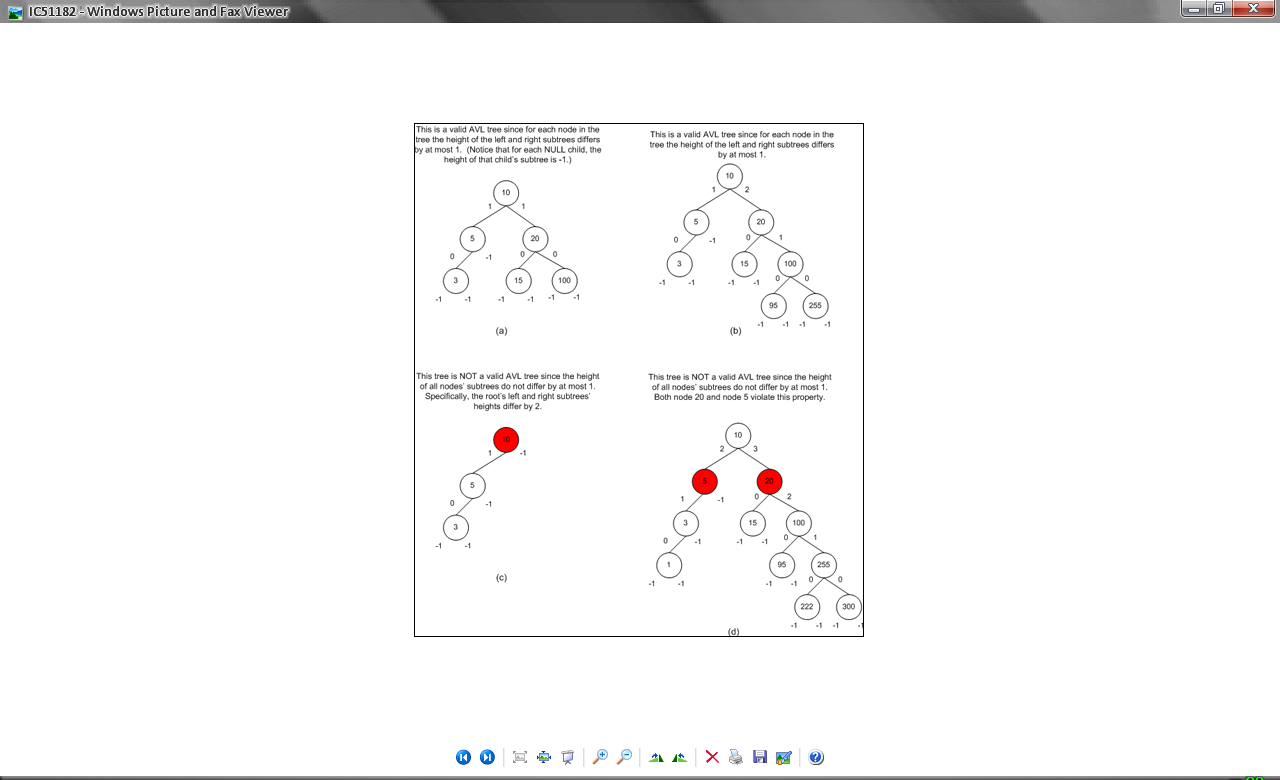
**AVL-Tree**

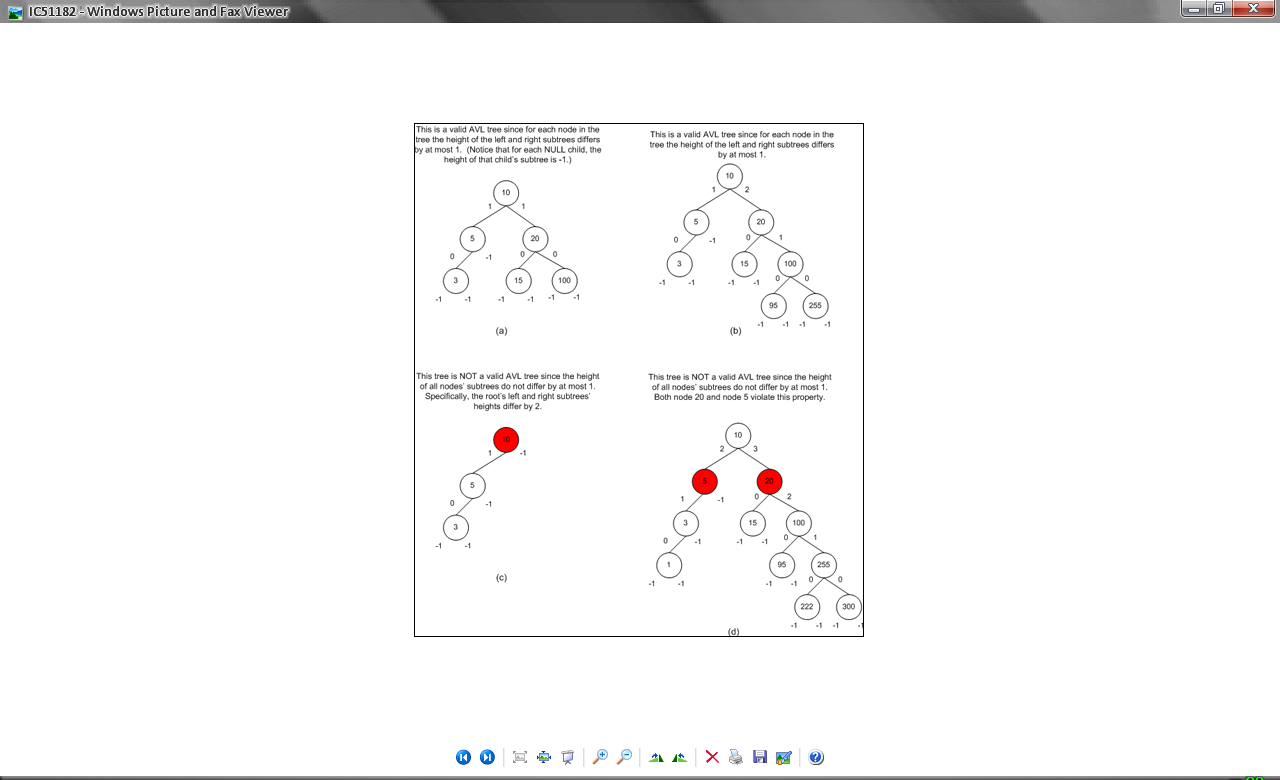
Kebanyakan aplikasi saat ini melakukan operasi penambahan dan penghapusan elemen secara terus-menerus tanpa urutan yang jelas urutannya. Oleh karena itu sangatlah penting untuk mengoptimasi waktu pencarian dengan menjaga agar pohon tersebut mendekati seimbang sepanjang waktu. Dan hal ini telah diwujudkan oleh 2 orang matematikawan Russia , G.M. Adel’son-Vel’skii dan E.M. Landis. Oleh karena itu *Binary Search Tree* ini disebut AVLtree yang diambil dari nama kedua matematikawan Russia tersebut. Tujuan utama dari pembuatan AVL-Tree ini adalah agar operasi pencarian, penambahan, dan penghapusan elemen dapat dilakukan dalam waktu O(log n) bahkan untuk kasus terburuk pun. Tidak seperti *Binary Search Tree* biasa yang dapat mencapai waktu O(1.44 log n) untuk kasus terburuk.

