Penerapan Teknik Binary Search Tree Sebagai Alternatif Penyimpanan Data

Reynald Alexander G 13509006
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
reynald.a.g.34@students.itb.ac.id

Abstract

Database merupakan kunci dari keberjalanan sebuah instansi baik perusahaan maupun sekolah ataupun universitas. Di makalah ini ditawarkan BST sebagai alternatif pengolahan database karena keunggulan-keunggulan yang tidak dimiliki oleh pengolahan database yang lainnya seperti pemasukan data yang terurut, search yang efisien, tidak memerlukan sort, dll. Tetapi kelebihan utama yang dapat ditawarkan oleh BST tentunya adalah efisiensi dalam pengaksesan data. Selain kelebihan tentunya BST memiliki kelemahan, namun kelemahan tersebut tidak terlalu mendasar dan tidak menyebabkan BST tidak dapat digunakan sebagai alternatif pengolah database yang baru.

***Kata Kunci*—Pengolah Database, Binary Search Tree**

I. PENDAHULUAN

Database merupakan kegiatan yang sangat erat dalam kegiatan keinformatikaan. Mengapa demmikian? Bisa kita tinjau seperti data mahasiswa, data pegawai, data nasabah, data peserta ujian dan lain-lain tentunya adalah database. Database-database tersebut tentunya membutuhkan pengolahan yang sangat baik dari adminnya. Apa maksudnya? Apabila data diolah dengan baik, sehingga data terurut dengan baik dan tersusun secara kontigu, maka pengaksesan data pun akan jauh lebih mudah.

Lalu mengapa ditawarkan metode binary search tree, khusunya dengan linked list? Karena pada dasarnya teknik penyimpanan dengan cara binary search tree ini dapat mempermudah kinerja, menghemat waktu proses(tingkat efisiensi tinggi). Maksud kinerja tersebut adalah misalnya dalam pengurutan data, pencarian data, data insertion, dan data deletion.

II. DATABASE

Database adalah kumpulan informasi yang disimpan di dalam komputer secara sistematik dan terurut sesuai dengan keyword sehingga dapat diperiksa menggunakan suatu program komputer untuk memperoleh informasi dari basis data tersebut. Basis data tersebut dapat direpresentasikan dengan skema. Skema-skemanya pun dapat beraneka ragam seperti skema relasional, skema hierarkis, dll.

III. TREE

Sebelum membiacarakan Binary Search Tree, mari kita bahas dulu bab yang melingkupi BST tersebut. Sebenarnya, apa itu pohon? Definisi pohon dari buku Struktur Diskrit penerbit ITB adalah graf tak berarah yang terhubung dan tidak mengandung sirkuit.

Sifat-sifat pohon antara lain :

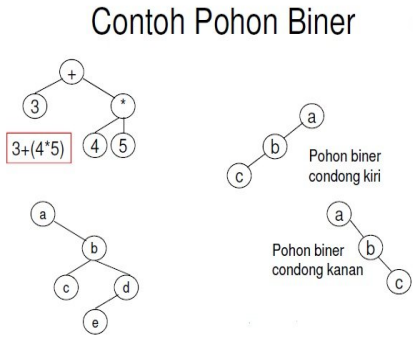
- Setiap pasang simpul pada pohon terhubung dengan lintasan tunggal
- G terhubung dan memiliki m = n – 1 buah sisi
- G tidak mengandung sirkuit
- Penambahan satu sisi pada graf hanya akan membentuk suatu sirkuit
- Setiap sisinya adalah jembatan yang apabila diputus akan menyebabkan pohon terpecah menjadi dua bagian

Pohon memiliki beberapa macam :

- Pohon n-ary
- Pohon terurut
- Pohon biner

Pohon biner sendiri memiliki beberapa terapan seperti:

- Pohon ekspresi
- Pohon keputusan
- Kode awalan
- Kode Huffman
- Pohon pencarian biner / binary search tree



Gambar 3.1 Berbagai contoh pohon biner

IV. BINARY SEARCH TREE

Binary Search Tree adalah salah satu bentuk dari pohon. Di mana masing-masing pohon tersebut hanya memiliki dua buah upapohon, yakni upapohon kiri dan upapohon kanan.

