Self Balancing Binary Search Tree

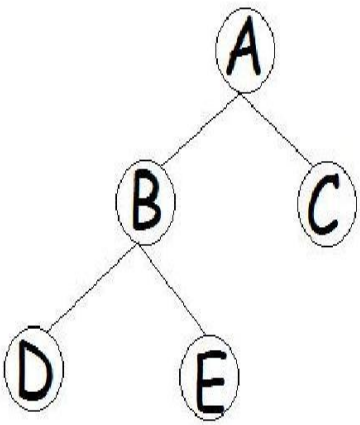
Kevin Wibowo-13509065
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13509065@std.stei.itb.ac.id

Abstrak - ***Makalah ini akan memuat tentang penjelasan dari Self Balancing Binary Search Tree. Self Balancing Binary Search Tree sendiri merupakan salah satu variansi dari pohon pencarian biner yang memiliki beberapa kriteria tersendiri. Di dalam makalah ini, juga akan diberikan beberapa contoh dari Self Balancing Binary Search Tree dan penjelasannnya.***

***Kata Kunci*—Akar, Pohon, Simpul, Struktur Data.**

I. PENDAHULUAN

Pohon adalah salah satu jenis graf yang tidak memiliki lintasan atau sirkuit. Pohon biasanya memiliki hanya satu simpul *orangtua* dan simpul *anak* yang bisa nol atau lebih dari nol. Pada pohon juga terdapat yang disebut pohon berakar yaitu pohon yang satu buah simpulnya dijadikan sebuah akar lalu anak dari akar tersebut akan memiliki simpul *anak* dan simpul anak tersebut akan terus memiliki *anak* sehingga akan ada simpul yang tidak memiliki *anak* yang disebut simpul daun.

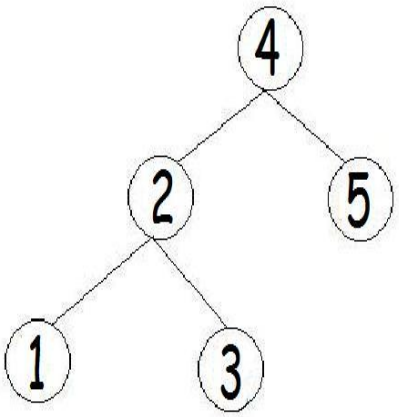


Gambar 1-1. Gambar Pohon Berakar

Dalam gambar pohon berakar di atas(Gambar 1-1). Dapat dilihat beberapa terminologi:
A adalah *akar* dari pohon tersebut
B dan C adalah *anak* dari A
B adalah *orangtua* dari D dan E
C,D dan E adalah *daun* dari pohon
A,B,C,D,E adalah *simpul* dari pohon
D dan E adalah *cucu* dari A
Tinggi pohon tersebut adalah 2(akar tidak dihitung)
Dan berlaku hubungan lainya seperti pohon keluarga

Dalam pohon terdapat berbagai jenis pohon, antara lain adalah pohon biner, dan pohon pencarian biner.

Pohon Pencarian Biner adalah pohon yang memiliki beberapa kriteria khusus, yaitu semua simpul anak di kiri nilainya selalu lebih kecil daripada simpul orangtua. Dan semua simpul anak di kanan selalu lebih besar daripada orangtua.



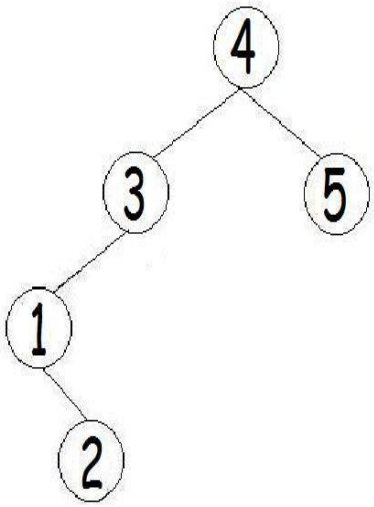
Gambar 1-2. Pohon pencarian Biner

Pohon Pencarian Biner biasa digunakan sebagai struktur data dalam ilmu komputer dan teknologi informasi.

