BAB II

MATERI PENUNJANG

Dalam bab ini dijelaskan beberapa teori yang dapat menunjang materi pada Bab III antara lain konsep dasar teori yang meliputi graph, pohon, notasi big O, running time, dan algoritma. Pada bagian akhir bab ini dibahas mengenai teori pohon biner dan pohon pencarian biner.

2.1. Graph

Definisi 2.1.1

Suatu graph G dinotasikan G = (V, E) adalah himpunan titik (*verteks*) V, dengan V = { v_1, v_2, v_3, ... , v_n} yang berhingga dan tidak kosong, dan himpunan garis (*edge*) E, dengan E ={ e_1, e_2, e_3, ..., e_n }.

Definisi 2.1.2

Graph berarah (*directed graph*) adalah graph yang semua garis – garisnya memiliki arah. Graph tak berarah (*undirected graph*) adalah graph yang semua garis – garisnya tidak memiliki arah.

Definisi 2.1.3

Titik – titik v_i dan v_j disebut titik akhir (*endpoint*) dari e_r jika e_r menghubungkan titik v_i dan v_j .

Definisi 2.1.4

Suatu garis e_r dikatakan *incident* dengan v_j jika v_j merupakan titik akhir dari e_r.

Definisi 2.1.5

Dua titik v_i dan v_j pada V disebut *titik adjacent* jika dihubungkan oleh sebuah garis e_r.

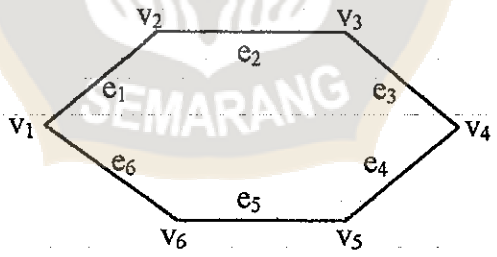
Definisi 2.1.6

Dua garis e_r dan e_s pada E disebut *garis adjacent* jika incident pada titik yang sama.

Definisi 2.1.7

Derajat (*degree*) v_i , untuk $v_i \in V$ adalah banyaknya garis yang incident dengan titik v_i dan dinotasikan dengan deg (v_i).

Contoh 2.1.1



Gambar 2. 1. 1.

(i). Titik v_1 dan v_2 adalah titik akhir dari e_1

(ii). Garis e_1 dan e_2 incident dengan titik v_2

(iii). Titik v_1 dan v_2 merupakan titik adjacent karena dihubungkan oleh garis e_1

(iv) Garis e_1 dan e_2 merupakan garis adjacent karena incident pada titik v_2

(v). Derajat masing – masing titik adalah 2

Definisi 2.1.8

Graph G = (V, E) adalah graph lengkap (*complete graph*) untuk setiap pasang v_i dan v_j anggota V adjacent. Jika G = (V, E) adalah graph lengkap dan banyaknya anggota V adalah n, maka G disebut graph lengkap atas n titik.

Contoh 2.1.2

Pada gambar 2.1.1, graph tersebut merupakan graph lengkap karena setiap titiknya adjacent dengan titik lainnya. Misalkan titik v_1 adjacent dengan titik v_2 $v_2 - v_3$, $v_3 - v_4$, $v_4 - v_5$, $v_5 - v_6$, dan $v_6 - v_1$. Dengan demikian graph tersebut merupakan graph lengkap atas 6 titik.

Definisi 2.1.9

Walk dari suatu graph G adalah barisan simpul dan garis yang bergantian, yang dimulai dan diakhiri dengan simpul. Setiap garis dalam walk menghubungkan dua simpul, yaitu simpul – simpul yang berada tepat sebelum dan sesudahnya.

Definisi 2.1.10

Sebuah walk yang semua simpul dan garisnya berbeda disebut *path.*

Definisi 2.1.11

Sebuah walk yang semua garisnya berbeda disebut *trail.*

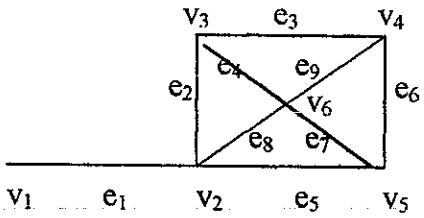
Definisi 2.1.12

Trail yang tertutup disebut sirkuit (*circuit*).

Definisi 2.1.13

Sebuah sirkuit dengan banyaknya simpul yang diulang satu kali ($v_0 = v_n$) disebut *cycle.*

Contoh 2.1.2



Gambar 2.1.2.

Dengan menggunakan graph pada gambar 2.1.2, maka

(i). W (v_1, v_3) = v_1 e_1 v_2 e_8 v_6 e_9 v_4 e_6 v_5 e_5 v_2 e_2 v_3 adalah walk.

(ii). P (v_1, v_3) = v_1 e_1 v_2 e_5 v_5 e_6 v_4 e_3 v_3 adalah path.

(iii). W (v_1, v_5) = v_1 e_1 v_2 e_8 v_6 e_9 v_4 e_3 v_3 e_4 v_6 e_7 v_5 adalah trail.

(iv). W (v_3, v_3) = v_3 e_3 v_4 e_9 v_6 e_8 v_2 e_5 v_5 e_7 v_6 e_4 v_3 adalah sirkuit.

(v). W (v_3, v_3) = v_3 e_3 v_4 e_4 v_2 e_2 v_3 adalah cycle.

Definisi 2.1.14

Graph terhubung (*connected graph*) adalah jika setiap pasang titik – titiknya dihubungkan oleh suatu path dan sebaliknya disebut graph tidak terhubung.

2.2. Pohon

Definisi 2.2.1

Pohon (*tree*) adalah suatu graph tak berarah dan terhubung (*connected*) yang tidak memuat *cycle*. Untuk selanjutnya dalam pembahasan pohon, titik dalam pohon disebut dengan simpul (*node*).

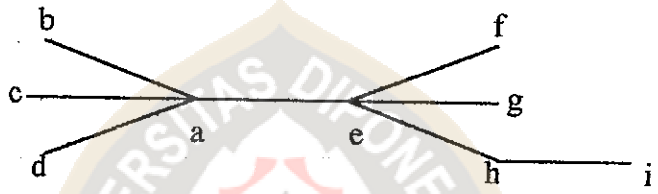
Definisi 2.2.2

Forest adalah suatu graph tak terhubung (unconnected) dan tidak memuat cycle.

Definisi 2.2.3

Simpul dengan derajat satu dalam suatu pohon dinamakan daun (*leaf*) atau simpul terminal (*terminal node*), sedangkan simpul dengan derajat lebih dari satu dinamakan simpul cabang (*branch node*) atau *simpul internal* .

Contoh 2.2.1



Gambar 2.2.1.

Gambar 2.2.1. merupakan pohon dengan 9 simpul. Simpul b, c, d, g, dan i adalah simpul daun atau simpul terminal, dan simpul a, e, h adalah simpul cabang atau simpul internal.

Definisi 2.2.4

(i). *Pohon berarah* adalah suatu pohon yang semua garisnya berarah.

(ii). Suatu pohon yang salah satu simpulnya dijadikan akar (*root*) disebut *pohon berakar*.

