BAB II

MATERI PENUNJANG

2.1 Himpunan

Himpunan merupakan kumpulan obyek-obyek yang berada dalam suatu keterikatan atau batasan yang mempunyai sifat keterikatan diantara anggota-anggotanya. Obyek-obyek ini disebut elemen-elemen atau anggota-anggota dari himpunan.

Himpunan-himpunan dinyatakan dengan huruf-huruf besar A, B, X, Y, ..., sedangkan elemen-elemen dalam himpunan dinyatakan dengan huruf-huruf kecil a, b, x, y, ...

Dalam menyajikan suatu himpunan terdapat dua cara, yaitu :

1. Dengan menyebutkan setiap elemen himpunan yang diletakkan dalam dalam tanda kurung kurawal.
2. Dengan menyebutkan sifat atau syarat keanggotaan.

contoh :

Himpunan A terdiri atas bilangan-bilangan 1, 3, 7 dan 10, maka ditulis :

A = { 1, 3, 7, 10 }

contoh :

B adalah himpunan dari semua bilangan-bilangan genap, maka dipergunakan suatu huruf, biasanya x, untuk menyatakan sebarang elemen dan ditulis :

B = { x| x genap } yang dibaca " B adalah himpunan dari bilangan x dimana x genap".

Jika suatu obyek x adalah elemen dari suatu himpunan A, artinya A mengandung x sebagai salah satu dari elemen-elemennya , maka ditulis : $x \in A$ yang juga dapat dibaca " x elemen A" atau " x di dalam A". Jika dipihak lain, suatu obyek x bukanlah anggota sebuah himpunan A, artinya A tidak mengandung x sebagai salah satu dari elemen-elemennya, maka ditulis : $x \notin A$.

Himpunan-himpunan dapat *berhingga* (*finite*) atau *tak berhingga* (*infinte*). Sebuah himpunan adalah berhingga bila terdiri atas sejumlah tertentu elemen-elemen yang berbeda, artinya , bila dihitung elemen-elemen yang berbeda dari himpunan ini, maka proses perhitungannya dapat berakhir. Bila tidak demikian, maka himpunannya adalah tak berhingga.

Definisi 2.1.1 :

Diberikan A dan B suatu himpunan. Himpunan A adalah *subhimpunan* atau *himpunan bagian* (*subset*) dari B, jika dan hanya semua elemen A adalah elemen B, dinotasikan dengan $A \subset B$, dibaca "A subhimpunan B ".

$$A \subset B \Leftrightarrow (\forall x) x \in A \Rightarrow x \in B$$

Contoh :

Himpunan C = { 1, 3, 5 } adalah subhimpunan dari D = { 5, 4, 3, 2,1} karena tiap-tiap bilangan 1, 3 dan 5 yang termasuk dalam C juga aggota D.

Definisi 2.1.2 :

Dua himpunan A dan B adalah sama, yaitu A = B, jika dan hanya jika $A \subset B$ dan $B \subset A$.

Teorema 2.1.1 :

Jika A adalah subhimpunan B dan B subhimpunan C maka A adalah subhimpunan C, yaitu jika $A \subset B$ dan $B \subset C$ maka berarti $A \subset C$

Bukti :

Dibuktikan dengan menggunakan kontradiksi. Diketahui A subhimpunan B dan B subhimpunan C. Andaikan A bukan merupakan subhimpunan C, artinya ada elemen x yang berada dalam A, $x \in A$, yang tidak berada dalam C, $x \notin C$.

A subhimpunan B, artinya semua elemen A, $x \in A$, merupakan elemen dari B, $x \in B$. Dan B subhimpunan C, artinya semua elemen B, $x \in B$, merupakan elemen dari C, $x \in C$. Konsekuensinya semua elemen dalam A , $x \in A$, berada dalam C, $x \in C$. Dengan demikian terjadi kontradiksi,sehingga pengandaian diingkar. Yang benar A subhimpunan C. ■

Operasi-operasi himpunan

Definisi 2.1.3 :

Diberikan A dan B suatu himpunan. *Perpaduan* atau *gabungan* (*union*) A dan B adalah himpunan dari semua elemen A atau B atau keduanya. Gabungan A dan B dinotasikan dengan $A \cup B$, dibaca " A gabungan B ".

$$A \cup B = \{x \in A \text{ atau } x \in B\}$$

Contoh :

Misalkan S = { a, b, c, d } dan T = { f, b, d, g }
Maka $S \cup T = \{a, b, c, d, f, g\}$

Definisi 2.1.4 :

Diberikan himpunan A dan B. *Irisan* (*intersection*) A dan B adalah himpunan dari elemen-elemen yang dimiliki bersama oleh A dan B, yaitu elemen-elemen yang termasuk A dan juga termasuk B. Irisan A dan B dinyatakan dengan $A \cap B$, yang dibaca " irisan A dan B ".