Dalam makalah ini akan dibahas lebih lanjut mengenai salah satu jenis pohon pencarian biner yaitu pohon pencarian biner yang dapat menyeimbangkan dirinya sendiri(*Self Balancing Binary Search Tree*).

II. SELF BALANCING BINARY SEARCH TREE

Self Balancing Binary Search Tree adalah pohon pencarian biner yang dapat menyeimbangkan dirinya sendiri, yaitu meminimalisasikan kedalaman pohon, pada saat pohon tersebut di tambahkan atau dikurangi simpulnya.



Gambar 2-1. Pohon Pencarian Biner Tak Seimbang

Pada umumnya dalam *Self Balancing Binary Search Tree*, bentuk pohon yang seharusnya seperti gambar 2-1, akan secara otomatis diubah menjadi seperti gambar 1-2. Algoritma penyeimbangan pohon akan berjalan saat dimasukan simpul baru atau saat sebuah simpul dalam pohon dihapus.

Pada umumnya *Self Balancing Binary Search Tree* mempunyai tinggi maksimal sebesar $\lfloor {}^{2}\log n \rfloor$ dengan n adalah jumlah simpul, hal ini disebabkan oleh pohon pasti mempunyai ketinggian minimal dan baru akan menambah tinggi saat pohon sudah mempunyai daun maksimal di suatu ketinggian, hal ini juga menyebabkan sebagian besar algoritma untuk mengerjakan pohon ini mempunyai kompleksitas sebesar O(log n).

Self Balancing Binary Search Tree sangat berguna untuk mempercepat kinerja sebuah pohon pencarian biner, selain itu pohon ini juga mempunyai beberapa aplikasi untuk mempermudah kinerja suatu algoritma seperti antrian dengan prioritas dan tabel data.

Self Balancing Binary Search Tree mempunyai beberapa contoh pohon yaitu:

1. *AA tree*
2. *AVL tree*
3. *Red-black tree*
4. *Scapegoat tree*
5. *Treap*

III. JENIS-JENIS SELF BALANCING BINARY SEARCH TREE

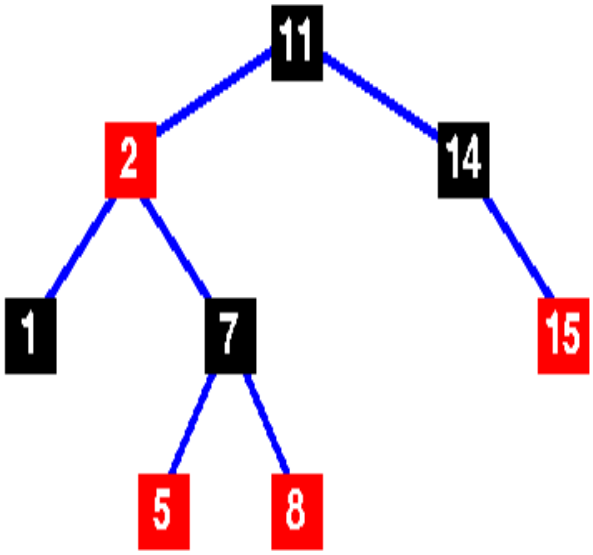
A. Red-black Tree

Red-black Tree adalah salah satu jenis *Self Balancing Binary Search Tree* yang diciptakan oleh Rudolf Bayer di tahun 1972 dan bernama *Symmetric binary B-tree*, lalu pada tahun 1978 diperbaharui menjadi *Red-black* Tree oleh Leonidas J. Guibas dan Robert Sedgewick.

Dalam *Red-black Tree* ini pohon, simpul ditambahkan dan dihapus dengan algoritma khusus sehingga pohon tidak perlu menyeimbangkan dirinya.

