BAB III

ALGORITMA PENYEIMBANGAN POHON AVL DENGAN MENGGUNAKAN STRUKTUR DATA PASCAL

Pencarian data tabel *look-up* atau *storage* dan *retrival information* adalah suatu proses untuk mengumpulkan sejumlah informasi di dalam pengingat komputer dan kemudian mencari kembali informasi yang diperlukan secepat mungkin. Seringkali informasi yang diperlukan sedikit tetapi banyak informasi sehingga perlu menghapus informasi- informasi yang tidak digunakan tersebut.

Algoritma pencarian *(searching algorithm)* merupakan algoritma yang menerima K sebagai kunci dan dengan langkah–langkah tertentu akan mencari rekaman yang kuncinya bernilai K. Setelah proses pencarian dilaksanakan akan diperoleh salah satu dari dua kemungkinan yaitu data yang dicari ditemukan *(succsesfull)* atau data tidak ditemukan *(unsuccesfull)* .

Dengan algoritma pencarian pohon biner seimbang akan dihasilkan suatu bentuk pohon biner seimbang (*balanced binery trees)* dari sembarang pohon biner. Mengingat pohon biner seimbang merupakan tipe yang sangat penting dari struktur pohon dan banyak dijumpai dalam berbagai terapan. Misalnya pohon biner seimbang dapat digunakan untuk mencegah terjadinya kemrosotan pencarian pohon di dalam list di bawah tingkat penyisipan.

3.1 Pohon Pencarian Biner

Pohon pencarian biner merupakan suatu teknik penting untuk pemeliharaan struktur kamus *(dictionary)*, tabel simbol file dan direktori. Di samping itu struktur pohon biner untuk pencarian akan memberikan keuntungan :

1. Bila tabel pohon biner adalah statis, yaitu tidak memungkinkan adanya pemasukan dan penghapusan data, maka struktur pohon yang eksplisit dapat dimanfaatkan dengan ketentuan frekuensi akses setiap data diketahui.
2. Bila tabel pohon biner adalah dinamis, yaitu perubahan dari tabel berupa penambahan dan penghapusan data dimungkinkan, maka struktur pohon yang eksplisit dapat dimanfaatkan sehingga jumlah operasi pencarian dapat diperoleh dalam order log dari waktu.

Suatu pohon pencarian biner (*binery search tree)* adalah suatu pohon dengan transversal secara inordernya akan memberikan urutan data yang sudah terurut menurut urutan key. Dengan setiap simpulnya berlaku bahwa nilai key data di cabang kiri lebih kecil dari nilai key data pada simpul tersebut dan nilai data pada simpul tersebut lebih kecil dari nilai data dari cabang kanan. Untuk pembahasan selanjutnya istilah subpohon diubah menjadi cabang . Kunjungan inorder pada sebuah pohon biner menghasilkan file dalam urutan kunci key naik.

Salah satu pohon pencarian biner adalah pohon pencarian biner seimbang yang dikenalkan G.M Andelson dan E.M Landish pada tahun 1962 sehingga disebut pula pohon AVL. Dengan menggunakan metoda AVL diperlukan dua ekstra bit per simpul

dan untuk pencarian suatu tree atau menyisipkan sebuah kunci diperlukan maksimal O(log n) operasi. Pembahasan selanjutnya akan dibahas dalam bagian 3.3.

Definisi 3.1.1

Pohon pencarian biner adalah pohon biner yang kosong atau setiap simpulnya memuat sebuah kunci key dan memenuhi kondisi sebagai berikut :

1. Kunci dalam cabang kiri pada sebuah simpul (jika ada) lebih kecil dari kunci simpul akar.
2. Kunci dalam cabang kanan pada sebuah simpul (jika ada) lebih besar dari kunci simpul akar.
3. Cabang kiri dan cabang kanan dari akar merupakan pohon pencarian biner.

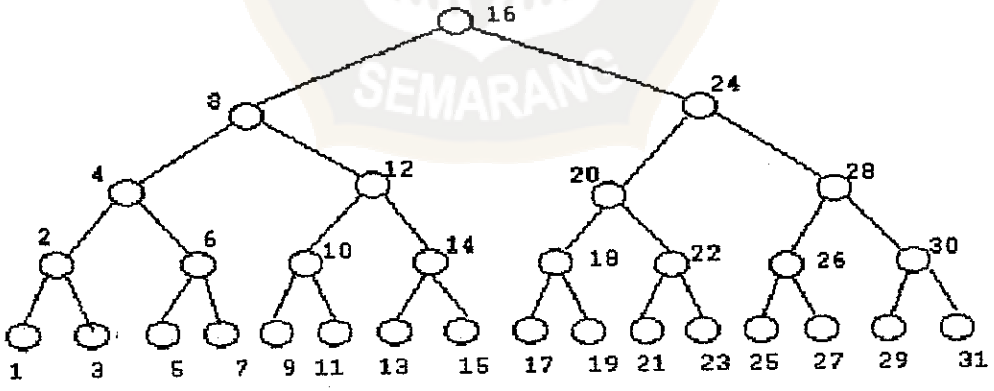
3.2 Membangun Pohon Pencarian biner

Dalam menyusun pohon pencarian biner pertama kali diasumsikan bahwa terdapat sebuah list yang tediri atas simpul-simpul yang telah terurutkan atau sebuah file yang terdiri atas record-record dengan kunci terurutkan secara alphabetik. Jika simpul–simpul ini digunakan sebagai tempat penyimpanan informasi ,tempat penambahan simpul baru atau perubahan lainnya, maka list atau file yang terdiri atas simpul-simpul itu dapat dibuat dalam sebuah pohon pencarian biner.

Hal ini dapat dilakukan dengan menggunakan algoritma penyisipan pohon yang diawali dengan sebuah pohon kosonng dan sederhana. Selanjutnya memasukkan simpul-simpul tersebut ke dalam pohon . Tetapi simpul-simpul yang telah diurutkan

tersebut akan menghasilkan pohon pencarian berupa sebuah deretan panjang dan penggunaaanya lebih lambat dibanding dengan kecepatan pencarian squensial dari pencarian biner.

Dari gambar 3.2.1 diperoleh bentuk pohon biner lengkap dengan cacah simpul n =31 terdiri atas simpul dengan label bilangan 1 sampai 31. Di mana semua label simpul daunya terdiri atas bilangan yang tidak bisa dibagi 2. Sedangkan simpul satu level di atasnya adalah bilangan 2, 6, 10, 14, 18, 22, 26 ,dan 30. Bilangan–bilangan ini semuannya habis dibagi 2 tetapi tidak habis dibagi 4. Selanjutnya satu level diatasnya adalah bilangan–bilangan yang habis dibagi 4 yaitu 4, 12, 20, dan 28 tetapi tidak habis dibagi 8. Berikutnya adalah simpul dengan label kunci 8 dan 24 yang berada satu level di atasnya. Akar pohon merupakan simpul dengan label kunci 16.