1. **Definisi AVL-Tree**

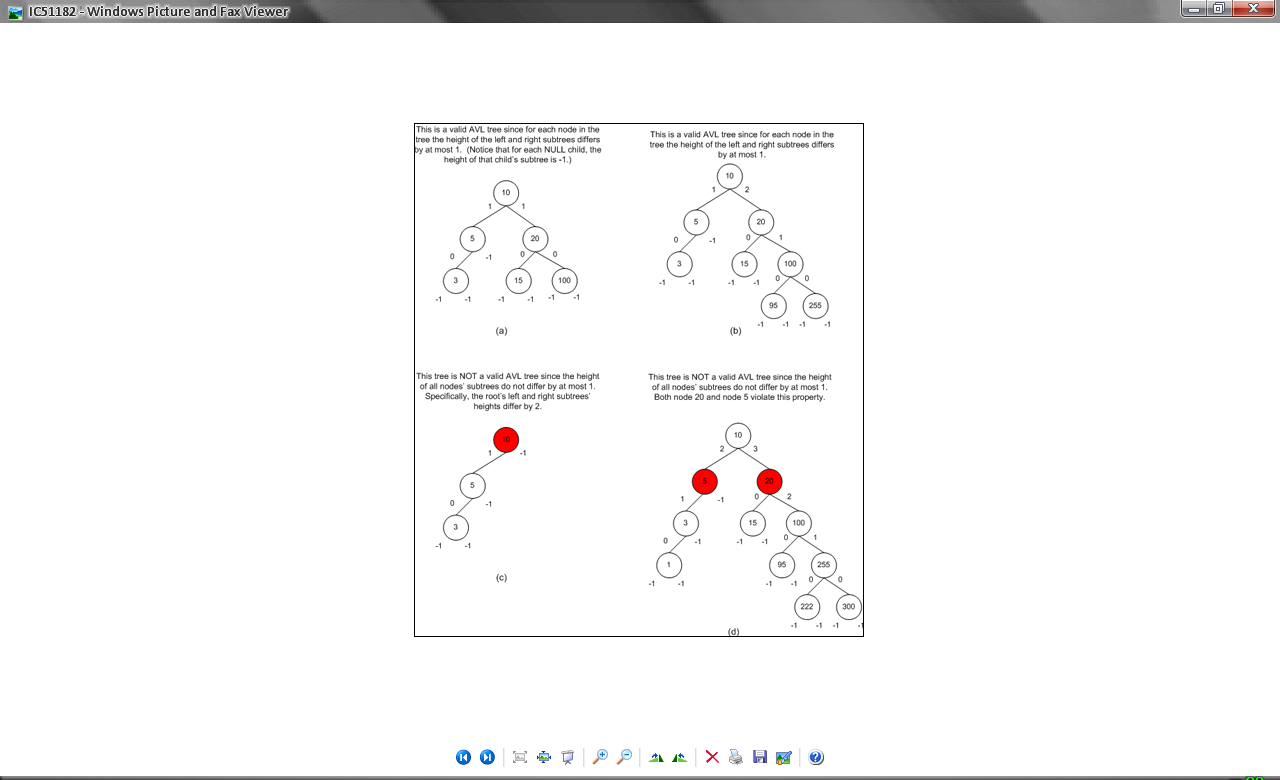
Dalam pohon yang benar-benar seimbang, upapohon kiri dan kanan dari setiap simpul mempunyai tinggi yang sama. Walaupun kita tidak dapat mencapai tujuan ini secara sempurna, setidaknya dengan membangun *Binary Search* *Tree* dengan metode penambahan elemen yang nantinya akan kita bahas, kita dapat meyakinkan bahwa setiap upapohon kiri dan kanan tidak akan pernah berselisih lebih dari 1. Jadi, sebuah *AVL-Tree* merupakan *Binary Search* *Tree* yang upapohon kiri dan kanan dari akarnya tidak akan berselisih lebih dari 1 dan setiap upapohon dari *AVL-Tree* juga merupakan *AVLTree*. Dan setiap simpul di *AVL-Tree* mempunyai faktor penyeimbang (*balance factor*) yang bernilai *left-higher* (upapohon kiri > kanan), *equal-height* (upapohon kiri = kanan)*, righthigher* (upapohon kiri < kanan)*.*