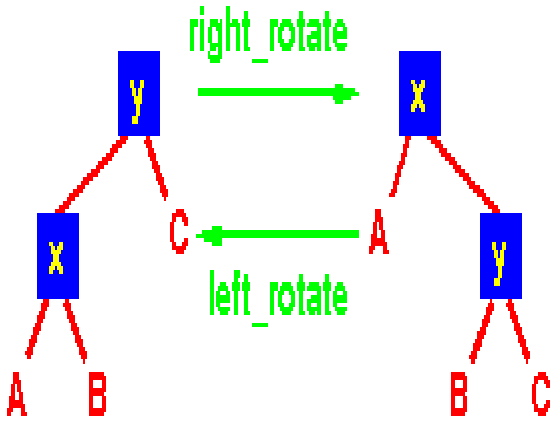
Selain kriteria pohon pencarian biner biasa, ada beberapa kriteria tambahan berlaku untuk *Red-Black Tree*:

1. Simpul bewarna merah atau hitam.
2. Akar berwarna hitam. (Aturan ini dapat dihilangkan karena akar dapat diubah ke merah dan tidak mempengaruhi.)
3. Anak dari tiap simpul merah adalah hita
4. Setiap jalur dari akar sampai simpul sampai daun keturunannya mempunyai jumlah simpul hitam yang sama.



Gambar 3-1. Contoh *Red-black Tree*
Sumber: http://www.cs.auckland.ac.nz/~jmor159/PLDS210/red_black.html

Pada pohon ini algoritma menambahkan dan mengurangkan bergantung pada kasus dan beberapa diantaranya menggunakan rotasi pohon seperti pada gambar 3-2.



Gambar 3-2. Rotasi sebuah pohon biner
Sumber: http://www.cs.auckland.ac.nz/~jmor159/PLDS210/red_black.html

Pohon ini menambahkan dan mengurangkan dengan cara melihat dimana simpul baru akan ditambahkan dan tergantung dimana simpul tersebut ditambahkan akan ada algoritma tersendiri.

Contoh beberapa kasus :

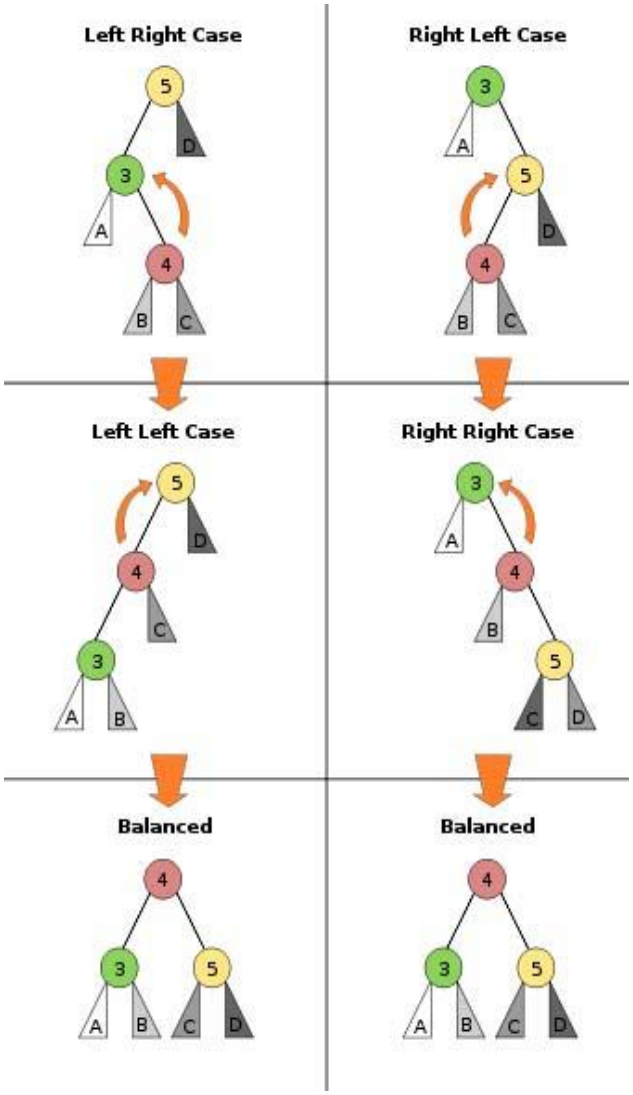
1. Simpul baru menjadi akar.
2. Simpul orangtua bewarna hitam.
3. Simpul orangtua bewarna merah.

