# Penggunaan Pohon Biner Sebagai Struktur Data untuk Pencarian

Rita Wijaya/13509098
*Program Studi Teknik Informatika*
*Sekolah Teknik Elektro dan Informatika*
*Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia*
*13509098@std.stei.itb.ac.id*

***Abstract*—Pohon adalah graf tak-berarah terhubung yang tidak mengandung sirkuit. Pohon biner terurut dan pohon seimbang adalah dua dari berbagai jenis pohon yang pernah kita pelajari. Kedua jenis pohon ini memiliki manfaatnya masing-masing. Dalam makalah ini, kita akan membandingkan keefektifan pencarian menggunakan pohon biner terurut seimbang dibandingkan pohon biner terurut lainnya.**

***Index Terms*—Pencarian, Pohon biner, Pohon biner seimbang, Pohon biner terurut.**

## I. PENDAHULUAN

Bagi orang yang berkecimpung dalam dunia keinformatikaan, terutama programmer, pemilihan jenis struktur data yang akan digunakan dalam program membawa dampak yang besar. Kesalahan pemilihan jenis struktur data dapat mengakibatkan buruknya kinerja program karena program yang efisien menuntut pemilihan struktur data yang tepat.

Salah satu hal yang paling sering dilakukan dalam suatu program adalah proses pencarian dan sorting. Struktur data yang berbeda tentu saja memerlukan teknik pencarian yang berbeda. Hal ini mengakibatkan waktu yang diperlukan untuk pencarian juga berbeda. Struktur data itu dapat berupa tabel, list berkait, termasuk pohon.

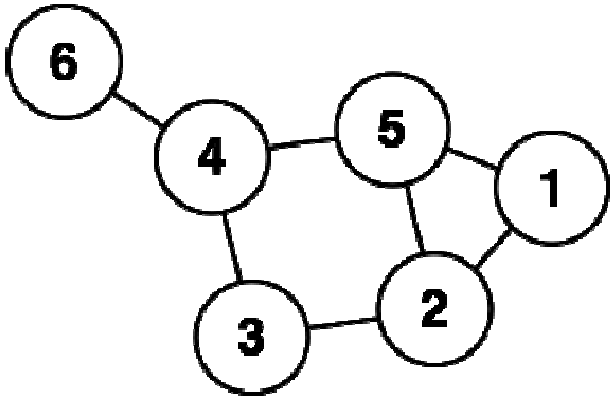
Kata pohon pasti sudah tidak asing lagi di telinga kita Namun, pohon sendiri memiliki berbagai pengertian tergantung konteks pembicaraan yang ada. Pohon yang akan kita bahas di sini bukanlah pohon yang merupakan tumbuhan dengan batang kayu, yang hidup dan dapat tumbuh. Pohon yang dimaksud dalam makalah ini merupakan salah satu dari aplikasi graf. Pohon ini adalah graf tak-berarah terhubung yang tidak mengandung sirkuit.

Pohon sendiri dikelompokkan lagi menjadi berbagai kategori, di antaranya pohon biner, pohon biner terurut, pohon n-ary, dan pohon seimbang. Masing-masing jenis pohon memiliki kelebihan dan kekurangannya sendiri. Untuk pencarian dan sorting yang lebih cepat, umumnya digunakan pohon biner terurut. Namun, apakah pencarian pada pohon biner terurut ini sudah merupakan yang paling mangkus? Jika tidak, maka jenis pohon apa yang lebih efisien dalam hal pencarian ini? Pencarian menggunakan struktur data pohon inilah yang akan kita bahas lebih lanjut dalam makalah ini.

## II. TEORI DASAR

### A. Graf

Graf G didefinisikan sebagai pasangan himpunan (V,E) ditulis dengan notasi G=(V,E), yang dalam hal ini V adalah himpuynan tidak-kosong dari simpul-simpul (vertices atau node) dan E adalah himpunan sisi (edges atau arcs) yang menghubungkan sepasang simpul.




