Binary Search Tree (BST)

Cormen Sect 12.1, 12.2, 12.3

Predecessor Problem

Given a set $S \subseteq U$ of n keys, build a DS to support the following operations:

Dictionary Problem
- Insert (S, x)
- Delete (S, x)
- Search (S, x) / Lookup (S, x)

PQ
- Minimum (S)
- Maximum (S)

- Predecessor (S, x) returns the predecessor of x in S (even if $x \notin S$)
  i.e. the largest key in S which is smaller than ~~or equal to~~ x

$S = \{1 \quad 3 \quad 7 \quad 10\}$

Predecessor $(8) = 7$

- Successor (S, x) returns the successor of x in S (even if $x \notin S$)

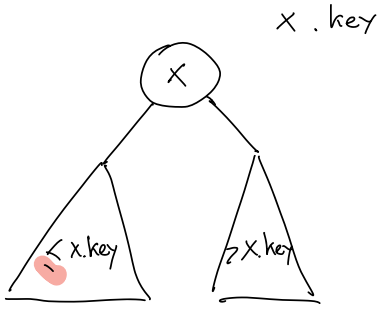
$S = \{1 \quad 3 \quad 7 \quad 10\}$

Successor $(8) = 10$

Solutions for this problem are often called Ordered dictionary / map

Static version of the problem solved with Binary Search on the sorted set (Pred/Succ/Search in $\Theta(\log n)$ time)

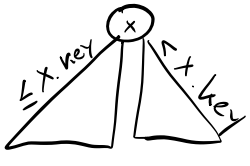
Keys of S "sorted" in a binary tree T



Binary Search tree property

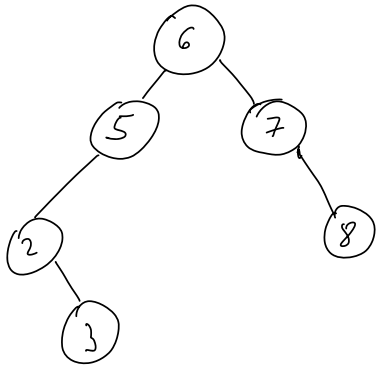
Let x any node of a BST

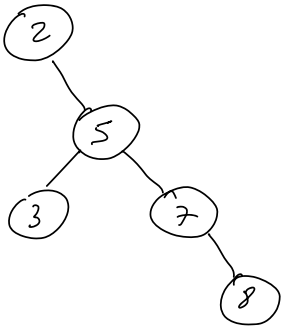
- If y is a node in left subtree of x, then $y.key \leq x.key$
- If y is a node in right subtree of x, then $x.key < y.key$

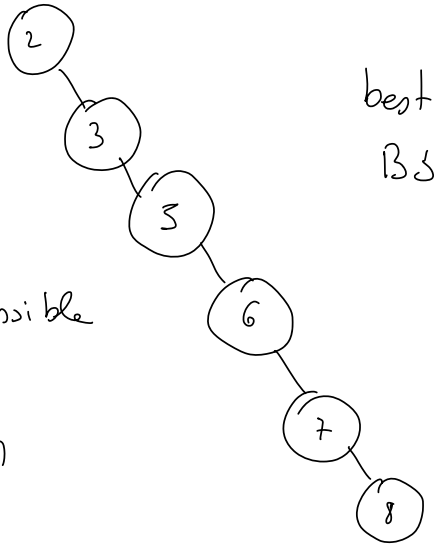


Several different BSTs for a set

$S = \{2\ 3\ 5\ 6\ 7\ 8\}$





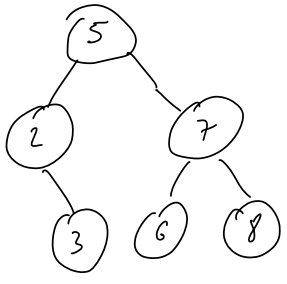


worst possible BST

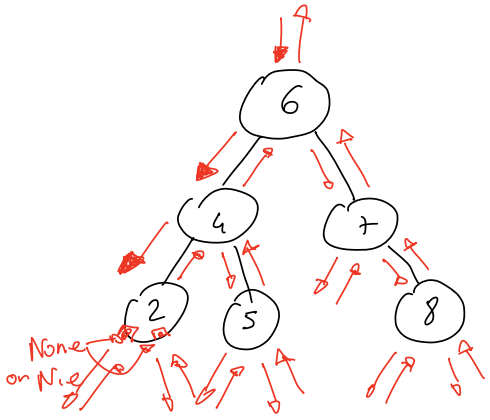
$h = n$

best possible $h = \Theta(\log n)$

BST = mimic Binary Search



Important property: if we print keys by traversing the tree inorder, then we get S sorted




```
Inorder-Visit(u)
    if u ≠ NIL
        Inorder-Visit(u.left)      # preorder
        print(u.key)               # postorder
        Inorder-Visit(u.right)
```