Definisi 2.2.5

Jika uv adalah garis berarah dalam suatu pohon berakar, maka u adalah simpul orang tua (*parent*)dari v dan v adalah simpul anak (*child*) dari u. Suatu simpul dalam (*simpul internal*) dari pohon berakar adalah simpul yang mempunyai anak.

Dengan demikian setiap simpul dalam pohon berakar kecuali akar mempunyai satu orang tua tetapi suatu simpul dapat mempunyai beberapa anak.

Definisi 2.2.6

(i). Jika u dan v adalah simpul dalam suatu pohon berakar, maka v adalah keturunan (*descendant*) dari u dengan $u \neq v$ dan u adalah simpul dari suatu path tunggal akar r menuju v. Simpul dengan *parent* sama disebut bersaudara (*sibling*).

(ii). Jika u adalah simpul dari suatu pohon berakar, subpohon dengan akar u adalah pohon yang terdiri atas u, keturunan u, dan semua garis berarah dari u menuju yang lainnya.

Definisi 2.2.7

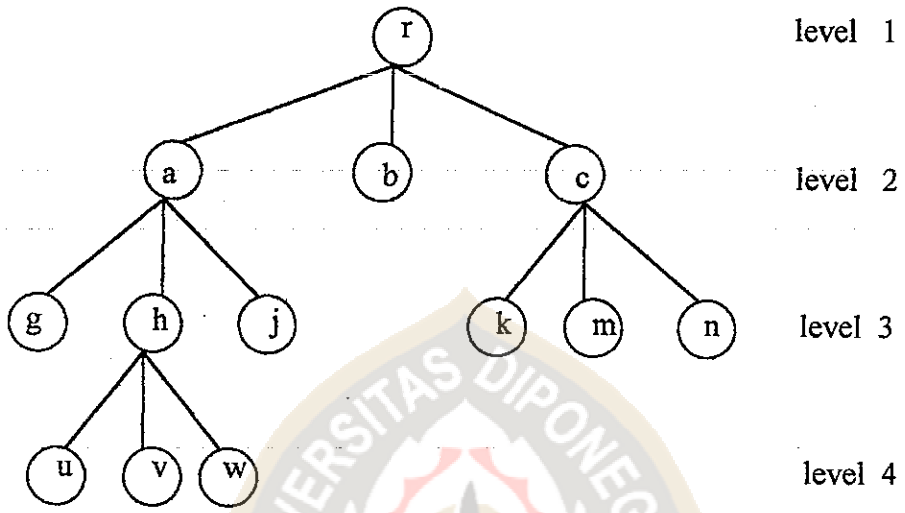
Misalkan T adalah suatu pohon berakar dengan akar r, maka

(i). Tingkat (*level*) dari r adalah satu dan tingkat dari simpul v ($v \neq r$) adalah panjang path tunggal dari r ke v.

(ii). Tinggi (*height*) atau kedalaman (*depth*) dari pohon T adalah tingkat maksimum dari simpul dalam pohon tersebut dikurangi satu.

(iii). *Ancestor* suatu simpul adalah semua simpul yang terletak dalam satu jalur dengan simpul tersebut dari akar sampai simpul yang ditinjau.

Contoh 2.2.2



Gambar 2.2.2

(i). Gambar 2.2.2 merupakan suatu pohon berakar dengan akar r

(ii). Simpul child dari r adalah a, b, dan c. simpul parent dari u adalah h dan simpul grandparent u adalah a. Simpul sibling g adalah h, j, k, m, dan n.

(iii). Subpohon dengan akar a terdiri atas a, tiga simpul child yaitu g, h, j, dan tiga simpul grand child yaitu u, v, w, serta garis yang menghubungkan antar simpul tersebut.

(iv). Dari gambar 2.2.2 dapat diketahui bahwa tingkat simpul r adalah 1, tingkat simpul g, h, j, k, m, dan n adalah 3 dan tingkat simpul u, v, dan w adalah 4. Dengan demikian tinggi (height) pohon tersebut adalah 3 karena tinggi maksimum dikurangi satu.

(v). Ancestor dari simpul v adalah r, a, dan h.

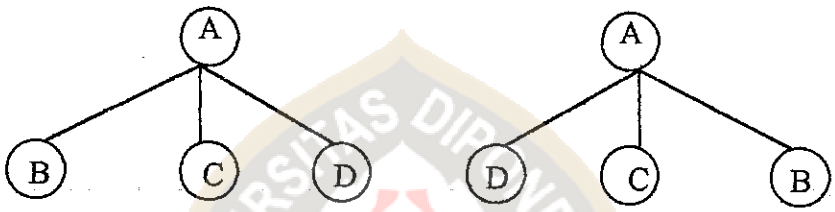
(vi). Jika dalam gambar 2.2.2, simpul r dihapus maka akan diperoleh sebuah hutan (forest) dengan 3 pohon.

Definisi 2.2.8

Suatu pohon T dikatakan sebagai pohon beraturan (*ordered tree*) jika urutan hubungan antara satu simpul dengan simpul yang lain sangat diperhatikan.

Contoh 2.2.3

Berikut ini disajikan dua pohon beraturan yang berbeda satu sama lain.



Gambar 2.2.3. Dua pohon beraturan yang berbeda

2.3. Algoritma

Algoritma berasal dari kata *algoris* dan *ritmis*, yang pertama kali diungkapkan oleh *Abu Ja'far Muhammad Ibnu Musa Al Khowarizmi* (825 M) dalam bukunya yang berjudul *Al-Jabr Wa-al Muqabla.*

Di dalam bidang pemrograman, algoritma didefinisikan sebagai metode khusus yang tepat dan terdiri dari serangkaian langkah yang terstruktur dan dituliskan secara sistematis yang akan dikerjakan untuk menyelesaikan suatu masalah dengan bantuan komputer.

Hubungan antara algoritma, masalah, dan solusi dapat digambarkan sebagai berikut :

Gambar 2.1.8. Diagram hubungan masalah, algoritma, dan solusi.

Proses dari masalah hingga terbentuk suatu algoritma disebut tahap pemecahan masalah, sedangkan tahap dari algoritma hingga terbentuk suatu solusi disebut dengan tahap implementasi. Solusi yang dimaksud adalah suatu program yang merupakan implementasi dari algoritma yang disusun.

Algoritma yang baik memiliki beberapa kriteria sebagai berikut :

1. Adanya output

   Dalam menyelesaikan suatu permasalahan, algoritma harus memiliki output yang merupakan solusi dari masalah yang sedang diselesaikan.

2. Efektifitas dan Efisiensi

   Suatu algoritma dikatakan efektif jika algoritma tersebut dapat menyelesaikan suatu solusi yang sesuai dengan masalah yang diselesaikan.

   Algoritma yang efisien adalah algoritma yang waktu prosesnya lebih singkat dan penggunaan memorinya relatif lebih sedikit.

3. Jumlah langkahnya berhingga

   Barisan instruksi yang dibuat dalam suatu urutan tertentu, dimaksudkan agar masalah yang dihadapi dapat diselesaikan. Banyaknya instruksi atau langkah-langkah harus berhingga.

4. Berakhir

   Proses di dalam mencari penyelesaian suatu masalah harus berhenti atau berakhir. Hasil akhir yang didapat merupakan solusinya atau informasi tidak ditemukannya solusi.

5. Terstruktur

Urutan dari barisan langkah-langkah yang digunakan harus disusun sedemikian rupa agar proses penyelesaiannya tidak berbelit-belit, sehingga memungkinkan waktu prosesnya akan menjadi relatif lebih singkat.