Contoh :

Misalkan S = { a, b, c, d } dan T = { f, b, d, g }
Maka $S \cap T = \{b, d\}$

Definisi 2.1.5 :

Diberikan himpunan A dan B. *Selisih* himpunan A dan B adalah himpunan dari elemen-elemen yang termasuk A tetapi tidak termasuk B. Dinyatakan selisih A dan B dengan A - B, yang dibaca " selisih A dan B ".

2.2 Relasi Dan Fungsi

Definisi 2.2.1 :

Suatu Relasi R adalah determinatif pada Semesta S, atau antara anggota anggotanya semesta S, bila hanya bila, kalimat " a R b " adalah kalimat deklaratif untuk setiap a, b dalam S.

contoh :

Relasi habis dibagi pada bilangan-bilangan bulat.

Karena relasi habis dibagi adalah determinatif diantara bilangan.

Definisi 2.2.2 :

Suatu relasi biner dari dua himpunan A dan B merupakan subhimpunan dari hasil ganda A x B. Jika (a, b) $\in$ R, maka ditulis " a R b ".

Definisi 2.2.3 :

Suatu fungsi dari A ke B adalah suatu relasi untuk setiap anggota dari A menentukan dengan tunggal satu anggota dalam B.

Definisi 2.2.4 :

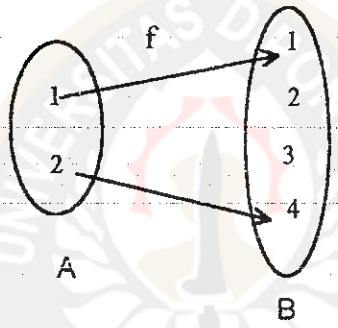
Diberikan dua himpunan A dan B. Suatu fungsi f adalah relasi biner pada A x B, untuk a $\in$A, dapat ditemukan tepat satu b $\in$ B, sedemikian sehingga

(a,b) $\in$ f. Himpunan A dinamakan daerah asal (domain) dan Himpunan B disebut daerah hasil (kodomain).

Definisi 2.2.5 :

Jika f suatu fungsi dan a $\in A_f$, maka terdapat dengan tunggal elemen y sedemikian sehingga (a,b) $\in$ f, dinotasikan dengan f(a) = b dan disebut dengan ***nilai*** (***value***) dari f pada a.

Contoh :



Gambar 2.1.1

f(1) = 1, nilai dari 1 adalah 1

f(2) = 4 nilai dari 2 adalah 4

2.3 Graph

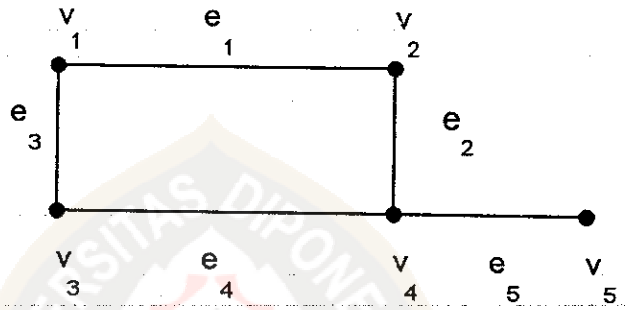
Definisi 2.3.1 :

Suatu Graph G dinotasikan G = (V,E) adalah himpunan *titik* (*vertex*) V, dimana V = (v_1 , v_2 , v_3 , . . . , v_n) yang berhingga dan tidak kosong, dan himpunan *garis* (*edge*) E, dimana E = (e_1 , e_2 , e_3 , . . . , e_m).

