Cara Kerja B-tree dan Aplikasinya

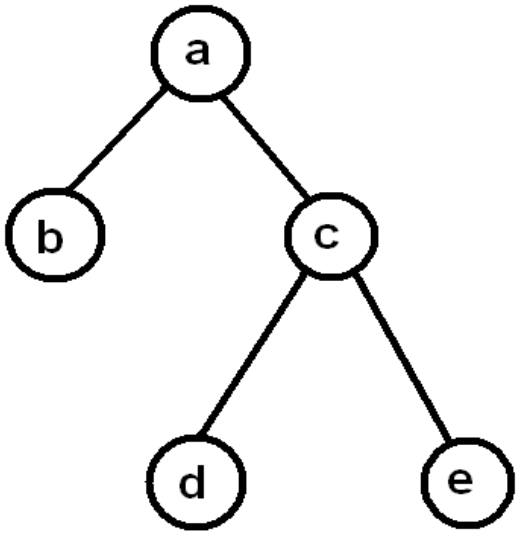
Paskasius Wahyu Wibisono - 13510085
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
paskasius.wahyu@students.itb.ac.id

Abstrak – B-tree adalah implementasi khusus dari struktur data pohon. B-tree memungkinkan data tetap terurut. Proses penambahan data, penghapusan data, dan pencarian juga dapat dilakukan dengan efisien. B-tree juga bagus dalam membagi data ke beberapa media penyimpanan berbeda, sehingga efektif untuk digunakan pada data berukuran besar. Di dalam makalah ini akan dijelaskan cara kerja B-tree, variasi, dan berbagai aplikasinya.

Kata kunci – pohon, b-tree, basis data, berkas sistem.

I. PENDAHULUAN

Dalam teori graf, pohon adalah graf tak berarah di mana dua simpul acak hanya terhubung oleh tepat satu lintasan. Dengan kata lain, pohon adalah graf terhubung yang tidak memiliki siklus. Graf tak terhubung yang semua elemennya adalah pohon disebut sebagai hutan. Simpul 'teratas' pada sebuah pohon berakar, yaitu simpul yang tidak memiliki simpul orang tua disebut juga sebagai simpul akar. Sementara simpul yang tidak memiliki simpul anak disebut juga sebagai simpul daun. Simpul dalam adalah sebutan untuk simpul yang bukan merupakan simpul akar ataupun simpul daun.



Gambar 1. Ilustrasi pohon berakar

Pada gambar di atas, dapat kita lihat beberapa elemen dari sebuah pohon berakar

- b dan c adalah anak dari a
- d dan e adalah anak dari c
- a adalah simpul akar
- b, d, dan e adlah simpul daun
- c adalah simpul dalam

Pohon sering kali dipakai dalam kehidupan sehari-hari maupun dalam bidang informatika. Dalam kehidupan sehari-hari, pohon antara lain dipakai untuk menggambarkan silsilah keluarga, struktur kepengurusan organisasi, pengambilan keputusan, penjadwalan pertandingan olahraga, dan lain-lain.

Sementara dalam bidang informatika, pohon digunakan sebagai sebuah struktur data. Pohon digunakan karena memiliki beberapa kelebihan dibandingkan struktur data lain. Kelebihan-kelebihan itu antara lain dalam algoritma pencarian dan pengurutan, serta menjadi basis untuk struktur data lain seperti set, multiset, dan larik asosiatif.

Pohon memiliki banyak variasi, antara lain B-tree, pohon pencarian biner, heap, dan trie. Variansi yang beraneka ragam tersebut dikarenakan tiap permasalahan membutuhkan struktur data yang mampu mengolah data dalam ukuran, waktu, dan efisiensi yang berbeda-beda.