Output = 2 4 5 6 7 8

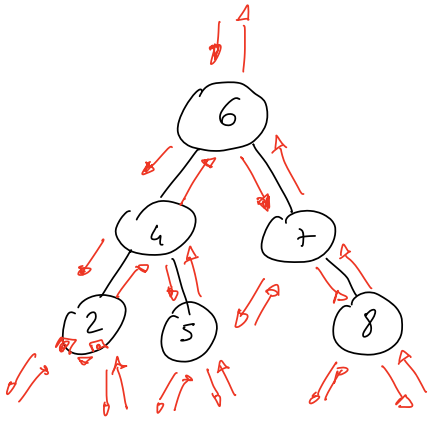
Complexity = $\Theta(n)$ time

x.key key stored in x

x.left Children of x

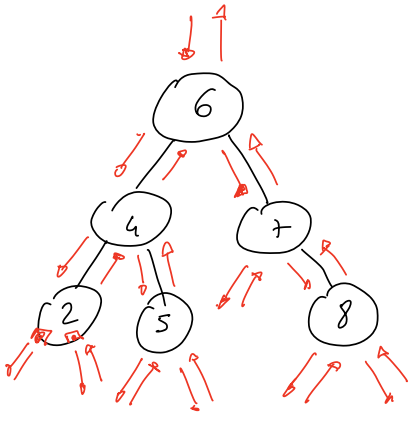
x.right NIL if child does not exist

Preorder visit



output: 6 4 2 5 7 8

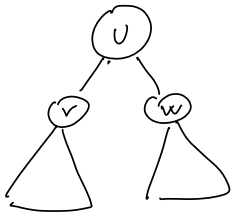
Post order visit



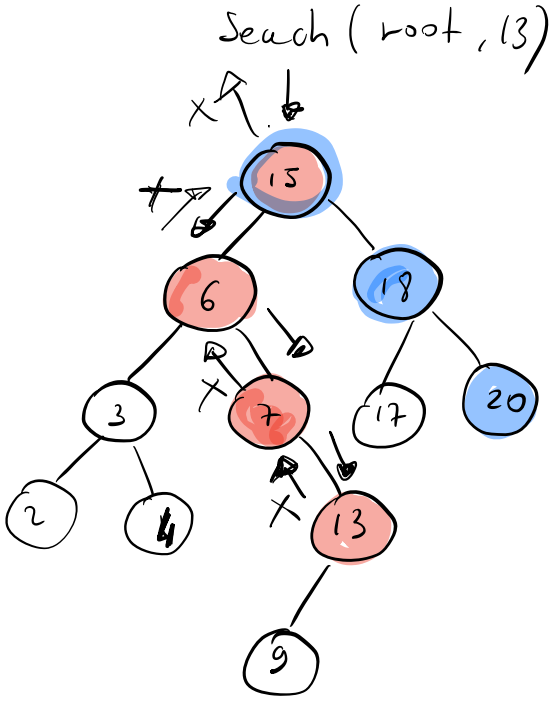
output: 2 5 4 8 7 6

```
height (u)
    if u == NIL return 0
    hl = height (u.left)
    hr = height (u.right)
    hu = max ( hl , hr ) + 1
    return hu
```

$\max(\text{height}(v), \text{height}(w)) + 1$



Searching



Search (13)

Search (21)

Search (x , k)

```
1  if x == NIL or x.key == k
2          return x
3  if k < x.key
4          return Search ( x.left , k)
5  else
6          return Search ( x.right , k)
```

Complexity $\Theta(h)$ time

Minimum and Maximum

```
Min (x)
1   if x.left == NIL
2       return x.key
3   return Min (x.left)
```

```
Min (x)
1   while x.left ≠ NIL
2       x = x.left
3   return x.key
```