**Gambar 1 Graf dengan 6 Simpul dan 7 sisi**

Sebagai contoh definisi dari graf pada gambar 1 di atas adalah :

$V$ = {1,2,3,4,5,6}

$E$ = {(1,2),(1,5),(2,3),(3,4),(4,5),(5,2),(4,6)}

Berdasarkan orientasi arah pada sisi, maka secara umum graf dibedakan atas 2 jenis:

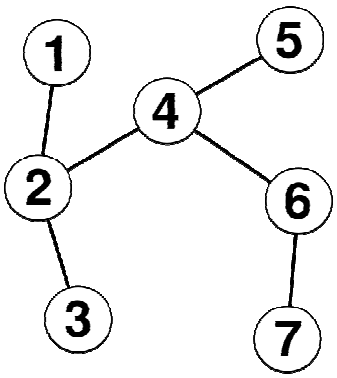
1. Graf tak-berarah
   Graf tak berarah adalah graf yang sisinya tidak memiliki orientasi arah. Graf pada gambar 1 di atas merupakan salah satu contoh dari graf tak-berarah. Pada grak tak-berarah, sisi (u,v) dan (v,u) adalah sama.
2. Graf berarah

Graf berarah adalah graf yang setiap sisinya diberikan orientasi arah. Untuk busur (u,v) pada grak berarah, simpul u diamakan simpul asal dan simpul v dinamakan simpul terminal.

Dalam graf, dikenal juga istilah lintasan. Lintasan yang panjangnya n dari simpul awal $v_0$ ke simpul tujuan $v_n$ di dalam graf G ialah barisan berselang-seling simpul-simpul dan sisi-sisi yang berbentuk $v_0$, $e_1$, $v_1$, $e_2$, $v_2$, …, $v_{n-1}$, $e_n$, $v_n$ sedemikian sehingga $e_1 = (v_0, v_1)$, $e_2 = (v_1, v_2)$, …, $e_n = (v_{n-1}, v_n)$ adalah sisi-sisi dari graf G. Lintasan yang berawal dan berakhir pada simpul yang sama disebut sirkuit atau siklus. Suatu graf tak-berarah G disebut graf terhubung jika untuk setiap simpul u dan v di dalam himpunan V terdapat lintasan dari u ke v (yang juga berarti ada lintasan dati v ke u). Jika tidak, maka G disebut graf tak-terhubung.

*B. Pohon*

Pohon adalah graf tak-berarah terhubung yang tidak mengandung sirkuit.

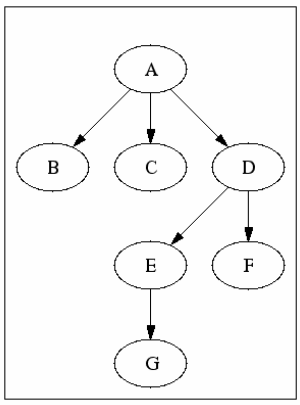



**Gambar 2 Pohon**

Misalkan G = (V, E) adalah graf tak-berarah sederhana dan jumlah simpulnya n. Maka, semua pernyataan di bawah ini adalah ekivalen:

1. G adalah pohon.
2. Setiap pasang simpul di dalam G terhubung dengan lintasan tunggal.
3. G terhubung dan memiliki m = n-1 buah sisi.
4. G tidak mengandung sirkuit dan memiliki m = n-1 buah sisi.
5. G tidak mengandung sirkuit dan penambahan satru sisi pada graf akan membuat hanya satu sirkuit
6. G terhubung dan semua sisinya adalah jembatan (jembatan adalah sisi yang bila dihapus menyebabkan graf terpecah menjadi dua komponen.)

Pohon yang sebuah simpulnya dianggap sebagai akar sisi-sisinya diberi arah menjauh dari akar dinamakan pohon berakar.




