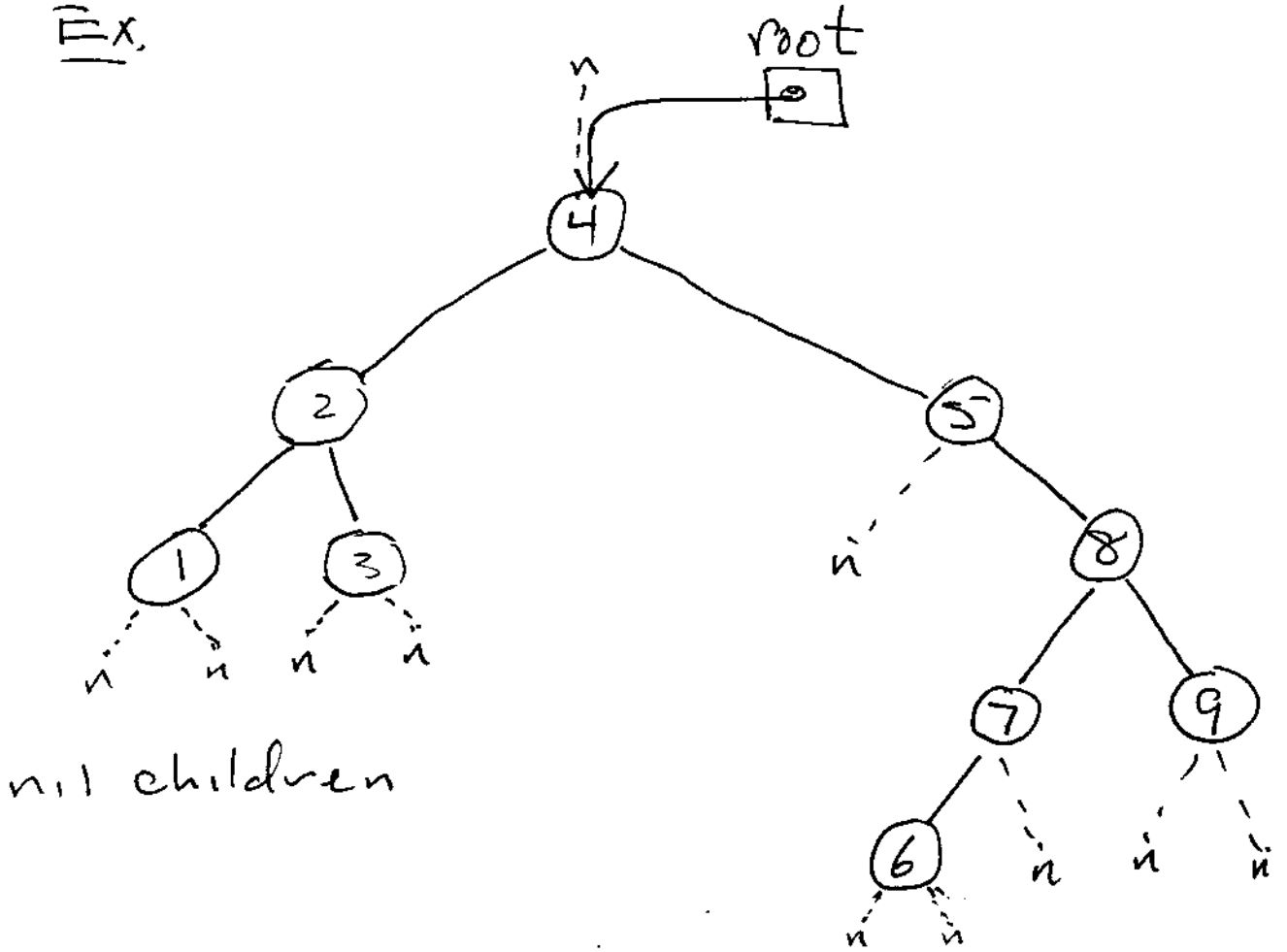
BST Properties:

let x, y be nodes in a BST. Then

(1) if y is in left-subtree of x,
then key[y] ≤ key[x]

(2) if y is in the right subtree of x,
then key[x] ≤ key[y]

Ex.



Defn

a leaf is a node with no children.

an internal node is a non-leaf.

[Recommend: Create a dummy node called nil as a field in DictionaryObj.]

tree traversal:

book:
x.left
C
x->left

InOrderTreeWalk(x)

1. if $x \neq nil$
2. InOrderTreeWalk(left[x])
3. Print(key[x])
4. InOrderTreeWalk(right[x])

InOrderTreeWalk(root) prints all keys in order.