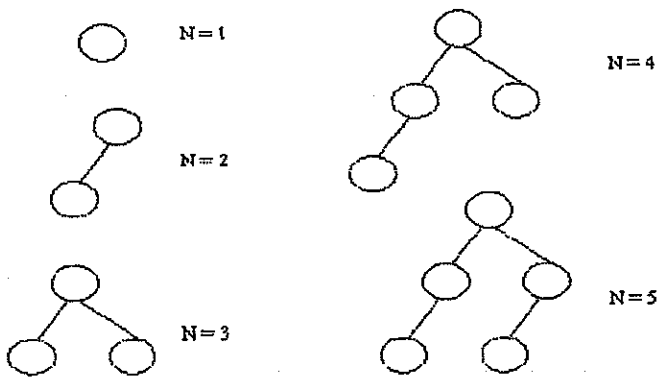
Gambar 3.2.1 Pohon biner lengkap dengan 31 simpul

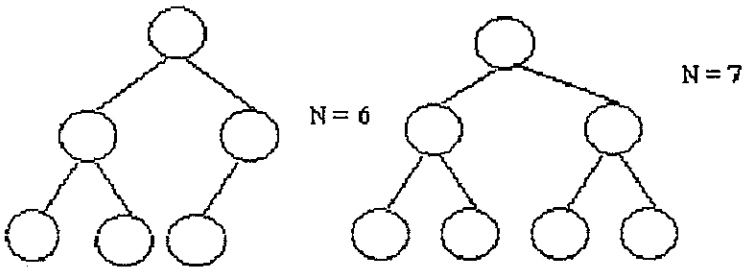
3.3 Penyajian Pohon AVL

Pohon biner dapat digunakan untuk menyajikan himpunan data yang elemennya dapat dicari lewat suatu kunci. Dimisalkan bahwa pohon yang akan dibangun memuat simpul yang bertipe sebagai

```
Type node = record
        op : char ;
        Left, right : ^ node ;
    end ;
```

Dengan nilai simpulnya, sebanyak n buah dibaca dari berkas masukan. Pohon biner dapat diperoleh dengan mengalokasikan sebanyak mungkin simpul pada semua aras kecuali aras yang paling rendah dengan mendistribusikan simpul-simpul yang masuk secara merata ke sisi kiri dan kanan setiap simpul maka akan diperoleh sebuah pohon biner. Hal ini dapat ditunjukkan pada gambar 3.3.1 berikut ini





Gambar 3.3.1 pohon biner seimbang

Adapun aturan pendistribusian merata sebanyak n simpul paling baik dapat dinyatakan sebagai aturan rekursif berikut :

1. Menggunakan sebuah simpul sebagai akar (root)
2. Membangkitkan cabang kiri yang terdiri atas nl = n div 2 simpul
3. Membangkitkan cabang kanan yang terdiri atas nr = n – nl – 1 simpul

Dalam sebuah pohon seimbang sempurna , cabang kiri dan cabang kanan mempunyai ketinggian sama . Sehingga cacah simpul maksimalnya sebanyak $2^k - 1$. Untuk definisi seimbang secara umum dapat disajikan sebagai berikut :

Definisi 3.1.1

Jika T adalah sebuah pohon yang tidak kosong dengan T_L sebagai subpohon kiri dan T_R sebagai subpohon kanan, T disebut pohon biner seimbang jika :

1. T_L dan T_R telah mencapai keseimbangan

2. $|H_L - H_R| \leq 1$ dimana HL adalah ketinggian cabang kiri (T_L) dan H_R ketinggian cabang kanan (T_R)

Definisi 3.3.2

Tingkat (*level*) suatu simpul ditentukan dengan menentukan akar sebagai tingkat satu. Jika suatu simpul dinyatakan sebagai tingkat n, maka simpul –simpul yang merupakan anaknya dikatakan berada dalam tingkat n+1.

Definisi 3.3.3

Tinggi *(height)* atau kedalaman dari suatu pohon adalah tingkat maksimum dari simpul dalam pohon tersebut.

Berikut ini adalah algoritma untuk membangun pohon seimbang sempurna :

Algoritma pembentukan pohon pencarian biner seimbang :

Langkah 0 : Test kondisi apakah n=0

Jika ya, maka pohon = nil teruskan langkah ke 6

Jika tidak kerjakan langkah 1 dan 2

Langkah 1: Tentukan nr = n div 2

{ menyatakan pohon pada cabang kiri }

Langkah 2 : tentukan nr = n – nl – 1

Langkah 3 : baca data x

Langkah 4 : Tentukan Newnode^. Key := x

Newnode^. Left := tree(nl)

Newnode^.right := tree(nr)

Langkah 5 : Tentukan tree := newnode

Langkah 6 : Selesai.

Algoritma I

```
Program Buildtree (input, output) ;
Type  ref = ^ Node;

Node = record key  : integer ;
Left,right : ref
End;
Var  n : integer ;
Root :  ref;
  Function tree(n : integer): ref;
        Var newnode : ref;
              X,nl,n r : integer ;
               Begin
                 If n = 0 then tree :  = nil else
      Begin
               nl  := n div 2;
               nr  :=  n- nl – 1;
               Read (x) ; newnode^ do ;
               With newnode do
            Begin
                 key : = x ; left : = tree (n1)
                 Right : = tree (nr) ;
              End ;
                   Tree : = newnode ;
          End ;
       End ; (tree)
```

3.4 Operasi-Operasi Pada Pohon AVL

Dalam melakukan penyeimbangan pohon biner dari bentuk tidak seimbang *(unbalanced)* menjadi bentuk seimbang sempurna *(complete balanced)*, hal yang harus diperhatikan bahwa keadaan akan selalu seimbang apabila cabang kanan dan cabang kiri tingginya sama. Karena prosedur penyisipan untuk menjadikan pohon ke keadaan seimbang sempurna memerlukan operasi yang cukup rumit, maka perbaikan yang mungkin dilakukan terletak pada keluwesan definisi keadaan seimbang. Dengan mengurangi ujuk kerja pencarian rata-rata ini kriteria seimbang tak sempurna akan mempermudah prosedur pengorganisasian kembali.

Prosedur ini tidak hanya sederhana, tetapi juga lebih memudahkan prosedur penyeimbangan simpul-simpulnya dan panjang jalur pencarian rata-rata secara praktis sama dengan pada pohon seimbang sempurna.