**Gambar 3 Pohon Berakar**

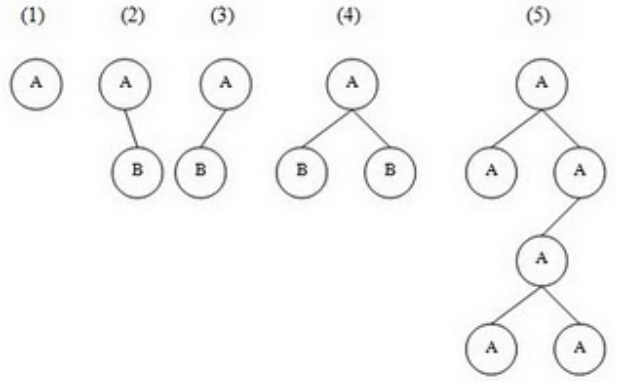
Berikut beberapa istilah yang berkaitan dengan pohon berakar:

1. Anak dan Orangtua
   Jika p merupakan simpul pada pohon berakar dan q merupakan anak dari simpul p (p orangtua dari q), maka harus terdapat sisi dari simpul p ke q. Pada gambar 3, simpul A merupakan orangtua dari simpul B, C, D. Dengan kata lain, simpul B, C, D merupakan anak-anak dari A.
2. Lintasan
   Lintasan antara simpul $v_1$ dan simpul $v_k$ adalah runtutan simpul-simpul dari $v_1$, $v_2$, $v_3$, $v_4$, ..., $v_k$ sedemikian rupa sehingga $v_i$ adalah orang tua dari $v_{(i+1)}$ untuk $1 \leq i < k$.
3. Keturunan
   Jika x dan y adalah simpul dalam pohon dan terdapat lintasan dari x ke y, maka x disebut leluhur dari y dan y adalah keturunan dari x.
4. Saudara kandung
   Simpul-simpul yang memiliki orang tua yang sama disebut saudara kandung.
5. Upapohon
   Yang dimaksud dengan upapohon dengan x (x adalah simpul di dalam pohon) sebagai akarnya ialah upagraf T' = (V', E') sedemikian sehingga V' mengandung x dan semua keturunannya dan E' mengandung sisi-sisi dalam semua lintasan yang berasal dari x.
6. Derajat
   Derajat adalah jumlah anak pada simpul tersebut. Simpul D pada gambar 3 berderajat 2 karena simpul D memiliki 2 anak yaitu E dan F.
7. Daun
   Daun adalah simpul yang berderajat 0. Pohon pada gambar 3 memiliki 4 daun, yaitu B, C, G, F.
8. Simpul Dalam
   Simpul dalam ialah simpul yang memiliki anak.
9. Aras atau Tingkat
   Jika akar dianggap memiliki aras = 0, maka aras simpul lainnya merupakan panjang lintasan dari akar ke simpul tersebut.
10. Tinggi (*height*) atau Kedalaman (*depth*)

Tinggi pohon adalah panjang lintasan maksimum dari akar ke daun. Pohon pada gambar 3 memiliki tinggi 3.

### *C. Pohon Biner*

Pohon biner adalah pohon yang setiap simpulnya memiliki anak paling banyak berjumlah 2. Anak-anak tersebut dibedakan antara anak kiri (*left child*) dan anak kanan (*right child*). Karena ada perbedaan urutan anak, maka pohon biner adalah pohon terurut. Pohon biner merupakan jenis pohon yang paling populer karena sering dimanfaatkan. Pohon biner memang memiliki banyak aplikasi, misalnya dalam sorting atau untuk menghitung hasil evaluasi suatu ekspresi aritmatika.




**Gambar 4 Hutan yang Terdiri dari 4 Pohon Biner**

Gambar 4 merupakan gambar hutan yang di dalamnya terdapat 4 pohon biner. Anak kiri dan anak kanan dibedakan berdasarkan letaknya dari akar. Pada pohon 2, simpul B merupakan anak kanan dari A, sedangkan pada pohon 3, simpul B merupakan anak kiri dari A. Oleh sebab itu, pohon 2 dan pohon 3 berbeda.

### *D. Pohon Biner Seimbang*

Pohon seimbang memenuhi kriteria berikut:

1. Pohon seimbang tingginya, perbedaan tinggi subpohon kiri dengan subpohon kanan maksimum 1.
2. Pohon seimbang jumlah simpulnya, perbedaan banyaknya simpul subpohon kiri dengan subpohon kanan maksimum 1.