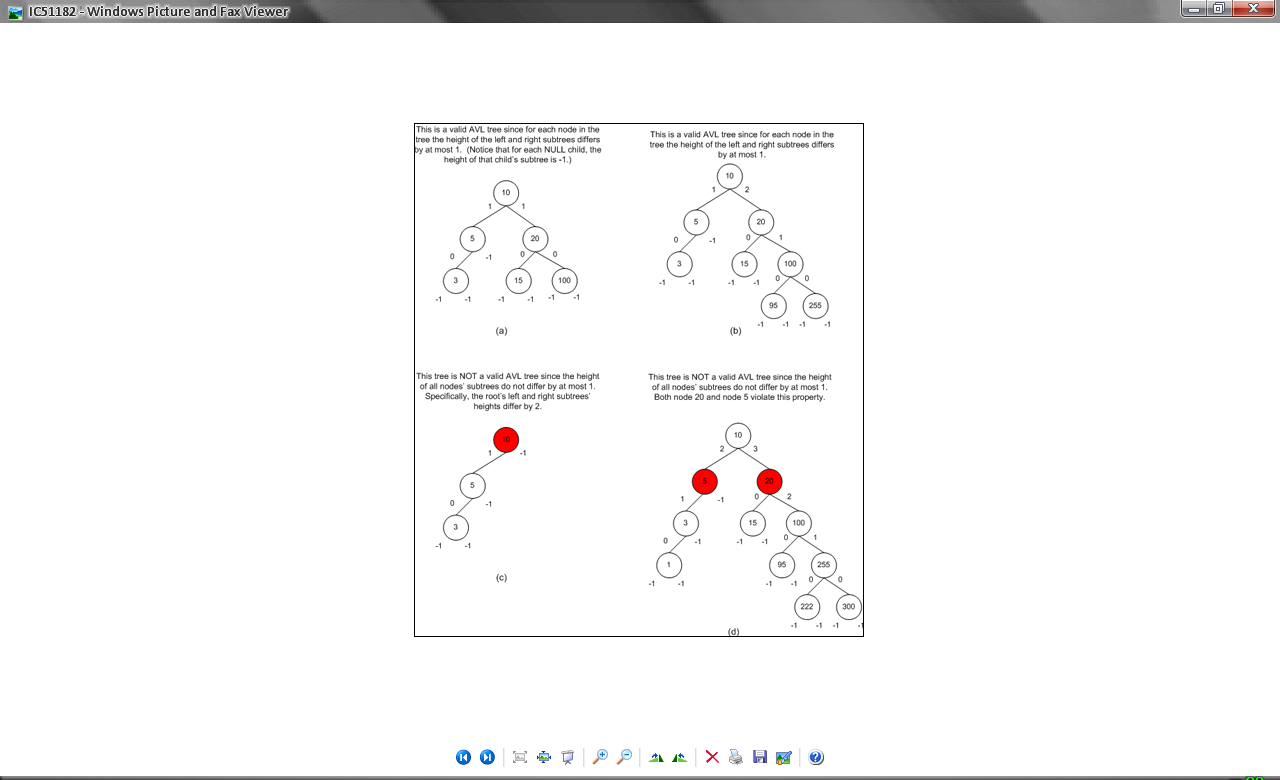
Berikut ini contoh gambar *AVL-Tree.*



**

Berikut ini contoh gambar pohon sembarang.

****

****

1. **Operasi Penambahan Elemen**

Karena *AVL-Tree* ini juga merupakan *binary search tree* maka penambahan elemen pada *AVL-Tree* ini juga dapat kita lakukan denganmetode yang sama dengan *binary search tree.* Pada kebanyakan kasus, penambahan elemen itusering sekali tidak membuat tinggi dariupapohonnya meningkat sedemikian sehinggadapat membuat selisih tinggi dari upapohon kiridan kanan lebih dari 1. Namun, bukan tidakmungkin hal ini bisa terjadi. Satu-satunya kasusdimana masalah ini muncul adalah ketika simpulbaru itu ditambahkan ke upapohon yang lebihtinggi dari upapohon lainnya sehingga tingginyaakan bertambah dan membuat selisih tinggi nyamenjadi lebih dari 1.Berikut gambar contoh penambahan simpulsederhana, kita menggunakan ‘/’ sebagai tandabahwa faktor penyeimbang bernilai *left-higher*,‘-‘ untuk *equal-height* dan ‘\’ untuk *right-higher*.

****

**Gambar, Penambahan Elemen Pada AVL-Tree yang Sederhana**

Sekarang mari kita tinjau masalah dimana penambahan elemen tersebut akan membuat *AVL-Tree* tidak lagi memenuhi syarat sebuah *AVL-Tree*. Solusi yang ditawarkan untuk menyelesaikan masalah ini adalah dengan merotasi upapohon tersebut sedemikian sehingga selisih tinggi setiap upapohon nya tidak lebih dari 1. Lebih jelasnya lagi, mari kita asumsikan kita telah menambahkan simpul baru sehingga selisih tinggi setiap upapohonnya lebih dari 1. Ada 3 kasus yang muncul pada keadaan ini yakni:

1. Upapohon kanan lebih tinggi

Yang harus kita lakukan adalah merotasi kiri / *left rotation* yaitu merotasi *right\_tree* tersebut ke *root* dan menjadikan *root* tersebut menjadi upapohon kiri dari *right\_tree*. Lalu T2 dijadikan upapohon kanan dari *root*.