Definisi 2.3.2 :

Graph tak berarah adalah graph yang semua garisnya tidak mempunyai arah dan biasanya disebut dengan graph pada Definisi 2.3.1

Contoh :



Gambar 2.3.1

Gambar 2.3.1 adalah suatu graph dengan :

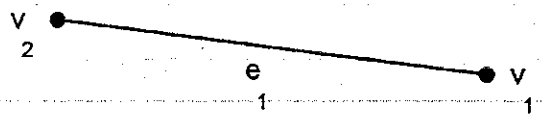
$V = \{ v_1 , v_2 , v_3 , v_4 , v_5 \}$

$E = \{ e_1 , e_2 , e_3 , e_4 , e_5 \}$

Definsi 2.3.3 :

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

Contoh :



Gambar 2.3.2

Pada Gambar 2.3.2 titik v_1 dan v_2 titik akhir dari e_1 .

Definisi 2.3.4 :

Suatu garis e_i dikatakan incident dengan titik v_j jika v_j titik akhir dari e_j.

Contoh :

Pada Gambar 2.3.1 garis e_1 dan e_2 incident dengan titik v_2 .

Definisi 2.3.5 :

Dua titik dikatakan adjacent jika dihubungkan oleh sebuah garis.

Contoh :

Pada Gambar 2.3.1 , titik v_1 adjacent dengan v_2 .

Definisi 2.3.6 :

Dua garis dikatakan adjacent jika mereka incident pada titik yang sama.

Contoh :

Pada Gambar 2.3.1 , e_1 dan e_2 adjacent.

Definisi 2.3.7 :

Derajat (***degree***) dari titik v_i dinotasikan d (v_i) adalah banyaknya garis yang incident dengan titik v_i .

Contoh :

Seperti pada Gambar 2.3.1 , maka :

d (v_1) = 2

Yang harus dicatat bahwa setiap garis dalam suatu barisan yang dibicarakan mempunyai satu titik akhir yang bersamaan dengan garis sebelumnya dan titik akhir lain dengan garis sesudahnya.

Contoh :

Dalam barisan (e_3 , e_4 , e_2) pada contoh sebelumnya, titik v_3 merupakan titik akhir dari e_4 juga merupakan titik akhir dari garis sebelumnya yaitu e_3, dan titik v_2 adalah titik akhir dari e_4 yang juga merupakan titik akhir dari garis sesudahnya.

Jika garis yang demikian dalam suatu barisan muncul hanya sekali, barisan ini dinamakan suatu *train* garis.

Definisi 2.3.8 :

Suatu train garis disebut train garis open jika initial titik dan final titiknya berbeda, sebaliknya jika titik initial dan titik finalnya sama disebut train garis tertutup.

Contoh :

Pada Gambar 2.3.3 , barisan (e_3 , e_4 , e_2 , e_5 , e_7) merupakan train garis terbuka yang initial titiknya v_1 dan titik finalnya v_4 . Barisan (e_1 , e_4 , e_3) adalah train tertutup.

Definisi 2.3.9 :

Path antara titik v_i dan v_j adalah himpunan semua garis dalam train garis terbuka yang memenuhi sifat :

1. Titik initial dan titik finalnya adalah titik v_i dan v_j .

2. Setiap titik selain titik v_i dan v_j derajatnya dua dan titik v_i dan v_j derajatnya satu.

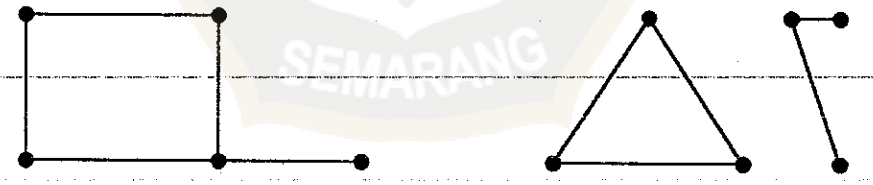
Contoh :

Himpunan semua garis (e_1 , e_4 , e_5) pada Gambar 2.3.3 merupakan path antara titik v_1 dan v_2 .

Definisi 2.3.10 :

Graph terhubung (*connected graph*) adalah jika setiap pasang titik-titiknya dihubungkan oleh suatu path. Dan sebaliknya disebut *graph tak terhubung* (*disconnected graph*).

Contoh :



(a) (b)

Gambar 2.3.4

Gambar 2.3.4 (a) merupakan graph terhubung , sedangkan Gambar 2.3.4 (b) merupakan graph tak terhubung.

Definisi 2.3.11 :

Sirkuit adalah graph terhubung yang setiap titiknya berderajat dua. Dari definisi ini, dapat dilihat hahwa suatu himpunan semua garis dalam train garis tertutup akan menjaadi sirkuit jika derajat setiap titiknya dua.

