cse 101 5-7-20

Pa4: ext 1 last day

Back To BSTs .....

Defn

let x be a node in a BST, T.

The successor of x is the next node in T to be processed in an in-order tree walk of T.

note: if key[x] is maximum in T, then x has no successor.

The predecessor of x is the node processed before x in an in order tree walk (exists if x not min)

TreeSuccessor(x) { return successor of x if it exists, nil otherwise

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

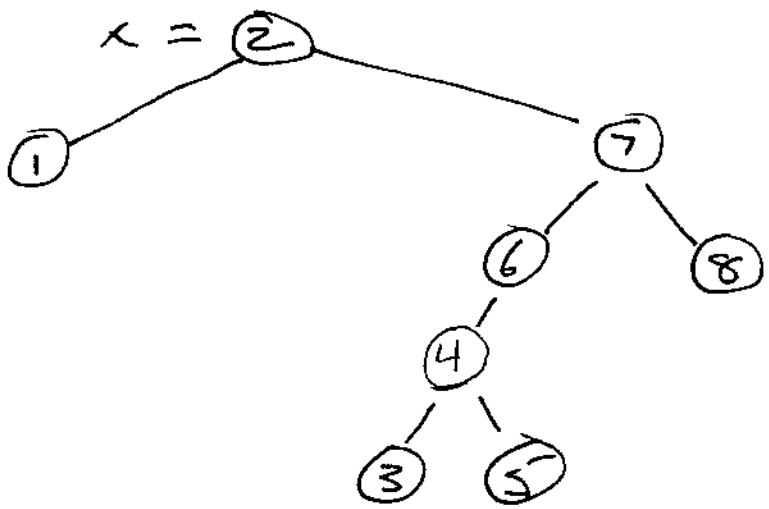
case 1: x has a right child.

In this case, the successor is left most node in x's right subtree.

Case 2: x has no right child

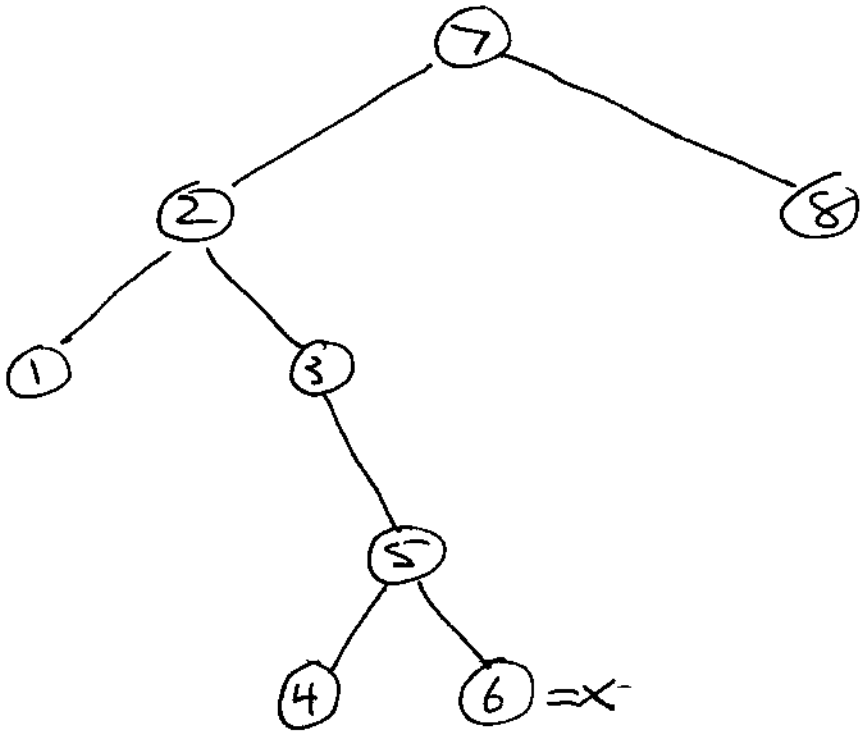
in this case, if x has a successor y, then y is the lowest ancestor of x whose left child is also an ancestor of x (or x itself). [See Prob. 12.2-6].

Ex. case 1 : x has right child




Successor(2) = 3

Ex Case 2 : x has no right child




Successor(6) = 7

Exercise

write TreePredecessor(x), which returns next node in reverse-order tree walk, if it exists. otherwise returns nil.

## 12.3 Insertion and Deletion

Goal: maintain BST Properties

to insert node z, let $k = key[z]$. set parent[z] = left[z] = right[z] = nil, load satellite data.

