Penerapan Pohon Untuk Memanipulasi dan Meritrieve Data

Calvin Irwan – NIM 13507010
Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika
Intstitut Teknologi Bandung
email calvin_alonso@yahoo.com

ABSTRAK

Stuktur pohon memiliki sifat-sifat tertentu yang khas. Berbagai sifat pohon tersebut memungkinkan penggunaan struktur pohon yang luas yang mencakup pada berbagai lapangan kehidupan.

Penggunaan struktur pohon tersebut antara lain dalam memodelkan permasalahan yang biasa ditemui dalam bidang informatika. Dengan pemodelan masalah dengan menggunakan struktur pohon manipulasi informasi dan penyelesaian masalah akan menjadi lebih mudah.

Salah satu penerapan pohon misalnya dalam bidang informatika. Makalah ini membahas tentang studi dan penggunaan struktur pohon untuk memodelkan dan memanipulasi data dan informasi dalam bidang informatika yaitu tentang salah satu metode pencarian yang memnfaatkan algoritma rekursif dari tree dan juga sering dipakai saat ini, yaitu Binary Search Tree BST beserta teori – teori dasarnya..

Kata kunci: BST, Searching, Pohon (Tree).

1. PENDAHULUAN

Sejak masuk ke abad 21, internet mulai diminati orang di seluruh penjuru dunia. Hal tersebut ditunjang oleh berbagai hal: kemajuan teknologi komputasi, penemuan-penemuan penting seperti semi konduktor ataupun transistor, dan rasa ingin tahu manusia untuk terus maju mengembangkan ilmu pengetahuannya.

Kita berada di era informasi dimana saat ini semua informasi yang kita butuhkan sudah tersedia di dunia maya. Informasi disajikan dalam bentuk data. Umumnya data tersebut dibuat dalam suatu paket. Dalam jaringan, dikenal yang namanya OSI Layer yaitu tahapan-tahapan dalam pengiriman data. OSI Layer merupakan model ideal dari koneksi logis yang harus terjadi agar komunikasi data dalam jaringan dapat berlangsung.

Dalam prakteknya perpindahan informasi data berlangsung dengan sangat cepat, namun tak dapat disangkal jika segala sesuatu pasti ada batasnya. Jika data sangat besar pemrosesan akan semakin rumit dan membutuhkan waktu yang lebih lama.

Dalam kasus pencarian data misalnya, jika data yang tersedia adalah semua komputer yang ada di seluruh dunia, maka jika kita harus mencari satu persatu masuk ke komputer tersebut, tidak dapat dibayangkan berapa lama Anda akan menemukan data tersebut.

Tidak semua prosedur harus dilakukan secara brute force yang memiliki pengertian metode menemukan data dari semua kemungkinan yang ada. Misalnya mesin pencari data, mesin pelacak sandi lewat, atau dalam kehidupan seharihari:

Anda mencari buku di rak perpustakaan yang tidak terorganisir dengan baik dengan mengecek satu persatu rak yang ada. Oleh karena itu diperlukan metode yang dapat meningkatkan kemudahan menemukan data yang dicari. Ada dua hal yang dapat dilakukakan. Pertama manajemen penyimpanan data harus dilakukan dengan baik. Misalnya pemberian label pada setiap buku di perpustakaan lalu menyimpannya dalam katalog buku atau komputer, pemberian kata kunci pada artikel di blog Anda. Kedua, dengan mengoptimalkan metode atau media pencarian data yang digunakan.

Peningkatan penggunaan internet juga tidak terlepas dari keberadaan media untuk menemukan data. Mesin pencari data atau yang sering dikenali oleh kita sebagai search engine banyak membantu pengguna untuk menemukan data dari seluruh belahan dunia. Google, Astalavista, dan Yahoo, adalah salah satu contohnya.