Contoh :

Himpunan garis dalam train garis tertutup (e_1 , e_3 , e_4) pada Gambar 2.3.3 merupakan sirkuit tetapi himpunan garis dalam train garis tertutup (e_6 , e_7 , e_5, e_2 , e_4 , e_3) pada gambar yang sama bukan merupakan sirkuit.

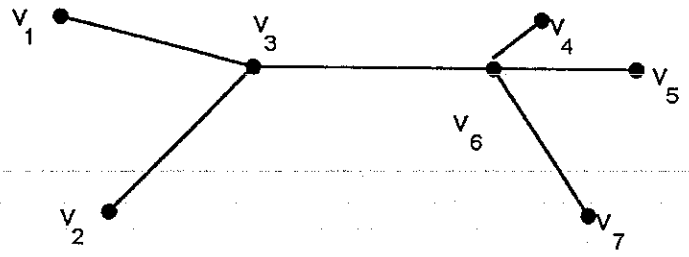
2.4 Pohon

Definsi 2.4.1 :

Pohon (*tree*) adalah suatu graph *terhubung* (*connected*) yang tidak memuat *sirkuit* (*circuit*).

Contoh :

Gambar 2.4.1 merupakan suatu tree.



Gambar 2.4.1

Teorema 2.4.1 :

Jika G = (V, E) suatu tree dan u , v berada dalam V, maka terdapat suatu path tunggal (*uni que*) antara titik u dan v.

Untuk selanjutnya dalam pembahasan tree , titik dalam tree disebut dengan *simpul* (*node*) .

Bukti :

G suatu tree, maka setiap simpulnya terhubung. Akan dibuktikan bahwa terdapat suatu path tunggal antara u dan v, jika u , v ∈ V. Akan dibuktikan dengan menggunakan kontradiksi.

Misalkan terdapat dua path P_1 dan P_2 yang menghubungkan u dan v dalam G, sehingga $P_1 \neq P_2$. Ambil suatu path e yang berada dalam P_1 sedemikian sehingga e tidak berada dalam P_2 .

Maka $(P_1 \cup P_2) - \{e\}$ terhubungkan, konsekuensinya $P \cup \{e\}$ adalah suatu sirkuit dalam G. Hal ini menunjukkan adanya kontradiksi, karena G suatu tree. Yang benar path antara u dan v adalah tunggal. ■

Teorema 2.4.2 :

Diberikan graph G = (V, E) . Jika setiap pasangan u dan v suatu simpul dalam G terdapat path tunggal antara u dan v, maka G terhubung dan e=v-1.

Bukti :

Ambil sebarang pasangan u dan v sehingga dalam G terdapat path tunggal antara u dan v. Jelas bahwa G terhubung. Akan dibuktikan bahwa e = v - 1 dengan menggunakan cara induksi matematik. Ambil

S = { n ∈ N; jika G graph dengan n simpul sedemikian sehingga setiap pasangan u dan v terdapat path tunggal dalam G, maka e = v - 1 }

Jika G suatu graph dengan satu simpul, maka G tidak mempunyai garis. Dimana 1 ∈ S.

Misal n ∈N dan { 1, 2, 3, ..., n } ⊆ S. Ambil G = (V, E) suatu graph dengan n + 1 simpul yang memenuhi aturan bahwa antara u dan v terdapat path tunggal dalam G. Dipilih e = {a,b} anggota dari E. Karena e suatu path tunggal, maka tidak ada (a,b) dalam G - {e}. Maka G tak terhubung. Ambil G_1 dan G_2 yang merupakan komponen dari G - {e} . Maka untuk setiap i = 1, 2 dan setiap pasangan simpul c dan d mempunyai path tunggal antara c dan d dalam G. Jika terdapat dua path dalam G_i , maka juga terdapat dua path dalam G. Maka jumlah semua simpul untuk G_i adalah kurang dari atau sama dengan n dan { 1, 2 , 3 , ... , n) ⊆ S, $e(G_i) = v(G_i) - 1$. Maka

$$e(G) = e(G_1) + e(G_2) + 1$$

$$= [\, v(G_1) - 1 \,] + [\, v(G_2) - 1 \,] + 1$$

$$= v(G) - 1$$

Karena $n + 1 \subseteq S$, maka bukti selesai. ■

Akibat 2.4.1 :

Jika G suatu tree, maka e = v - 1.