Ciri khas yang melekat pada binary search tree ini yang bisa juga dibilang sebagai keunggulan dari BST adalah peletakan isi dari nodenya yang terurut berdasarkan besarnya dari isinya tersebut. Isinya bisa saja berupa integer, karakter, atau apapun sesuai dengan spesifikasi BST yang ada.

Operasi dasar dari Binary Search Tree(BST) ini sendiri sangatlah sederhana, yakni hanya fungsi perbandingan dan fungsi rekursif. Di mana anak pohon sebelah kiri node adalah anak pohon yang lebih kecil dari node, sedangkan anak pohon sebelah kanan node memiliki isi yang lebih besar daripada isi node. Biasanya penyimpanan data di dalam BST ini bisa juga berupa record di mana pengurutannya hanya tinggal melihat key dari record tersebut, missal nomor absen, NIM, tanggal, dll.

V. KEUNGGULAN BINARY SEARCH TREE

A. Search Method

Metode search adalah penentu paling fundamental dalam program atau bisa dibilang nyawa dari program database. Kenapa search sangat penting untuk efisien.? Bayangkan saja apabila ada miliaran data yang ada di dalam database pada sebuah komputer server dan harus diolah dan selalu harus diupdate setiap hari ataupun diakses pengguna untuk dicari isi data yang diinginkanya. Lalu, yang kita gunakan adalah struktur data yang tidak efisien, di mana suatu saat computer server akan mengalami minimal hang atau yang paling parah adalah server down sehingga tidak bisa diakses dan akhirnya justru membuang waktu bahkan lebih parahnya bisa menyebabkan data dalam harddisk server menghilang. Tentunya kita sangat menghindari hal semacam itu terjadi bukan?

Metode Search sendiri ada banyak, seperti yang sudah sempat disinggung sedikit pada pendahuluan. Bahwa search macamnya ada linear search, binary search, binary search tree, dll. Di sini hanya akan dibahas 3 contoh tersebut.

Linear search merupakan tipe search yang paling mendasar dalam dunia search. Pada dasarnya cara kerja linear search ini sangat sederhana. Yakni memulai pencarian dari elemen pertama kemudian bergeser terus ke elemen berikutnya hingga menemukan isi elemen yang dicari. Apabila elemen yang dicari tidak ada, maka linear search akan terus melakukan traversal hingga elemen terakhir. Mari kita bayangkan apabila terdapat banyak sekali data yang harus ditraversal dan data yang dicari tidak ada di dalam database(worst case possibility). Efektivitas dari linear search ini adalah dengan O(n) worst case tergantung dari jumlah elemen yang ada di dalam database tersebut.

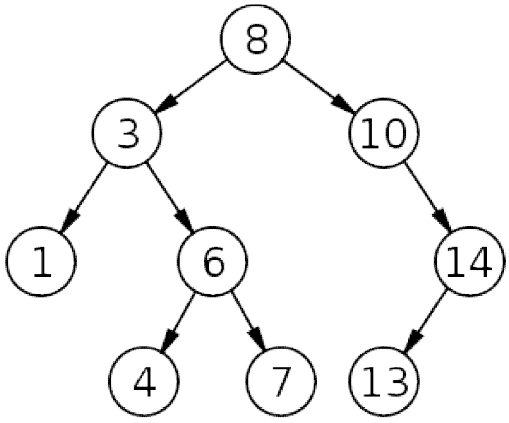
Metode search yang berikutnya yakni binary search, bedanya binary search dari linear search adalah dengan pembagian jumlah elemen menjadi dua bagian. Sehingga pencarian bisa lebih efektif dengan cara membagi pencarian ke dua arah. Worst case dari binary search ini adalah O(log n) sehingga cara ini bisa dibilang cukup efisien.

Metode search yang terakhir adalah binary search tree(sebenarnya bukan merupakan metode search). Binary search tree memiliki kompleksitas algoritma O(log n) yang tidak ada bedanya dengan metode binary search. Tetapi seharusnya metode search pada BST bisa lebih singkat, karena jalur yang dipilih sudah tergolong spesifik, sehingga tidak perlu memeriksa semua elemen untuk mencari elemen tertentu.