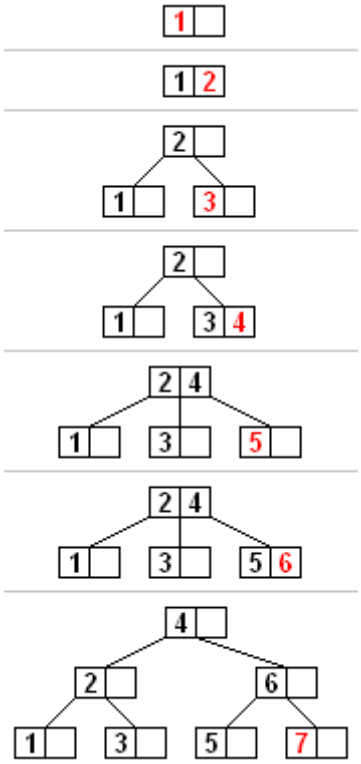
Makalah ini akan membahas implementasi B-tree dan primitif-primitif yang dibutuhkan, serta implementasinya, terutama dalam basis data dan berkas sistem.

II. B-TREE

B-tree diciptakan oleh Rudolf Bayer dan Ed McCreight ketika mereka bekerja di Laboratorium Riset Boeing pada tahun 1971. *B-tree* adalah pohon seimbang yang dioptimasi khusus untuk situasi di mana sebagian pohon harus disimpan di media yang berbeda dengan bagian pohon yang lain. Penggunaan *B-tree* meminimalisir jumlah akses ke media penyimpanan. Karena itulah *B-tree* digunakan dalam berkas sistem, di mana data yang diproses sering kali tidak dapat dimuat dalam *Random Access Memory* (RAM), sehingga harus disimpan di media penyimpanan sekunder. *B-tree* juga mampu melakukan pemindahan data berukuran besar secara efisien. Hal ini menyebabkan *B-tree* banyak dipakai dalam basis data yang berukuran besar.

B-tree adalah pohon yang menjaga data tetap terurut, dan dapat melakukan operasi akses, pencarian, penambahan, dan penghapusan dalam waktu logaritmik. Secara struktur, *B-tree* merupakan generalisasi dari pohon pencarian biner. Bedanya, *B-tree* bersifat *multiway*, sehingga tiap simpul dapat mempunyai lebih dari dua anak. Biasanya sebuah *B-tree* mempunyai batas atas dan batas bawah jumlah simpul anak. Batsan-batasan tersebut berbeda-beda dan disesuaikan dengan kebutuhan implementasi tersebut. Misalnya dalam *2-3 tree*, setiap simpul hanya boleh memiliki dua atau tiga anak. Simpul akar adalah pengecualian, karena simpul tersebut tidak mempunyai batas bawah jumlah simpul anak.

Setiap simpul dalam pada *B-tree* memiliki sejumlah kunci. Jumlah kunci adalah bilangan di antara d dan $2d$. Jika sebuah simpul dalam mempunyai $2d$ kunci, maka penambahan kunci pada simpul tersebut dapat dilakukan dengan membagi simpul tersebut menjadi dua simpul dengan d kunci, dan menambahkan satu kunci ke simpul orang tua. Proses ini akan menghasilkan dua simpul dengan jumlah kunci minimum. Sementara itu jika dua buah simpul dalam masing-masing memiliki d kunci, maka penghapusan kunci dari salah satu simpul tersebut dapat dilakukan dengan menggabungkan kedua simpul tersebut dan menambahkan satu kunci dari simpul orang tua. Sehingga terbentuklah sebuah simpul penuh dengan $2d$ kunci.



Gambar 2. Penambahan elemen pada sebuah *B-tree*
Sumber :
http://en.wikipedia.org/wiki/File:B_tree_insertion_example.png

Kompleksitas waktu untuk operasi-operasi pada *B-tree* adalah sebagai berikut :

	Rata-rata	Kasus terburuk
Ruang	O(n)	O(n)
Pencarian	O(log n)	O(log n)
Penambahan	O(log n)	O(log n)
Penghapusan	O(log n)	O(log n)