2.4. Notasi Big O

2.4.1. Beberapa Konsep Dasar Matematis

Definisi 2.4.1

Apabila $b^x = n$, dengan n adalah bilangan positif dan b adalah bilangan positif yang tidak sama dengan 1, maka eksponen x adalah logaritma dengan bilangan pokok b, dan ditulis dengan $x = \log_b n$.

Pada pembahasan analisa algoritma tugas akhir ini digunakan dua jenis logaritma yaitu logaritma berbasis 10 yang dinyatakan dengan notasi $\log_{10} n$ atau log n, dan logaritma berbasis 2 yang dinyatakan dengan notasi $\log_2 n$ atau lg n.

Contoh 2.4.1

(i). Karena $10^2 = 100$ maka 2 adalah logaritma dari 100 dengan bilangan pokok 10, sehingga $2 = \log_{10} 100$ atau $2 = \log 100$

(ii). $\log_2 8$ adalah bilangan x sedemikian hingga bilangan pokok 2 harus dipangkatkan agar menghasilkan 8 atau $2^x = 8$, $x = 3$ sehingga $\log_2 8 = 3$.

Definisi 2.4.2

Misalkan x sebarang bilangan riel, maka

(i). Floor dari bilangan x adalah integer bilangan terbesar yang lebih kecil atau sama dengan bilangan x, ditulis dengan $\lfloor x \rfloor$.

(ii). Ceiling dari bilangan x adalah integer bilangan terkecil yang lebih besar atau sama dengan bilangan x, ditulis dengan $\lceil x \rceil$.

Contoh 2.4.2

Misalkan x = 2,5 maka $\lceil 2{,}5 \rceil = 3$ dan $\lfloor 2{,}5 \rfloor = 2$

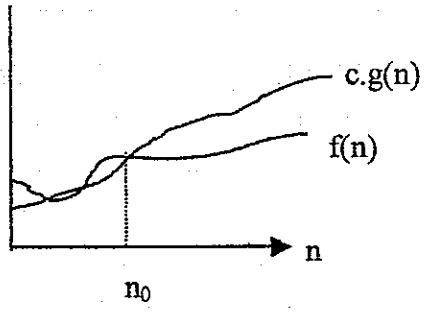
2.4.2. Pengertian dan Sifat – Sifat Notasi Big O

Definisi 2.4.3

Diberikan S, himpunan semua fungsi yang mempunyai domain D. Untuk D=N dan R yang mewakili himpunan bulat dan Riil. Fungsi f(n) dan g(n) anggota S. Dikatakan bahwa f(n) = O(g(n)), dibaca '*f(n) adalah big O dari g(n)*' jika dapat ditemukan bilangan positif c dan n_0 sedemikian hingga

$$|f(n)| \leq c.|g(n)|$$

untuk semua $n \in D$ dan $n \geq n_0$



Gambar 2.4.1. Fungsi f(n) = O(g(n))

Pada gambar 2.4.1 menunjukkan gambaran dari fungsi f(n) dan g(n) untuk f(n) = O(g(n)). Notasi O memberikan suatu batas atas untuk suatu fungsi pertumbuhan. Suatu fungsi f(n) termasuk dalam himpunan O(g(n)) jika terdapat bilangan positif c dan n_0 sedemikian hingga f(n) berada dibawah c | g(n) |. Penulisan f(n) = O(g(n) berarti bahwa fungsi f(n) merupakan anggota dari O(g(n)) atau f(n) ∈ O(g(n)).

Terdapat beberapa cara dalam menentukan notasi big O, diantaranya dengan menggunakan aturan limit dan teorema d'L Hospital.

(i). Penerapan aturan limit untuk menentukan notasi big O.

Aturan limit digunakan untuk menentukan kecepatan pertumbuhan dua fungsi. f(n) = O(g(n)) atau $|f(n)| \leq c.|(g(n)|$ jika

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$, untuk suatu c ∈ R.

jika limit dari rasio f(n) terhadap g(n) ada dan tidak sama dengan ∞ maka pertumbuhan fungsi f(n) tidak lebih cepat dibandingkan fungsi g(n). Jika limitnya adalah ∞, maka pertumbuhan fungsi f(n) lebih cepat daripada g(n).

(ii).Penerapan aturan d'L hospital

Jika $\lim_{n\to\infty} f(n) = \lim_{n\to\infty} g(n) = \infty$, maka $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{f'(n)}{g'(n)}$

dengan syarat derivatif dari f' dan g' ada.

Contoh 2.4.3

(i).Misalkan $g(n)=n^3/2$ dan $f(n)=37n^2+120n+17$. Akan ditunjukkan bahwa f(n) = O(g(n)) tetapi g(n) ≠ O(f(n)).

(ii). Misalkan suatu f(n) = n lg n dan g(n) = n^2. Akan ditunjukkan bahwa f(n) = O(g(n)), tetapi g(n) ≠ O(f(n)).

Analisa :

(i). - Untuk membuktikan f(n) = O(g(n)) digunakan limit dari rasio f(n) terhadap g(n).

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=\lim_{n\to\infty}\frac{37n^2+120n+17}{n^3/2}=\lim_{n\to\infty}\left(\frac{74}{n}+\frac{240}{n^2}+\frac{34}{n^3}\right)$$

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=0$$

Karena limit dari rasio f(n) terhadap g(n) ada, maka f(n) = O(g(n)).

- Untuk membuktikan bahwa g(n) ≠ O(f(n))

$$\lim_{n\to\infty}\frac{g(n)}{f(n)}=\lim_{n\to\infty}\frac{n^3/2}{37n^2+120n+17}=\lim_{n\to\infty}\frac{n^3}{74n^2+240n+34}$$

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=\infty$$

Karena limit dari rasio g(n) terhadap f(n) adalah ∞, maka g(n) ≠ O(f(n)).

(ii). Untuk menentukan f(n) = O(g(n)) digunakan aturan d'l hospital.

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=\lim_{n\to\infty}\frac{n\lg n}{n^2}=\lim_{n\to\infty}\frac{\lg n}{n}$$

$$\lim_{n\to\infty}\frac{f'(n)}{g'(n)}=\lim_{n\to\infty}\frac{\lg e/n}{1}=\lim_{n\to\infty}\frac{\lg e}{n}\quad\text{karena } f'(n)=\frac{\lg e}{n}$$

maka $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$

Karena limit dari rasio f(n) terhadap g(n) ada, maka f(n) = O(g(n)).

- Untuk membuktikan g(n) ≠ O(f(n)) maka

$$\lim_{n\to\infty} \frac{g(n)}{f(n)} = \lim_{n\to\infty} \frac{n^2}{n\lg n} = \lim_{n\to\infty} \frac{n}{\lg n}$$

$$\lim_{n\to\infty} \frac{g'(n)}{f'(n)} = \lim_{n\to\infty} \frac{1}{\lg e/n} = \lim_{n\to\infty} \frac{n}{\lg e}$$

$$\therefore \lim_{n\to\infty} \frac{g(n)}{f(n)} = \infty$$

Karena limit dari rasio g(n) terhadap f(n) adalah ∞, maka g(n) ≠ O(f(n)).

