1. *Tries*: tree in which every node has 26 NULL and non-NULL children.
   1. Code
      1. struct trie Node {
         1. int count; //Value of struct’s “count field”
         2. struct trieNode\* children[26];
      2. };
   2. *n*th node named after the *n*th letter
   3. Functions (*k* = string length)
      1. Insertion [O(*k*) 🡪 O(*k*)]
      2. Look-up [O(1) 🡪 O(*k*)]
      3. Deletion [O(1) 🡪 O(*k*)]
         1. Deletion of word frequency
            1. Case 1

Decrement count at terminal node to 0

Delete all nodes without children, as well as the first one with children

* + - * 1. Case 2

Decrement count at terminal node

* + - * 1. Case 3

Decrement count field at terminal node, but no nodes get teleted.

* + 1. Accessing
       1. Example: apple
          1. “a” = 0, “p” = 15, “l” = 11, and “e” = 4
          2. root->children[0]->children[15]->children->[11]->children[4]->count
          3. Without knowing “p” index

root->children[‘a’-‘a’]->children[‘p’-‘a’]->children[

* 1. Applications
     1. Spell-check
     2. Document word count/frequency
     3. Find all words beginning with prefix
     4. Word prediction/recognition

1. *Adelson-Velskii-Landis (AVL) Trees*: balanced binary trees; the height of the left subtree can differ by at most 1 from the height of the right subtree.
   1. Minimum number of nodes in AVL tree with height H: SH = SH-1 + SH-2 + 1
      1. S0=1 and S1=2
      2. Use induction
         1. True for an integer *k*
         2. True for integer *k* + 1
         3. True for 1
         4. Therefore true for all positive numbers
      3. Induction
         1. Base Cases
            1. H=0: LHS = 1, RHS = F3 - 1 = 2 - 1 = 1
            2. H=1: LHS = 2, RHS = F4 - 1 = 3 - 1 = 2
         2. Hypothesis: For an arbitrary integer k <= H, assume that Sk = Fk+3 -1.
         3. Inductive step: prove for H=k+1 that Sk+1 = Fk+1+3 -1

Sk+1 = Sk + Sk-1 + 1 (because to form an AVL tree with the min.

number of nodes of height k+1, one side of

the root must have height k and the other

k-1. This is because we need the sides to

be within one, but we want to minimize the

number of nodes. The only other option

would have been k and k, which would

NOT minimize the desired value.)

= (Fk+3 -1) + (Fk+2 -1) +1, using the I.H. twice

= (Fk+3 + Fk+2) - 1

= Fk+4 -1, using the defn. of Fibonacci numbers,

to complete proof.

It can be shown through recurrence relations, that

Fn ≈ 1/√5 [(1 + √5)/2]n

So now, we have the following:

Sn ≈ 1/√5 [(1 + √5)/2]n+3

This says that when the height of an AVL tree is n, the minimum number of nodes it contains is 1/√5 [(1 + √5)/2]n+3.

So, in order to find the height of a tree with n nodes, we must replace Sn with n and replace n with h? Why is this the case?

n ≈ 1/√5 [(1 + √5)/2]h+3

n ≈ (1.618)h

h ≈ log 1.618 n

h = O(log 2 n)

1. How to maintain AVL property when adding/deleting a node
   1. Rework the binary search
2. Symmetric cases
   1. insertion into the left subtree of the left child of the root.
   2. insertion into the right subtree of the left child of the root.
   3. insertion into the left subtree of the right child of the root.
   4. insertion into the right subtree of the right child of the root.
3. Insertion into AVL tree
   1. Do normal binary tree insert.
   2. Restore tree based on this leaf node.
      1. Calculate the heights of the left and right subtrees, use this to set the potentially new height of the node.
      2. If they are within one of each other, recursively restore the parent node.
      3. If not, then perform the appropriate restructuring described above on that particular node, THEN recursively call the method on the appropriate parent node.
   3. No recursive call is made if the node in question is the root node and has no parents.
   4. Also, one rebalancing will always do the trick, though we must make the recursive calls to move up the tree so that the heights stored at each node are properly recalculated.