Jumlah simpul anak dari sebuah simpul pada *B-tree* adalah jumlah kunci pada simpul tersebut ditambah satu. Sebuah *B-tree* dideskripsikan dengan parameter (d+1) – (2d+1), atau cukup (2d+1). Jika m adalah jumlah simpul anak maksimum dari sebuah simpul, maka kasus terbaik untuk tinggi sebuah *B-tree* adalah :

$$\log_m(n)$$

sementara kasus terburuknya adalah :

$$\log_{m/2}(n)$$

Sebuah *B-tree* menjaga dirinya sendiri agar tetap seimbang dengan mengharuskan tiap daun untuk berada di tingkat yang sama. Tinggi *B-tree* akan bertambah seiring bertambahnya jumlah kunci. Namun, struktur B-tree membuat penambahan tingkat pada *B-tree* jarang terjadi bila dibandingkan dengan jenis pohon yang lain. Imbasnya, jumlah akses simpul berkurang dan penyeimbangan ulang lebih jarang terjadi. Jumlah simpul anak yang besar akan meningkatkan performa dari sebuah *B-tree*, meskipun tingkat kerumitannya juga bertambah.

Karena kebutuhan implementasi yang berbeda-beda, maka *B-tree* sendiri memiliki beberapa variansi, di samping beberapa jenis pohon yang sebenarnya identik dengan *B-tree*.

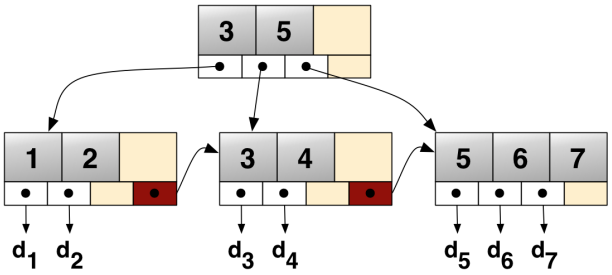
III. VARIASI B-TREE

A. B+ tree

B+ tree adalah variasi khusus dari *B-tree* di mana kunci pada tiap simpul bukan merupakan isi data dari pohon itu sendiri, melainkan hanya sebuah penunjuk ke isi data yang sebenarnya (*record*). Semua *record* disimpan di simpul daun, sementara simpul-simpul dalam hanya berisi kunci. Tidak seperti *B-tree*, berapapun derajat sebuah *B+ tree*, simpul akarnya mempunyai jumlah simpul anak minimum dua.

Fungsi utama dari *B+ tree* adalah untuk menyimpan dan mengakses secara efisien data pada penyimpanan berorientasi blok, teruatama berkas sistem. Hal ini dikarenakan *B+ tree* memiliki *fanout* yang tinggi, tidak seperti pohon pencarian biner. *B+ tree* digunakan dalam pengindeksan metadata pada berkas sistem NTFS, ReiserFS, NSS, XFS, dan JFS. *B+ tree* juga digunakan

pada sistem manajemen basis data relasional, seperti IBM DB2, Informix, Microsoft SQL Server, Oracle 8, Sybase ASE, dan SQLite. Selain itu, *B+ tree* juga didukung sistem manajemen basis data bernilai-kunci, seperti CouchDB dan Tokyo Cabinet.



Gambar 3. Ilustrasi *B+ tree*
Sumber : http://en.wikipedia.org/wiki/File:Bplustree.png

Untuk sebuah *B+ tree* dengan derajat *b* dan *h* tingkat, berlaku sifat-sifat berikut :

- Jumlah *record* maksimum,
$$n_{max} = b^h - b^{(h-1)}$$
- jumlah record minimum,
$$n_{min} = 2\left\lceil \frac{b}{2} \right\rceil^{(h-1)}$$
- jumlah kunci minimun,
$$n_{kmin} = 2\left\lceil \frac{b}{2} \right\rceil^{h} - 1$$
- Ruang yang dibutuhkan untuk menyipan pohon adalah *O(n)*
- Penambahan sebuah *record* memerlukan $O(\log_b n)$ operasi
- Pencarian sebuah *record* memerlukan $O(\log_b n)$ operasi
- Penghapusan sebuah *record* memerlukan $O(\log_b n)$ operasi

