Implementasi Pohon AVL sebagai Struktur Data Pohon Biner Terurut Seimbang

Timotius Nugroho Chandra - 13508002

Program Studi Teknik Informatika – Sekolah Teknik Elektro dan Informatika – Institut Teknologi Bandung
Jalan Ganesha 10 - Bandung 40132
e-mail: if18002@students.if.itb.ac.id

ABSTRAK

Makalah ini membahas tentang struktur data pohon AVL. Pohon AVL pada dasarnya merupakan pohon biner terurut seimbang biasa, namun pohon AVL memiliki kekhasan, yaitu dapat menyeimbangkan dirinya sendiri. Dalam sebuah pohon AVL, perbedaan tinggi upapohon kiri dan upapohon kanan dari suatu simpul maksimal 1. Pohon AVL ditemukan oleh G.M. Adelson –Velskii dan E.M. Landis. Nama AVL merupakan gabungan singkatan kedua nama mereka.

Keunggulan pohon AVL dibandingkan pohon biner terurut biasa adalah, primitif-primitif dasar untuk sebuah pohon, seperti pencarian, penyisipan, dan penghapusan dapat dilakukan dengan lebih efisien, dan dengan demikian, menjadikan pohon AVL suatu struktur data yang lebih efisien dibandingkan pohon biner terurut biasa. Dalam sebuah pohon AVL, pencarian, penyisipan, dan penghapusan sebuah elemen semuanya memiliki kompleksitas O(log n), dimana n merupakan jumlah simpul di dalam sebuah pohon AVL.

Kelebihan yang dimiliki oleh pohon AVL, disebabkan setiap kali dilakukan penyisipan dan penghapusan, selalu diikuti dengan satu atau lebih rotasi pohon untuk tetap menjaga keseimbangan tingginya.

Kata kunci: pohon, pohon biner, pohon biner terurut, pohon biner terurut seimbang, rotasi, AVL.

1. PENDAHULUAN

Konsep pohon merupakan konsep yang sangat penting dalam bidang informatika. Banyak informasi yang dapat direpresentasikan dengan menggunakan struktur pohon. Penggunaan pohon juga memungkinkan kita untuk memiliki cara yang akses yang unik terhadap elemen-elemennya.

Pohon juga sering digunakan dalam kehidupan kita sehari-hari. Silsilah keluarga, klasifikasi makhluk hidup, struktur organisasi, organisasi pertandingan, dan lain - lain.

Di dalam bidang informatika, struktur data dan konsep pohon seringkali digunakan untuk merepresentasikan pohon keputusan, pohon ekspresi aritmatika, atau untuk sekedar menyimpan elemen-elemen tertentu (kata, angka, dll)

Dalam penyimpanan elemen-elemen tertentu, struktur data pohon banyak diminati orang sebagai representasi dikarenakan struktur lojiknya yang khas dan dapat dimanfaatkan untuk mempercepat proses-proses yang sering dilakukan terhadap elemen-elemen yang disimpan tersebut. Proses-proses yang sering dilakukan tersebut adalah pencarian, penyisipan, dan penghapusan.

Contoh-contoh yang ada dalam kehidupan kita sehari-hari adalah penyimpanan buku-buku di perpustakaan, penyusunan kata-kata di dalam sebuah kamus, dll. Keuntungan menggunakan struktur data dan konsep pohon dalam merepresentasikan elemen-elemen persoalan di atas adalah waktu yang dibutuhkan untuk mencari, menambahkan, dan menghapus sebuah elemen dapat dilakukan dengan lebih cepat dan efisien.

Jenis-jenis pohon yang ada sekarang ini pun cukup beragam jenisnya. Jenis strktur pohon yang paling terkenal adalah pohon biner. Struktur data pohon biner menawarkan kemudahan dan kecepatan serta efisiensi dalam proses pencarian, penyisipan, dan penghapusan elemen-elemennya. Namun demikian, masih ada jenis pohon lainnya yang menawarkan kecepatan dan efisiensi yang lebih baik daripada pohon biner, yaitu pohon biner terurut seimbang.