Teorema 2.4.1

Misalkan $f(n) = a_p n^p + a_{p-1} n^{p-1} + \ldots + a_1 n + a_0$ adalah fungsi polinomial dan g(n) $= n^p$ maka f(n) = O(g(n)).

Bukti :

Dipilih $c = |a_p| + |a_{p-1}| + \ldots + |a_1| + |a_0|$ dan n > 1 untuk n ∈ R.

Maka $|f(n)| = |a_p n^p + a_{p-1} n^{p-1} + \ldots + a_1 n + a_0|$

$$\leq |a_p n^p| + |a_{p-1} n^{p-1}| + \ldots + |a_1 n| + |a_0|$$

$$\leq |a_p| n^p + |a_{p-1}| n^{p-1} + \ldots + |a_1| n + |a_0|$$

$$\leq |a_p| n^p + |a_{p-1}| n^p + \ldots + |a_1| n^p + |a_0| n^p$$

$$= (|a_p| + |a_{p-1}| + \ldots + |a_1| + |a_0|) n^p$$

$$= c.n^p \quad . \ |f(n)| = c. \ |g(n)|$$

Terbukti bahwa f(n) = O(g(n)).

Teorema 2.4.2

Misalkan S adalah himpunan semua fungsi dengan domain D, dengan D = N, Z, atau R dan kodomain (0,∞). Fungsi-fungsi $f_1(n)$, $f_2(n)$, $g_1(n)$, dan $g_2(n)$ adalah anggota-anggota S sedemikian hingga $f_1(n) = O(g_1(n))$ dan $f_2(n) = O(g_2(n))$ maka berlaku :

(a). $f_1(n) + f_2(n) = O(\max\{ g_1(n), g_2(n)\})$.

(b). $f_1(n) \cdot f_2(n) = O(g_1(n) \cdot g_2(n))$.

Bukti :

(a). Karena $f_1(n) = O(g_1(n))$ dan $f_2(n) = O(g_2(n))$ maka terdapat c_1 dan c_2, k_1 dan k_2 sedemikian hingga jika $n \in D$ dan $n > k_1$, dan $n > k_2$, maka berlaku:

$$|f_1(n)| + |f_2(n)| = |f_1(n)| + |f_2(n)|$$
$$\leq c_1 . |g_1(n)| + c_2 . |g_2(n)|$$
$$= c_1 . g_1(n) + c_2 . g_2(n)$$
$$\leq c_1 . \max\{g_1(n), g_2(n)\} + c_2 . \max\{g_1(n), g_2(n)\}$$
$$= (c_1 + c_2) \max\{g_1(n), g_2(n)\}$$
$$= c . \max\{g_1(n), g_2(n)\}$$

maka terbukti bahwa

$f_1(n) + f_2(n) = O(\max\{g_1(n), g_2(n)\})$.

(b). Karena $f_1(n) = O(g_1(n))$ dan $f_2(n) = O(g_2(n))$ maka terdapat c_1, c_2, k_1 dan k_2 bilangan positif sedemikian hingga jika $n \in D$ dan $n > k_1$ maka $|f_1(n)| \leq c.g_1(n)$ dan jika $n > k_2$ berlaku $|f_2(n)| \leq c.g_2(n)$, selanjutnya pilih $k = \max(k_1, k_2)$ dan $c = c_1.c_2$ maka:

$$|(f_1.f_2)(n)| = |f_1(n)|.|f_2(n)|$$

$$\leq c_1.|g_1(n)|.c_2.|g_2(n)|$$

$$= c.|g_1(n).g_2(n)|$$

sehingga $f_1(n).f_2(n) = O(g_1(n).g_2(n))$ terbukti.

Contoh 2.4.4

Jika suatu fungsi $f(n)=3n^3+2n^2$ merupakan fungsi dari waktu tempuh suatu algoritma maka big O nya adalah n^3, yang dinotasikan $f(n) = O(n^3)$.

Analisa

Berdasarkan definisi 2.4.3 bahwa jika $f(n) = O(n^3)$ maka akan terdapat dua konstanta bulat positif c dan n_0 yaitu :

$f(0) = 0 \rightarrow n_0 = 0$, dan $f(1) = 5 \rightarrow c = 5$.

Sedemikian hingga berlaku $3n^3+2n^2 \leq 5n^3$ untuk setiap $n \geq 0$.

Jadi terbukti bahwa big O dari $f(n)=3n^3+2n^2$ adalah n^3 yang dinotasikan sebagai $f(n) = O(n^3)$.

2.5. Running Time

Pada suatu analisa algoritma untuk menentukan kecepatan prosesnya digunakan istilah *running time* (waktu tempuh) dan dinotasikan dengan T(n). Proses suatu algoritma di dalam mencari solusi dari suatu masalah memerlukan waktu tertentu.

Adapun hal-hal yang mempengaruhi running time adalah :

(i). Banyaknya langkah

Makin banyak langkah atau intruksi yang digunakan makin lama running time yang dibutuhkan dalam proses tersebut.

(ii). Besar dan jenis input data

Ukuran besar serta jenis input data yang digunakan akan sangat berpengaruh pada proses perhitungan.

(iii).Jenis operasi

Waktu tempuh juga dipengaruhi oleh jenis operasi yang digunakan. Jenis operasi tersebut meliputi operasi aritmetika, operasi nalar atau logika.

(iv). Komputer dan kompilator

Faktor ini di luar dari rancangan atau pembuatan algoritma yang efisien, dan lebih mengacu pada sistem komputer yang digunakan.

Dalam melakukan analisa algoritma terdapat tiga hal yang sering digunakan, yaitu :

(1). *Worst case running time*

Suatu keadaan 'terburuk' dari proses di dalam suatu algoritma, sehingga waktu yang ditempuh oleh algoritma tersebut adalah waktu yang maksimum.

(2). *Average case running time*

Analisa waktu rata-rata yang digunakan untuk melaksanakan algoritma. Akan tetapi hal ini tidak sering digunakan, karena adanya kesulitan dalam menentukan suatu data yang secara umum dapat mewakili keadaan rata-rata data yang digunakan.

(3). *Best case running time*

Suatu keadaan 'terbaik' dari proses di dalam suatu algoritma, sehingga waktu yang ditempuh oleh algoritma tersebut adalah waktu yang minimum.

Selanjutnya dalam menentukan running time suatu algoritma dipergunakan worst case running time, hal ini berdasarkan pada beberapa alasan yaitu :

(1). Worst case running time merupakan waktu maksimal yang dibutuhkan suatu algoritma dalam menyelesaikan masalah.

(2). Worst case running time dari suatu algoritma merupakan batas atas dari running time. Dengan demikian akan memberikan suatu jaminan bahwa algoritma yang digunakan mempunyai akhir proses.

(3). Untuk beberapa algoritma, worst case running time merupakan kejadian yang sering terjadi. Sebagai contoh, dalam proses pencarian suatu data base untuk mendapatkan suatu informasi, worst case running time selalu terjadi ketika informasi yang dibutuhkan tidak berada dalam data base.

Karena suatu algoritma mempunyai ciri-ciri khusus yang mungkin berbeda satu sama lain, maka dibutuhkan kemampuan untuk mengidentifikasi algoritma secara signifikan. Walaupun demikian, diperlukan suatu kerangka utama dalam menganalisa suatu algoritma sehingga memberikan kemudahan dalam melakukan analisa.