Operasi –operasi yang dilakukan atas pohon AVL dengan O(log n) satuan waktu adalah :

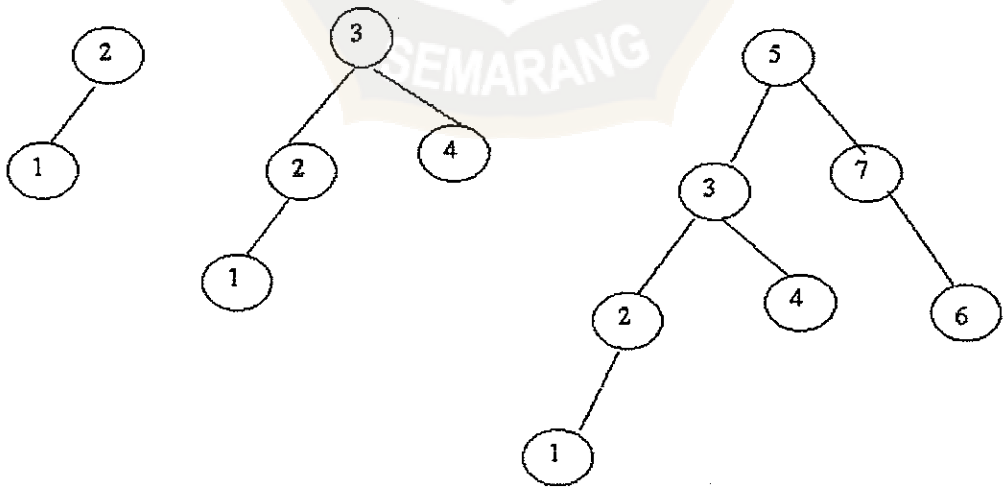
1. Menyisipkan simpul dengan kunci tertentu.
2. Mencari simpul dengan kunci tertentu.
3. Menghapus simpul dengan kunci tertentu.

Pembahasan lebih lengkapnya akan dibahas dalam bagian 3. 4. 1 dan bagian 3. 4. 2 . Menurut Adelson Velski dan Landish, yang telah membuktikan bahwa pohon biner seimbang tidak mempunyai kelebihan tinggi lebih dari 45% dibanding pohon

seimbang sempurna , tak gayut terhadap cacah simpul yang ada. Nilai optimumnya akan dicapai apabila pohon dalam keadaan seimbang sempurna untuk $n = 2^k - 1$.

Untuk mencari tinggi maksimum h dari semua pohon seimbang dengan n simpul, dimisalkan bahwa tinggi h-nya tetap dan membentuk pohon seimbang dengan cacah simpul minimum. Strategi ini sangat disarankan karena seperti dalam kasus minimal h, nilai ini dapat diperoleh dengan nilai n tertentu.

Misalkan pohon dengan tinggi h sebagai Th, To sebagai pohon kosong dan T1 pohon dengan sebuah simpul. Untuk membentuk pohon Th dengan h>1, disiapkan akar yang mempunyai dua buah cabang pohon dengan cacah simpul minimal. Sehingga sebuah cabang pohon harus mempunyai tinggi h – 1, dan cabang lainya mempunyai tinggi kurang dari h-1 misalnya h-2. Contoh pohon seimbang dengan tinggi 2, 3 dan 4.ditunjukkan dalam gambar 3.4.1.



Gambar 3. 4.1 pohon Fibonacci dengan tinggi 2, 3 dan 4

Karena komposisinya menyerupai deret Fibonacci, maka disebut pohon Fibonacci dan didefinisikan sebagai berikut :

Definisi 3. 4. 1

Pohon kosong adalah pohon Fibonacci dengan ketinggian 0. Sebuah Sebuah pohon adalah pohon Fibonacci dengan ketinggian 1. Jika Th-1 dan T2 pohon Fibonacci dengan ketinggian h-1 dan h—2, maka Th = (Th-1, x, Th-2) adalah pohon Fibonacci dengan ketinggian h.

Banyaknya simpul pada Th didefinisikan sebagai hubungan rekursif sederhana.:

$$Nh = N_{h-1} + 1 + N_{h-2}$$

Dengan syarat awal $N_0 = 0$ dan $N_1 = 1$

3.4.1 Menyisipkan Simpul Baru

Dalam melakukan penyisipan simpul baru, satu hal yang harus dipertahankan adalah kunjungan inordernya. Untuk mengetahui cabangnya seimbang, miring kiri atau ke kanan digunakan tanda sebagai berikut :

0, Apabila seimbang, yaitu jika jalur terpanjang kedua cabangnya sama.

-1, Miring ke kiri, yaitu jika jalur terpanjang dari cabang kiri satu lebih besar dibanding cabang kanannya.

+1, Miring ke kanan, yaitu jika jalur terpanjang dari cabang kanan satu lebih besar dibanding cabang kiri.

Jika sebuah simpul disisipkan pada salah satu cabangnya, misalkan cabang kirinya, maka terdapat tiga kasus yang berbeda :

1. $H_L = H_R$; tinggi L dan R menjadi tidak sama, akan tetapi kriteria seimbang tidak dilanggar
2. $H_L < H_R$; tinggi L dan R menjadi sama sehingga diperoleh keadaan seimbang.
3. $H_L > H_R$; kriteria seimbang dilanggar, sehingga pohon harus ditata kembali.

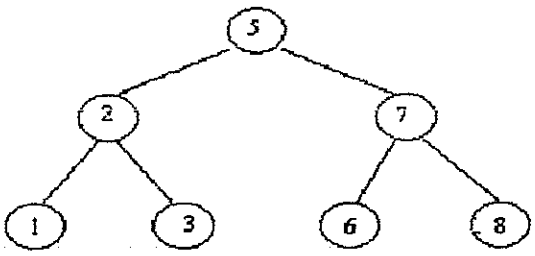
Definisi 3. 4.1

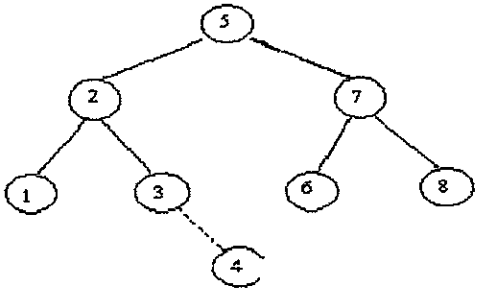
Faktor keseimbangan BF(T), pada sebuah simpul T dalam suatu pohon biner didefinisikan sebagai selisih tinggi antara cabang kiri dengan cabang kanan (hl-hr), dimana h_L dan h_R merupakan tingkat ketinggian cabang kiri dan cabang kanan pada T. Sehingga untuk beberapa simpul T dari pohon biner seimbang (AVL tree).BF(T) = -1, 0 atau +1 .

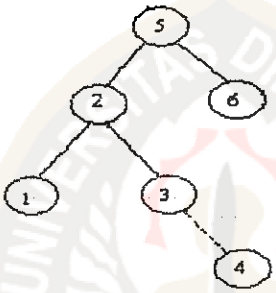
-1 menujukkan cabang miring ke kiri.