B. B-tree*

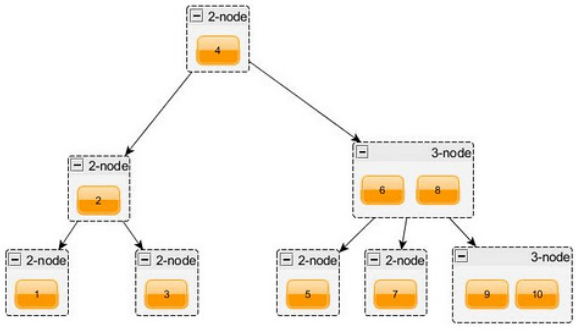
B-tree* menyeimbangkan simpul-simpul yang bertetangga untuk menjaga agar simpul-simpul dalam lebih padat. Pohon ini mengharuskan simpul non-akar 2/3 penuh. Untuk melakukannya, ketika sebuah kunci ditambahkan ke sebuah simpul yang sudah penuh, simpul tersebut tidak dibagi menjadi dua, melainkan beberapa kuncinya dipindahkan ke simpul tetangganya. Jika baik simpul tersebut maupun tetangganya sudah penuh, maka kedua simpul tersebut dibagi menjadi tiga. *B*-tree* digunakan dalam berkas sistem HFS dan Reiser4.

C. UB-tree

UB-tree adalah pohon seimbang yang didesain untuk menyimpan dam mengambil data multi-dimensi secara efisien. Pada dasarnya, *UB-tree* adalah sebuah *B-tree* di mana data disimpan berdasarkan kurva *Z-order*. Algoritma penambahan dan penghapusan pada *UB-tree* sama dengan *B-tree* biasa. Namun untuk melakukan pencarian pada data multi-dimensi dibutuhkan sebuah algoritma dengan kompleksitas eksponensial, yang menjadi kelemahan utama dari struktur data ini.

D. 2-3 tree

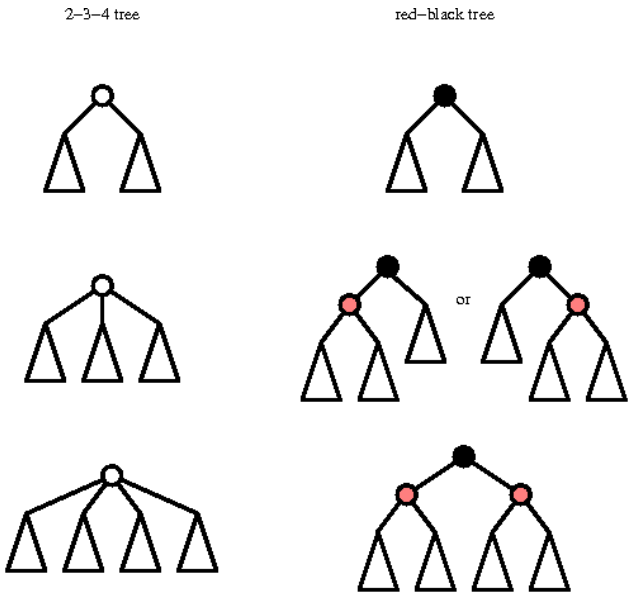
2-3 tree adalah *B-tree* di mana tiap simpul internal mempunyai dua atau tiga simpul anak. *2-3 tree* adalah *B-tree* yang sederhana, sehingga dapat digunakan sebagai pengganti pohon biner biasa dalam algoritma-algoritma sederhana. *2-3 tree* merupakan struktur data yang ekivalen dengan *AA tree*.



Gambar 4. Ilustrasi *2-3 tree*
Sumber : http://v2matveev.blogspot.com/2010/03/data-structures-2-3-tree.html

E. 2-3-4 tree

2-3-4 tree adalah *B-tree* di mana tiap simpul internal mempunyai minimal dua dan maksimal empat simpul anak. *2-3-4 tree* biasanya digunakan untuk mengimplementasikan larik asosiatif. Pohon ini mempunyai properti dan algoritma penambahan, pengurangan, serta pencarian yang sama dengan *B-tree* pada umumnya. *2-3-4 tree* adalah struktur data yang ekivalen dengan *red-black tree*. Namun karena *red-black tree* lebih sederhana untuk diimplementasikan, struktur data tersebut lebih sering digunakan daripada *2-3-4 tree*.