### *E. Pohon Biner Terurut*

Pohon biner terurut memenuhi sifat:

1. Semua simpul subpohon kiri dari sebuah simpul selalu lebih kecil dari atau sama dengan nilai pada simpul tersebut.
2. Semua simpul subpohon kanan dari sebuah simpul selalu lebih besar dari atau sama dengan nilai pada simpul tersebut.

## III. PEMBAHASAN

### *A. Pencarian pada Pohon Biasa*

Berikut algoritma untuk pencarian pada pohon biner biasa dalam bahasa C.

```
typedef int infotype;

/*** Definisi Type Pohon Biner ***/
typedef struct tnode *address;
typedef struct tnode
{   infotype Info;
    address Left;
    address Right;
} node;
typedef address BinTree;

#define Akar(P) (P)->Info
#define Left(P) (P)->Left
#define Right(P) (P)->Right
```

Header (deklarasi tipe dan *getters*)

```
/*** Searching ***/
boolean Search (BinTree P, infotype X)
    //Mengirimkan true jika ada node dari P yang bernilai X
{   if(IsTreeEmpty(P))
        return false;
    else
    {   if(Akar(P)==X)
            return true;
        else
            return Search(Left(P),X)||Search(Right(P),X);
    }
}
```

Algoritma pencarian simpul bernilai X

Jumlah operasi pada proses pencarian terhadap pohon biasa dengan jumlah simpul n akan dilakukan sebanyak n kali. Hal ini karena penelusuran dilakukan terhadap seluruh simpul yang ada, setelah itu barulah proses pencarian dihentikan dan diterminasi. Oleh karena itu, proses pencarian pada pohon biner biasa dengan jumlah simpul n memiliki T(n) = O(n).

### *B. Pencarian pada Pohon Biner Terurut*

Berikut algoritma untuk pencarian pada pohon biner biasa dalam bahasa C.

```
/*** Definisi Type Pohon Biner Terurut***/
typedef struct{
    int key;
    int count;
}infotype;
typedef struct tnode *address;
typedef struct tnode
{   infotype info;
    address root;
    address left;
    address right;
} node;
typedef address BinTree;

#define info(P) (P)->info
#define key(P) (P)->info.key
#define count(P) (P)->info.count
#define right(P) (P)->right
#define left(P) (P)->left
#define root(T) (P)->root
```

Header (deklarasi tipe dan *getters*)

```
/*** Searching ***/
boolean BSearch (BinTree P, infotype X)
{
//KAMUS LOKAL
    BinTree Pt;
//ALGORITMA
    Pt = Alokasi(X);
    if (IsEmpty(P))
        {return false;}
    else
        if (key(P) == key(Pt))
            {return true;}
        else
            {if (key(Pt) < key(P))
                {return (BSearch (left(P), X));}
            else {return(BSearch (right(P), X));}
            }
}
```

Algoritma pencarian simpul bernilai X

Berdasarkan algoritma di atas, dapat kita lihat bahwa untuk kasus terbaik di mana nilai yang dicari ditemukan langsung pada operasi pertama, maka jumlah operasi yang dilakukan hanya berjumlah 1 kali. Kasus terburuk terjadi saat pohon biner tersebut merupakan pohon condong kiri, sedangkan nilai yang kita cari lebih kecil atau sama dengan nilai pada daun terkiri. Pada kasus terburuk ini, jumlah operasi yang dilakukan menjadi sebanyak n kali. Notasi Big-O yang dimiliki pada pencarian ini sama dengan pencarian pada pohon biner. Untuk proses pencarian pada pohon biner terurut dengan jumlah simpul n memiliki T(n) = O(n).