Ex. output: 1 2 3 4 5 6 7 8 9

PreOrderTreeWalk(x)

1. if $x \neq nil$
2. Print(key[x])
3. PreOrderTreeWalk(left[x])
4. PreOrderTreeWalk(right[x])

Ex. tree PreOrderTreeWalk(root)

output: 4 2 1 3 5 8 7 6 9

PostOrderTreeWalk(x)

1. if $x \neq nil$
2. PostOrderTreeWalk(left[x])
3. PostOrderTreeWalk(right[x])
4. Print(key[x])

Ex. output: 1 3 2 6 7 9 8 5 4

Queries

TreeSearch(x, k)

```
1. if x == nil or k == key[x]
2.     return x
3. else if k < key[x]
4.     return TreeSearch(left[x])
5. else // k > key[x]
6.     return TreeSearch(right[x])
```

TreeMinimum(x)

Pre: $x \neq nil$

```
1. while left[x] ≠ nil
2.     x = left[x]
3. return x
```

Exercise re-write this so it returns nil if x is nil.

TreeMaximum(x)

1.

2.

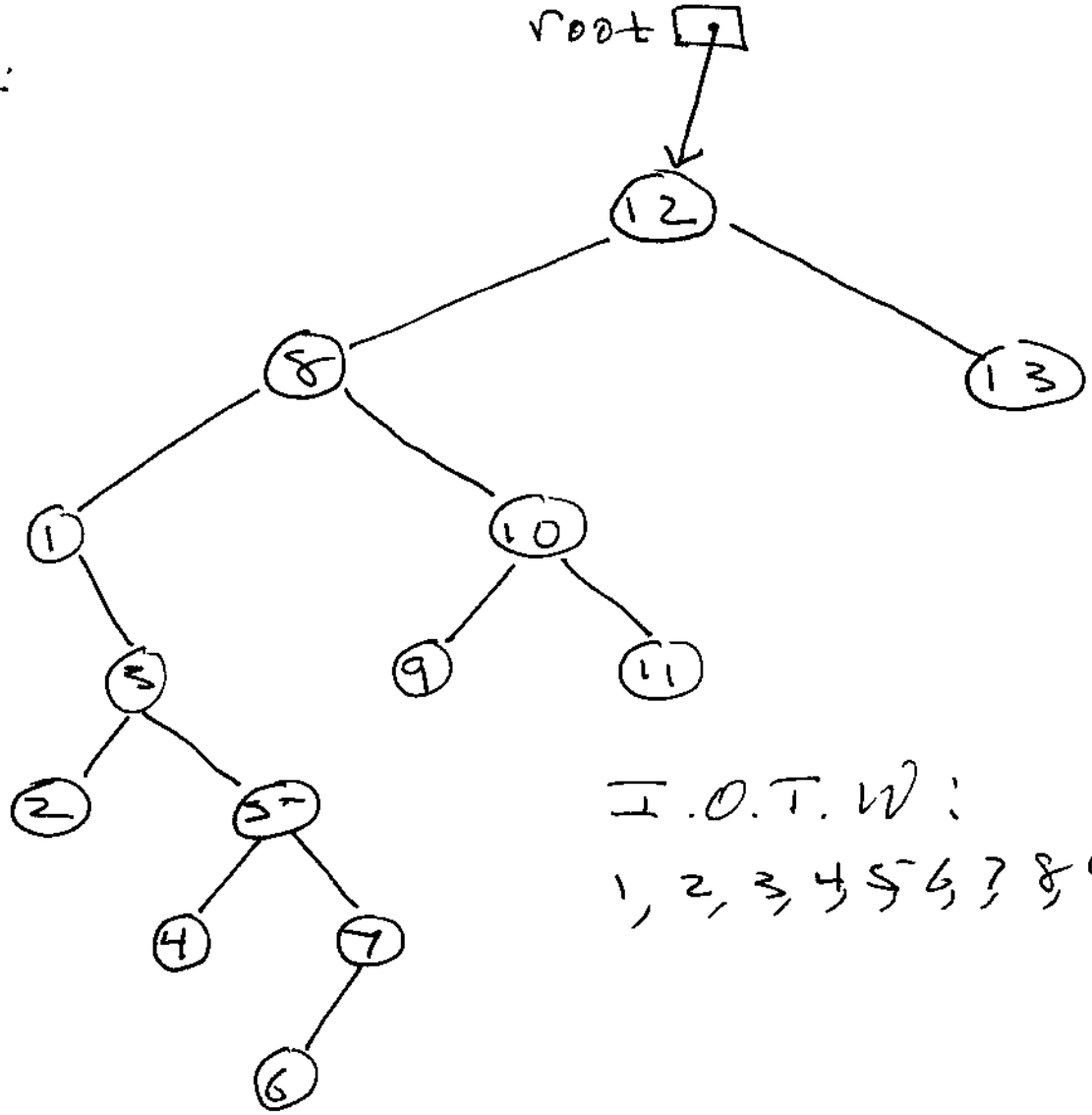
3.

Exercise: do this

Defn

The Successor of a Node x is the next Node to be Processed in an in order tree Walk.

Ex.



I.O.T.W:
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13

successor(1) = 2

successor(8) = 9

successor(7) = 8

TreeSuccessor(x)

```
1. if right[x] ≠ nil
2.     return TreeMinimum(right[x])
3. y = parent[x]
4. while y ≠ nil and x = right[y]
5.     x = y
6.     y = p[y]
7. return y
```