Apa yang mesin pencari lakukan. Pertama melakukan manajemen dalam menerima masukan data dengan kaidah tertentu. Kedua mengoptimalkan algoritma pencarian data. Umumnya jika data sudah ditata dengan rapi dalam media penyimpanan kita, maka semakin mudah kita memperoleh data tersebut.

Dalam kasus ini metode yang akan kita bahas adalah metode pencarian dengan pohon biner. Metode ini sangat

terkenal di dunia struktur data. Kemudahan dan kecepatan ditawarkan oleh metode ini. Seperti yang telah disampaikan sebelumnya bahwa selain metode pencarian data, manajemen memasukkan atau mengeluarkan data juga diperlukan agar pencaria data semakin optimal.

2. Pohon

Pohon adalah graf yang tak berarah , terhubung dan juga tidak membentuk sebuah sirkuit.

Struktur pohon adalah struktur yang sangat penting dalam memodelkan masalah terutama di bidang informatika, yang memungkinkan kita untuk:

- Mengorganisasi informasi berdasarkan suatu struktur logik.
- Memungkinkan cara akses yang khusus terhadap suatu elemen.

Berikut ini beberapa terminologi yang akan digunakan di dalam makalah ini.

1. Orang Tua (Parent)

   Sebuah simpul dikatakan sebagai orang tua apa bila simpul tersebut memiliki simpul lain yang terhubung dengannya dan simpul yang terhubung tersebut letaknya berada dibawah simpul orang tua.
   Pada Gambar 1.1 simpul B adalah orang tua bagi simpul G, H dan I.

2. Anak (Child)

   Sebuah simpul dikatakan sebagai anak apa bila simpul tersebut memiliki simpul lain yang terhubung dengannya dan simpul yang terhubung tersebut letaknya berada diatas simpul anak.
   Pada Gambar 1.1 simpul O adalah orang tua bagi simpul J.

3. Saudara kandung (Sibling)

   Sebuah simpul dikatakan sebagai saudara kandung apa bila simpul tersebut memiliki simpul lain yang memiliki parent yang sama.
   Pada Gambar 1.1 simpul M adalah saudara kandung N.

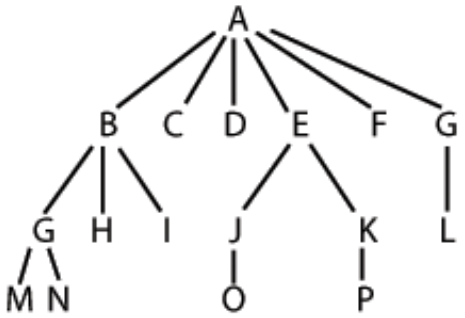
4. Pohon n-ary

   Pohon n-ary merupakan pohon yang jumlah anak maksimum setiap simpulnya adalah sebanyak n, gambar 1.1 merupakan contoh dari pohon n-ary.

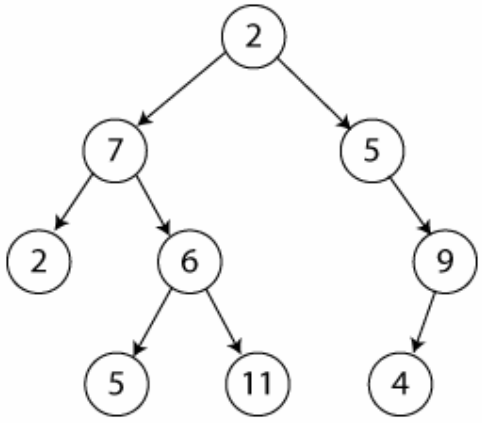
5. Pohon biner

   Pohon biner adalah pohon n-ary yang jumlah anak maksimum atau n nya berjumlah 2. Pohon jenis ini sangat sering digunakan di dalam dunia informatika karena kemudahan struktur datanya. Banyak sekali terapannya dalam komputer. Beberapa contohnya adalah kode Huffman dan juga Binary Search Tree (BST)

   Karena hanya memiliki jumlah maksimum anak = 2, maka anak dari pohon biner dibagi 2, yaitu *left* dan *right.*
   Gambar 1.2 merupakan contoh dari pohon biner.