## 12.3 Insertion & Deletion

We maintain BST properties by careful insertion & deletion.

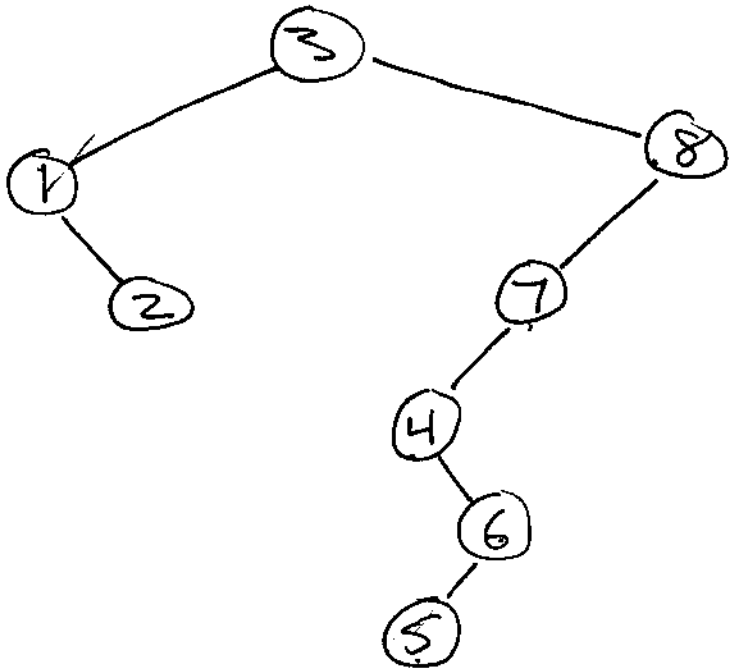
To insert a node z, first set key[z] = k, p[z] = left[z] = right[z] = nil, and load satellite data.

**Insert(T, z)**

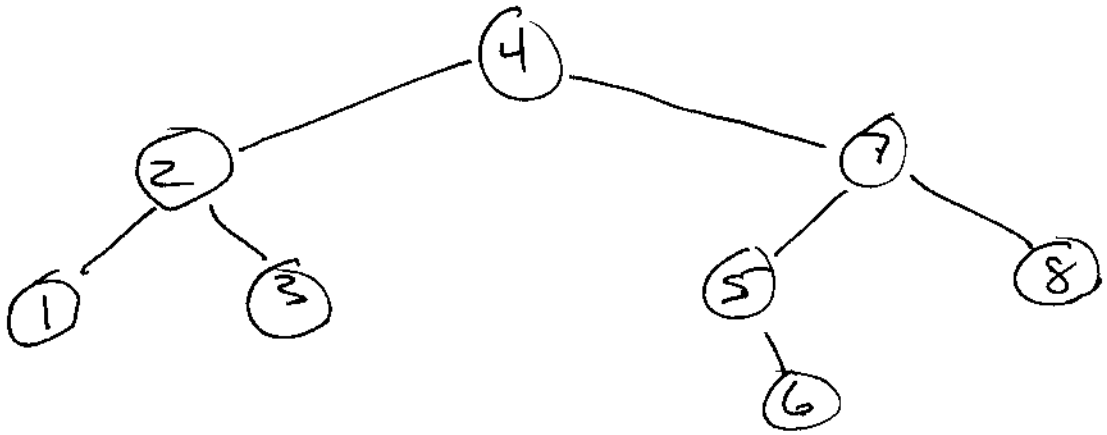
```
1.)  y = nil
2.)  x = root[T]
3.)  while x ≠ nil
4.)      y = x
5.)      if key[z] < key[x]
6.)          x = left[x]
7.)      else
8.)          x = right[x]
9.)  p[z] = y
10.) if y == nil      // T was empty
11.)     root[T] = z
12.) else if key[z] < key[y]
13.)     left[y] = z
14.) else
15.)     right[y] = z
```

Runtime: $\Theta(\text{height}(T))$.

Ex. insert keys : 3 8 1 2 7 4 6 5




Ex. insert same keys : 4 2 7 3 8 1 5 6



Runtime of algorithm that climbs up (or down) a line of ancestry is, in worst case $\Theta$(height of tree).