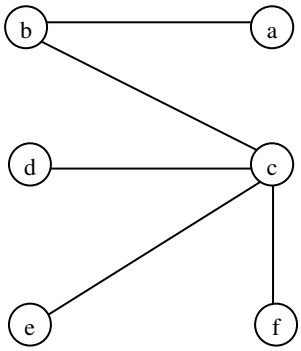
Makalah ini akan membahas salah satu implementasi yang digunakan untuk membangun struktur data pohon biner seimbang yang dapat menyeimbangkan dirinya sendiri, yaitu pohon AVL. Di sini akan dibahas primitif-primitif untuk membentuk suatu pohon AVL sehingga keseimbangan tingginya dapat tetap terjaga.

2. POHON[1]

Apabila terdapat sub-bab, maka judul dari tiap sub-bab ditulis dengan tipe huruf Times New Roman dengan ukuran 12pt, cetak tebal, serta penomoran dengan huruf (1.1, 1.2, 1.3,... 2.1, 2.2, ...).

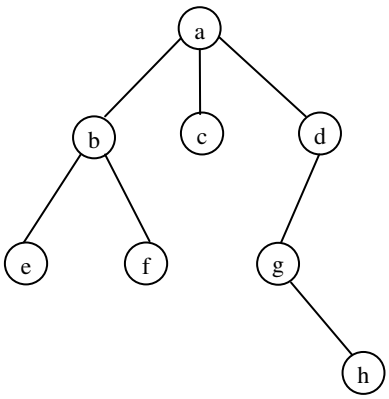
2.1. Definisi Pohon dan Pohon Berakar

Pada dasarnya, pohon adalah sebuah graf yang khusus. Definisi pohon adalah : graf tak-berarah terhubung yang tidak mengandung sirkuit.



Gambar 1. Contoh pohon

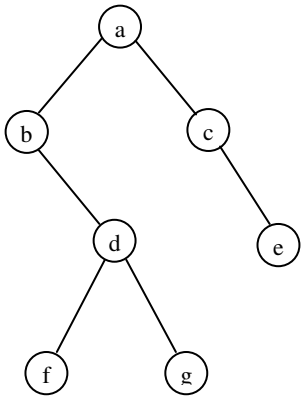
Pada kebanyakan penggunaan pohon di dunia informatika, suatu simpul tertentu dibedakan dari yang lain yang kemudian disebut sebagai akar (*root*). Jika sebuah simpul sudah ditetapkan sebagai akar, maka simpul-simpul lainnya dapat dikunjungi dengan melewati sisi-sisi pohon yang diberi arah (disebut sebagai anak / *children*). Pohon berakar dapat didefinisikan sebagai : pohon yang sebuah simpulnya diperlakukan sebagai akar dan sisi-sisinya diberi arah sehingga menjadi graf berarah.



Gambar 2. Contoh pohon berakar. Tanda panah pada sisi dibuang

2.2. Definisi Pohon Biner dan Pohon Biner Seimbang

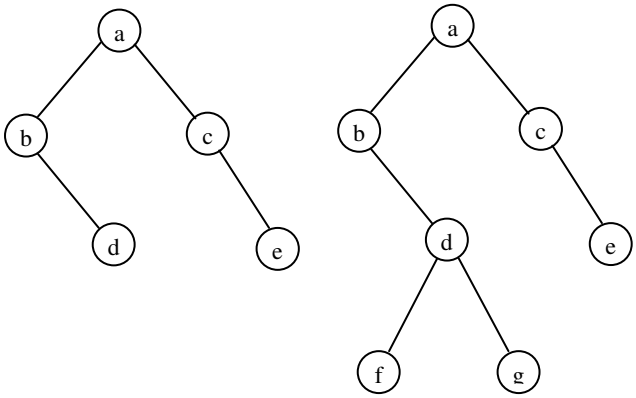
Pohon biner adalah suatu jenis pohon berakar yang setiap simpulnya memiliki maksimal 2 anak. Kedua anak pada pohon biner disebut sebagai anak kiri (*left child*) dan anak kanan (*right child*).