Gambar 1.1 Pohon n-ary



Gambar 1.2 Pohon n-ary

3. Binary Search Tree

Searching adalah hal yang sering dilakukan manusia. Mulai dari sekedar mencari data secara manual (melihat di buku telepon) sampai ketemu. Namun seiring berkembanganya zaman informasi-informasi pun semaki

banyak dan semakin banyak hal-hal yang ingin dicari pun semakin banyak sehingga lama-kelamaan manusia membuat cara penyimpanan data, seperti menyimpan di komputer. Oleh karena itu cara mencari data pun berkembang, hingga sekarang sudah ada berbagai macam metode. Namun Salah satu cara yang akan dibahas di makalah ini adalah Binary Search Tree.

Dalam ilmu komputer, sebuah BST adalah sebuah pohon biner yang memiliki sifat-sifat berikut:

- Setiap *node* memiliki sebuah nilai.
- Subpohon kiri dari sebuah *node* hanya memuat nilai-nilai yang lebih kecil atau sama dengan dengan nilai dari *node*.
- Subpohon kanan dari sebuah *node* hanya memuat nilai-nilai yang lebih besar atau sama dengan dengan nilai dari *node*.

Kelebihan utama dari pohon pencarian biner adalah keterkaitannya dengan algoritma pengurutan dan algoritma pencarian yang dapat lebih efisien, seperti *in-order traversal*.

Pohon pencarian biner adalah sebuah struktur data dasar yang digunakan untuk membentuk struktur data yang lebih abstrak seperti set, multiset, dan array asosiatif.

Jika PPB memperkenankan nilai-nilai duplikat, maka PPB merupakan sebuah multiset. Pohon jenis ini menggunakan ketaksamaan longgar (*non-strict inequalities*), sehingga semua yang berada di subpohon bagian kiri dari sebuah node adalah lebih kecil atau sama dengan nilai dari node, dan semua yang berada di subpohon bagian kanan dari node adalah lebih besar atau sama dengan dengan nilai dari node.

Jika PPB tidak memperkenankan nilai-nilai duplikat, maka PPB merupakan sebuah set dengan nilai-nilai unik, sama seperti set pada matematika (himpunan). Pohon tanpa nilai-nilai duplikat menggunakan ketaksamaan kaku (*strict inequalities*), artinya subpohon kiri dari sebuah node hanya memuat node-node dengan nilai yang lebih kecil dari nilai node, dan subpohon kanan hanya memuat nilai-nilai yang lebih besar.

Beberapa definisi PPB menggunakan sebuah ketaksamaan longgar hanya pada satu sisi, sehingga nilai-nilai duplikat diperkenankan. Walaupun demikian, definisi-definisi PPB tersebut membatasi dengan baik bagaiman sebuah pohon dengan banyak nilai duplikat dapat diseimbangkan

3.1 Operasi-operasi

3.1.1 Pencarian

Pencarian sebuah nilai tertentu pada pohon biner adalah sebuah proses yang dapat dilakukan secara rekursif karena nilai-nilai yang disimpan adalah terurut. Pencarian dimulai dengan memeriksa akar (root). Jika nilai yang dicari sama dengan akar, maka nilai ditemukan. Jika nilai yang dicari kurang dari akar, maka pencarian dilakukan terhadap subpohon kiri, sehingga kita secara rekursif mencari subpohon kiri dengan cara yang sama. Jika nilai yang dicari lebih besar dari akar, maka pencarian dilakukan terhadap subpohon kanan sehingga kita secara rekursif mencari subpohon kanan dengan cara yang sama. Jika kita mencapai sebuah ujung (leaf) dan belum menemukan yang dicari, maka nilai tersebut tidak ada dalam pohon. Sebuah pembandingan dapat dibuat dengan pencarian biner, yang beroperasi hampir mirip degan pengaksesan acak pada sebuah array.