These algorithms are

TreeMinimum(x)

TreeMaximum(x)

Successor(x)

Predecessor(x)

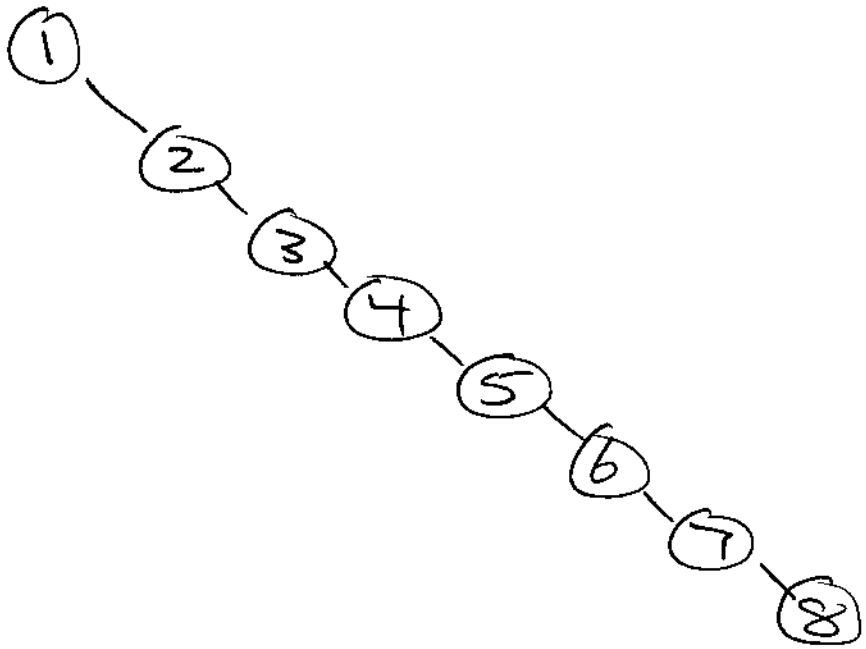
insert(x), delete(x)

all run in time $\Theta$(height of tree)

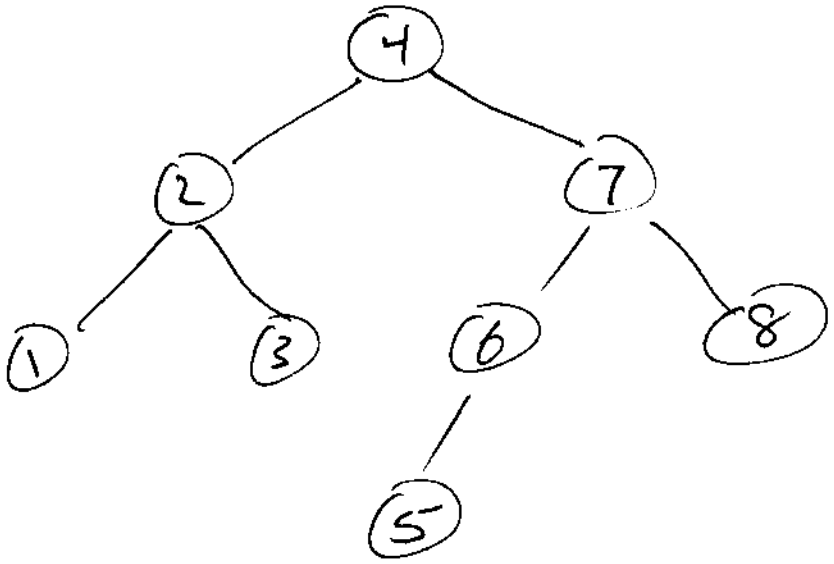
The traversals (in, Pre, Post-order) run in time $\Theta$(# nodes) = $\Theta(n)$

let $n$ = # nodes.

Ex insert: 1 2 3 4 5 6 7 8




how small can height be?

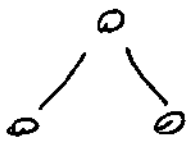



insertion order: 4 2 7 1 3 6 8 5

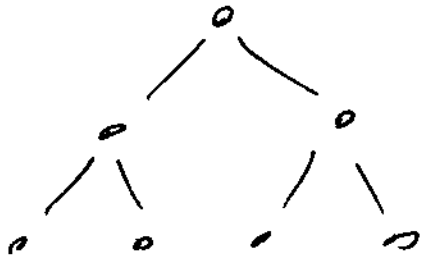
seq. of best trees :