Bukti :

Diberikan G suatu tree. Maka, dengan Teorema 2.3.1, untuk setiap pasangan u dan v terdapat path tunggal dalam G. Sehingga, dengan Teorema 2.3.2 diperoleh e = v - 1. ■

Definisi 2.4.2 :

Suatu tree yang satu simpulnya dijadikan *akar (root)* simpul-simpul yang lain disebut *tree berakar* (*rooted tree*).

Teorema 2.4.3:

Jika G = (V, E) suatu tree yang tidak kosong, maka G paling sedikit mempunyai dua simpul dengan degree 1.

Bukti :

Diberikan G = (V, E) suatu tree yang tidak kosong, maka G terhubung dan $E \neq \varnothing$. Akan dibuktikan bahwa V mempunyai paling sedikit dua simpul yang masing-masing mempunyai degree satu.

Dengan mengingat persamaan 2.3.1 :

$$\sum_{i=n}^{n} \deg(v_i) = 2e$$

dan dengan Akibat 2.3.1 :

e = v - 1 , maka

2e = 2v - 2 ..... (2.3.2)

Karena tree dengan lebih dari satu simpul tidak memiliki simpul yang terisolasi, berarti sedikitnya ada dua simpul yang berada dalam tree yang memenuhi persamaan 2.3.2, yang masing-masing berderajat satu . Dengan demikian bukti selesai . ■

Definisi 2.4.3 :

Simpul dengan derajat satu dalam suatu tree disebut dengan *daun* (*leaf*)

Teorema 2.4.4 :

Jika G suatu graph terhubung sedemikian sehingga e = v - 1 , maka G suatu tree.

Bukti :

Akan dibuktikan dengan menggunakan kontradiksi. Diberikan suatu graph terhubung sedemian sehingga e = v - 1.

Misalkan G memiliki suatu sirkuit C_1 . Dipilih e_1 suatu garis di dalam C_1 . Maka G - $\{e_1\}$ terhubung, v (G - $\{e_1\}$) = v (G) dan e (G - $\{e_1\}$)= e (G) -1. Jika G - $\{e_1\}$ mempunyai sirkuit C_2 , dipilih garis e_2 dalam C_2 sedemikian sehingga G-$\{e_1 , e_2\}$ terhubung, v (G - $\{e_1 , e_2\}$) = v (G), dan e (G - $\{e_1 , e_2\}$)= e (G) - 2. Jika langkah ini dilanjutkan , akan dapat ditemukan graph terhubung H dengan v simpul yang tidak memiliki sirkuit dan memiliki garis yang kurang dari v - 1. Hal ini merupakan kontradiksi karena graph terhubung tanpa sirkuit adalat tree dan dengan Akibat 2.3.1 memiliki

e = v -1 ■

Teorema-torema 2.4.1 , 2.4.2 dan 2.4.4 merupakan suatu ekuivalensi dengan mengikuti statement dibawah dari suatu graph G = (V , E) :

1. G adalah suatu tree.
2. Jika u, v ∈V, dan u ≠ v maka terdapat path tunggal antara u dan v.
3. G terhubung, dan e = v - 1 .

Definisi 2.4.4 :

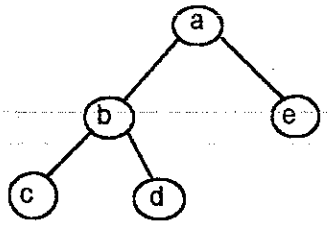
Jika uv adalah garis berarah dalam suatu tree berakar, maka u adalah *orang tua* (*parent*) dari v dan v adalah *anak* (*child*) dari u. Suatu *simpul dalam* (*internal node*) dari tree berakar adalah simpul yang mempunyai anak.

Dengan demikian setiap simpul dalam tree berakar keculi akar mempunyai satu orang tua, tetapi suatu simpul dapat mempunyai beberapa anak.

Definisi 2.4.5 :

Jika u dan v adalah simpul dalam suatu tree berakar, maka v adalah *keturunan* (*descendant*) dari u dimana $u \neq v$ dan u adalah simpul dari suatu path tunggal dari akar r menuju v. Simpul dengan orang tua yang sama disebut *bersaudara* (*sibling*). Jika u adalah simpul dari suatu tree berakar, subtree dengan akar u adalah tree yang terdiri atas u, keturunan u, dan semua garis berarah dari u menuju keturunan dari u, dan semua garis berarah dari satu keturunan dari u menuju yang lainnya.

