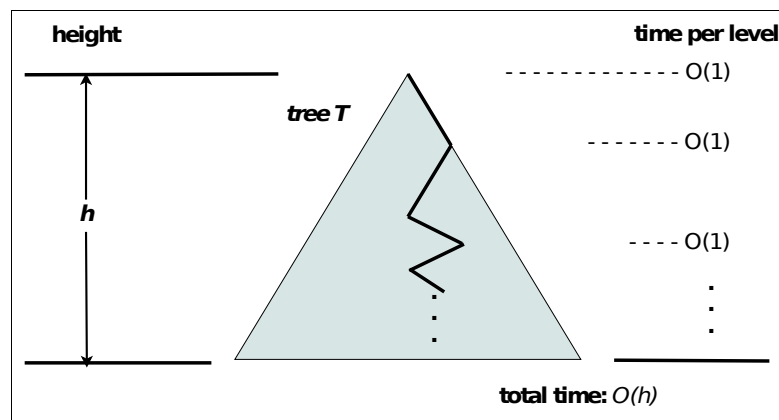


Reading: GT pp. 25-26 (the part of 1.3.3 on induction). Skim the stuff on BST & AVL from last week. Pay particular attention to p. 147, p. 151, p. 153 this time. [time 30 - 40 minutes]

Ingredients: Performance of BSTs and performance of AVL-Trees

Performance of BSTs



- Search executes a constant number of operations per node.
- It visits h nodes at most, where h is the height of the tree.
- Running time is $O(h)$.
- Remember that h itself is $\Omega(\log n)$ and $O(n)$.

Conclusion: BST search runs in time between $\Omega(\log n)$ and $O(n)$, depending on whether the tree is well balanced.

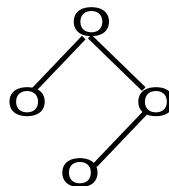
Method	BST	Sorted Array	HashTable worst case	HashTable best case
SIZE, isEmpty	$O(1)$			
FINDELEMENT, INSERTITEM, REMOVEELEMENT	$O(h)$			
FINDALLELEMENTS, REMOVEALLELEMENTS	$O(h+s)$			

Q. How much space a BST uses? How does this compare with (a) sorted array (b) hash tables. Answer the questions asymptotically and precisely (how many bytes per element).

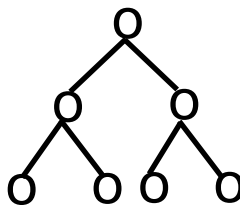
Performance of AVL Trees

Theorem 3.2 [p. 153] *The height of an AVL tree storing n items is $O(\log n)$.*

Observe that this problem is non-trivial. An AVL tree is not a complete binary tree – it still can have quite a lot of “space” inside. For example the following tree could be an AVL tree:



A complete tree of height 2 looks like that:



It has almost twice as many nodes!

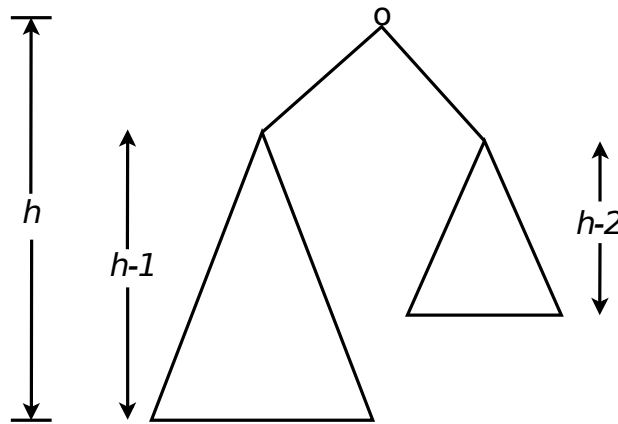
Proof. Let $n(h)$ be the minimum number of nodes in an AVL tree of height h .

Notice that $n(1)=1$ and $n(2)=2$.

- A tree of height 1 must have one node, a tree of height 2 can have two or three nodes, so the minimum is 2.

Consider $h \geq 3$:

- An AVL tree with height $h \geq 3$ must have two subtrees.
- One needs to have height at least $h-1$, the other at least $h-2$ (Otherwise the balance would exceed 1). For example:



$$\text{So } n(h) = 1 + n(h-1) + n(h-2)$$

Observe that $n(h)$ is a strictly increasing function: 1 2 4 7 ...

$$\text{So } n(h-2) \leq n(h-1)$$

$$\text{and } n(h) \geq 1 + n(h-1) + n(h-2) \geq 1 + 2n(h-2) \geq 2n(h-2) .$$

This means that $n(h)$ doubles each time h increases by 2. So $n(h)$ grows exponentially.

When height grows from 1 to h we will increment height by 2 at least $\lfloor \frac{h}{2} \rfloor$

times. So we will double $\lfloor \frac{h}{2} \rfloor$ times:

$$n(h) \geq 2^{\lfloor \frac{h}{2} \rfloor} n(1) = 2^{\lfloor \frac{h}{2} \rfloor}$$

Take a logarithm of both sides (log is monotonic):

$$\log n(h) \geq \log 2^{\lfloor \frac{h}{2} \rfloor}$$

$$\log n(h) \geq \lfloor \frac{h}{2} \rfloor \geq \frac{h}{2} - 1$$

$$h \leq 2 \log n(h) + 2$$

So an AVL tree storing n keys has height not larger than $(2 \log n + 2)$.

Conclusion: $h = O(\log n)$.

Concluding Data Structures

You already know a lot data structures implemented in the Java library by Sun. C5 is a library of algorithms and data structures for the .NET platform, developed in C# by Peter Sestoft. See <http://www.itu.dk/research/c5/>

You may find the following interview with Peter Sestoft interesting:

<http://channel9.msdn.com/ShowPost.aspx?PostID=370368>

Both video and audio (for your iPod) are available. Makes sense to listen even if you are not (yet) a .NET programmer.