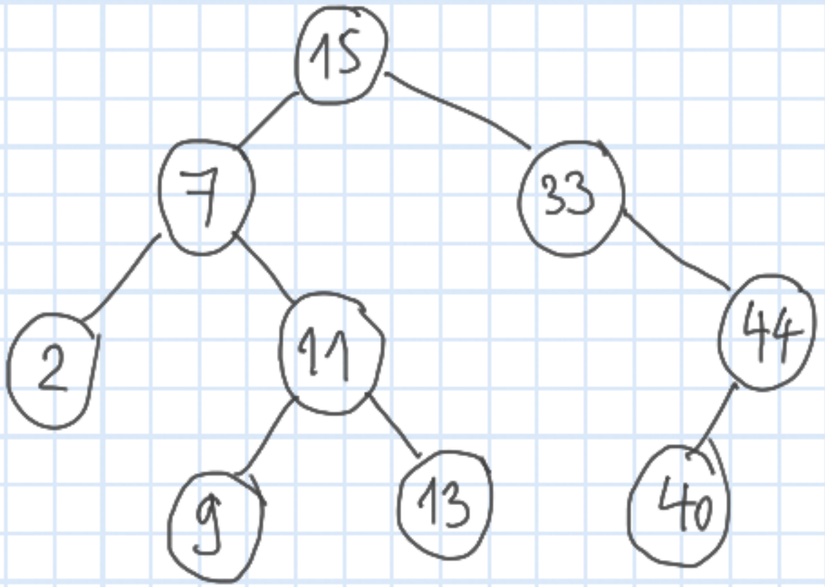
November 19, 2020

Quiz 2 → December 3, 16:00 - 17:30

(lecture time)

Binary search trees

Data structure:

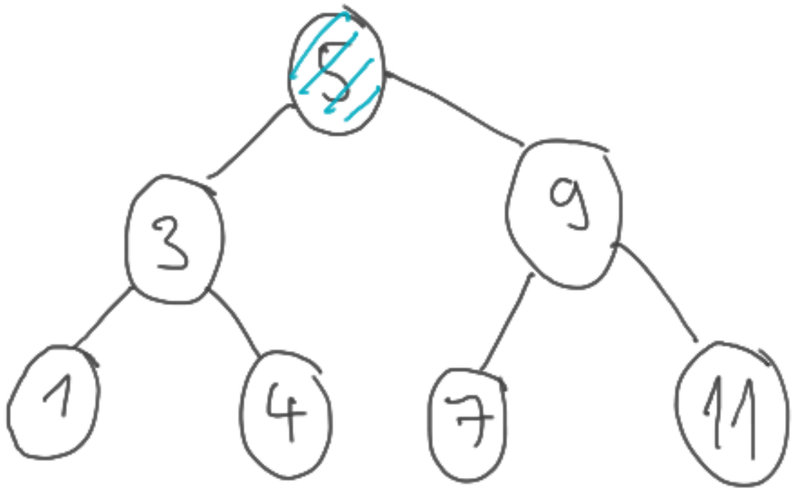


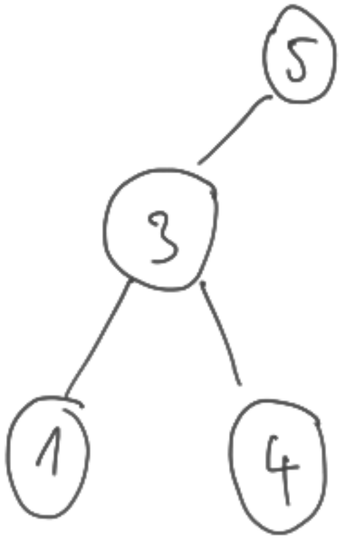
```
RecursiveSearch ( x, k )                    ← (root[T], k)
if  x = nil OR k = x→key
    Return x
if  k < x→key
    then Return RecursiveSearch (x→left, k)
    else Return RecursiveSearch (x→right, k)
```