0 menunjukkan cabang seimbang.

+1 menunjukkan cabang miring ke kanan.







Gambar 3. 4.2 Perubahan tanda simpul

Algoritma untuk penyisipan dan penyeimbangan kembali tergantung dari penyimpanan informasi pohon AVL. Salah satu penyelesaiannya adalah dengan menyimpan keadaan seimbang dalam pohon itu sendiri, sehingga faktor keseimbangan simpul harus diketahui setiap kali ada penyisipan simpul baru, cara yang lain adalah dengan menyimpan faktor keseimbangan pada setiap simpul. Oleh karena itu, simpulnya didefinisikan

```
Type node = record key : integer ;
                   Count : integer ;
                   Left, right : ref ;
                   Bal : -1....+1 ;
                  End ;
```

Adapun proses penyisipan simpul baru terdiri atas 3 bagian yaitu :

1. Menelusuri jalur pencarian sampai diverifikasi bahwa simpul yang disisipkan belum ada dalam pohon *(tree)*.
2. Menyisipkan simpul baru dan menentukan faktor keseimbangannya
3. Mengulangi penelusurannya dan mengecek kembali faktor keseimbangan pada setiap simpulnya

Walaupun cara ini mengandung beberapa redudansi pengecekan (setelah keadaan seimbang diperoleh, tidak diperlukan lagi pengecekan ke nenek moyang simpul itu), tetapi skema ini dapat diimplementasikan lewat pengembangan murni dari prosedur pencarian dan penyisipan sebagai berikut :

Langkah 0 : Test kondisi apakah p = nil

Jika ya, kerjakan langkah 1 dan langkah 2.

Jika tidak, kerjakan langkah 3

Langkah 1 : Tentukan begin new(p);

Langkah 2 : Tentukan begin p^. Key := x;

P^.count := 1;

P^.left := nil;

P^.right := nil;

Langkah 3 : Test kondisi apakah x < p^.key

Jika ya, kerjakan langkah 4

Jika tidak, kerjakan langkah 5

Langkah 4 : { Mencari di cabang kiri }

Search (x, p^.left)

Langkah 5 : Test kondisi apakah x>p^.key

Jika ya, kerjakan langkah 6

Jika tidak, kerjakan langkah 7

Langkah 6 : {Mencari di cabang kanan}

Search(x, p^.right);

Langkah 7 : tentukan p^.count := p^.count + 1;

Langkah 8 : Tentukan root := nil;

Lankag 9 : baca data k

Langakah 10 : Tentukan search(k, root);.

Langkaj 11 : Selesai.

Algoritma 2

```
Procedure search (x:integer ; var p : ref) ;
Begin
  If p = nil then
     Begin
       New (p) ;
      With p do
        Begin
        Key : = x ; count : = 1 ; left : = nil ;
        Right : = nil ;
       End
     End
     Else
```

```
If x < p^.key then search(x, p^.left)
  Else
    If x> p^.key then search (x, p^.right)
      Else
          P^.count  :=  p^.count + 1
    End
     Begin
    Root := nil;
Begin
 Read (k);
Search(k, root)
end ;
```

Prosedur ini menjelaskan operasi pencarian yang diperlukan oleh setiap simpul dan sifatnya rekursif, hal ini dapat mengakomodasikan operasi tambahan "pada saat kembali kejalur pencarian" pada setiap langkah, informasi harus dilewatkan tidak tergantung apakah tinggi cabang pohon (tempat simpul disisipkan) telah berubah. Sehingga perlu penambahan daftar parametar dengan peubah boolean h yang berarti bahwa "tinggi cabang pohon telah bertambah". Jadi h harus merupakan parameter peubah karena perubahan nilainya juga diperlukan.

Untuk menunjukkan perbedaan tinggi subpohon sebelum penyisipan dilaksanakan perlu dibuat suatu ketentuan sebagai berikut :

1. $H_L < H_R$, P^.bal = +1 kondisi tak seimbang pada p telah diseimbangkan
2. $H_L = H_R$, P^bal = 0 bebannya miring ke kiri
3. $H_L > H_R$, P^.bal = -1 perlu penyeimbangan kembali

Selanjutnya akan dibahas bagaimana menyeimbangkan pohon biner yang tidak seimbang dengan cara pemutaran. Terdapat dua kasus yang bisa terjadi yang masing-masing dibagi lagi menjadi dua subkasus yang serupa, yaitu :

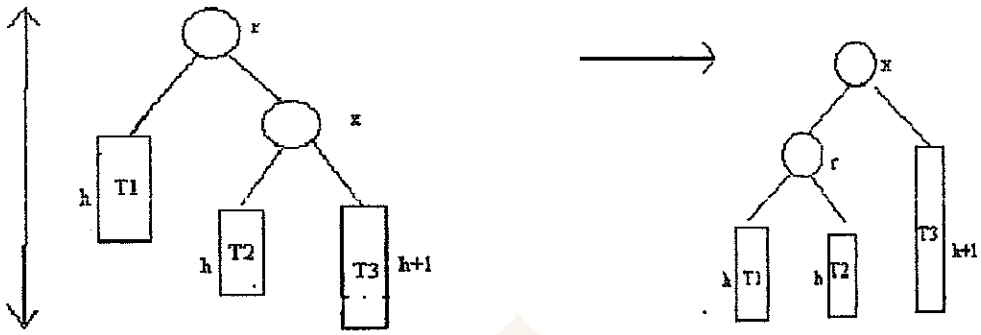
1. Pemutaran tunggal *(single rotation)* yang bisa dibagi menjadi pemutaran tunggal ke kiri (karena cabang pohon kanan lebih berat), dan pemutaran tunggal ke kanan (karena cabang kiri lebih tinggi)
2. Pemutaran ganda*(double rotation*) yang bisa dibagi menjadi pemutaran ganda kiri-kanan dan pemutaran ganda kanan-kiri.

3.4.1.1 Pemutaran Tunggal (*Single Rotation*)

Penambahan simpul baru pada suatu pohon biner seimbang (*balanced binery trees)* akan menyebabkan pohon tersebut berubah dari bentuk semula. Untuk menjaga agar pohon yang disisipi simpul baru tetap seimbang, maka perlu dilakukan pemutaran (*rotation*).