$n = 1$ $h = 0$

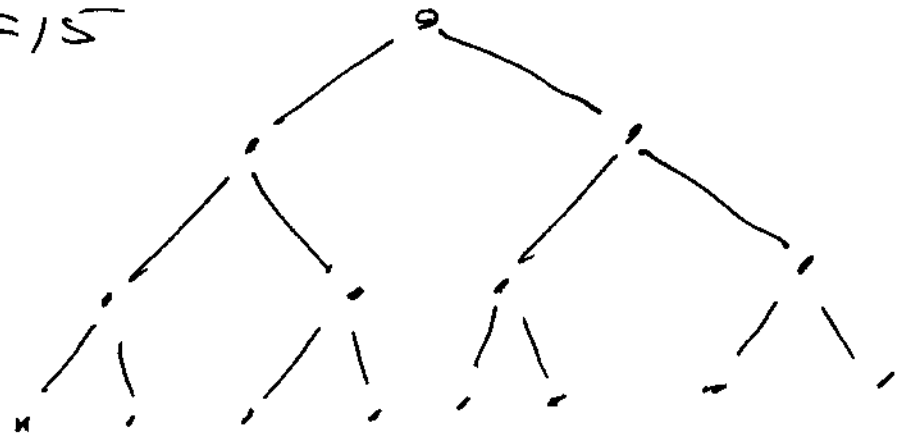
$n = 3$ $h = 1$



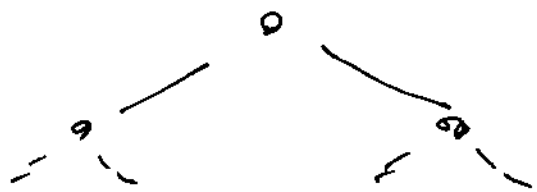
$n = 7$ $h = 2$



$n = 15$ $h = 3$



$n = 31$ $h = 4$



(Exercise)

These are called complete binary trees. (CRT). In a CRT

$$n = 2^{h+1} - 1$$

so $$h = \lg(n+1) - 1$$

Theorem If $T$ is a binary tree with $n$ nodes and height $h$, then

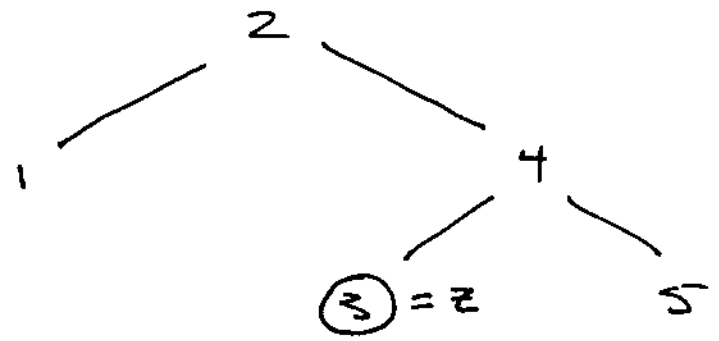
$$\boxed{\lfloor \lg(n) \rfloor \leq h \leq n-1}$$

Procedure for deletion has 3 cases. say z is the node to be deleted.

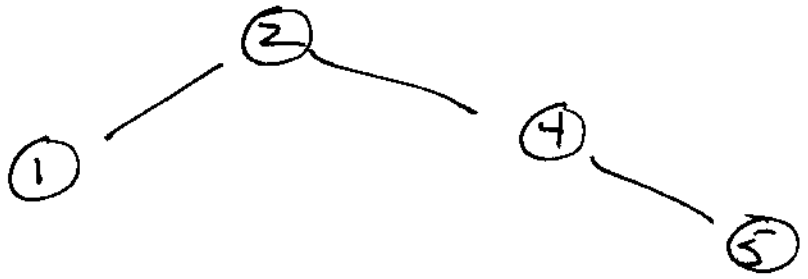
Case 1: z is a leaf.

modify P[z] have nil in place of z

Ex




delete(z)