Dalam melakukan analisa algoritma digunakan suatu aturan umum yang dapat menyederhanakan perhitungan dalam menentukan running time, yaitu perhitungan hanya dikenakan pada suatu loop atau iterasi dari algoritma.

Aturan-aturan yang diperlukan dalam melakukan analisa algoritma adalah:

(1). Loop

Running time dari suatu loop adalah jumlah iterasi yang dilakukan untuk menyelesaikan statemen-stetemen dalam loop.

(2). Loop berkalang

Jika dalam suatu algoritma terdapat loop berkalang, yaitu loop yang berada dalam loop, maka yang menjadi pedoman adalah loop yang terdalam. Dan running time-nya adalah hasil dari perkalian ukuran semua input pada loop yang diselesaikan.

(3). Statemen berurutan

Apabila terdapat statemen berurutan, maka penentuan running time-nya menggunakan running time maksimal dari running time yang ada. (sesuai dengan teorema 2.4.2.1)

(4). Keadaan if-then-else

Untuk suatu penggalan algoritma : if (kondisi) then S_1 else S_2

Running time dari keadaan it-then-else tidak akan melebihi dari running time test kondisi ditambah dengan running time terbesar dari S_1 dan S_2.

2.6. Pohon Biner

Salah satu jenis pohon yang yang sering digunakan dalam struktur data adalah pohon biner. Dalam bagian ini akan dibahas mengenai pengertian pohon biner, beberapa karakteristik khusus pohon biner dan sifat-sifat pohon biner.

2.6.1. Konsep Dasar

Definisi 2.6.1

Pohon biner adalah himpunan terbatas simpul-simpul yang kosong atau terdiri atas sebuah akar dan dua subpohon yang saling asing yang disebut dengan subpohon kiri dan subpohon kanan.

Dalam hal ini istilah subpohon dapat disebut juga cabang. untuk pembahasan selanjutnya istilah cabang kiri untuk menyatakan subpohon kiri dan cabang kanan untuk menyatakan subpohon kanan.

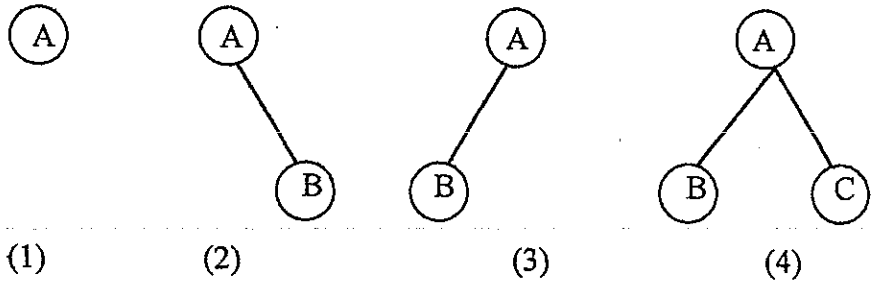
Sesuai dengan definisi pohon biner, terdapat tiga jenis karakteristik khusus dari pohon biner, yaitu:

(1). Setiap simpul paling banyak hanya memiliki dua buah anak, dengan kata lain derajat tertinggi dari setiap simpul dalam pohon biner adalah dua.

(2). Dalam pohon biner dibedakan antara cabang kanan dan cabang kiri.

(3). Pohon biner dimungkinkan untuk memiliki simpul-simpul kosong.

Berikut ini adalah ilustrasi dari beberapa pohon biner yang mungkin terjadi.



Gambar 2.6.1.

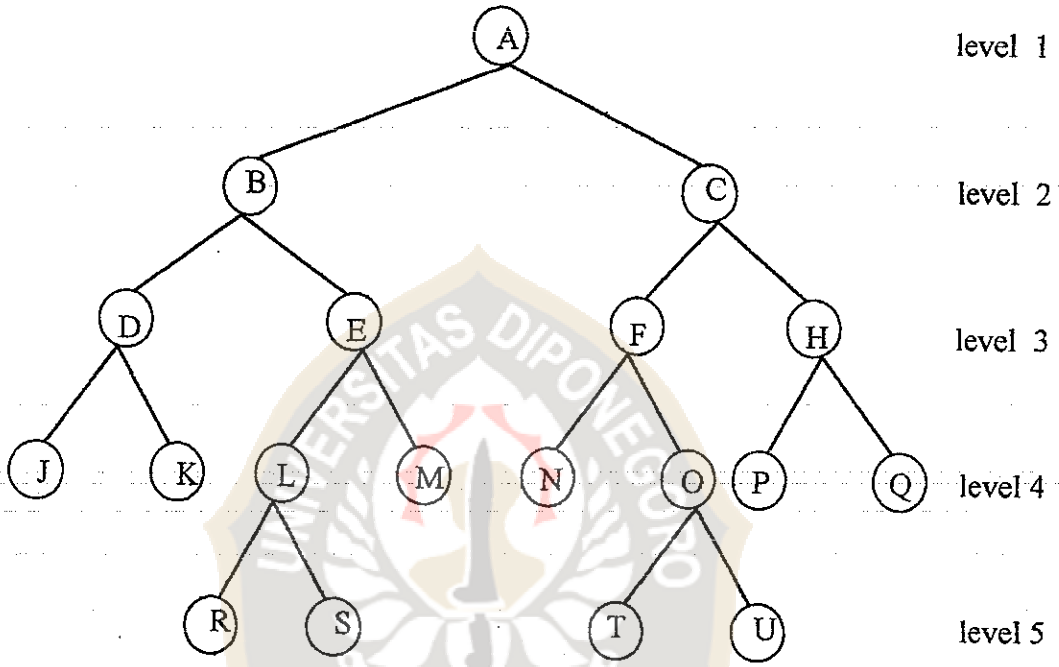
Pada gambar 2.6.1., pohon biner (2) dan (3) menurut definisi adalah berbeda, karena pada pohon biner (2) memiliki cabang kiri kosong, sedangkan pohon biner (3) memiliki cabang kanan yang kosong.

Pengertian daun , ayah, anak, tingkat (level), tinggi dan derajat yang berlaku dalam pohon, juga berlaku dalam pohon biner. berdasarkan bentuk strukturnya pohon biner dapat dibedakan menjadi dua bagian, yaitu :

(1). Pohon biner lengkap (*Complete binary tree*)

Pohon biner lengkap bertingkat i adalah sebarang pohon biner yang semua daunnya terdapat pada tingkat i dan semua simpul yang bertingkat lebih kecil dari i selalu memiliki cabang kanan dan kiri.

Pada gambar 2.6.2., menunjukkan pohon biner lengkap tingkat 4, tetapi bukan pohon biner lengkap pada tingkat 5.

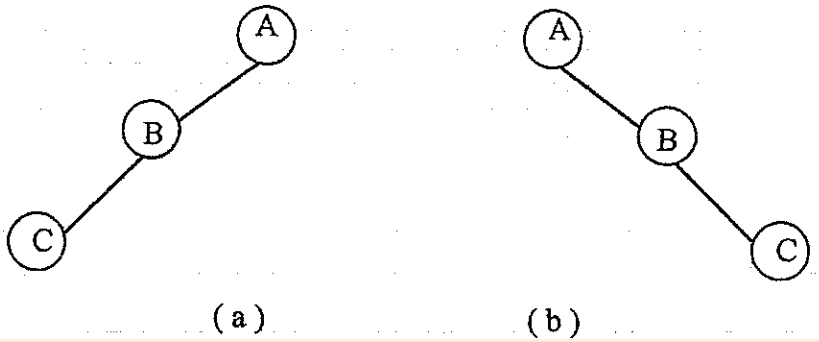


Gambar 2.6.2.