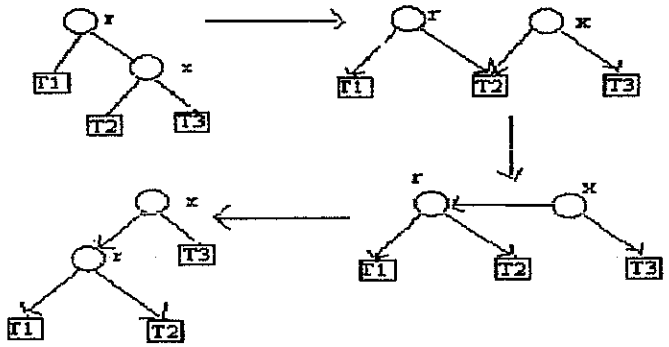
Pemutaran tidak harus dilakukan pada bagian akar dari pohon ; tetapi dapat dilakukan juga pada beberapa simpul dalam pohon, karena simpul merupakan akar pada subpohon *(subtree) t*ersebut

Pemutaran tunggal (single rotation) dilakukan untuk memperbaiki pohon biner seimbang (AVL tree) apabila penysisipan simpul baru menyebabkan salah satu simpulnya kehilangan sifat-sifat keseimbangannya. Gambar 3. 4. 2 dan gambar 3. 4..3 berikut ini menggambarkan pohon AVL yang memerlukan pemutaran tunggal.



Gambar 3. 4.3 Pohon Avl cabang kanan lebih tinggi

Dari gambar 3.4.3 nampak bahwa penambahan simpul baru pada cabang kanan menyebabkan pohon AVL kehilangan sifat keseimbangannya. Kondisi seimbang diperoleh dengan pemutaran tunggal ke kanan , yaitu dengan cara memutar simpul x ke atas menjadi akar pohon dan menurunkan r menjadi cabang kiri x. Adapun langkah- langkahnya sebagaimana ditunjukkan dalam gambar 3. 4. 4 berikut ini



Gambar 3. 4.4 Langkah-langkah penyeimbangan

Dari gambar 3. 4.4 diperoleh bahwa :

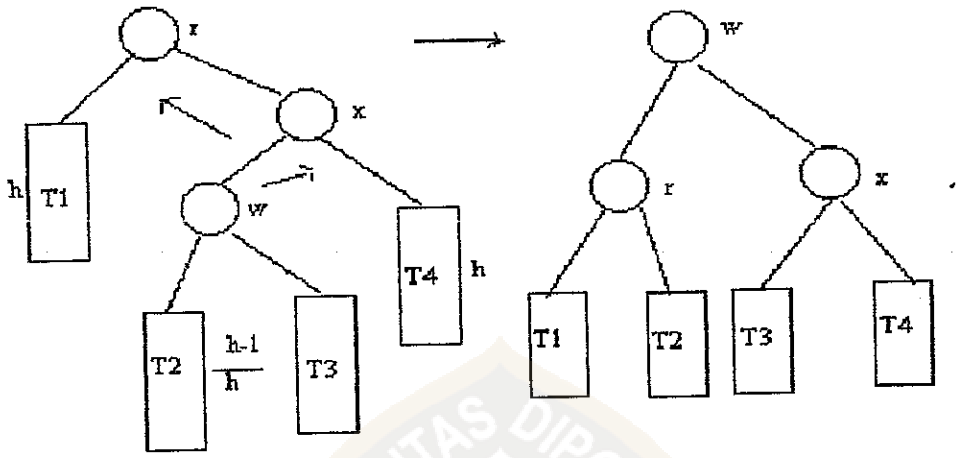
1. Nilai r < x
2. Nilai T1 < r
3. Nilai x < T3
4. T2 terletak antara r dan x

3.4.1.2 Pemutaran ganda *(double rotation)*

Pemutaran ganda dilakukan pada pohon biner yang tidak seimbang karena pemutaran tunggal tidak mampu mengembalikan keadaan seimbang . Seperti halnya pemutaran tunggal, pemutaran ganda terdiri atas pemutaran ganda kanan- kiri *(right-left double rotation)* dan pemutaran ganda kiri-kanan *(left-right double rotation).*

Case I: cabang kiri lebih tinggi

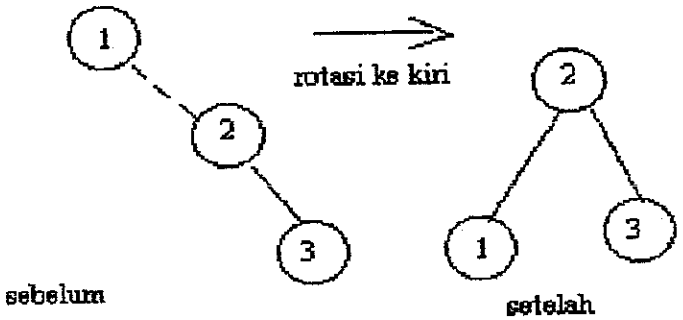
Kasus ini terjadi apabila cabang kirinya lebih tinggi, dengan faktor keseimbangan x lebih tinggi cabang kirinya. Sehingga perlu memindahkan dua level karena simpul w yang merupakan akar cabang kiri x terdapat simpul baru. Hal ini dapat dijelaskan dalam gambar 3.3.6, dimana dilakukan pemutaran simpul w ke kanan sehingga x menjadi cabang w. Selanjutnya w diputar kekiri sehingga w menjadi akar baru pohon tersebut.



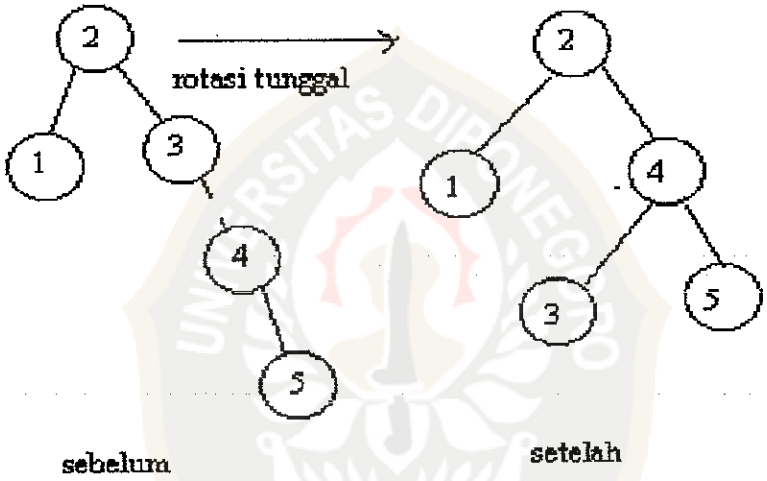
Gambar 3.4.5 Pemutaran ganda

Contoh kasus

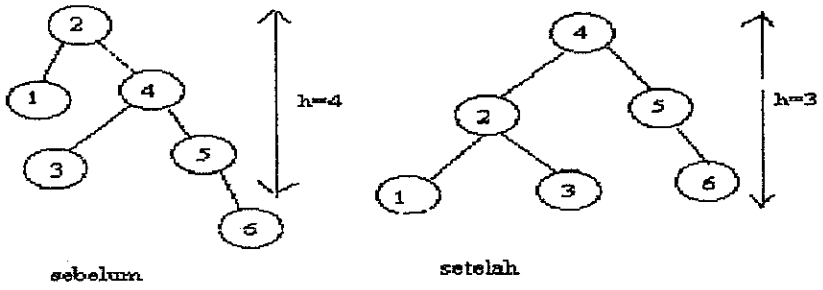
Misal diberikan sebuah pohon kosong, kemudian disisipkan kunci 1 sampai dengan kunci 7 secara urut. Masalah yang pertama kali muncul adalah saat penyisipan simpul dengan kunci 3 (sebab sifat pohon AVL dilanggar) . Dengan melakukan pemutaran tungggal antara akar dan cabang kanan akan diperoleh kondisi seimbang.