Berikut adalah algoritma prosedural pencarian dalam Binary Search Tree

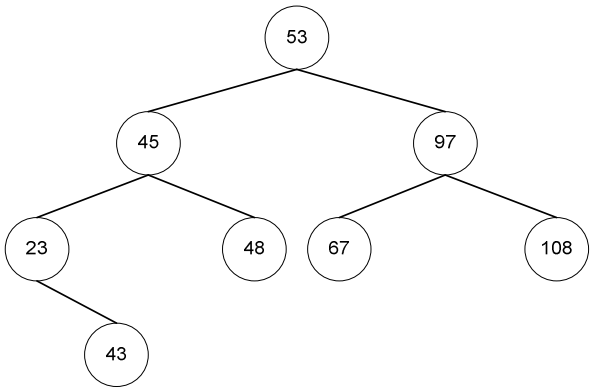
Procedure InsSearchTree (input X : int, input/output P :Bintree)

```
If (IsEmpty(P)) then
        "tidak ada X di pohon P"
Else {rekurens}
        Depend on X, Key P
        X = Akar(P) : Contain (Akar (P))
        X < Akar(P) : InsSearchTree (X, Left(P))
        X > Akar(P) : InsSearchTree (X, Right(P))
```

Jadi pada algoritma di atas yang terjadi adalah memanggil prosedur searching *InsSearchTree* kalau P(Pohon biner) kosong maka akan menampilkan error message. Bila pohon tidak kosong maka akan dilihat Akar(P) (simpul yang ditemui pertama). Bila X lebih kecil dari Akar P maka Left(P) maka dilakukan InsSearchTree (X, Left(P)) yaitu membandingkan X dengan Left (P). Bila X lebih besar dari Akar P maka Right(P) maka dilakukan InsSearchTree (X, Right(P)) yaitu membandingkan X dengan Right (P). Bila X = Akar(P) maka procedure akan memanggil contain (Akar(P)) yaitu menampilkan isis dari Akar(P).

Seperti pada Gambar 2.1. Bila X = 43, maka saat masuk X langsung dibandingka dengan 53. Karena 43 bernilai lebih kecil, maka InsSearchTree (X, Left(P)). Kemudian X dibandingkan dengan 45 dan karena 43 bernilai lebih kecil, maka InsSearchTree (X, Left(P)). Kemudian X dibandingkan dengan 23 dan karena 43 bernilai lebih besar, maka InsSearchTree (X, Right(P)). Setelah itu X dibandingkan dengan 43, karena mmiliki nilai yang sama antara X dan Akar(P) maka lakukan Contain (Akar(P)).

Ket : Contain(Akar(P)) = fungsi untuk misalnya menampilkan isi dari file bernomor tersebut.



Gambar 2.1 Binary Search Tree

3.1.2 Penyisipan

Penyisipan dimulai sebagaimana sebuah pencarian dilakukan. Jika akar tidak sama dengan nilai sisipan, kita mencari subpohon kiri atau kanan seperti di atas. Pada suatu saat kita akan mencapai sebuah node luar dan menambahkan nilai sisipan sebagai anak kiri atau anak kanan, bergantung pada nilai node. Dengan kata lain, kita memeriksa akar dan secara rekursif menyisipkan node yang baru ke subpohon kiri jika nilai yang baru lebih kecil atau sama dengan akar, atau menyisipkan ke subpohon kanan jika nilai yang baru lebih besar dari root.

