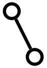
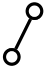
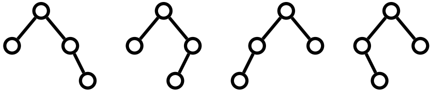
AVL-Tree: Worst Case Imbalance

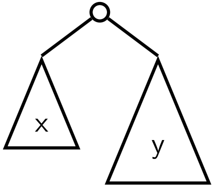
Fibonacci trees = Minimal AVL trees of a given height

How many nodes does the tree have if the balance of each (non-leaf) node is either 1 or -1?

- If the height is 1 – two options:  or  $\Longrightarrow$ size is 2
- If the height is 2: 

$\Longrightarrow$ size is *always* 4

- In general, we obtain the **Fibonacci tree of height h+2** (called T_{h+2}), from the Fibonacci trees of height h and $h+1$ (called T_h and T_{h+1}, respectively) as:



$\Longrightarrow$ the size of $T_{h+2} = 1 +$ size of $T_h +$ size of T_{h+1}

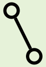
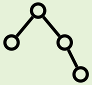
We see that there are two **minimal** AVL trees of height 1 and four **minimal** AVL trees of height 2. However, those minimal trees are all the same, except for the ordering of children. Similarly, the minimal AVL trees of larger heights are also of the same size.

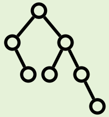
For now, we are only interested in the **size** of a minimal AVL tree of a certain height. Because all minimal AVL trees of a given height have the same size, we can pick just one representative AVL tree for every height.

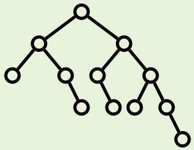
The following procedure describes a construction of **Fibonacci trees** $T_{-1}, T_0, T_1, T_2, T_3, \ldots,$ where T_h is the minimal AVL tree of height h (up to ordering of children):

- T_{-1} is the empty tree
- T_0 is the one element tree
- T_{h+2} is obtained by making T_h and T_{h+1} children of the root node (as shown in the picture on the previous slide).

For example, to construct T_3 we combine T_1 and T_2. Because

$T_1 =$  $T_2 =$ 

we obtain that T_3 is the following tree 

and T_4 is the following tree 

and so on.

Fibonacci trees and Fibonacci numbers

Denote the size of T_h as $|T_h|$:

h	$\|T_h\|$
-1	0
0	1
1	$1 + \|T_0\| + \|T_1\| = 2$
2	$1 + \|T_1\| + \|T_2\| = 4$
3	$1 + \|T_2\| + \|T_3\| = 7$
4	$1 + \|T_3\| + \|T_4\| = 12$
5	$1 + \|T_4\| + \|T_5\| = 20$
⋮	⋮

k	F_k
0	0
1	1
2	$F_0 + F_1 = 1$
3	$F_1 + F_2 = 2$
4	$F_2 + F_3 = 3$
5	$F_3 + F_4 = 5$
6	$F_4 + F_5 = 8$
⋮	⋮

$$|T_{h+2}| = 1 + |T_h| + |T_{h+1}|$$
$$1 + |T_{h+2}| = (1 + |T_h|) + (1 + |T_{h+1}|)$$

vs $\quad F_{k+2} = F_k + F_{k+1}$

initial values plus equation $\implies |T_h| + 1 = F_{h+3}$

Computing the bounds

If an AVL tree has height h then its size is

- $\leq$ the size of the perfectly balanced tree of height h, and
- $\geq$ the size of Fibonacci tree of height h (that is, $|T_h|$).

Therefore (because $|T_h| = F_{h+3} - 1$)

$$F_{h+3} - 1 \leq \text{the size of the tree} \leq 2^{h+1} - 1$$

Binet's formula: $F_k = \frac{\left(\frac{\sqrt{5}+1}{2}\right)^k - \left(\frac{\sqrt{5}-1}{2}\right)^k}{\sqrt{5}} \approx O(1.61^k)$

$\implies$ the size of an AVL tree is exponential in its height

$\implies$ the height of an AVL tree is logarithmic in its size

$\implies$ **an AVL tree of size n has height $\mathcal{O}(\log n)$**

If we have a tree of height h which is AVL, we know that the size of our tree could be as small as $|T_h|$, or as big as the size of the perfectly balanced tree of height h. However, in general it is somewhere in between.

Let n be the size of an AVL tree, then we have that

$$F_{h+3} - 1 \leq n \leq 2^{h+1} - 1$$

These are conditions on size, given that we know the height of our AVL tree. Conversely, if we know the size and we know that the tree is AVL, then what implications does this have for the height? Let's express the conditions for height in terms of n.

For example $n \leq 2^{h+1} - 1$, gives us that

$$\log_2 n \leq \log_2(2^{h+1} - 1) \leq \log_2(2^{h+1}) = h + 1.$$

In other words, height h is at least $\log_2 n - 1$.

Consequences for time complexities

For a Binary Search Tree implemented as a height-balanced tree (e.g. AVL tree), where $n =$ the number of nodes of the tree:

- `search(x)` goes through at most $O(\log n)$-many levels
  $\implies$ $O(\log n)$ steps
- `insert(x)`:
  1. We first find the leaf where to insert `x` $\implies$ $O(\log n)$ steps,
  2. then, insert it there $\implies$ $O(1)$ steps,
  3. finally, on the way up, in each node we do balancing
     $\implies$ $O(\log n)$-many times we do $O(1)$ steps
     $\implies$ $O(\log n)$ steps in total
- `delete(x)` is similar to `insert`, it also takes $O(\log n)$ steps