Penambahan simpul 4 pada cabang kanan tidak menimbulkan masalah, akan tetapi setelah disisipkan simpul 5 menyebabkan simul 3 kehilangan sifat keseimbangannya sehingga memerlukan pemutaran tunggal ke kiri.



Penambahan simpul dengan kunci 6 menyebabkan akar kehilangan sifat keseimbangan, karena cabang kiri tingginya 1 dan cabang kanan tingginya 3. Sehingga pemutaran tunggal diperlukan untuk menyeimbangkan antara simpul 2 dan simpul 4.



3.4.2 Menghapus Simpul

Menghapus elemen simpul dalam pohon AVL seperti halnya kasus dalam pohon pencarian biner. Penghapusan simpul nyata lebih sulit dibanding menyisipkan simpul baru, terutama dalam hal mengatur kembali pointer-pointernya. Jika sebuah simpul dihapus, kecuali simpul yang bertindak sebagai daun maka posisinya akan digantikan simpul lain yang merupakan urutan predesesornya apabila dibaca secara inorder (atau simpul paling kanan dari cabang kiri).

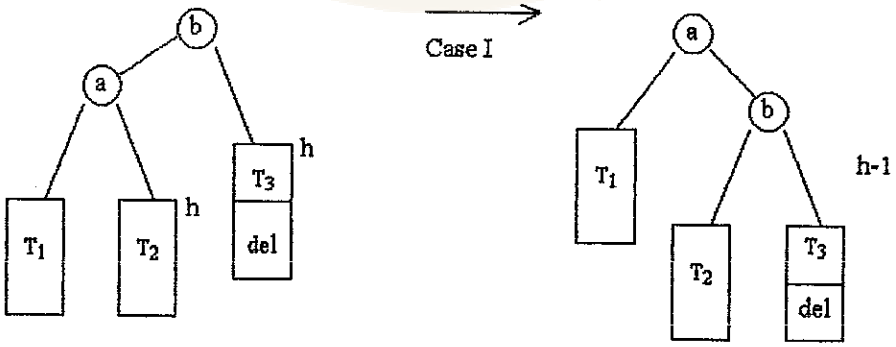
Setelah sebuah simpul dihapus, maka kondisi setiap simpul dalam pohon perlu dicek untuk meyakinkan keadaan seimbangnya. Apabila ditemukan sebuah simpul yang cabang kiri dan cabang kanannya tidak seimbang, maka pada simpul tersebut perlu dilakukan penyeimbangan kembali dengan salah satu bentuk penyeimbangan yang telah dijelaskan sebelumnya.

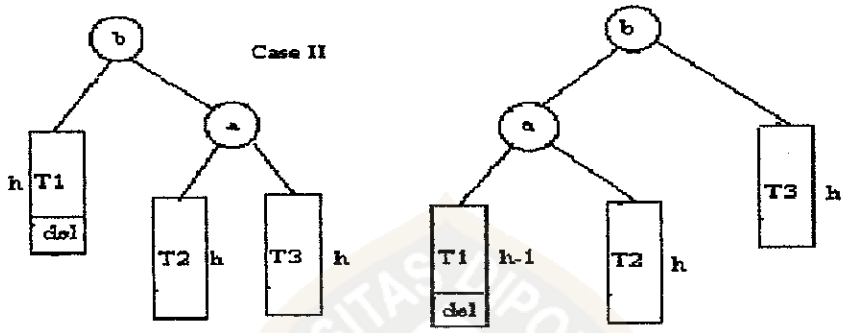
Terdapat tiga kasus yang bisa terjadi setelah penghapusan salah satu simpul pohon AVL, yaitu :

1. jika simpul akar kondisinya telah seimbang 0, maka penghapusan salah satu simpulnya tidak berdampak pada simpul akar tersebut. Tetapi faktor keseimbangan berubah menjadi +1 apabila yang dihapus simpul anak pada cabang kirinya dan berubah menjadi -1 jika yang dihapus simpul anak pada cabang kanan.

2. Jika simpul akar kondisinya telah seimbang +1 , maka penghapusan simpul cabang kanan menyebabkan faktor keseimbangan simpul akar menjadi 0, dalam artian seimbang sempurna. Sebaliknya apabila simpul akar kondisinya telah seimbang -1, maka penghapusan simpul cabang kiri menyebabkan faktor keseimbangan simpul akar menjadi 0.

3. Apabila simpul akar keadaan seimbangnya +1 dan cabang kiri dihapus atau simpul akar keadaan tingginya -1, dan cabang kanan dihapus, maka keadaan seimbang pada simpul akar harus diseimbangkan kembali.

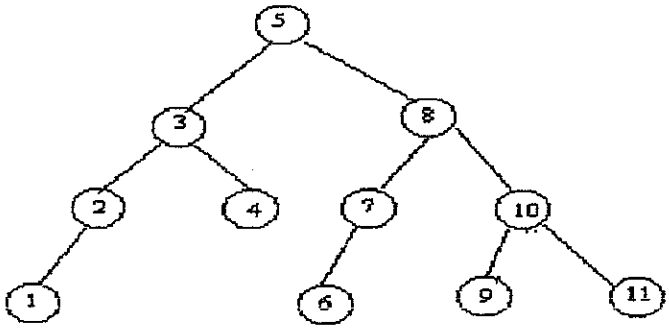




Gambar 3. 4.6 Penyeimbangan dengan pemutaran tunggal

Contoh kasus :