### *C. Pencarian pada Pohon Biner Seimbang*

```
typedef int infotype;

/*** Definisi Type Pohon Biner ***/
typedef struct tnode *address;
typedef struct tnode
{   infotype Info;
    address Left;
    address Right;
} node;
typedef address BinTree;

#define Akar(P) (P)->Info
#define Left(P) (P)->Left
#define Right(P) (P)->Right
```

Header (deklarasi tipe dan *getters*)

```
/*** Searching ***/
boolean Search (BinTree P, infotype X)
    //Mengirimkan true jika ada node dari P yang bernilai X
{   if(IsTreeEmpty(P))
        return false;
    else
    {   if(Akar(P)==X)
            return true;
        else
            return Search(Left(P),X)||Search(Right(P),X);
    }
}
```

Algoritma pencarian simpul bernilai X

Algoritma pencarian pada pohon seimbang memang sama saja dengan algoritma pencarian pada pohon biasa. Perbedaan pohon biasa dan pohon seimbang hanya terletak pada proses pembentukan, penambahan simpul baru, dan penghapusan. Namun, baik pada pohon biasa maupun pohon seimbang, pembentukannya sama-sama tidak diurut berdasarkan nilai simpul. Hal ini menyebabkan pengecekan tetap harus dilakukan terhadap seluruh simpul. Jadi, jumlah operasi pada proses pencarian terhadap pohon biasa dengan jumlah simpul n tetap dilakukan sebanyak n kali. Dengan kata lain, proses pencarian pada pohon biner biasa dengan jumlah simpul n memiliki T(n) = O(n).

### *D. Pencarian pada Pohon Biner Terurut Seimbang*

```
/*** Definisi Type Pohon Biner Terurut***/
typedef struct{
    int key;
    int count;
}infotype;
typedef struct tnode *address;
typedef struct tnode
{   infotype info;
    address root;
    address left;
    address right;
} node;
typedef address BinTree;

#define info(P) (P)->info
#define key(P) (P)->info.key
#define count(P) (P)->info.count
#define right(P) (P)->right
#define left(P) (P)->left
#define root(T) (P)->root
```

Header (deklarasi tipe dan *getters*)

```
/*** Searching ***/
boolean BSearch (BinTree P, infotype X)
{
//KAMUS LOKAL
    BinTree Pt;
//ALGORITMA
    Pt = Alokasi(X);
    if (IsEmpty(P))
        {return false;}
    else
        if (key(P) == key(Pt))
            {return true;}
        else
            {if (key(Pt) < key(P))
                {return (BSearch (left(P), X));}
            else {return(BSearch (right(P), X));}
            }
}
```

Algoritma pencarian simpul bernilai X

Pohon biner terurut seimbang merupakan pohon biner terurut yang kondisi pohonnya selalu dijaga agar seimbang. Pohon biner terurut seimbang ini memenuhi semua sifat pohon biner terurut dan pohon biner seimbang. Jadi, pohon jenis ini dapat dikatakan sebagai gabungan dari pohon terurut biasa dengan pohon seimbang.

Algoritma pencarian pada pohon biner terurut seimbang ini memang tidak berbeda dengan algoritma pencarian

pada pohon biner terurut biasa. Lantas, apa yang bebeda dari pencarian pada pohon terurut biasa dengan pencarian pada pohon terurut seimbang ini? Seperti yang telah kita ketahui, jumlah operasi yang terjadi pada kasus terburuk untuk pohon biner terurut sama dengan tinggi dari pohon tersebut. Pada pohon dengan jumlah simpul n dan tinggi h memiliki tinggi:

$h \geq {}^2\log (n+1) - 1 \geq {}^2\log n$.

Pohon seimbang yang perbedaan antara setiap subpohon kiri dan subpohon kanannya hanya 1, akan selalu menghasilkan pohon dengan tinggi minimum. Dengan perpaduan sifat-sifat tersebut, pohon biner terurut seimbang menjadi pohon biner terurut dengan tinggi minimum yang dapat dibentuk oleh n simpul ($^2\log n$). Dengan keadaan sedemikian, kasus terbaik pada pencarian terhadap pohon biner terurut seimbang tetap O(1). Namun, pada kasus terburuknya, operasi yang dilakukan maksimum hanya sebanyak tinggi yang dimiliki pohon tersebut yaitu $^2\log n$, sehingga $T(n) = O(\log n)$.