Gambar 5. Perbandingan antara *2-3-4 tree* dengan *red-black tree*

Sumber :
http://theory.cs.uvic.ca/inf/tree/RedBlackTree.html

F. (a,b)-tree

(a,b)-tree adalah sebuah pohon pencarian. Setiap simpul dalam pada *(a,b)-tree* mempunyai antara *a* dan *b* anak, di mana *a* dan *b* adalah bilangan bulat sehingga $2 \leqslant a \leqslant (b+1)/2$. Simpul akar bisa mempunyai minimal 0 anak, dan maksimal *b* anak. Semua lintasan dari akar ke daun mempunyai panjang yang sama. Sebuah *(a,b)-tree* menjadi sebuah *B-tree* bila persamaannya diganti menjadi $1 \leqslant a \leqslant (b+1)/2$.

G. Dancing tree

Dancing tree diciptakan oleh Hans Reiser untuk digunakan pada berkas sistem Reiser4. *Dancing tree* mampu menyeimbangkan dirinya sendiri, tapi hal tersebut hanya dilakukan saat membilas ke media penyimpanan., baik karena batasan memori ataupun karena proses pemindahan data yang sudah selesai. Hal ini dilakukan untuk mempercepat operasi sistem dengan menunda optimisasi pohon dan hanya menulis ke media penyimpanan saat diperlukan. Hal tersebut disebabkan karena menulis ke media penyimpanan ribuan kali lebih lambat dibandingkan menulis ke memori. Karena optimisasi menjadi jauh lebih jarang dibandingkan struktur data pohon lain, optimisasi yang dilakukan dapat dijadikan lebih ekstensif dari sebelumnya.

Sifat dari *Dancing tree* membuatnya cocok untuk digunakan pada media dengan kecepatan penulisan rendah., karena hasil akhir selalu seimbang namun tanpa proses penulisan mid-proses, sehingga mengurangi proses penambahan dan pengurangan simpul. Kelemahan dari struktur data ini dibandingkan pohon biasa adalah kesulitan untuk menyelamatkan data yang rusak akibat *shutdown* mendadak atau penulisan data yang terhenti sebelum selesai. Salah satu cara untuk mengatasi hal ini adalah dengan membuat *log* penulisan data atau mengembangkan algoritma untuk mendeteksi berkas mana saja yang telah berhasil ditransfer secara sempurna.

H. Htree

Htree adalah pohon yang mirip dengan B-tree yang digunakan untuk mengindeks direktori. Tinggi *Htree* selalu konstan, yaitu satu atau dua tingkat. *Htree* memiliki faktor *fanout* yang tinggi, menggunakan nama berkas sebagai *hash*, dan tidak membutuhkan penyeimbangan. *Htree* digunakan pada berkas sistem Linux ext3 dan ext4. *Htree* ditambahkan pada kernel Linux mulai versi 2.5.40. Sebenarnya struktur data *Btree* dan *B*-tree* juga dapat digunakan untuk mengindeks direktori. Namun kedua struktur data tersebut membutuhkan algoritma penyeimbangan ulang, sehingga kalah efisien dibandingkan dengan *Htree,* terutama untuk data yang berukuran besar.

IV. APLIKASI B-TREE

A. Basis Data

Dalam mengaplikasikan suatu struktur data ke dalam sebuah basis data, ada beberapa hal yang harus diperhatikan. Hal-hal tersebut antara adalah :

- Lama pencarian sebuah elemen. Untuk data yang sangat besar, pencarian bisa memakan waktu yang lama. Waktu pencarian bisa menjadi lebih lama lagi apabila data tidak disimpan secara terurut, dan berada dalam media penyimpanan dengan kecepatan tulis lambat.
- Pengindeksan data. Dengan memberi indeks pada blok-blok tertentu data, kita dapat memangkas waktu pencarian dengan hanya mencari di blok tertentu, dan bukannya di seluruh data.
- Penghapusan dan Penambahan data. Kedua operasi tersebut membuat manajemen data dan indeksnya menjadi lebih rumit. Penambahan data bisa berbahaya karena harus mengalokasikan tempat untuk data yang baru, yang berarti harus menggeser data-data yang sudah ada sebelumnya. Salah satu cara mengatasinya adalah dengan sengaja membiarkan sebagian memori tetap kosong, dan dipakai hanya jika ingin menambahkan elemen baru.