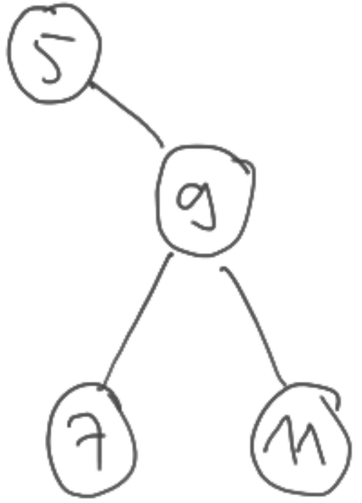
Minimum and Maximum

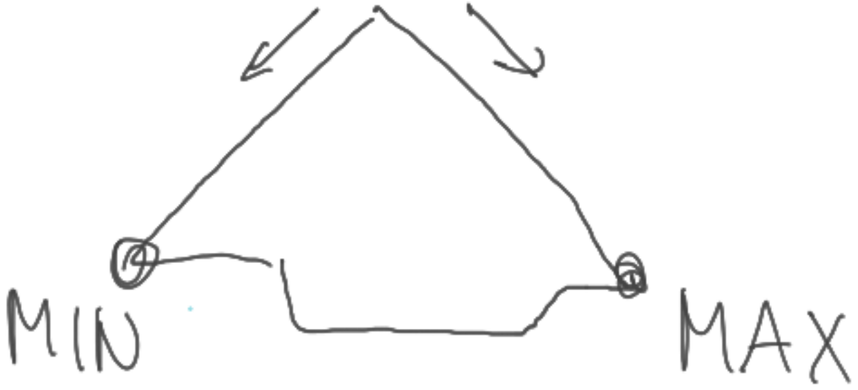
Let x be the root of a subtree of T. If x doesn't have a right subtree then it is the maximum of the subtree, and, similarly, if x doesn't have a left subtree then it is the minimum of the subtree.

Otherwise the minimum of the subtree is the minimum of the left subtree of x and the maximum of the subtree is the maximum of the right subtree of x.


5
3
9
1
4
7
11


5
3
1
4


5
9
7
11


MIN
MAX

```
Minimum (T, x)
while  x -> left ≠ nil  do
    x := x -> left
return  x
```

```
Maximum (T, x)
while  x -> right ≠ nil do
    x := x -> right
return  x
```

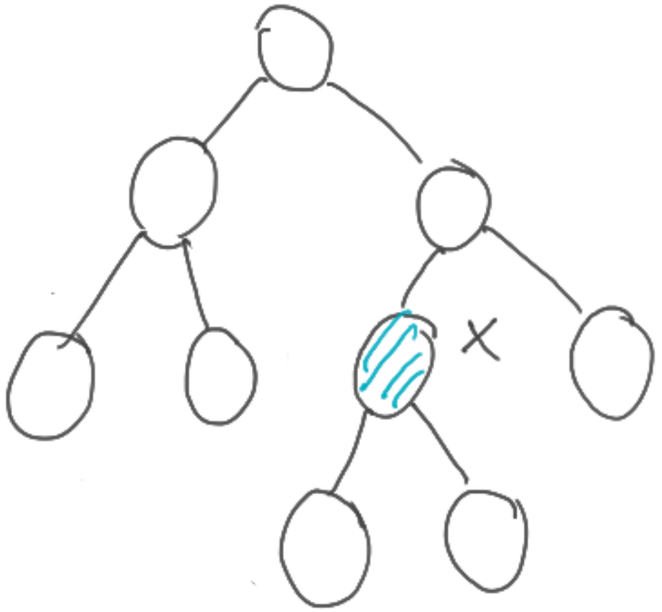
Cost : $O(h)$ again [h : depth of T]

Previous and Next (according to the inorder traversal)

```
Previous (T, x)
if x -> left ≠ nil then
    Return Maximum (T, x -> left)
y := x -> parent
while y ≠ nil and x = y -> left do
    x := y , y := y -> parent
return y
```

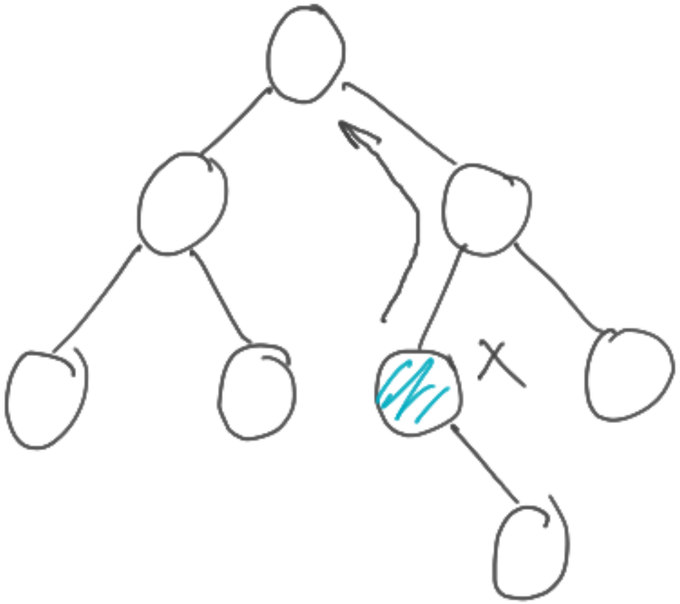
```
Next (T, x)
if x -> right ≠ nil then
    Return Minimum (T, x -> right)
y := x -> parent
while y ≠ nil and x = y -> right do
    x := y , y := y -> parent
return y
```

Previous (T, x)



$x \to left \neq nil$

$\Rightarrow$ Prev in Max (T, x->left)