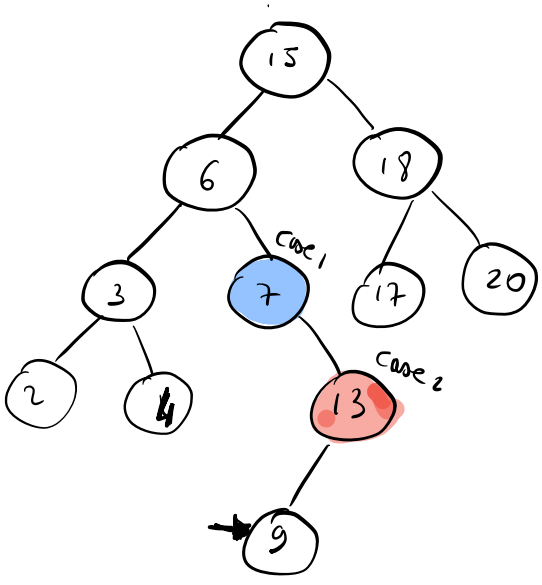
```
Max (x)
1   if x.right == NIL
2       return x.key
3   return Max (x.right)
```

```
Max (x)
1   while x.right ≠ NIL
2       x = x.right
3   return x.key
```

Complexity $\Theta(h)$ time

Predecessor and Successor

Focus: Reporting successor of a node x



there are two cases:

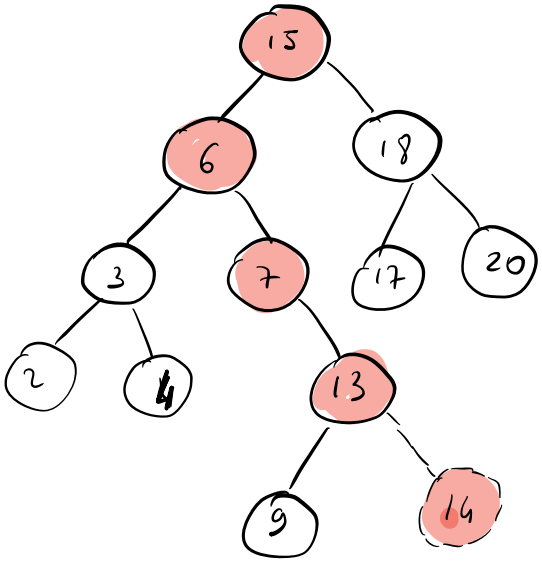
- if x has right subtree, x's successor is min in this sub tree
- if x has no right subtree, and x successor is node y. y is the lowest ancestor whose left child is also an ancestor of x (or x itself)

```
Successor(x)
1 if x.right ≠ NIL    # case 1
2     return min(x.right)
  # case 2
3 y = x.p
4 while y ≠ NIL and x == y.right
5         x = y
6         y = x.p
7 return y
```

x.p = points to the parent of x

Complexity = $\Theta(h)$ time

Insert

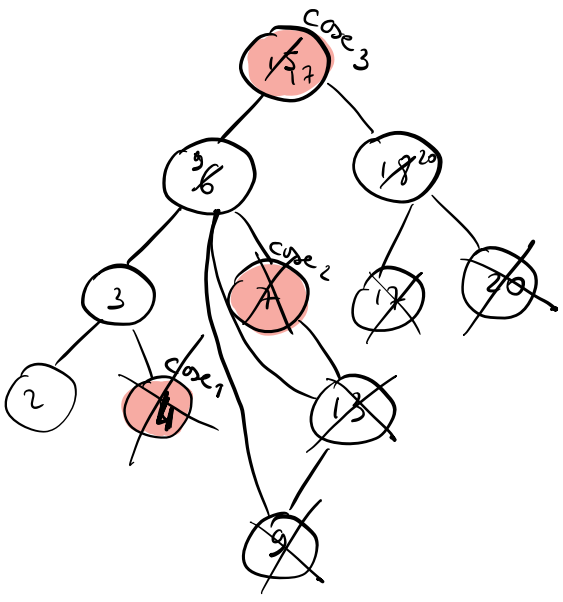


Insert (14)

Complexity : $\Theta(h)$ time

```
Insert (T, z)        // z is a new node { z.key = k, z.left = NIL, z.right = NIL
1   y = NIL
2   x = T.root
3   while x ≠ NIL
4       y = x
5       if z.key ≤ x.key
6           x = x.left
7       else
8           x = x.right
9   z.p = y
10  if y == NIL      // tree was empty
11      T.root = z
12  else if z.key ≤ y.key
13      y.left = z
14      else y.right = z
```

Delete



there are 3 cases for delete (x)

1) x is a leaf, just remove it

2) x has just one child, just remove x and connect x.p with x only child

3) x has two childrem.
   - replace x with its successor y
   - y is the min in x's rigth subtree
   - remove y which is easy as it is either case 1 or case 2

Complexity : $\Theta(h)$ time