Contoh :



Gambar 2.4.2

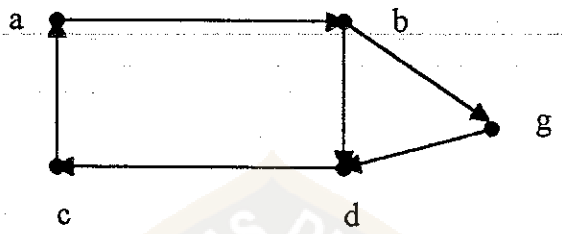
Gambar 2.4.2, merupakan suatu pohon berakar dengan akar a. Anak dari a adalah d dan e. Saudara c adalah d, dan subtree dengan akar b terdiri atas b dan kedua anaknya yaitu c dan d, serta garis yang menghubungkan b dengan c dan d.

Definisi 2.4.6 :

Jika uv adalah garis berarah dalam suatu graph berarah, maka u disebut *simpul initial* (*initial node*) dan v dinamakan *simpul terminal* (*terminal node*) dari suatu graph berarah. Jika v simpul dari suatu graph berarah G, *indegree* dari v adalah jumlah garis berarah dalam G yang dimiliki v sebagai simpul terminal dan *outdegree* dari v adalah jumlah garis berarah dalam G yang dimiliki v sebagai simpul initial.

Contoh :

Diberikan suatu graph berarah seperti pada Gambar 2.4.3 :



Gambar 2.4.3

Indegree dari simpul a adalah 1, indegree dari simpul b adalah 1, indegree dari simpul c adalah 1, indegree dari simpul d adalah 2, indegree dari simpul g adalah 1. Sedangkan outdegree dari simpul a adalah 1, outdegree dari simpul b adalah 2, outdegree dari simpul c adalah 1 , outtdegree dari simpul d adalah 1 dan outdegree dari simpul g adalah 1.

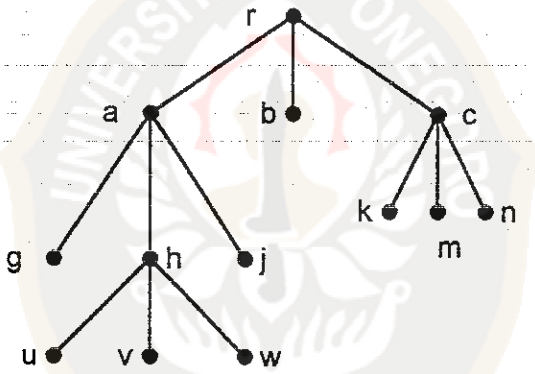
Definisi 2.4.7 :

Diberikan n ∈N. Suatu tree berakar T adalah *m-ary tree* jika outdegree dari setiap simpul kurang dari atau sama dengan m. Suatu 2-ary tree dinamakan *pohon biner* (*binary trees*). Suatu *regular ary tree* adalah m-ary tree sedemikian sehingga outdegree dari setiap simpul kecuali daun adalah m. Dengan demikian, *regular binary tree* adalah pohon biner sedemikian sehingga outdegree dari setiap simpul kecuali daun adalah dua.

Definisi 2.4.8 :

Diberikan T suatu tree berakar dengan akar r. ***Tingkat*** (***level***) dari r adalah 0, dan tingkat dari simpul v ($v \neq r$) adalah panjang dari path tunggal dari r ke v. ***Tinggi*** (***height***) dari T adalah ltingkat maksimum dari suatu simpul dalam T.

Contoh :



Gambar 2.4.4

Dari Gambar 2.4.4 dapat diketahui bahwa tingkat dari simpul r sama dengan 0. Tingkat dari simpul-simpul g, h, j, k, m dan n sama dengan 2 dan tingkat dari simpul-simpul u, v, dan w adalah 3. Dengan demikian tinggi tree adalah 3.

Definisi 2.4.9 :

Diberikan m $\in$ N. Suatu full m-ary tree (full binary tree) adalah regular m-ary tree (regular binary tree) sedemikian sehingga tingkat dari setiap daun adalah tinggi dari tree.

Teoreme 2.4.5 :

Diberikan m $\in$N. Jika T suatu full m-ary tree dengan tinggi h, maka T mempunyai m^h daun, $(m^h - 1)/(m - 1)$ orang tua dan $(m^{h+1} - 1)/(m - 1)$ simpul.