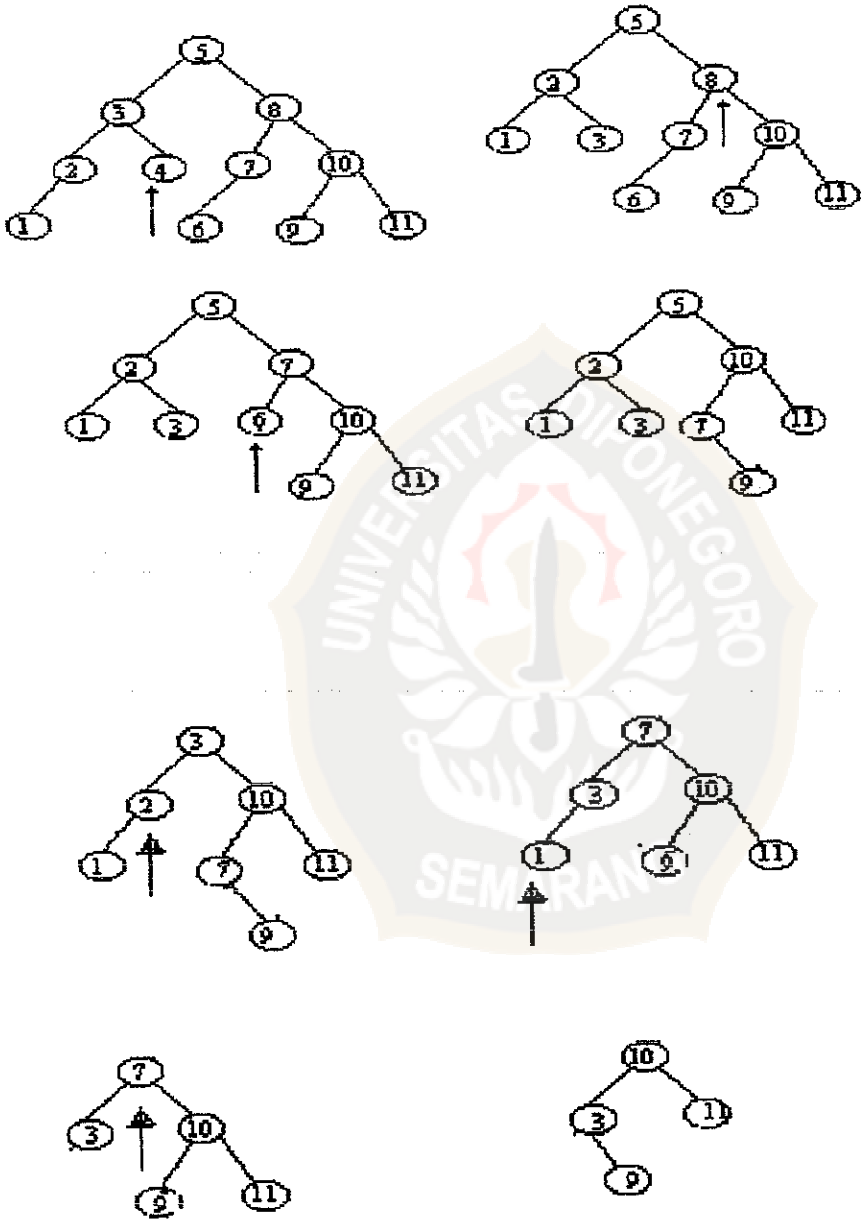
Diberikan sebuah pohon AVL sebagaimana ditunjukkan pada gambar 3. 4.7 sebagai berikut :



Gambar 3. 4.7 Pohon biner seimbang

Dari gambar 3. 4. 7 penghapusan simpul secara berturut-turut dengan kunci 4, 8, 6, 5, 2, 1 dan 7 akan menghasilkan pohon pada gambar 3. 3. 8. Penghapusan kunci 4 dengan mudah dapat dilakukan , karena merupakan simpul daun. Tetapi menyebabkan keadaan tak seimbang pada simpul-simpul sehingga memerlukan operasi penyeimbangan kembali dengan menggunakan pemutaran tunggal. Penyeimbangan kembali diperlukan lagi setelah penghapusan simpul 6. Cabang kanan dari akar dengan kunci simpul 7 diseimbangkan dengan pemutaran tunggal.

Selanjutnya menghapus simpul 2, meskipun dapat dikerjakan secara mudah karena hanya mempunyai satu turunan, tetapi memerlukan pemutaran ganda kanan-kiri. Pemutaran ganda kiri-kanan, dipanggil pada saat menghapus simpul simpul 7, yang akan diganti pertama kali oleh elemen paling kanan dari cabang kiri adalah simpul dengan kunci 3.



Gambar 3. 4.8 Menghapus pohon seimbang

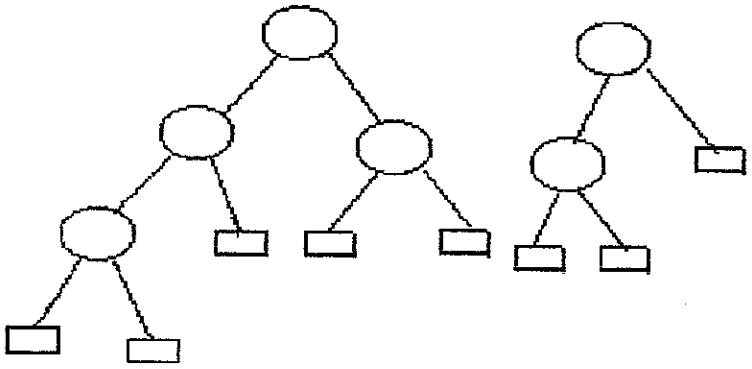
Penghapusan elemen dalam pohon seimbang dapat dilakukan dengan dalam kasus terjelek – O(log n) operasi. Meskipun demikian, perbedaan prosedur penyisipan

dan penghapusan tidak bisa diabaikan. Penyisipan sebuah kunci paling banyak akan menyebabkan sebuah pemutaran tunggal (dari dua atau tiga simpul), sedangkan penghapusan simpul seringkali memerlukan pemutaran setiap simpul sepanjang jalur pencarian. Misal dalam menghapus sebuah simpul paling kanan dari pohon fibonacci . Dalam kasus ini penghapusan sembarang simpul tunggal akan menyebabkan tinggi pohon berkurang dan penghapusan simpul paling kanan memerlukan cacah pemutaran maksimum.

3.5 Menghitung runing time

Untuk melakukan analisa runing time pencarian pada pohon AVL dan pohon pencarian biner, perlu ditambahkan simpul tambahan di tempat-tempat dengan pointer dari cabang pohon berharga **nil**. Di mana simpul pohon Avl itu merupakan daun sehingga untuk membedakan simpul tambahan dari simpul pohon biner yang asli maka simpul pohon biner yang asli diberi tanda lingkaran dan disebut simpul dalam *(internal node)* sedangkan simpul tambahan disebut simpul luar *(external node)* diberi tanda kotak. Selanjutnya pohon biner yang mempunyai simpul tambahan (*external node*) disebut pohon yang diperluas (*extended binary trees*) .