### *E. Perbandingan Kemangkusan Pencarian antar Struktur Data Pohon Biner*

Berdasarkan analisis sebelumnya, maka dapat kita buat tabel sebagai berikut.

| Jenis pohon | Kasus terbaik | Kasus rata-rata | Kasus terburuk |
|---|---|---|---|
| Pohon biner biasa | O(n) | O(n) | O(n) |
| Pohon biner terurut | O(1) | O(log n) | O(n) |
| Pohon biner seimbang | O(n) | O(n) | O(n) |
| Pohon biner terurut seimbang | O(1) | O(log n) | O(log n) |

Untuk kasus terbaik pada pohon biner terurut maupun pohon biner terurut seimbang, kompleksitas keduanya sama, yaitu O(1). Baik untuk kasus terbaik maupun kasus terburuk, pohon biner biasa dan pohon biner seimbang memiliki kompleksitas O(n). Kasus terburuk pada pencarian terhadap pohon biner terurut biasa kompleksitasnya O(n), dan untuk kasus terburuk pada pencarian terhadap pohon biner terurut seimbang kompleksitasnya hanya O(log n).

Untuk nilai n yang kecil, mungkin jenis struktur data apa pun yang dipilih tidak akan membawa pengaruh yang besar. Namun, apabila data yang diolah sangat banyak jumlahnya, maka jenis struktur data yang dipilih akan sangat mempengaruhi kecepatan pengolahan data.

Berikut tabel perbandingan jumlah operasi pada pencarian pada kasus terburuk untuk masing-masing jenis pohon dan n.

| n | Pohon biner biasa | Pohon biner terurut | Pohon biner seimbang | Pohon biner terurut seimbang |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 |
| 10 | 10 | 10 | 10 | 3 |
| 100 | 100 | 100 | 100 | 7 |
| 1000 | 1000 | 1000 | 1000 | 10 |
| $10^5$ | $10^5$ | $10^5$ | $10^5$ | 17 |
| $10^6$ | $10^6$ | $10^6$ | $10^6$ | 20 |
| $10^7$ | $10^7$ | $10^7$ | $10^7$ | 23 |

Untuk nilai yang kecil, perbedaan kecepatan antar jenis pohon yang dipilih memang tidak terasa. Namun apabila data yang diolah sangat besar jumlahnya, untuk satu kali proses pencarian mungkin saja perlu waktu berhari-hari. Pertumbuhan jumlah data atau elemen (simpul) pada pohon biner biasa, pohon biner terurut, maupun pohon biner seimbang berbanding lurus dengan lama pencarian (untuk kasus terburuk dan rata-rata). Bahkan pada kasus terburuk, perbandingan antara jumlah simpul dan banyak operasi yang dilakukan selama pencarian adalah 1:1. Namun, dengan menggunakan pohon biner terurut seimbang, lama waktu yang diperlukan untuk satu kali proses pencarian tidak tumbuh secepat jumlah data yang bertambah. Untuk kasus terburuknya saja, perbandingan antara jumlah simpul (n) dan banyak operasi yang dilakukan selama pencarian adalah $n : {}^2\log n$.

## IV. KESIMPULAN

Pohon biner terurut seimbang lebih efisien untuk digunakan sebagai struktur data diukur dari kecepatan proses pencariannya. Kompleksitas algoritma pencarian suatu nilai pada pohon biner terurut seimbang dengan n simpul yaitu baik pada kasus rata-rata maupun kasus terburuk yaitu $T(n) = O(\log n)$, dengan kasus terbaik $T(n)=O(1)$.

## REFERENCES


[1] Munir, Rinaldi, "Matematika Diskrit", Informatika Bandung, Bandung, Agustus 2005.

[2] Liem, Inggriani, "Diktat Struktur Data Edisi 2008", Progran Studi Teknik Informatika, ITB, 2008.

[3] http://id.wikipedia.org/wiki/Pohon_%28struktur_data%29. Diakses pada tanggal 14 Desember 2010, pukul 14.15.

[4] http://id.wikipedia.org/wiki/Teori_graf. Diakses pada tanggal 14 Desember 2010, pukul 14.20.

[5] http://id.wikipedia.org/wiki/Pohon_biner_terurut. Diakses pada tanggal 14 Desember 2010, pukul 14.35.


## 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, 14 Desember 2010

ttd

Rita Wijaya (13509098)