**Gambar, Penyeimbangan AVL-Tree dengan Merotasi kiri (*Left Rotation*)**

1. Upapohon kiri lebih tinggi

Pada kasus kedua ini permasalahannya sedikit lebih rumit dimana kita harus memindahkannya sejauh 2 tingkat ke simpul sub\_tree. Proses ini disebut rotasi ganda / *double rotation* karena transformasi dilakukan dalam 2 tahap. Pertama rotasi *sub\_tree* ke *right\_tree* sedemikian sehingga *right\_tree* menjadi upapohon kanan dari *sub\_tree* dan upapohon kanan *sub\_tree* sebelumnya menjadi upapohon kiri *right\_tree*. Lalu rotasi *sub\_tree* ke *root* sedemikian sehingga *root* menjadi upapohon kiri dari *sub\_tree* dan upapohon kiri *sub\_tree* yang sebelumnya menjadi upapohon kanan dari *root*.

****

**Gambar, Penyeimbangan AVL-Tree dengan Merotasi ganda (*double Rotation*)**

1. Upohon dengan tinggi yang sama

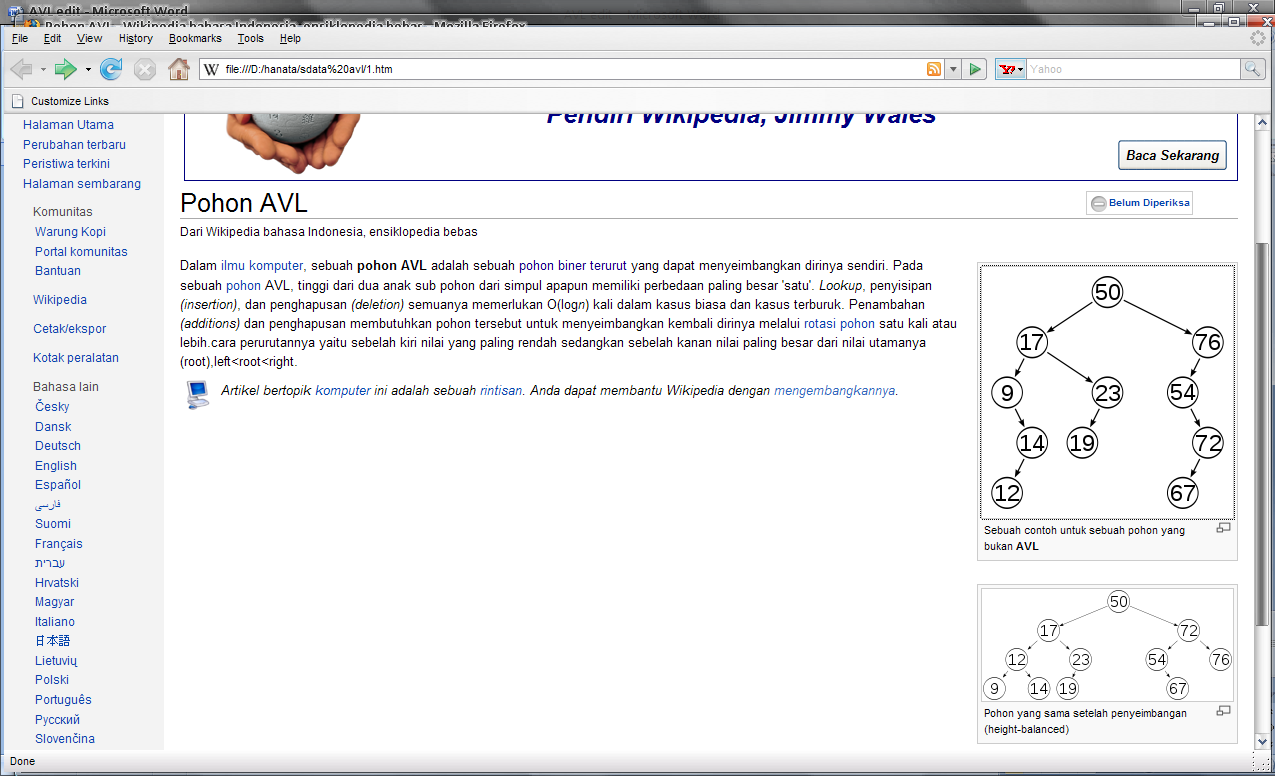
Kasus terakhir adalah upapohon kiri dan kanan memiliki tinggi yang sama, namun pada kenyataannya kasus ini tidak akan pernah terjadi. Berikut contoh *penambahan* elemen *AVL-Tree* yang memerlukan rotasi :