B. AVL Tree

AVL Tree adalah salah satu jenis *Self Balancing Binary Search Tree* yang ditemukan oleh G.M. Adelson-Velskii and E.M. Landis pada tahun 1962. Di pohon ini terdapat *balance factor* yaitu nilai ketinggian anak pohon dikiri kurang ketinggian anak pohon kanan. Dan pohon dianggap seimbang bila *balance factor* bernilai -1,0, atau 1.

Dalam *AVL Tree* penambahan dan pengurangan dilakukan sama seperti pohon pencarian biner biasa, hanya saja setelah penambahan atau pengurangan tersebut akan ditambahkan prosedur rotasi pohon untuk menyeimbangkan.

Dalam penambahan atau pengurangan, pertama akan di cek *balance factor* apakah -2 atau +2. Lalu setelah itu akan dicek kembali *balance factor* dari kedua sub-pohon dari pohon utama. Lalu akan dilakukan rotasi sesuai kebutuhan.



Gambar 3-3. Diagram algoritma AVL Tree
Sumber: http://en.wikipedia.org/wiki/AVL_tree

Dapat dilihat pada gambar 3-3 pohon berputar sesuai dengan kedaan pohon tersebut dan pada akhirnya pohon menjadi seimbang.

C. AA Tree

AA Tree adalah salah satu jenis *Self Balancing Binary Search Tree* yang ditemukan oleh Arne Andersson. Pohon ini merupakan varian dari *Red-black Tree.* Dalam pohon ini terdapat beberapa syarat seperti *Red-Black Tree.* Dalam persyaratan sebagai pengganti warna digunakan bilangan *level* simpul.

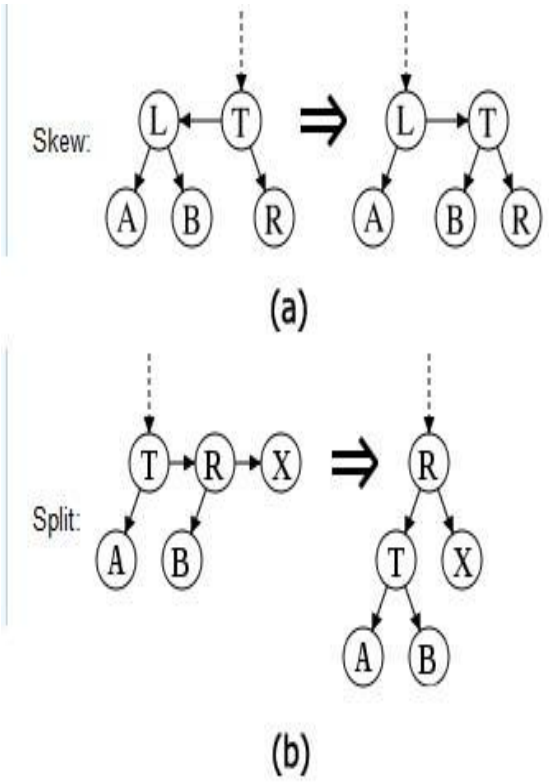
Beberapa persyaratannya adalah:

1. *Level* dari daun adalah 1
2. *Level* dari simpul kiri harus lebih kecil dari simpul orangtua
3. *Level* dari simpul kanan bisa lebih kecil atau

sama dengan orangtua.

4. *Level* dari cucu kanan selalu lebih besar dari *level* kakeknya
5. Simpul yang *level*-nya lebih besar dari satu harus mempunyai dua anak

Pada saat menambahkan atau mengurangkan pada pohon ini berlaku pengurangan dan penambahan seperti pohon pencarian biner biasa. Hanya saja untuk menyeimbangakn akan ditambahkan dua prosedur yaitu *Skew* dan *Split*(Gambar 3-4). Dan karena dua pohon tersebut aka nada keadaan dimana ada simpul horizontal untuk sementara.