Gambar 3. Contoh pohon biner

Pohon biner juga dikategorikan sebagai **pohon terurut**. Dikatakan terurut karena urutan anak-anaknya penting. Hal ini terlihat jelas karena kedua anak pohon biner dibedakan menjadi anak kiri dan anak kanan.

Di berbagai macam penggunaan, pohon biner diinginkan memiliki perbedaan yang seimbang, yaitu perbedaan tinggi upapohon kiri dan upapohon kanan maksimum 1. Tinggi suatu pohon adalah lintasan terpanjang dari simpul tersebut ke daun. Pohon seperti ini disebut sebagai pohon biner seimbang.



Gambar 4. Contoh - contoh pohon biner seimbang

2.3 Pohon Biner Terurut Seimbang

Pohon biner terurut seimbang memiliki memenuhi sifat – sifat sebagai berikut :

- Semua nilai simpul upapohon kiri selalu lebih kecil dari nilai simpul tersebut
- Semua nilai simpul upapohon kanan selalu lebih besar dari nilai simpul tersebut

- Semua nilai pada setiap simpul berbeda
- Perbedaan tinggi upapohon kiri dan upapohon kanan setiap simpul maksimum 1.

Dalam sebuah pohon biner, primitif – primitif yang sering dipanggil adalah pencarian, penyisipan, dan penghapusan data.

Struktur data pohon biner terurut seimbang memiliki kekhasan, yaitu struktur data ini dapat menyeimbangkan dirinya sendiri, sehingga setiap saat perbedaan tinggi upapohon kiri dan upapohon kanannya selalu bernilai maksimum 1. Keseimbangan tersebut dapat tetap terjaga karena setiap kali primitif penyisipan dan penghapusan dipanggil, selalu diikuti dengan prosedur rotasi yang dilakukan jika ada suatu simpul yang menjadi tidak seimbang.

2.3. Tipe Data Abstrak dan Primitif Dasar Pohon Biner[2]

Di dalam merepresentasikan pohon biner agar dapat dikenali oleh komputer, perlu didefinisikan sebuah tipe data abstrak untuk pohon biner tersebut. Sebagai contoh, dipilih representasi pohon biner dengan elemennya adalah sebuah bilangan bulat (*integer*) sebagai berikut :

```
type infotype : int
type address : pointer to node
type BinTree : address
type node : <    info : infotye,
                 left : BinTree,
                 right : BinTree
            >
```

Representasi di atas dikenal sebagai representasi yang menggunakan *pointer*.

Primitif-primitif dasar untuk tipe data abstrak di atas adalah :

```
procedure MakeTree(input X : infotype, input L, R : BinTree, output P : BinTree)

function SearchElement(input X : infotype, input P : BinTree) -> boolean

procedure InsertElement(input X : infotype, input/output P : BinTree)

procedure DelElement(input/output P : BinTree, input X : infotype)
```

Primitif `MakeTree` digunakan untuk menghasilkan sebuah pohon biner P dengan L sebagai anak kiri dan R sebagai anak kanan. Fungsi `SearchElement` digunakan untuk mengetahui apakah ada elemen X di pohon biner P. Primitif `InsertElement` digunakan untuk melakukan penyisipan elemen X ke pohon biner P, sedangkan primitif `DelElement` berguna untuk menghapus suatu elemen X dari pohon biner P. Algoritma untuk tiap-tiap primitif tersebut di atas kurang lebih sebagai berikut :

`MakeTree : alokasi sebuah tempat untuk P dengan elemen X, jika alokasi berhasil isi anak kiri dari P dengan L dan anak kanan P dengan R`

`SearchElement : didefinisikan secara rekursif dengan`

basis :
`jika pohon biner P kosong, maka kembalikan false.`

rekurens :
`bergantung pada nilai X dan nilai akar P :`
`Jika X sama dengan akar P maka return true.`
`Jika X kurang dari akar P maka return SearchElement(X,Left(P))`
`Jika X lebih dari akar P maka return SearchElement(X,Right(P))`