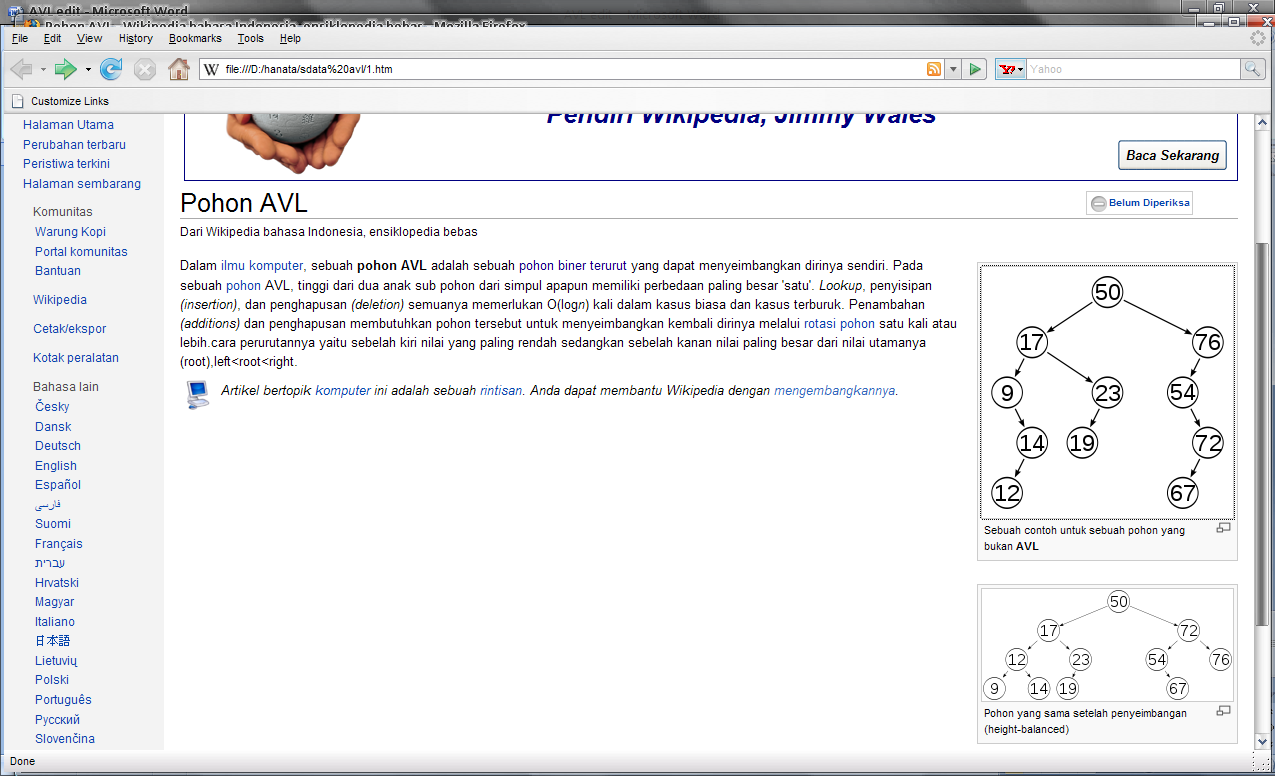
**Gambar, Penambahan AVL-Tree dengan Rotasi**

Berikut ini contoh pohon sembarang. Gambar (a) dibawah ini mengindikasikan tidak seimbang. Mengapa bisa dikatakan tidak seimbang? Faktor keseimbangan dari sebuah simpul adalah tinggi dari subtree kanan minus tinggi subtree kiri dan keseimbangan sebuah node dengan faktor 1, 0, atau -1 dianggap seimbang. Sebuah simpul dengan faktor keseimbangan lain dianggap tidak seimbang dan membutuhkan rebalancing pohon. Faktor keseimbangan baik langsung disimpan di setiap node atau dihitung dari ketinggian dari sub pohon.

Kedua sub pohon kiri dan kanan memiliki ketinggian 4. Subtree kanan kiri pohon memiliki ketinggian 3 yang masih hanya 1 kurang dari 4.

****

Gambar (a) Pohon sembarang(bukan AVL)



Gambar (b) Pohon sembarang setelah penyeimbangan (height balance)