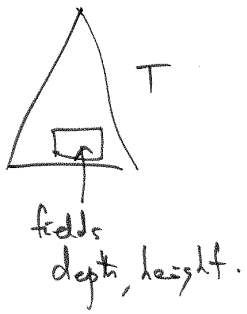
Finding depth, height of all nodes in a tree




```
depth.height (Tree t)
    dh(t.root , 0)

int dh (Entry<?> node, int d)
  // @ return - height of node in tree
  if node = null then
      return -1
  else
      node.depth ← d          maxh ← -1
      maxh ← -1
      for each child c of node
          h ← dh(c, d+1)
          maxh ← max(maxh, h)
      node.height ← maxh + 1
      return node.height
```

RT = O(n).

```
checkAVL ( AVLTree<T> t )
    checkAVL (t.root, -∞, ∞)

(Boolean, int) checkAVL ( AVLEntry<T> node, T minBound, T maxBound )
  // minBound < Any value in tree rooted at node < maxBound
  // @return: <boolean - is it a valid AVL tree?
  //           int - height of that tree>
  if node = null then
      return <true, -1>
  else if node.element = null then return <false, -1>
      <lb, lh> ← checkAVL(node.left, minBound, node.element)
      <rb, rh> ← checkAVL(node.right, node.element, maxBound)
      h ← max(lh, rh) + 1
      if |lh - rh| > 1 or node.height ≠ h or
         !lb or !rb or minbound ≥ node.element or node.element ≥ maxBound
      then return <false, h> else return <true, h>
```

RT = O(n)

Optimal BST's (simpler version of problem from Cormen et al's Intro to algorithms)

$a_1 < a_2 < \cdots < a_n$ — n keys

$w_1 \quad w_2 \quad \cdots \quad w_n$ — weights

a_i has weight w_i $\Rightarrow$ #of calls to find(a_i) is proportional to w_i.

Q: What is the best BST that minimizes the total cost of all the searches?

Define C_{ij} = cost of best BST for $a_i\, a_{i+1} \cdots a_j$

$$W_{ij} = \sum_{k=i}^{j} w_k = w_i + w_{i+1} + \cdots + w_j$$

Recurrence for C_{ij} =

$$\begin{cases} 0 & \text{if } j < i \\ w_i & \text{if } i = j \\ \min\limits_{i \le r \le j} \{ C_{i,r-1} + C_{r+1,j} + W_{ij} \} \end{cases}$$

Implement using a dynamic program = $O(n^3)$

Q: If $w_1 \ldots w_n$ is derived from some unknown distribution, how can we design a data structure to minimize total cost of search?

A: Splay trees!

splay tree = self-adjusting BST

= BST + no balance condition, no color, ..

Main operation: splay(x) — Bottom up, rotate tree to bring node with x to the root

(Zig-Zig, Zig-Zag, Zig rotations)

After splay(x) — x is at root of tree.

Find(x) — splay(x), check if root is x

(search for x, node where you end the search is splayed to the root)

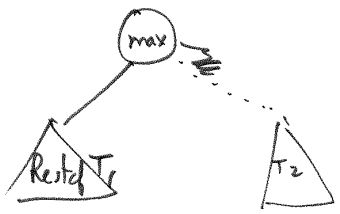
Delete(x) — splay(x)

Join(x.left, x.right)

Join (T_1, T_2) // all keys of T_1 < all keys of T_2

splay $(T_1.\max)$

Attach T_2 as right child of max.



Symmetric:

splay $(T_2.\min)$

Attach T_1 as left child of min.

Theoretical facts about splay trees

1. start from an empty tree, perform m operations on tree

   Amortized cost per operation = $O(\log n)$ — n = max size of dictionary at any time.

   Total cost of m operations = $O(m \log n)$ ← worst case.

2. Unknown distribution $w_1, \ldots w_n$ — best bst T for this distribution.

   RT of splay tree = Θ (RT of T)