Bukti :

Diberikan T suatu full m-ary tree mempunyai h dengan akar r. Karena T adalah regular m-ary tree, maka terdapat m simpul , v_1 , v_2 , v_3 , . . . , v_m dengan level number 1. Dimana untuk setiap i = 1, 2, ... , m , v_i mempunyai m anak. Maka terdapat m^2 simpul dengan level number 2. Apabila langkah ini diteruskan untuk setiap n $\leq$ h jumlah dari simpul dengan level number n adalah m^n . Karena jumlah daun dari T adalah m^h maka jumlah simpul dalam tree adalah:

$$\sum m^i = \frac{m^{h+1}-1}{m-1}$$

Akhirnya , jumlah orang tua adalah :

$$\frac{m^{h+1}-1}{m-1} - m^h = \frac{m^{h+1}-1-m^k(m-1)}{m-1}$$

$$= \frac{m^m-1}{m-1}$$

■

Definisi 2.4.10 :

Jika suatu orang tua dari binary tree terurut T memiliki dua anak, anak pertama disebut *anak kiri* (*left child*) dan anak kedua disebut *anak kanan* (*right child*) . Subtree dari T dimana akarnya adalah anak kiri dari orang tua u disebut *subtree kiri (left subtree*) dari u, dan subtree dari T dimana akarnya adalah anak kanan dari orang tua v disebut *subtree kanan* (*right subtree*) dari v.

2.5 Notasi Big O

Definisi 2.5.1 :

Diberikan S, himpunan semua fungsi yang mempunyai domain D. Dimana D = N dan R yang mewakili himpunan bilangan bulat dan riil. Fungsi f(x) dan g(x) anggota dari S. Dikatakan bahwa f(x) = O (g(x)) , dibaca " f(x) adalah big O dari g(x) " , jika dapat ditemukan bilangan positif c dan k sedemikian sehingga :

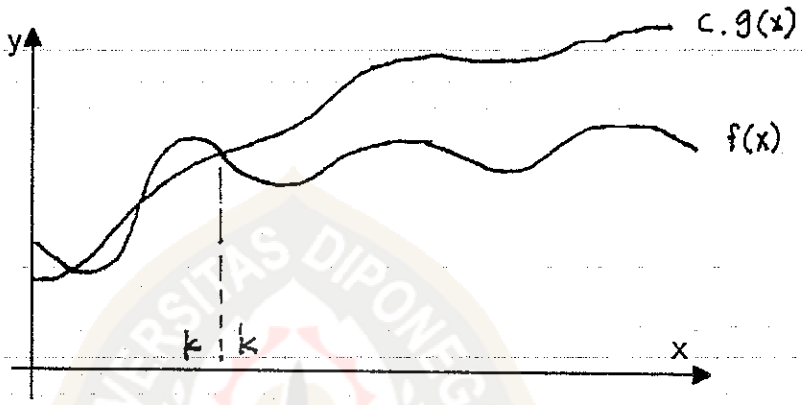
$$|f(x)| \leq c.|g(x)|$$

untuk semua x $\in$D dan x > k .

Suatu fungsi f(x) termasuk dalam himpunan O(g(x)) jka dapat ditemukan bilangan positif c dan k sedemikian sehingga f(x) kurang dari c. g(x) untuk suatu harga x > k . Penulisan f(x) = O(g(x)) menunjukkan bahwa f(x) anggota dari O(g(x))

atau $f(x) \in O(g(x))$. Gambar 2.4.1 memberikan gambaran dari fungsi f(x) dan g(x) dimana

$f(x) \in O(g(x))$.



Gambar 2.4.1

Contoh :

Diberikan S, himpunan semua fungsi dengan domain R, bilangan riil. Fungsi f(x) dan g(x) anggota dari S dan didefinisikan oleh :

$f(x) = x^3 + 7x^2 + 11x$ dan $g(x) = x^3$ maka $f(x) = O(g(x))$

Analisa :

Dipilih c = 19 dan berlaku untuk x > 1, diperoleh :

$$|f(x)| = x^3 + 7x^2 + 11x$$
$$\leq x^3 + 7x^3 + 11x^3$$
$$= 19\, x^3$$
$$\leq c.|g(x)|$$

dengan demikian f (x) = O(g(x)).

Teorema 2.5.1 :

Diberikan suatu fungsi polinomial berderajat n , $f(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$ dan $g(x) = x^n$, maka $f(x) = O(g(x))$.