Berikut adalah algoritma prosedural penyisipan dalam Binary Search Tree

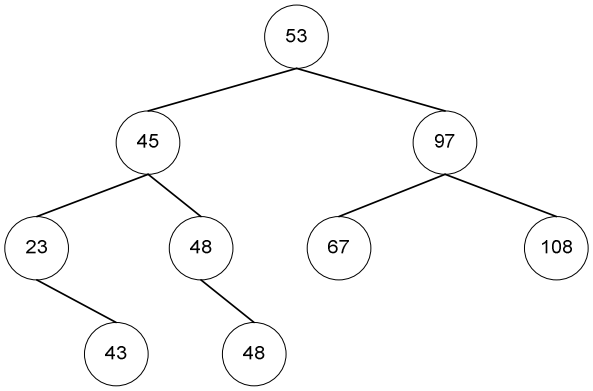
Function Insert (X : int, P :Bintree)

```
If (IsEmpty(P)) then
        MakeTree(X, nil, nil, P)
Else {rekurens}
        If ((X < Akar(P)) and (Left(P) = nil)) then
                Left(P) <= X
        If ((X >= Akar(P)) and (Right(P) = nil)) then
                Right(P) <= X

        Else
           If (X < Akar(P)) then
                Insert (X, Left(P))
           Else
                Insert (X, Right(P))
    ⇨   P
```

Penjelasan dari fungsi di atas adalah pertama bila Pohon tidak memiliki isi maka fungsi akan memanggil prosedur **maketree** yang berguna untuk membangung sebuah pohon dengan akar yang diisi dengan nilai X dan anak kanan dan kiri diisi dengan *nil* (kosong), singkat kata prosedur *maketree* membangun pohon *one element* dangan akar = X.
Bila X lebih kecil dari Akar(P) dan Left(P) = nil, maka Left(P) diisi oleh X. Bila X lebih besar sama dengan Akar(P) dan rightt(P) = nil, maka Right(P) diisi oleh X.

Bila X lebih kecil dari Akar(P) dan Left(P) tidak = nil, maka InsSearchTree (X, Left(P)). Bila X lebih besar sama dengan Akar(P) dan right(P) tidak = nil, maka InsSearchTree (X, Right(P)).



Gambar 2.2 Insert pada Binary Search Tree

Pada Gambar 2.2 terlihat Bila X = 48, maka saat masuk X langsung dibandingka dengan 53. Karena 48 bernilai lebih kecil, maka Insert (X, Left(P)). Kemudian X dibandingkan dengan 45 dan karena 48 bernilai lebih besar, maka Insert (X, Right(P)). Kemudian X dibandingkan dengan 48 dan karena 48 bernilai sama dengan X dan anak kanan dari 48 = nil pada awalnya, maka Right(P) diisi oleh X.

3.1.3 Penghapusan

Ada kasus-kasus yang mesti diperhatikan:

* Penghapusan sebuah ujung: Penghapusan sebuah node tanpa anak adalah mudah, cukup menghilangkanya dari pohon.
* Penghapusan sebuah node dengan satu anak: Menghapusnya dan mengganti dengan anaknya.
* Penghapusan sebuah node dengan dua anak: Misalkan node yang akan dihapus adalah N. Kita ganti nilai dari N dengan suksesor in-order (anak terkiri dari subpohon kanan) atau dengan predesesor in-order (anak terkanan dari subpohon kiri).

Berikut adalah algoritma prosedural peghapusan dalam Binary Search Tree

Procedure DelNode (input/output P : Bintree, input Y : Bintree)

Z : address

depend on P
Right(P) /= nil : DelNode (Right (P), Y)
Right(P) = nil : Z <= Left(Y)
Y <= P
P <= Left(P)
Left(Y) <= Z

Procedure DelBinTree (input/output P : Bintree, input X :int)

q : address
Y : address

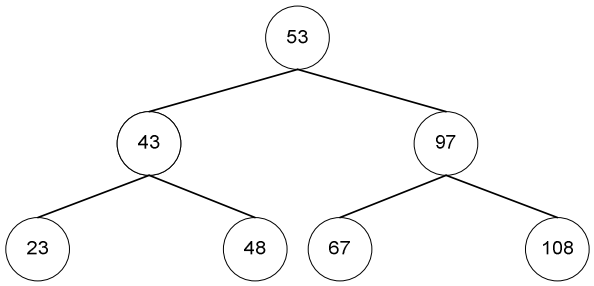
depend on X, key(P)
X < Akar(P) : DelBinTree (Left(P), X)
X > Akar(P) : DelBinTree (Right(P), X)
X = Akar(P) : q <= P
if (Right(P) = nil) then
P <= Left(q)
else if (Left (q) = nil) then
P <= Right(q)
Else
Y <= Left(q)
DelNode(Left(q), Y)