Dalam menganalisa masalah ini yangmenjadi perhatian adalah apakah simpul tersebut merupakan simpul dalam atau simpul luar. Secara umum pohon AVL yang mempunyai cacah simpul dalam sebanyak n akan mempunyai n+1 buah simpul luar. Hal ini ditunjukkan dalam gambar 3. 5. 1



(a) (b)

Gambar 3.5.1 a.Pohon fibonacci dengan 4 simpul dalam

b. Pohon fibonacci dengan 2 simpul dalam

Untuk menghitung kecepatan pada pohon pencarian biner seimbang di definisikan besaran-besaran berikut ; panjang jalur luar *(external path lenght)* dinotasikan E adalah jumlah dari panjang jalur semua daun pohon biner. Sedang panjang jalur dalamnya *(internal path lenght)* dinotasikan dengan I yaitu jumlah semua jalur simpul dalam dari pohon biner tersebut.

Dari gambar 3.5.1 pohon fibonacci dengan 4 simpul dalam dan 2 simpul dalam dapat di peroleh :

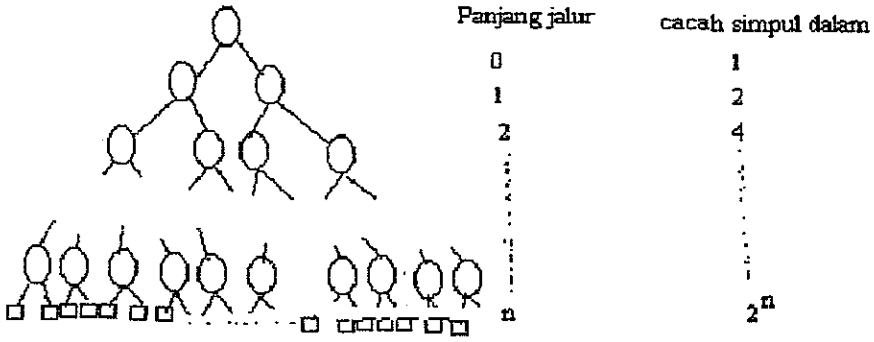
$I = 0 + 1 + 1 + 2 = 4$

$E = 2 + 2 + 2 + 3 + 3 = 12$ dan

$I = 0 + 1 = 1$

$E = 1 + 2 + 2 = 5$

Secara umum panjang jalur luar dapat dirumuskan sebagai $E = I + 2n$, dengan n adalah cacah simpul dalam sebuah pohon biner. Sebuah pohon AVL dengan cacah simpul n dapat diillustrasikan seperti gamabr 3.5.2 berikut ini :



Gambar 3.5.2 Pohon AVL

Dari gambar 3.5.2 dapat diperoleh bahwa :

$$I = (0 \times 1) + (1 \times 2) + (2 \times 4) + (3 \times 8) + \ldots + (n \times 2^n)$$

$$I = \Sigma\ {}^2\log i$$

$$= O(n\ {}^2\log n)$$

$$= O(n \log n)$$

$$E = I + 2n$$

$$= O(n \log n) + 2n$$

Kecepatan rata-rata algoritma pencarian sangat tergantung pada besaran I dan E. Rata-rata kecepatan pencarian yang berhasil menemukan data akan sama dengan I dibagi dengan jumlah simpul dalam. Sedangkan kecepatan rata-rata gagal akan sama dengan E dibagi dengan jumlah daun.

Dengan demikian pada pohon pencarian biner seimbang kecepatan rata-rata pencarian sukses maupun gagal mempunyai kompleksitas linear O(log n).

Teorema 3.5.1

Setiap pohon biner seimbang *(AVL trees)* dengan cacah simpul n akan mempunyai ketinggian kurang dari 1.44 log (n)

Bukti.

Ambil Wh sebagai himpunan yang terdiri atas pohon AVL dengan cacah simpul seminimal mungkin, w_h adalah cacah simpul salah satu pohon AVL pada W_h dengan $w_0 = 0$ dan $w_1 = 1$. Selanjutnya, ambil $T \in W_h$ dengan $h \geq 2$ yang mempunyai dua buah cabang yaitu T_L sebagai cabang kiri dan T_R sebagai cabang kanan. Apabila T mempunyai ketinggian h, maka salah satu cabangnya ketinggian h-1. Dengan tanpa mengurangi sifat umumnya diasumsikan bahwa T_R mempunyai ketinggian h-1. Karena T adalah pohon AVL dan cabang kanannya mempunyai ketinggian h-1, maka cabang kirinya (T_L) harus mempunyai ketinggian h-1 atau h-2. Tetapi karena di asumsikan mempunyai cacah simpul seminimal mungkin, sehingga T_L harus merupakan anggota dari W_{h-2}. Dengan demikian cacah simpul pohon AVL pada ketinggian h dapat dinyatakan sebagai relasi rekursif sebagai berikut:

$$w_h = w_{h-1} + w_{h-2} + 1 \qquad (3.5.1)$$

dengan $w_0 = 0$, $w_1 = 1$ dan diperoleh nilai w_h secara berturut-turut adalah 0,1, 2, 4, 7, 12, 20, ... Secara umum, w_h dapat dinyatakan pula sebagai :

$$w_h = F_{h+2} - 1 \qquad (3.5.2)$$

dengan F_h adalah relasi rekursif pohon fibonacci yang dinyatakan sebagai berikut :

$$F_h = F_{h-1} + F_{h-2} \qquad (3.5.3)$$

Dengan $F_0 = 0, F_1 = 1$ sehingga diperoleh nilai F_h secara berturut-turut adalah 0, 1, 1, 2, 3, 5, 8, 13, ... Dari persamaan (3. 5. 3) diperoleh penyelesaian relasi rekursif :

$$F_h = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^h - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^h \qquad (3.5.4)$$

Karena nilai $\left|\frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^h\right| < 1$ maka persamaan (3. 5. 4) menjadi

$F_h \approx \Phi^h/\sqrt{5}$, dengan $\Phi = \left(\frac{1+\sqrt{5}}{2}\right)$. Misalkan cacah simpul pada ketinggian h adalah n maka akan dipenuhi bahwa :

$$n \geq w_h > \Phi^{h+2}/\sqrt{5} - 1$$

$$(n+1) > \Phi^{h+2}/\sqrt{5}$$

$$\frac{\sqrt{5}(n+1)}{\Phi^2} > \Phi^h$$

$$\log \frac{\sqrt{5}(n+1)}{\Phi^2} > h \log \Phi$$

$$\frac{\log \frac{\sqrt{5}(n+1)}{\Phi^2}}{\log \Phi} > h$$

Jika $\frac{\sqrt{5}\,(n+1)}{\Phi^2} < n$ atau dengan kata lain jika $n > \sqrt{5}/\left(\Phi 2 - \sqrt{5} \right)$,

$n > 5.9$ atau $n \geq 6$ maka diperoleh bahwa

$$h < \frac{\log n}{\log \Phi}$$

$$h < 1.44 \log n$$

Jadi terbukti bahwa untuk $n \geq 6$ ketinggian pohon AVL kurang dari 1.44 log n.