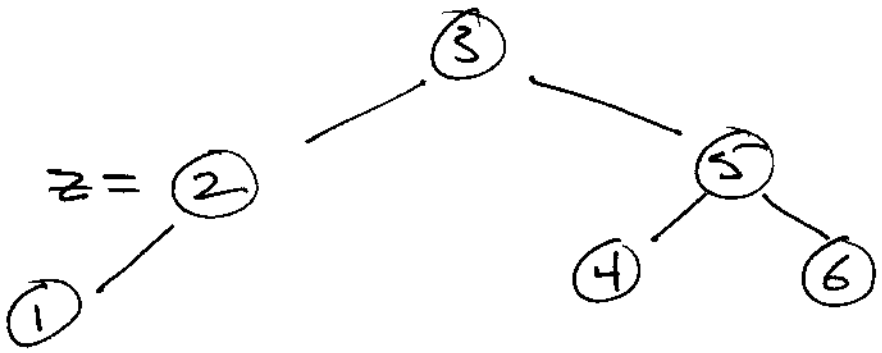


case 2 : z has 1 child

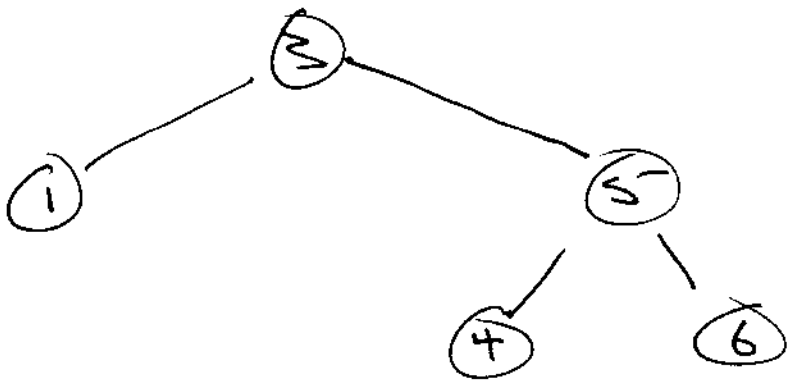
(2 subcases: right no left, left no right.)

splice out z, linking z's child to z's Parent.

Ex.



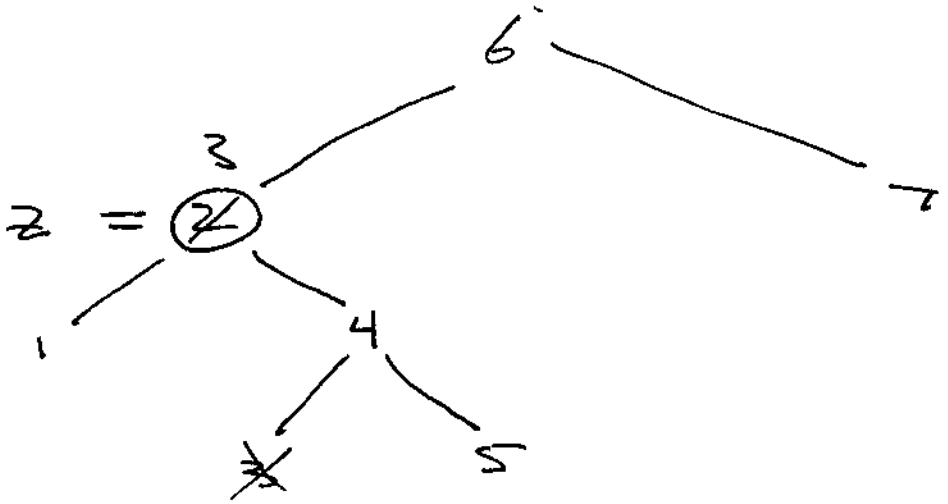

delete(2)



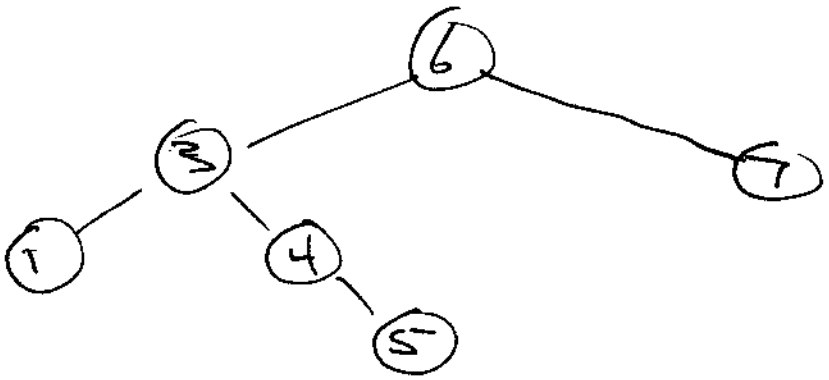
Case 3: z has 2 children

splice out z's successor y
(which has no left child!)
and copy data of y into z

Ex.




delete(2):



Exercise

write pseudo-code for delete in a BST.

Runtime: (worst case)

$O(\text{height of tree})$

Read book's version of delete()
(CLRS 3rd ed. sec. 12.3 P. 295-298)