B. Struktur Data

Struktur data merupakan keunggulan dari BST jika kita bandingkan dengan array pada umunya. Mengapa hal tersebut bisa dibilang sebagai keunggulan? Karne struktur data dari BST adalah linked list dengan elemennya sudah terurut. Dengan demikian apabila struktur datanya sudah terurut maka pengaksesannya pun jauh lebih mudah. Hal yang diakibatkan dari struktur data tersebut adalah kita tidak perlu melakukan sorting terhadap data yang sudah ada. Berbeda dengan struktur data yang lain, seperti struktur data array, linked list biasa membutuhkan sebuah fungsi sort untuk mengurutkan datanya.

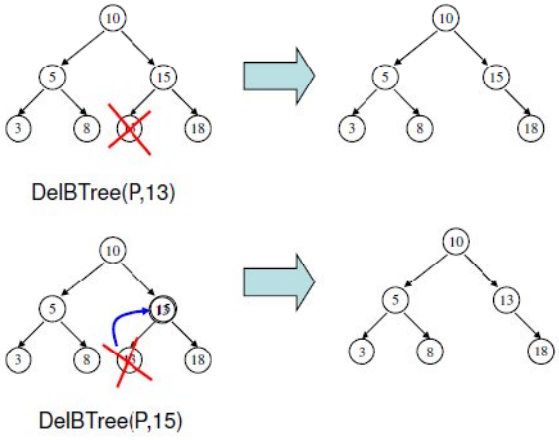
Keunggulan lain yang disebabkan struktur data dari pohon ini adalah kemudahan insertion process. Pada proses pemasukan data pada BST, pemasukan data sangatlah mudah, sama sekali tidak ada pergeseran data. Proses yang terjadi hanyalah pencarian lokasi yang tepat untuk data yang dimasukkan tersebut, kemudian dikaitkan ke data di atasnya sehingga dia menjadi anggota BST yang baru.



Gambar 3.1 Struktur Data BST

Kendati demikian, struktur data pada BST menyebabkan proses penghapusan data pada BST tergolong cukup rumit. Mengapa rumit? Karena harus ada traversal data untuk mencari node tertentu untuk menggantikan posisi dari node yang telah

dihapus(informasi akan diperjelas di dalam Gambar)\. Selain itu, kelemahan pemasukan data pada BST jika dibandingkan dengan Tabel Eksplisit adalah tabel eksplisit tidak memerlukan pengulangan untuk penambahan elemen baru, sedangkan bst membutuhkan proses sekuensial untuk menambahkan satu elemen baru. Namun hal tersebut dapat ditutupi dengan sorted element dari BST.



Gambar 3.2 Proses DelBTree pada BST

VI. BINARY SEARCH TREE DI DALAM DATABASE

Mengapa BST cocok untuk pengolahan database? Karena struktur datanya yang begitu unik dan rapi, BST bisa digunakan sebagai alternatif dalam database. Kemudian seperti yang telah dibicarakan pada bab sebelumnya, BST memiliki fitur-fitur yang dapat digunakan dalam database seperti Insert Element, Delete Element, Search Element, Display Element. Selain itu BST juga bisa melakukan penghitungan jumlah total elemen yang ada di dalam database. Prinsip penghitungan ini dapat dilakukan dengan prinsip penghitungan jumlah node yang ada pada BST tersebut. Kalkulasi elemen bilangan bulat atau real nya pun dapat dilakukan dengan mudah dengan menggunakan BST. Karena BST memiliki sifat perbandingan lebih besar dan lebih kecil, maka key pada database yang dibentuk dari BST haruslah unik. Hal ini memang sesuai dengan dunia nyata di mana sebuah NIM merepresentasikan hanya satu mahasiswa, NIP hanya mewakili seorang pegawai di sebuah pabrik, satu NO KTP hanyalah untuk seorang warga Negara dan tidak bisa lebih.