B-tree menangani semua masalah di atas dengan cara-cara berikut :

- Menjaga data tetap terurut untuk akses sekuensial.
- Menggunakan indeks hierarki untuk meminimalisir akses ke media penyimpanan.

- Menggunakan blok penuh-parsial untuk mempercepat penambahan dan penghapusan data.
- Indeks disesuaikan secara elegan menggunakan algoritma rekursif.
- Meminimalisir proses yang terbuang dengan memastikan semua simpul setidaknya setengah penuh.

Karena dapat mengatasi masalah-masalah tersebut itulah maka *B-tree* kerap dipakai dalam sistem manajemen basis data. Kelebihan lain dari *B-tree* adalah kemampuan untuk menangani operasi penambahan dan penghapusan data dalam jumlah yang tak terhingga, selama masih ada tempat di media penyimpanan. Variasi *B-tree* yang paling banyak digunakan dalam basis data adlah *B+ tree*.

B. Berkas Sistem

Selain dalam basis data, *B-tree* juga digunakan pada berkas sistem. *B-tree* digunakan karena memungkinkan akses acak ke blok mana pun pada sebuah berkas tertentu. Selain itu, *B-tree* juga mengatasi masalah mengubah alama blok berkas menjadi alamat blok fisik.*B-tree* digunakan dalam berkas sistem HFS dari Apple, NTFS dari Microsoft, dan beberapa berkas sistem Linux. Variasi dari *B-tree* yang juga banyak dipakai dalam berkas sistem adalah *B*-tree*. Selain itu, beberapa sistem operasi seperti TOPS-20 dan TENEX menggunakan struktur data serupa *B-tree* di dalamnya.

V. KESIMPULAN

B-tree adalah struktur data yang menjaga data tetap terurut dam memungkinkan operasi pencarian, penambahan, dan penghapusan dalamwaktu logaritmik. *B-tree* sangat sesuai untuk dipakai dalam basis data dan berkas sistem karena mampu menangani data yang berukuran sangat besar dan dapat membagi data ke dua media penyimpanan.

REFERENSI

[1] "B-Trees", http://cis.stvincent.edu/html/tutorials/swd/btree/btree.html, diakses pada tanggal 10 Desember 2011

[2] "B-Trees: Balanced Tree Data Structures", http://www.bluerwhite.org/btree/, diakses pada tanggal 10 Desember 2011

[3] "2-3 tree", http://www.en.wikipedia.org/wiki/2-3_tree, diakses pada tanggal 10 Desember 2011

[4] "2-3-4 tree", http://www.en.wikipedia.org/wiki/2-3-4_tree, diakses pada tanggal 10 Desember 2011

[5] "B-tree", http://www.en.wikipedia.org/wiki/B-tree, diakses pada tanggal 10 Desember 2011

[6] "B+ tree", http://www.en.wikipedia.org/wiki/B+_tree, diakses pada tanggal 10 Desember 2011

[7] "Dancing tree", http://www.en.wikipedia.org/wiki/Dancing_tree, diakses pad tanggal 10 Desember 2011

[8] "Htree", http://www.en.wikipedia.org/wiki/Htree, diakses pada tanggal 10 Desember 2011

[9] "UB-tree", http://www.en.wikipedia.org/wiki/UB-tree, diakses pada tanggal 10 Desember 2011

[10] "(a,b)-tree", http://www.en.wikipedia.org/wiki/(a,b)-tree, diakses pada tanggal 10 Desember 2011

[11] "A Directory Index for Ext2", http://www.linuxshowcase.org/2001/full_papers/phillips/phillips_html/index.html, diakses pada tanggal 10 Desember 2011

[12] "Tree (graph theory)", http://www.sccs.swarthmore.edu/users/08/ajb/tmve/wiki100k/docs/Tree_(graph_theory).html , diakses pada tanggal 10 Desember 2011

[13] "(a,b)-tree", http://xlinux.nist.gov/dads//HTML/abtree.html, diakses pada tanggal 10 Desember 2011

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, 11 Desember 2011

ttd

Paskasius Wahyu Wibisono
13510085