Dari Kedua algoritma diatas, lebih baik untuk memahami algoritma **DelNode** dahulu karena algoritma **DelBinTree** menggunakan algoritma **DelNode** di dalamnya.

DelNode(P, Y) adalah prosedur untuk mendelete sebuah simpul yang memiliki 2 buah anak (kanan dan kiri)dengan Y = alamat dari Akar(P). Bila kanan dari simpul memiliki anak maka melakukan rekursif dengan memanggil prosedur DelNode(Right(P), Y). Bila Right(P) bernilai nil maka Y (Alamat node yang akan di delete) digantikan oleh Right(P).Kemudian nilai P diisi oleh left(P).

DelBinTree(P, X) adalah prosedur untuk mendelete alamat yang bernilai X dari Pohon P.

Pertama-tama X diambil dan dibandingkan dengan akar(P). Bila X lebih besar dari Akar(P), maka DelBinTree (Right(P), X). Bila X lebih kecil dari Akar(P), maka DelBinTree (Left(P), X). Bila X = Akar(P), maka jika Right (P) = nil maka address P diisi oleh Left(q). jika Left (P) = nil maka address P diisi oleh Right(q). Jika (Right(P) /= nil) and (Left(P) /= nil) maka Y diisi oleh Left(q), kemudian panggil prosedur DelNode(Left(q), Y)



Gambar 2.3 Binary Search Tree yang sudah melakukan delete

Pada gambar diatas terjadi sebuah proses delete pada BST yang ada di gambar 2.1. Prosedur DelBinTree(P, 45), berarti yang dihapus adalah address yang berelemen 45. Perhatikan gambar 2.1, pertama-tama X yang bernilai 45 dibandingkan dengan Akar(P) yaitu 53. Karena X bernilai lebih kecil maka DelBinTree (Left(P), X). Karena nilai X dan Akar(P) yang sekarang sudah sama dan memiliki 2 buah anak (biner) maka DelNode (Left(q), Y) dimana Y = address yang memiliki nilai 45. Setelah itu karena Right(P) /= nil, maka DelNode(Right(P), Y). Sekarang Right (P) sudah = nil, proses yang terjadi selanjutnya adalah address Z diisi oleh Left(Y) yaitu 23. Kemudian nilai Y diisi oleh P yaitu 4, lalu P diganti oleh Left(P). Z diisi oleh Left(Y).

4. KESIMPULAN

Pohon Biner adalah sebuah struktur data yang sangat penting manfaatnya di bidang informatika, karena dengan memanfaatkan pola pikir rekursif, membuat algoritma menjadi mudah untuk diimplementasi dan juga waktu yang digunakan cenderung lebih efisien dari pada cari satu persatu. Sistem pencarian , pemasukan, penghapusan data pun menjadi lebih baik denga metode ini. Dari pada melakukan pencarian secara brute force ataupun secara traversal satu presatu data, karena dalam dunia informatika efisiensi sangatlah dihargai.

REFERENSI

[1] Munir,Rinaldi. “Bab 9 Pohon”,Matematika Diskrit, EdisiKempat, 2006, hal IX 1 –IX 31
[2] Liem, Inggriani. “Pohon”.DiktatStruktur Data Edisi 2008, Program Studi Tehnik Informatika hal 111-123
[3] www.msdn.microsoft.com, Tanggal akses : 12 Desember 2009, pukul 00.30
[4] www.en.wikipedia.org, Tanggal akses : 14 Desember 2009, pukul 17.13