Keunggulan BST sebagai database juga karena efisiensi memorinya. Maksud dari efisiensi memori di sini adalah karena sifat dari BST adalah dinamik, di mana jumlah memori fleksibel sesuai dengan jumlah elemen yang ada. BST juga dapat dibentuk menjadi sebuah multilist yang mampu mengakses elemen record lainnya layaknya multilist pada umumnya.

Namun ada kekurangan BST sebagai alternatif pengolahan database. Kelemahan tersebut adalah tidak semua elemennya terkait ke elemen lainnya. Misalkan kita berada di elemen di mana node tersebut adalah salah satu daun dari BST, kita tidak bisa langsung pergi ke elemen di upapohon lainnya. Selain itu, kita tidak bisa secara langsung memanggil sebuah indeks tabel untuk memanggil node tertentu, jika kita bandingkan dengan representasi tabel. Namun sebenarnya hal tersebut tidak terlalu berpengaruh karena pengaksesan suatu nilai dari pengguna tentunya selalu ketika pointer berada di node awal dan penampilan semua node tidak harus dengan cara demikian dan kita tidak selalu membutuhkan indeks tabel untuk mencari elemen tertentu.

VII. KESIMPULAN

Dari hasil ulasan di atas dapat kita lihat bahwa sesungguhnya BST memiliki potensi untuk menjadi salah satu alternatif untuk mengolah database. Dengan BST waktu pengolahan dapat dipersingkat karena tingkat efisiensinya yang tergolong tinggi untuk hamper setiap proses. Namun demikian BST masih memiliki beberapa kelemahan seperti tidak semua node memiliki akses ke sebuah node yang lain.

VIII. APPENDIX

Berikut adalah tampilan gambar dari predikat dan selektor primitif yang dibutuhkan untuk membuat binary search tree:

```
type infotype : < Key : ..., { terdefinisi }
                  Count : integer >
type Node : < Info : infotype,
              Left : BinTree,
              Right : BinTree >
type BinTree : address

{ Selektor : Jika P adalah BinTree,
  Key(P)   → akses bagian P.Info.Key
  Count(P) → akses bagian P.Info.Count
  Left(P)  → akses bagian P.Left
  Right(P) → akses bagian P.Right }
```

Gambar 8.1 Kode primitf BST

```
procedure InsSearchTree (input X : infotype,
                         input/output P : BinTree)
{ Menambahkan sebuah node X ke pohon biner pencarian P }
{ infotype terdiri dari key dan count. Key menunjukkan nilai unik,
dan Count berapa kali muncul }
{ Basis : Pohon kosong  }
{ Rekurens : Jika pohon tidak kosong, insert ke anak kiri jika
nilai < Key(P) }
{ Atau insert ke anak kanan jika nilai > Key(P) }
{ Perhatikan bahwa insert selalu menjadi daun terkiri/terkanan dari
subpohon }

KAMUS LOKAL

ALGORITMA
   if (IsTreeEmpty(P)) then { Basis: buat hanya akar }
      MakeTree(X,Nil,Nil,P)
   else {Rekurens }
      depend on X, Key(P)
         X.Key = Key(P) : Count(P) ← Count(P) + 1
         X.Key < Key(P) : InsSearchTree(X,Left(P))
         X.Key > Key(P) : InsSearchTree(X,Right(P))
```

Gambar 8.2 Penambahan elemen ke BST

```
procedure DelBTree (input/output P : BinTree, input X : infotype)
{ Menghapus simpul bernilai Key(P) = X.Key, asumsi: X.Key pasti
ada di P}
{ infotype terdiri dari key dan count. Key menunjukkan nilai
unik, dan Count berapa kali muncul }
{ Basis : ? ; Rekurens : ? }
KAMUS LOKAL
   q : address
   procedure DelNode (input/output P : BinTree)
   { … }
ALGORITMA
   depend on X, Key(P)
     X.Key < Key(P) : DelBTree(Left(P),X)
     X.Key > Key(P) : DelBTree(Right(P),X)
     X.Key = Key(P) : { Delete simpul ini }
                q ← P
                if IsEmptyTree(Right(q)) then P ← Left(q)
                else if IsEmptyTree(Left(q)) then P ← Right(q)
                     else
                          DelNode(Left(q))
                Dealokasi(q)
```