Gambar 3-4. (a) Skema Prosedur *Skew*. (b) Skema Prosedur *Split*
Sumber: http://en.wikipedia.org/wiki/AA_tree

Dalam penambahan dan pengurangan akan diadakan pengubahan level agar pohon sesuai dengan ketentuan yang berlaku.

D. Scapegoat Tree

Scapegoat Tree adalah salah satu jenis *Self Balancing Binary Search Tree* yang diciptakan oleh Igal Galperin, Jacob Tepec and Ronald L. Rivest.

Pohon pencarian biner akan dikatan seimbang beratnya jika jumlah simpul di sub-pohon kanan sama dengan jumlah simpul di pohon kiri. Dan dari ketentuan tersebut didapat rumus

size(left) <= α*size(node)
size(right) <= α*size(node)

Dimana size adalah jumlah node dan bila memenuhi rumus tersebut disebut *α-weight-balanced*.

Selain itu sebuah pohon yang seimbang beratnya juga harus seimbang tingginya dan dari sana didapt rumus.

height(tree) <= $\log_{1/\alpha}$(NodeCount)

dan bila memenuhi rumus tersebut disebut *α-height-balanced.*

Dan dalam *Scapegoat Tree* akan selalu dipastikan pohon akan memenuhi *α-height-balanced* walaupun belum dipastikan memenuhi *α-weight-balanced.*

α disini dapat diubah-ubah sesuai ketentuan, dan pengubahan α berpengaruh pada kinerja pohon (apakah pohon cepat saat menambahkan atau menghapus).

Penambahan pada pohon ini dilakukan dengan cara seperti penambahan pada pohon pencarian biner biasa tetapi dengan penambahan *scapegoat* dan perhitungan jumlah dan ketinggian simpul yang akan digunakan untuk penyeimbangan. Dalam penghapusannya juga akan menggunakan jumlah simpul untuk memastikan apakah penyeimbangan perlu dilakukan atau tidak.

E. Splay Tree

Scapegoat Tree adalah salah satu jenis *Binary Search Tree* yang dapat menjadi *Self Balancing Binary Search Tree*. diciptakan Daniel Dominic Sleator and Robert Endre Tarjan tahun 1985. Pohon ini mempunyai ketentuan tambahan yaitu elemen yang baru saja diakses akan mudah dan cepat diakses lagi.

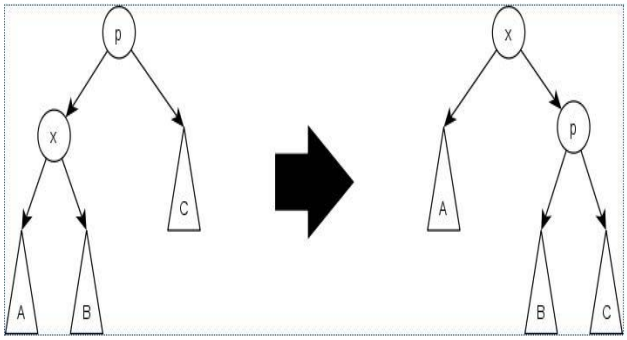
Semua kejadian yang diatur dalam *Splay Tree* disebut *Splaying.* Dalam *Splaying*, sebuah simpul akan diatur seperti biasa lalu dilanjutkan dengan rotasi pohon agar pohon tetap seimbang dan simpul yang mengandung elemen yang mudah diakses tersebut dipindah menjadi akar.