(2). Pohon biner miring (*Skewed binary tree*)

Pohon biner miring adalah pohon biner yang banyaknya simpul dalam cabang kiri tidak seimbang dengan banyaknya simpul dalam cabang kanan.

Pada gambar 2.6.3.(a). menunjukkan pohon biner miring ke kiri dan (b). miring ke kanan.



Gambar 2.6.3.

2.6.2. Sifat-sifat Pohon Biner

Dalam bagian ini akan dibahas jumlah maksimum dari simpul-simpul pohon biner dengan ketinggian h dan hubungan antara jumlah simpul daun dan jumlah dari simpul-simpul berderajat 2 dalam pohon biner.

Lemma 2.5.1

(1). Jumlah maksimum dari simpul-simpul pada tingkat (level) i dari pohon biner adalah 2^{i-1} , dengan $i \geq 1$

(2). Jumlah maksimum dari simpul-simpul dalam pohon biner dengan ketinggian h adalah 2^h-1 , dengan $h \geq 1$

Bukti :

(1). Proses pembuktian dengan induksi matematis.

- Basis induksi

  akar pohon biner berada dalam tingkat i = 1, dengan demikian jumlah maksimum simpul pada tingkat i = 1 adalah $2^{i-1}=2^{1-1}=2^0=1$.

- Hipotesis Induksi

  misal i adalah sebarang bilangan integer positif lebih besar dari 1. Asumsikan bahwa jumlah maksimum simpul pada tingkat i adalah 2^{i-2}.

- Langkah Induksi

  jumlah maksimum simpul pada tingkat i-1 berdasarkan hipotesis induksi adalah 2^{i-2}. Diketahui bahwa setiap simpul pada pohon biner memiliki derajat maksimum 2. Dengan demikian jumlah maksimum simpul-simpul pada tingkat i-1 atau 2^{i-1}.

(2). Maksimum jumlah dari simpul simpul pada pohon biner dengan ketinggian h adalah

$$\sum_{i=1}^{h} (\text{maksimum jumlah simpul-simpul pada tingkat i}) = \sum_{i=1}^{h} 2^{i-1} = 2^h - 1$$

Lemma 2.6.2

Diberikan suatu pohon biner tidak kosong T, jika n_0 adalah jumlah maksimum dari simpul-simpul daun dan n_2 adalah jumlah simpul-simpul berderajat 2 maka $n_0 = n_2 + 1$.

Bukti :

Misalkan n_1 adalah jumlah simpul-simpul berderajat 1 dan n jumlah total dari simpul. Diketahui semua simpul-simpul dalam T paling banyak berderajat 2, maka :

$$n = n_0 + n_1 + n_2 \qquad \ldots (2.6.1).$$

Jika dihitung jumlah cabang-cabang pada pohon biner, maka akan didapat setiap simpul kecuali akar memiliki sebuah cabang sebelumnya. Jika B adalah jumlah cabang-cabang, maka $n = B + 1$. Apabila diketahui semua cabang berasal dari derajat satu atau dua maka diperoleh $B = n_1 + 2n_2$. Karena itu diperoleh :

$$n = B + 1 = n_1 + 2n_2 + 1 \qquad \ldots (2.6.2)$$

Dengnan mensubstitusikan persamaan (2.6.1) terhadap persamaan (2.6.2) diperoleh :

$$n_0 + n_1 + n_2 = n = n_1 + 2n_2 + 1$$

$$n_0 = n_2 + 1.$$

Contoh 2.6.1

Pada gambar 2.6.2, pohon biner tersebut memiliki jumlah simpul daun $n_0 = 10$ dan jumlah simpul-simpul berderajat dua adalah $n_2 = 9$.

Dari lemma 2.6.1.(2). dengan mudah dapat diketahui bahwa ketinggian dari suatu pohon biner lengkap dengan simpul-simpul adalah $\lceil \lg (n+1) \rceil$.

2.7. Pohon Pencarian Biner

2.7.1. Konsep Dasar

Definisi 2.7.1

Pohon pencarian biner adalah pohon biner yang kosong atau setiap simpulnya memuat sebuah kunci dan memenuhi sebagai berikut :

(1). Kunci dalam subpohon kiri pada sebuah simpul (jika ada) lebih kecil dari kunci simpul akar.

(2). Kunci dalam subpohon kanan pada sebuah simpul (jika ada) lebih besar dari kunci simpul akar.

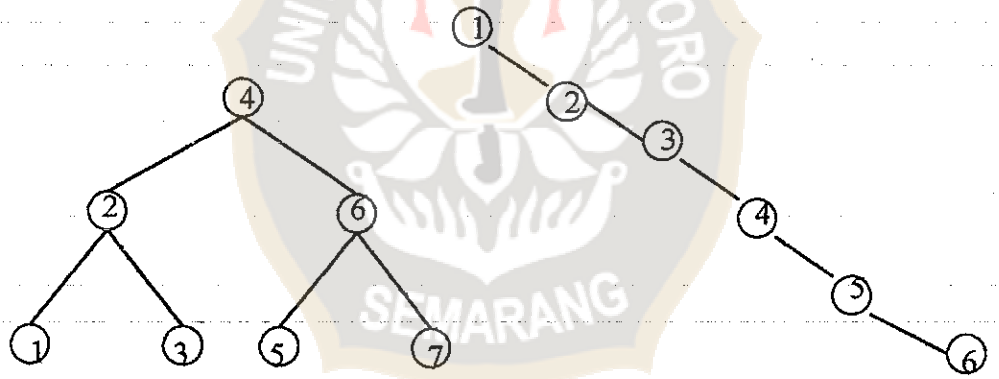
(3). Subpohon kiri dan subpohon kanan dari akar merupakan pohon pencarian biner.

Suatu pohon pencarian biner adalah suatu pohon biner dengan kunjungan secara inordernya akan memberikan urutan data yang sudah terurut menurut key. Untuk pembahasan selanjutnya istilah subpohon diubah menjadi cabang. Dalam pohon biner tersebut, setiap simpul berlaku bahwa nilai key data di kiri lebih kecil dari nilai key data pada simpul tersebut dan nilai data pada simpul tersebut lebih kecil dari nilai data dari cabang kanan.

Secara umum suatu key pada pohon pencarian biner selalu memenuhi aturan pohon pencarian biner, yaitu :

Diberikan simpul x pada pohon pencarian biner. Jika y simpul di cabang kiri dari x, maka key [y] < key [x], jika y adalah simpul di cabang kanan dari x maka key[x] < key [y].

Suatu key dalam pohon pencarian biner mempunyai field (medan) left, right dan p yang berkorespondensi dengan cabang kiri, cabang kanan, dan orang tua (parent). Jika anak atau orang tuanya tidak ada maka mempunyai nilai NIL. Akar simpul adalah satu simpul pada pohon dengan orang tuanya bernilai NIL.