`InsertElement : didefinisikan secara rekursif dengan`

basis :
`jika pohon biner P kosong, maka panggil MakeTree(X,Nil,Nil,P)`

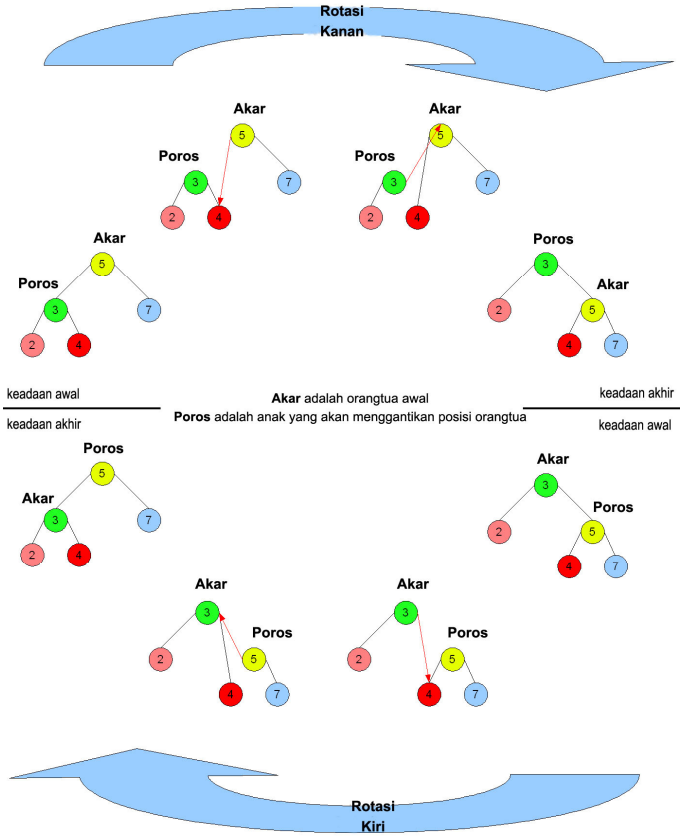
rekurens :
`bergantung pada nilai X dan nilai akar P :`
`Jika X sama dengan akar P maka tidak perlu melakukan apa-apa.`
`Jika X kurang dari akar P maka panggil SearchElement(X,Left(P))`
`Jika X lebih dari akar P maka panggil SearchElement(X,Right(P))`

`DelElement: Jika elemen yang akan dihapus adalah daun, maka langsung hapus elemen tersebut. Jika bukan daun, ganti elemen tersebut dengan elemen anak kirinya yang terbesar dan kemudian hapus elemen pengganti tersebut`

Primitif-primitif di atas dapat mempertahankan sifat dari pohon biner terurut, yaitu elemen di sebelah kiri akar selalu lebih kecil dan elemen di sebelah kanan akar selalu lebih besar. Namun demikian, primitif-primitif di atas tidak cukup untuk mempertahankan keseimbangan tinggi pohon biner terurut. Agar dapat tetap mempertahankan keseimbangan tinggi, setiap kali melakukan penyisipan dan penghapusan elemen, diperlukan suatu rotasi sehingga pohon hasil penambahan atau penghapusan tadi tetap terjaga keseimbangannya.

2.4. Rotasi Pohon Biner[3] dan Faktor Keseimbangan[4]

Rotasi pohon adalah suatu operasi pada pohon biner yang merubah struktur dari pohon biner tersebut tanpa merusak keterurutan dari elemen-elemennya. Rotasi pohon menaikkan sebuah simpul dan menurunkan sebuah simpul yang lain. Tujuan dari rotasi pohon adalah mereduksi tinggi dari pohon yang bersangkutan. Ada 2 macam rotasi : **rotasi kiri** dan **rotasi kanan**.



