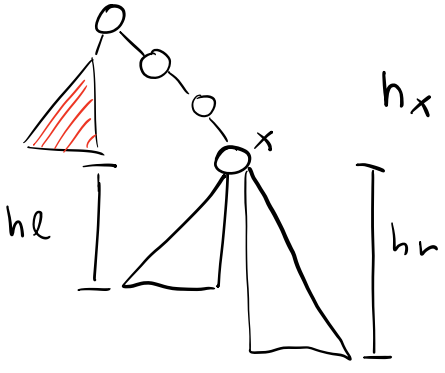
Trees

Exercise #1

Given a tree (binary) T, compute the height of T

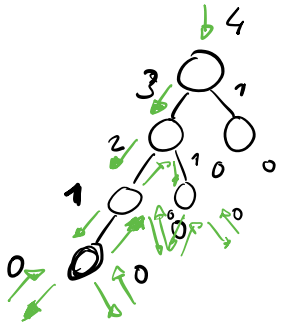
Solution



height of empty tree is 0

$h_x = 1 + \max(h_l, h_r)$

Information flow bottom-up

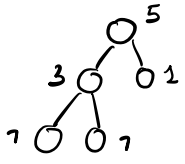



```
height (x)
    if x == NIL : return 0
    hl = height (x.left)
    hr = height (x.right)
    return 1 + max (hl, hr)
```

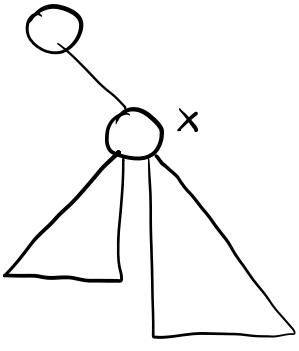
Exercise # 2

Given a tree T, compute the subtree size of every node

Example

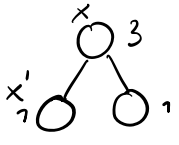


Solution



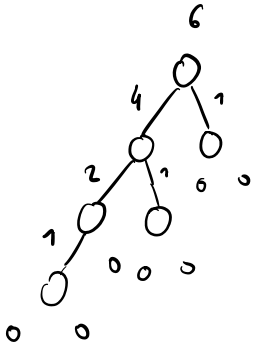
$s_x = s_l + s_r + 1$

base case $s_x = 0$



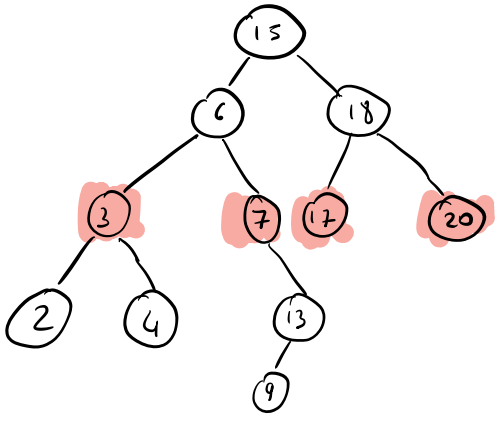
$s_{x'} = 0 + 0 + 1 = 1$

```
Subtree (x)
    if x == NIL : return 0
    sl = Subtree (x.left)
    sr = Subtree (x.right)
    return sl + sr + 1
```



Exercise # 3

Given a tree, print keys of every node at given level k



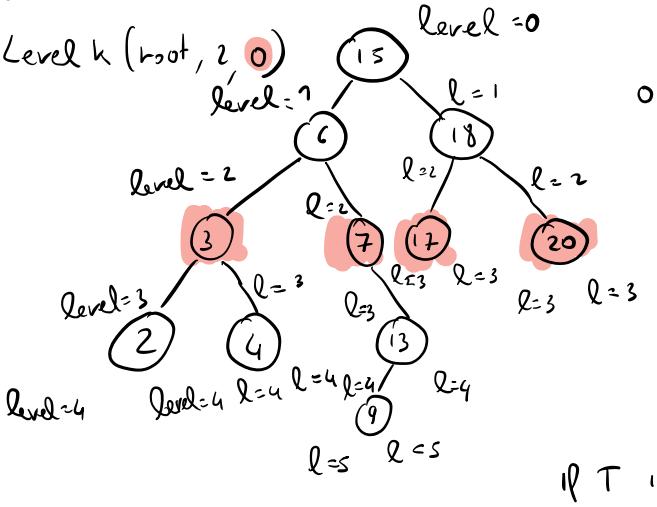
level
0
1
2
3
4

Information goes Top-down

```
PrintLevelk ( x, k, level)
    if  x == NIL : return
    PrintLevelk ( x.left, k, level + 1)
    PrintLevelk ( x.right, k, level + 1)
    if  level == k : print (x.key)
```

First call

PrintLevelk (root, 2, 0)



output 3 7 17 20

What about different visits for this Problem?