Gambar 8.3 Penghapusan pohon BST

```
procedure DelNode (input/output P: BinTree)
{ I.S. P adalah pohon biner tidak kosong }
{ F.S. q berisi salinan nilai daun terkanan }
{ Proses : }
{ Memakai nilai q yang global}
{ Traversal sampai NODE terkanan, copy nilai NODE terkanan P,
salin nilai ke q semula }
{ q adalah NODE TERKANAN yang akan dihapus }
KAMUS LOKAL

ALGORITMA
   depend on P
      Right(P) ≠ Nil : DelNode(Right(P))
      Right(P) = Nil : {P anak terkanan}
         Key(q) ← Key(P)    {salin info P ke q}
         Count(q) ← Count(P)
         q ← P              {q: yang akan dihapus}
         P ← Left(P)   {pastikan anak kiri "naik", jika ada}
```

Gambar 8.4 Penghapusan node

```
function NBDaun (P : BinTree) → integer
{ Prekondisi: Pohon Biner tidak mungkin kosong.
  Mengirimkan banyaknya daun pohon }
{ Basis: Pohon yang hanya mempunyai akar: 1 }
{ Rekurens:
  Punya anak kiri dan tidak punya anak kanan: NBDaun(Left(P))
  Tidak Punya anak kiri dan punya anak kanan : NBDaun(Right(P))
  Punya anak kiri dan punya anak kanan : NBDaun(Left(P)) +
                                         NBDaun(Right(P))       }
KAMUS LOKAL

ALGORITMA
   if (IsOneElmt(P)) then { Basis 1 : akar }
      → 1
   else { Rekurens }
      depend on (P)
        IsUnerLeft(P)  : → NBDaun(Left(P))
        IsUnerRight(P) : → NBDaun(Right(P))
        IsBiner(P)     : → NBDaun(Left(P))+NBDaun(Right(P))
```

Gambar 8.5 Kode contoh proses pada Pohon Biner

Traversal - Inorder

```
procedure InOrder (input P : BinTree)
{ I.S. Pohon P terdefinisi }
{ F.S. Semua node pohon P sudah diproses secara InOrder:
       kiri, akar, kanan }
{ Basis : Pohon kosong : tidak ada yang diproses }
{ Rekurens : Proses secara inorder (Left(P));
             Proses Akar(P);
             Proses secara inorder (Right(P)) }
KAMUS LOKAL

ALGORITMA
    if (P = Nil) then { Basis-0 }
       { do nothing }
    else { Rekurens, tidak kosong }
       InOrder(Left(P))
       Proses(P)
       InOrder(Right(P))
```

Gambar 8.6 Predikat untuk memproses BST(misal Print Node)

```
KAMUS
{ Deklarasi TYPE  POHON BINER }
   constant Nil  : ... { konstanta pohon kosong, terdefinisi }

   type infotype : ... { terdefinisi }
   type address  : ... { terdefinisi }
   { Type Pohon Biner }
   type BinTree  : address
   type node     : < Info  : infotype, { simpul/akar }
                     Left  : BinTree,  { subpohon kiri }
                     Right : BinTree   { subpohon kanan } >
   {Cara akses/selektor, P: Pohon: Akar(P), Left(P), Right(P)}

{ Tambahan struktur data list untuk pengelolaan elemen pohon } h2
   type ElmtNode : < Info : infotype,
                     Next : addressList >
   type ListOfNode : addressList
   { list linier yang elemennya adalah ElmtNode }
```

REFERENCES

Rinaldi Munir, *Matematika Diskrit edisi ketiga*, Informatika, 2005.
http://en.wikipedia.org/wiki/Binary_search_tree
HTTP://EN.WIKIPEDIA.ORG/WIKI/BINARY_SEARCH_ALGORITHM
Inggriani Liem, Diktat Struktur
http://macherish.blogspot.com/2010/06/binary-search-tree-itu-apa.html

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

ttd

Reynald Alexander G
13509006