Gambar 5. Ilustrasi Rotasi Kiri dan Rotasi Kanan pada Pohon Biner

Kode-semu untuk rotasi kanan :

```
Poros = Left(Akar)
Left(Akar) = Right(Poros)
Right(Poros) = Akar
Akar = Poros
```

Kode-semu untuk rotasi kiri :

```
Poros = Right(Akar)
Right(Akar) = Left(Poros)
Left(Poros) = Akar
Akar = Poros
```

Definisi dari faktor keseimbangan dari suatu simpul adalah : tinggi upapohon kiri dikurangi tinggi upapohon kanan.

Sebuah simpul dengan faktor keseimbangan -1, 0, +1 dikatakan seimbang tingginya, sedangkan simpul dengan faktor keseimbangan selain yang disebut di atas dikatakan tidak seimbang.

Seperti yang sudah dijelaskan di upabab 2.3, pada struktur data pohon biner terurut seimbang, setiap kali melakukan penyisipan dan penghapusan elemen, selalu diikuti dengan rotasi untuk mempertahankan keseimbangan pohon.

Setelah menyisipkan sebuah elemen, cek setiap leluhur dari simpul yang baru saja disisipkan tersebut sampai menemukan simpul yang tidak seimbang atau mencapai simpul paling atas (simpul yang tidak memiliki orangtua), jika selama pencarian ada simpul yang faktor keseimbangannya menjadi 2 atau -2, maka diperlukan rotasi dan kemudian peninjauan dapat dihentikan (tidak perlu ditinjau sampai ke simpul paling atas). Jika sudah mencapai simpul teratas dan faktor keseimbangan tetap -1, 0, atau +1, maka peninjauan selesai dan tidak diperlukan rotasi sama sekali.

Dalam melakukan rotasi pohon biner, ada 4 kasus yang perlu diperhatikan. Misalkan P adalah akar dari upapohon yang tidak seimbang, R adalah anak kanan dari P, dan L anak kiri dari P. Keempat kasus tersebut adalah :

- Kasus **Kanan-Kanan** dan **Kanan-Kiri** : jika faktor keseimbangan dari P adalah -2, maka tinggi upapohon kanan melebihi tinggi upapohon kiri, sehingga faktor keseimbangan dari R (anak kanan) harus dicek. Jika faktor keseimbangan dari R adalah -1, sebuah **rotasi kiri** perlu dilakukan dengan P sebagai akarnya. Jika faktor keseimbangan dari R adalah +1, maka **rotasi kiri ganda** perlu dilakukan. Rotasi pertama adalah **rotasi kanan** dengan R sebagai akarnya, kedua adalah **rotasi kiri** dengan P sebagai akar
- Kasus **Kiri-Kiri** dan **Kiri-Kanan** : jika faktor keseimbangan dari P adalah +2, maka tinggi upapohon kiri melebihi tinggi upapohon kanan, sehingga faktor kesembangan dari L (anak kiri) harus dicek. Jika faktor keseimbangan dari L adalah +1, sebuah **rotasi kanan** perlu dilakukan dengan P sebagai akarnya. Jika faktor keseimbangan dari L adalah -1, maka **rotasi kanan ganda** perlu dilakukan. Rotasi pertama adalah **rotasi kiri** dengan L sebagai akarnya, kedua adalah **rotasi kanan** dengan P sebagai akar.

Dalam melakukan penghapusan sebuah elemen, maka sama seperti setelah penyisipan, perlu ditinjau ulang apakah tinggi sebuah pohon seimbang atau tidak. Namun tidak seperti peninjauan yang dilakukan setelah penyisipan, peninjauan yang dilakukan setelah penghapusan harus dilakukan sampai ke simpul paling ujung atas, jadi dapat dimungkinkan, setelah penghapusan dilakukan lebih dari 2 kali rotasi.

3. Analisis Perbandingan Kompleksitas Algoritma Primitif Dasar pada Struktur Data Pohon Biner Terurut Seimbang dan Tidak Seimbang

Kompleksitas algoritma primitif-primitif dasar pada struktur data pohon biner teurut seimbang maupun tidak seimbang dapat dilihat dari kompleksitas algoritma pencariannya saja, karena pada dasarnya primitif penyisipan dan penghapusan juga secara tidak langsung dilakukan pencarian.