Setiap kali elemen x diakses, *Splay* akan membawanya ke akar sehingga terdapat beberapa dan tahap-tahap tersebut tergantung beberapa factor yaitu:

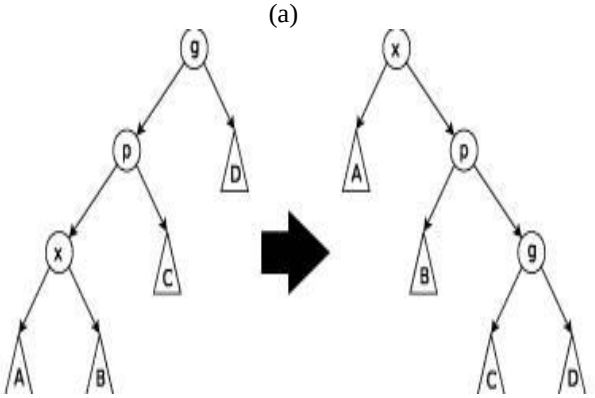
- Apakah x adalah simpul kanan atau kiri dari orangtuanya(simpul p)
- Apakah p akar atau bukan
- Apakah p adalah simpul kanan atau kiri dari Orangtuanya,g(kakek dari x)

Dan menurut faktor-faktor tersebut terdapat 3 cara *Splaying* yaitu:

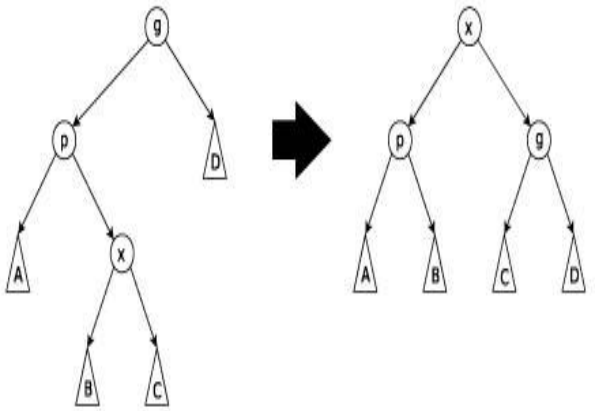
- *Zig Step(Gambar 3-5(a))*
- *Zig-Zig Step(Gambar 3-5(b))*
- *Zig-Zag Step(Gambar 3-5(c))*



(a)



(b)



(c)

Gambar 3-5. (a)Skema *Zig Step* (b)Skema *Zig-zig Step* (c)Skema *Zig-zag Step*
Sumber: http://en.wikipedia.org/wiki/Scapegoat_tree

Dapat dilihat pada gambar *Splay Tree* dapat menjadi *Self Balancing Binary Search Tree* bila terjadi proses *Zig-zag* pada akhirnya.

IV. KESIMPULAN

Self Balancing Binary Search Tree merupakan salah satu jenis pohon yang sangat berguna sebagai struktur data dalam penyelesaian masalah di bidang komputer dan juga terdapat berbagai jenis, sehingga dapat diadaptasikan sesuai permasalhan dan kebutuhan, bahkan terdapat pohon yang lebih fleksibel seperti *Splay Tree.*

V. DAFTAR PUSTAKA

[1] http://www.informatika.org/~rinaldi/Matdis/2008-2009/Pohon.ppt Minggu,12 Desember 2010
[2] http://en.wikipedia.org/wiki/Binary_search_tree Senin,13 Desember 2010
[3] http://en.wikipedia.org/wiki/Tree_%28data_structure%29 Senin,13 Desember 2010
[4] http://en.wikipedia.org/wiki/Self-balancing_binary_search_tree Senin,13 Desember 2010
[5] http://en.wikipedia.org/wiki/Red-black_tree Selasa,14 Desember 2010
[6] http://en.wikipedia.org/wiki/AVL_tree Selasa,14 Desember 2010
[7] http://www.cs.auckland.ac.nz/~jmor159/PLDS210/red_black.html Selasa,14 Desember 2010
[8] http://en.wikipedia.org/wiki/AA_tree Selasa,14 Desember 2010
[9] http://en.wikipedia.org/wiki/Scapegoat_tree Selasa,14 Desember 2010
[10] http://en.wikipedia.org/wiki/Splay_tree Rabu,15 Desember 2010

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 17 Desember 2010

KevinWibowo/13509065