(a) (b)

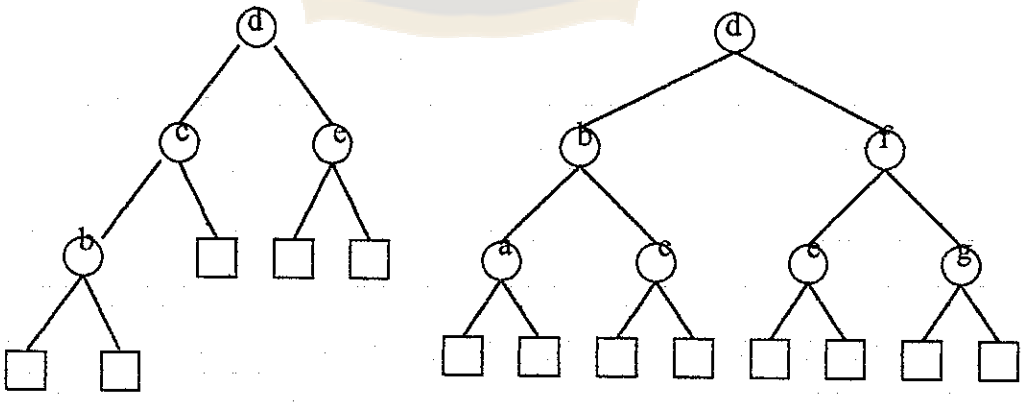
Gambar 2.7.1.

Suatu pohon pencarian biner seperti pada gambar 2.7.1 (a) mempunyai akar 4, sedangkan key 1, 2, dan 3 terletak pada cabang kiri karena tidak lebih besar dari 4. Untuk key 5, 6, dan 7 terletak pada cabang kanan karena tidak lebih kecil dari 4. Aturan yang sama juga berlaku untuk setiap simpul pada pohon gambar 2.7.1.(b) yang membentuk pohon biner miring karena key 2, 3, 4, 5, dan 6 tidak lebih kecil dari 1, sehingga pohon biner tersebut hanya mempunyai cabang kanan dan cabang kiri *NIL*.

2.7.2. Running Time Pohon Pencarian Biner

Dalam bagian ini akan ditentukan running time untuk proses pencarian pohon biner lengkap dan rata-rata kecepatan proses pencarian pohon biner dalam menentukan data key k.

Untuk melakukan analisa running time pencarian pada pohon biner, ditambahkan simpul-simpul tambahan di tempat-tempat dari pointer cabang yang berharga NIL. Pada pohon biner, simpul-simpul tersebut adalah daun. Untuk membedakan simpul tambahan dari simpul asli pohon biner maka simpul biner yang asli di beri tanda lingkaran sedangkan simpul tambahan diberi tanda kotak. Simpul yang asli disebut simpul dalam (*internal node*) sedang simpul daun tambahan disebut simpul luar (*external node*). Pohon biner yang mempunyai simpul-simpul tambahan disebut pohon biner yang diperluas.



(a) (b)

Gambar 2.7.2.

Secara umum suatu pohon biner dengan n buah simpul dalam akan mempunyai n + 1 buah daun. Selanjutnya untuk memudahkan analisa perhitungan kecepatan pada pohon biner didefinisikan besaran-besaran berikut. Panjang path luar dengan E adalah jumlah dari panjang path dari semua daun pohon biner. Dengan cara yang sama, didefinisikan panjang path dalam dengan I adalah jumlah semua path simpul dalam pada suatu pohon biner.

Contoh 2.7.1

Pada gambar 2.7.2 (a), diperoleh

$I = 0 + 1 + 1 + 2 = 4$

$E = 2 + 2 + 2 + 3 + 3 = 12$

dan pada gambar 2.7.2.(b), diperoleh

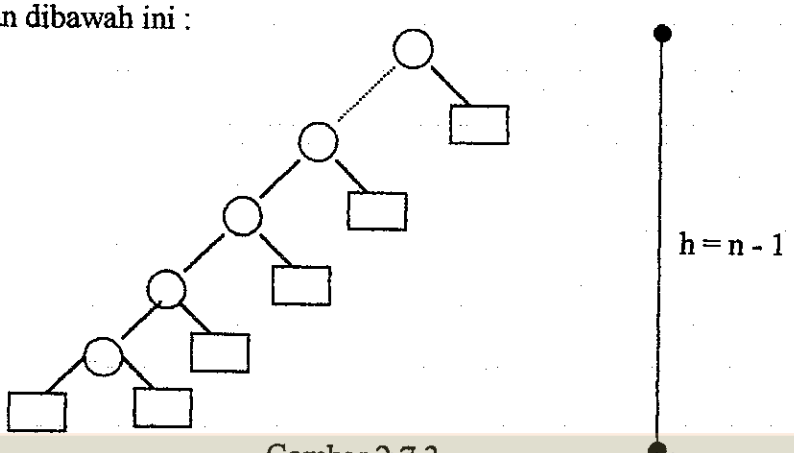
$I = 0 + 1 + 1 + 2 + 2 + 2 + 2 = 10$

$E = 3 + 3 + 3 + 3 + 3 + 3 + 3 + 3 = 24$

Secara umum pada pohon biner berlaku hubungan :

$E = I + 2n$; n = banyaknya simpul dalam dari suatu pohon biner

Suatu pohon pencarian biner miring ke kiri dengan n simpul dapat diillustrasikan dibawah ini :



Gambar 2.7.3.

Pohon pencarian biner yang miring ke kiri atau ke kanan dengan memiliki n simpul akan mempunyai panjang path yang sama dengan ketinggian pohon pencarian biner (n – 1).

Untuk suatu pohon pencarian biner yang miring ke kiri atau ke kanan saja dengan n simpul pada gambar 2.7.3., maka jumlah path dalam dan jumlah path luar adalah :

(i). Jumlah path dalam pohon biner miring ke kiri adalah

$$I = 0 + 1 + 2 + \ldots + n - 1$$

$$I = \sum_{j=1}^{n-1} j = \frac{n(n-1)}{2}$$

$$T(n) = O(n^2).$$

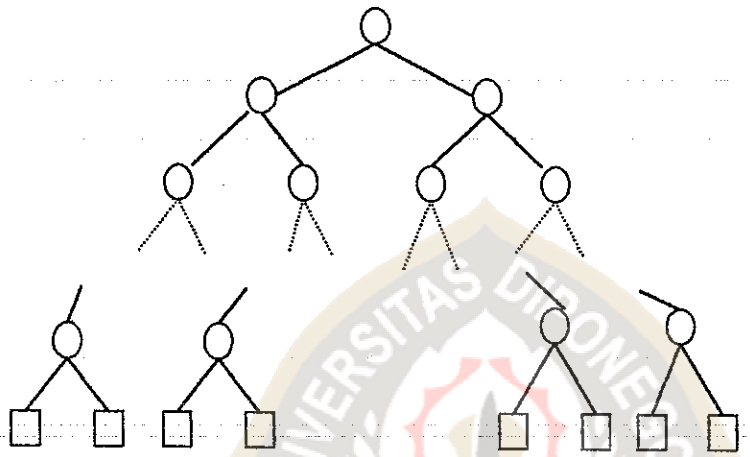
(ii). Jumlah path luar pohon biner miring ke kiri adalah

$$E = I + 2n = \frac{n(n-1)}{2} + 2n \quad \therefore \quad E = \frac{n(n+3)}{2}$$

$$T(n) = O(n^2).$$

Dengan demikian running time untuk pohon pencarian biner miring ke kiri atau ke kanan adalah $O(n^2)$, dengan n adalah jumlah simpul pada pohon pencarian biner tersebut.