Pada kasus terbaik, nilai elemen yang akan dicari ada di simpul paling atas (akar paling ujung). Sehingga pada kasus terbaik, algoritma pencarian hanya dipanggil sekali saja. Jadi pada kasus terbaik, kompleksitas algoritma pencarian elemen adalah **O(1)**. Pada pohon biner terurut tidak seimbang, kompleksitasnya juga sama **O(1)**.

Pada kasus rata-rata, nilai elemen yang akan dicari berada di tengah-tengah pohon. Misal **n** menyatakan jumlah simpul yang ada dalam sebuah pohon biner terurut seimbang, maka tinggi maksimum dari pohon tersebut adalah **log(n)**. (Pembulatan ke atas). Dengan demikian, pada kasus rata-rata di mana elemen yang dicari ada di tengah-tengah pohon, maka algoritma pencarian akan dipanggil (secara rekursif) sebanyak : **½ log(n)** kali, sehingga kompleksitas algoritma pencarian pada kasus rata-rata adalah **O(log n)**. Pada pohon biner terurut tidak seimbang, kasus rata-rata berarti pohonnya hampir mendekati pohon seimbang, sehingga kompleksitasnya juga sama yaitu **O(log n)**.

Pada kasus terburuk, maka nilai elemen yang dicari ada di daun (simpul terbawah). Maka algoritma pencarian akan dipanggil sebanyak : **log(n)** kali, sehingga kompleksitas algoritma pencarian pada kasus terburuk adalah **O(log n)**.

Bila dibandingkan dengan pohon biner terurut biasa (tidak seimbang). Maka pada kasus terburuk, elemen yang akan dicari ada di daun dan pohon biner terurutnya berbentuk pohon yang condong ke kanan atau ke kiri. Dengan demikian, tinggi pohon tersebut adalah **n**, sehingga algoritma pencarian akan dipanggil sebanyak **n** kali pula, yang mengakibatkan kompleksitas algoritma pencarian data pada pohon biner terurut yang tidak seimbang adalah **O(n)** pada kasus terburuk.

Tabel 1. Perbandingan Kompleksitas Algoritma Pohon Biner Terurut Seimbang dan Tidak Seimbang

Pohon/Kasus	Terbaik	Rata-rata	Terburuk
Seimbang	O(1)	O(log n)	O(log n)
Tidak Seimbang	O(1)	O(log n)	O(n)

Terlihat bahwa struktur data pohon biner terurut seimbang memiliki kompleksitas yang lebih baik pada kasus terburuk, yaitu O(log n).

4. Kesimpulan

Untuk membangun sebuah pohon biner terurut seimbang dan terus menjaga keseimbangannya, diperlukan suatu prosedur untuk mempertahankannya, yaitu rotasi pohon dengan terlebih dahulu dilakukan penghitungan faktor keseimbangan sebuah simpul.

Pohon biner terurut seimbang memiliki kelebihan dibanding pohon biner terurut biasa, yaitu kompleksitas algoritma pencarian, penyisipan, dan penghapusan data semuanya memiliki kompleksitas O(log n) baik pada kasus rata-rata maupun kasus terburuk, sehingga strktur data pohon biner terurut seimbang dapat dijadikan alternatif jika seseorang menginginkan kecepatan komputasi yang lebih baik.

REFERENSI

[1] Munir, Rinaldi. "Diktat Kuliah IF2091 Struktur Diskrit", edisi keempat. Program Studi Teknik Informatika, STEI ITB. 2003.

[2] Liem, Inggriani. "Draft Diktat Struktur Data", edisi 2008. Program Studi Teknik Informatika, STEI ITB. 2008.

[3] Wikipedia. The Free Encyclopedia. 2009. http://en.wikipedia.org/wiki/AVL_Tree. Tanggal akses : 19 Desember 2009, pukul 20:14 WIB.

[4] Wikipedia. The Free Encyclopedia. 2009. http://en.wikipedia.org/wiki/Tree_rotation. Tanggal akses : 19 Desember 2009, pukul 20:14 WIB.