$x \to left = nil$

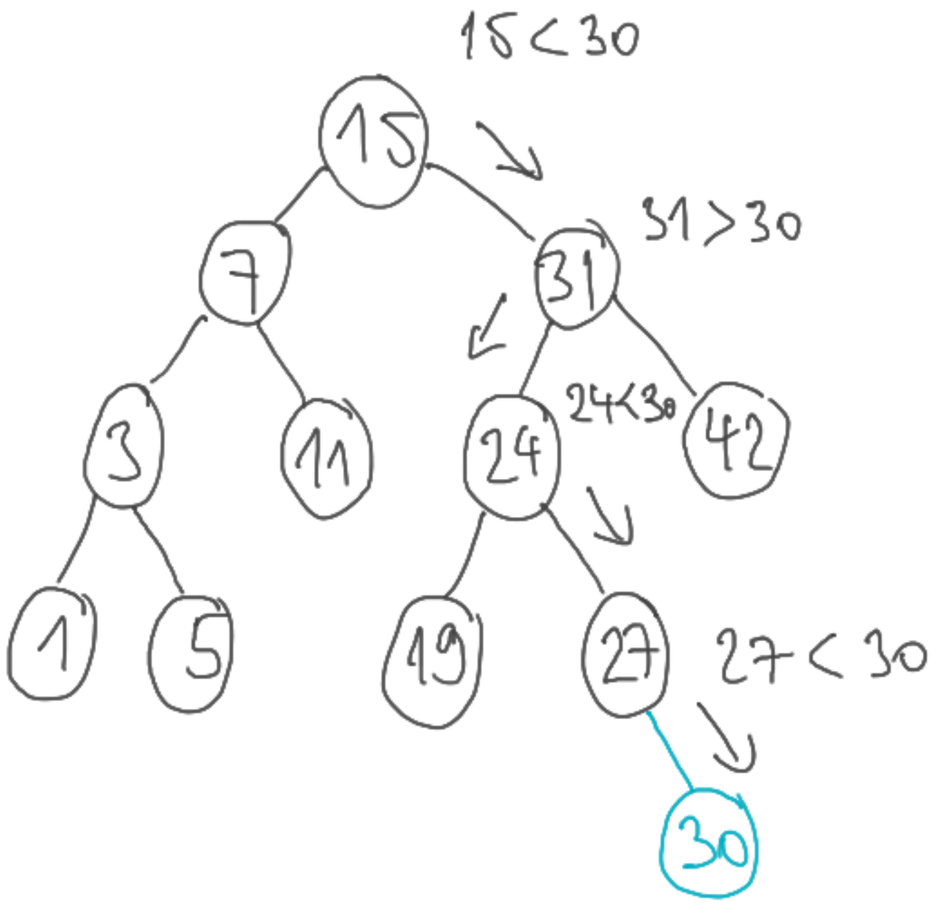
$\Rightarrow$ Prev is the first ancestor of x with the property that x is in the right subtree of this vertex

Next (T, x) is similar

Cost : O(h)

Insert (T, k)

We start with a search for k;
if the search is unsuccessful we insert the key where we leave the tree.



Insert (30)

Insert (T, z) z is a vertex object with key k

```
y := nil
x := root[T]
while x ≠ nil do
    y := x
    if k < x→key
        then x := x→left
        else x := x→right
z→parent := y
if y = nil
:
```

```
    then
        root[T] := z
    else
        if k < y → key
            then y → left := z
            else y → right := z
```

Cost: $O(h)$

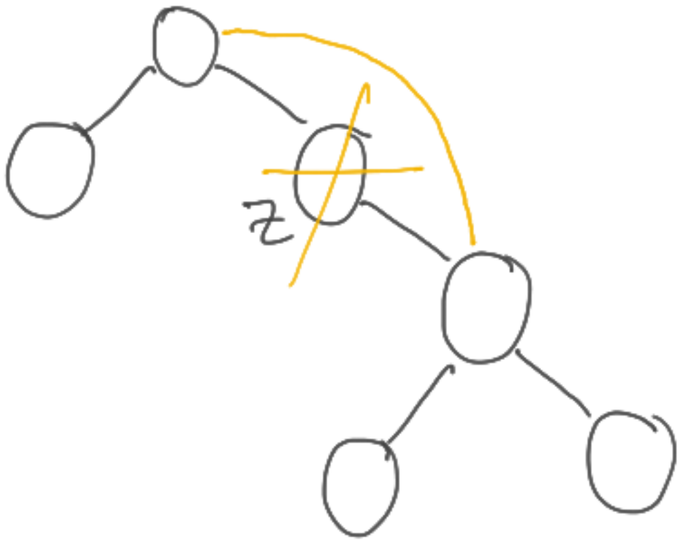
Delete (T, z)

z in the vertex object we want to delete
(esp. its key)

There are three cases:

① z is a leaf
cut z from the tree

② z has exactly one child



we can cut z off the tree again

③ z has two children

trick: find Prev(z) in the tree (this is the Max of the subtree rooted at z → left).

This vertex cannot have a left child, so what we do is this: Delete this vertex object from the tree (① or ② type) and replace the key of 2 by the key of this physically deleted vertex.