If T is a BST, print inorder

```
PrintLevelk ( x , k)                    just 2 parameters
    if x == NIL    return
    PrintLevelk(x.left , k-1)
    PrintLevelk(x.right , k-1)
    if k == 0    print (x.key)
```

Exercise #4

Given a tree T, print the key of every node x such that x.key equals the subtree size of x

Solution

```
p(x)
    if x == NIL: return 0
    sl = p(x.left)
    sr = p(x.right)
    sx = sl + sr + 1
    if sx == x.key : print(x.key)
    return sx
```

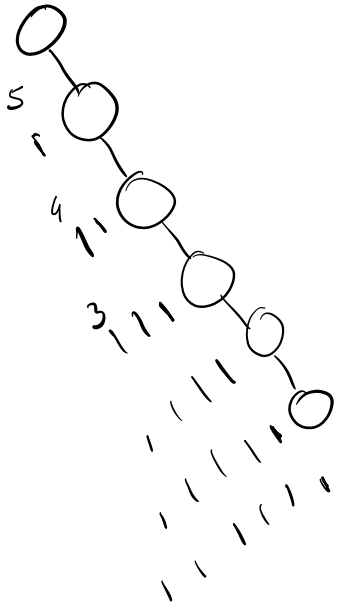
Exercise #4 bis

if T is a BST, print keys of node x (subtree(x) == X.key) sorted

Too hard without storing subtree sizes for every node

Very slow (and thus bad solution)

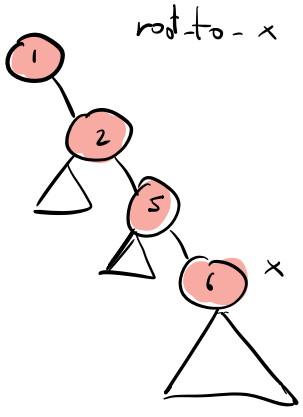
```
p(x)
    if  x == NIL:  return 0
    sl = p(x.left)
    sr = Subtree_size(x.right)
    sx = sl + 1 + sr
    if sx == x.key : print(x.key)
    sr = p(x.right)
    return sx
```



Exercise # 5

Given a tree T, print the key of every node x such that the sum of keys on the root-to-x path equals x's subtree size

Example

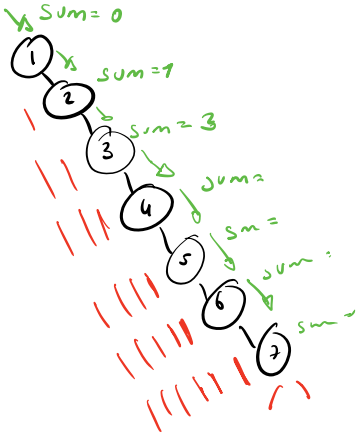


Solution

VERY BAD SOLUTION

```
PathSum(x, sum)
    if x == Nil    return
    sum = sum + x.key
    pathSum(x.left, sum)
    pathSum(x.right, sum)
    if sum == Subtreesize(x):
        print(x.key)
```

```
Subtreesize(x)
    if x == NIL  return 0
    sl = Subtreesize(x.left)
    sr = Subtreesize(x.right)
    return sl + sr + 1
```



VERY BAD: $\Theta(n^2)$ time

```
PathSum(x, sum)
    if x == NIL    return 0
    sum = sum + x.key
    sl = PathSum(x.left, sum)
    sr = PathSum(x.right, sum)
    sx = sl + sr + 1
    if sum == sx
            print(x.key)
    return sx
```

Complexity = $\Theta(n)$ time!

```
PathSum(x, sum)
    if x == NIL    return
    sum = sum + x.key
    pathSum(x.left, sum)
    pathSum(x.right, sum)
    if sum == Subtreesize(x)
            print(x.key)
```

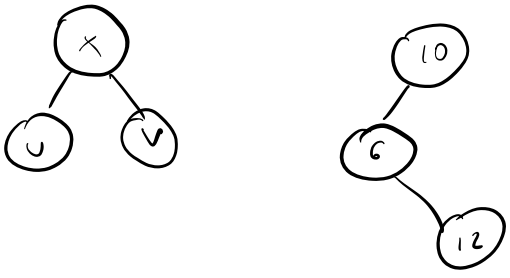
```
Subtreesize(x)
    if x == NIL  return 0
    sl = Subtreesize(x.left)
    sr = Subtreesize(x.right)
    return sl + sr + 1
```

Exercise # 6

Given a tree T, check if T is a BST

Wrong Solution

P_1 u.key $\leq$ x.key and x.key $<$ v.key



Exercise # 7 before doing 6

Check if T satisfies property P_1

```
prop(x)
    if x == Nil : return True
    bl = prop(x.left)
    br = prop(x.right)
    if x.left =/= Nil and x.left.key > x.key
                return False

    if x.right =/= Nil and x.right.key ≤ x.key
                return False

    return bl and br
```

Now Exercise # 6

VERY BAD SOLUTION Complexity $\Theta(n^2)$ time

```
Check BST (x)
    if x == NIL  return True
    b l = check BST (x.left)
    b r = check BST(x.right)
    M l = max tree (x.left)
    m r = min tree (x.right)
    if (M l > x.key) or (m r < x.key):
                return False
    return b l and b r

max tree (x)
    if x == NIL  return -∞
    M l = max tree (x.left)
    M r = max tree (x.right)
    return max (x.key, M l, M r)

minmax tree (x)
    if x == NIL  return +∞, -∞
    m l, M l = minmax tree (x.left)
    m r, M r = minmax tree (x.right)
    return min (x.key, m l, m r), max (x.key, M l, M r)
```