Suatu pohon pencarian biner seimbang dengan n buah simpul dapat diilustrasikan seperti gambar 2.7.4. di bawah ini :



Panjang path	Simpul dalam
0	1
1	2
2	4
⋮	⋮
j	2^j

Gambar 2.7.4.

Berikut ini akan dibahas hal – hal yang erat kaitannya dengan proses perhitungan running time pohon pencarian biner :

i : level atau tingkat pohon biner

2^{i-1} : jumlah simpul pada level ke-i

$2^i - 1$: jumlah simpul pada pohon biner

j : panjang path pada pohon biner

2^j : jumlah simpul pada panjang path ke-j

Jika $2^{i-1} = 2^j$ maka $i - 1 = j$ atau $j + 1 = i$. Dengan demikian jumlah simpul pohon biner pada panjang path ke-j adalah $n = 2^{j+1} - 1$.

Untuk suatu pohon pencarian biner yang seimbang dengan n simpul pada gambar 2.7.4. , maka jumlah path dalam dan path luar adalah

(i). Jumlah path dalam pohon pencarian biner seimbang dengan n simpul adalah

$$I = (0x1) + (1x2) + (2x4) + (3x8) + \ldots + (jx2^j)$$

$$I = \sum_{i=1}^{j} i.2^i = (j-1).2^{j+1} + 2, \qquad i = 1,2,3,\ldots,j$$

Jika diketahui $n = 2^{j+1} - 1$ dan $n+1 = 2^{j+1}$ maka dengan menggunakan logaritma basis 2 akan diperoleh $\lg(n+1) = j+1$ atau $j = \lg(n+1) - 1$.

Dengan demikian akan diperoleh :

$$I = \sum_{i=1}^{\lg(n+1)-1} i.2^i = (\lg(n+1)-2).2^{\lg(n+1)} + 2, \qquad i = 1,2,3,\ldots,\lg(n+1)-1$$

$$= (\lg(n+1)-2).(n+1) + 2$$

$$= (n+1).\lg(n+1) - 2n$$

$I = (n+1).\lg(n+1) - 2n$; untuk n adalah banyaknya simpul pohon biner

(ii). Jumlah path luar pohon pencarian biner seimbang dengan n simpul adalah

$E = I + 2n$

$E = (n+1).\lg(n+1) - 2n + 2n$

$E = (n+1).\lg(n+1)$

Berikut ini akan dibahas mengenai proses penentuan running time pohon pencarian biner dengan menggunakan sifat – sifat notasi big O.

Misalkan $f_1(n) = (n+1).\lg(n+1) - 2n + 2$; $f_2(n) = (n+1).\lg(n+1)$; $g(n) = n.\lg n$

(i). Untuk $f_1(n) = O(g(n))$.

$$\lim_{n\to\infty} \frac{f_1(n)}{g(n)} = \lim_{n\to\infty} \frac{(n+1).\lg(n+1) - 2n}{n.\lg n}$$

$$\lim_{n\to\infty} \frac{f_1(n)}{g(n)} = \lim_{n\to\infty} \left(\frac{n.\lg n}{n.\lg n} + \frac{n.\lg(1+1/n)}{n.\lg n} + \frac{\lg n}{n\lg n} + \frac{\lg(1+1/n)}{n\lg n} - \frac{2n}{n.\lg n} \right)$$

$$\lim_{n\to\infty} \frac{f_1(n)}{g(n)} = \lim_{n\to\infty} \left(1 + \frac{\lg(1+1/n)}{\lg n} + \frac{1}{n} + \frac{\lg(1+1/n)}{n\lg n} - \frac{2}{n.\lg n} \right)$$

$$\lim_{n\to\infty} \frac{f_1(n)}{g(n)} = 1+0+0+0-0=1$$

karena $f_1(n) = I$, maka running time untuk jumlah path dalam pohon pencarian biner adalah O(n.lg n).

(ii). Untuk $f_2(n) = O(g(n))$.

$$\lim_{n\to\infty} \frac{f_1(n)}{g(n)} = \lim_{n\to\infty} \left(\frac{(n+1).\lg(n+1)}{n.\lg n} \right) = \lim_{n\to\infty} \left(\frac{(n+1)(\lg n + \lg(1+1/n))}{n\lg n} \right)$$

$$\lim_{n\to\infty} \frac{f_1(n)}{g(n)} = \lim_{n\to\infty} \left(\frac{n.\lg n}{n.\lg n} + \frac{n\lg(1+1/n)}{n\lg n} + \frac{\lg n}{n\lg n} + \frac{\lg(1+1/n)}{n.\lg n} \right)$$

$$\lim_{n\to\infty} \frac{f_1(n)}{g(n)} = \lim_{n\to\infty} \left(1 + \frac{\lg(1+1/n)}{n\lg n} + \frac{1}{n} + \frac{\lg(1+1/n)}{n\lg n} \right) = 1+0+0+0$$

$$\lim_{n\to\infty} \frac{f_1(n)}{g(n)} = 1$$

karena $f_2(n) = E$, maka running time untuk jumlah path luar pohon pencarian biner adalah O(n.lg n).

Kedua aturan I dan E sangat menentukan kecepatan rata-rata algoritma pencarian. Rata-rata kecepatan pencarian yang berhasil menemukan data akan sama dengan I dibagi dengan jumlah simpul dalam, sedangkan kecepatan rata-rata pencarian yang gagal akan sama dengan E dibagi dengan jumlah daun, jadi :

- Kecepatan rata-rata pencarian yang berhasil = I/n
- Kecepatan rata-rata pencarian yang gagal = E/n +1

(i). Bila diketahui running time untuk pohon pencarian biner miring adalah $I = O(n^2)$ dan $E = O(n^2)$.

Maka dapat ditarik kesimpulan :

- Kecepatan rata – rata pencarian yang berhasil pada pohon pencarian biner miring adalah I/n = O(n).

  Dengan demikian running time-nya adalah O(n).

- Kecepatan rata – rata pencarian yang gagal pada pohon pencarian biner miring adalah E/n+1 = O(n).

  Dengan demikian running time-nya adalah O(n).

(ii). Bila diketahui running time untuk pohon pencarian biner seimbang adalah $I = O(n.\lg n)$ dan $E = O(n.\lg n)$.

Maka dapat ditarik kesimpulan :

- Kecepatan rata – rata pencarian yang berhasil pada pohon pencarian biner seimbang adalah I/n = O(n.lg n).

  Dengan demikian running time-nya adalah O(lg n).

- Kecepatan rata – rata pencarian yang gagal pada pohon pencarian biner seimbang adalah E/n+1 = O(lg n).

  Dengan demikian running time-nya adalah O(lg n).

Dari proses analisa algoritma diatas, untuk struktur pohon yang lengkap rata-rata pencarian yang berhasil atau tidak berhasil adalah O(lg n), sedangkan untuk pohon yang miring rata – rata pencarian yang berhasil atau tidak adalah O(n). Sehingga dapat dikatakan bahwa struktur pohon yang lengkap lebih baik untuk pencarian data dibandingkan dengan struktur pohon yang miring.