Bukti :

Dipilih $c = |a_n| + |a_{n-1}| + \ldots + |a_1| + |a_0|$ dan $x > 1$ untuk $x \in R$, maka :

$$|f(x)| = |a_n.x^n + a_{n-1}.x^{n-1} + \ldots + a_1.x + a_0$$

$$\leq |a_n.x^n| + |a_{n-1}.x^{n-1}| + \ldots + |a_1.x| + |a_0|$$

$$\leq |a_n|.x^n + |a_{n-1}|.x^{n-1} + \ldots + |a_1|.x + |a_0|$$

$$\leq |a_n|.x^n + |a_{n-1}|.x^n + \ldots + |a_1|.x^n + |a_0|x^n$$

$$\leq (|a_n| + |a_{n-1}| + \ldots + |a_1| + |a_0|).x^n$$

$$= c.x^n$$

$$= c.|g(x)|$$

Maka terbukti bahwa $f(x) = O(g(x))$. ■

Teorema 2.5.2 :

Diberikan S, himpunan semua fungsi dengan domain D, dimana D = N dan R dengan kodomain (0, ~). Fungsi-fungsi $f_1(x)$, $f_2(x)$, $g_1(x)$ dan $g_2(x)$ anggota S, sedemikian sehingga $f_1(x) = O(g_1(x))$ dan $f_2(x) = O(g_2(x))$, maka berlaku :

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

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

Bukti :

a. Karena $f_1(x) = O(g_1(x))$ dan $f_2(x) = O(g_2(x))$, maka terdapat c_1 , c_2 , k_1 dan k_2 sedemikian sehingga jika $x \in D$ dan $x > k_1$, maka berlaku $|f_1(x)| \le c_1 |g_1(x)|$, dan jika $x > k_2$ maka $|f_2(x)| \le c_2 |g_2(x)|$

Ambil $x \in D$ sedemikian sehingga $x > k_1$ dan $x > k_2$ serta dipilih $k = \max(k_1, k_2)$ dan $c = c_1 + c_2$, maka

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

Maka terbukti bahwa $f_1(x) + f_2(x) = O(\max \{g_1(x) , g_2(x)\})$ ■

b. Karena $f_1(x) = O(g_1(x))$ dan $f_2(x) = O(g_2(x))$, maka terdapat c_1 , c_2 , k_1 dan k_2 sedemikian sehingga jika $x \in D$ dan $x > k_1$, maka berlaku $|f_1(x)| \le c_1 |g_1(x)|$, dan jika $x > k_2$ berlaku $|f_2(x)| \le c_2 |g_2(x)|$.
Selanjutnya dipilih $k = \max \{ k_1 , k_2 \}$ dan $c = c_1 . c_2$, maka

$$|f_1(x) . f_2(x)| = |f_1(x)| + |f_2(x)|$$
$$\le c_1 . |g_1(x)| . c_2 |g_2(x)|$$

$$= c_1.c_2.|g_1(x)|.g_2(x)$$
$$= c.|g_1(x).g_2(x)|$$

Sehingga $f_1(x) . f_2(x) = O (g_1(x) . g_2(x))$ ■

2.6 Menghitung Running Time

Seperti telah disinggung dalam pendahuluan bahwa pemilihan suatu algoritma untuk menyelesaikan suatu permasalahan salah satu faktor yang menjadi pertimbangan adalah kecepatan algoritma yang dipilih dalam menyelesaikan permasalahan yang dihadapi. Dalam suatu analisa algoritma untuk menentukan kecepatan algoritma digunakan istilah running time, dinotasikan dengan T(n).

Untuk menentukan running time dari suatu algoritma tergantung dari beberapa faktor, antara lain adalah :

1. Masukan atau input yang digunakan.
2. Kemampuan kompiler yang digunakan dalam menjalankan program.
3. Kecepatan dari mesin dalam menjalankan instruksi dari suatu program.
4. Kompleksitas waktu dari algoritma yang digunakan.

Suatu kenyataan bahwa running time suatu algoritma tergantung terhadap masukan atau input yang akan diselesaikan, sehingga dikatakan bahwa running time suatu algoritma didefinisikan sebagai suatu fungsi dari masukan, masukan disini adalah jumlah data yang akan diproses.

Untuk faktor kedua dan ketiga, bahwa running time tergantung terhadap kompiler dan mesin yang digunakan dalam menyelesaikan program juga

merupakan faktor yang cukup mempengaruhi kecepatan proses. Akan tetapi kedua faktor tersebut bersifat sangat teknis dan tergantung pada komputer yang digunakan. Oleh karena itu dalam menentukan running time suatu algoritma , diasumsikan bahwa running time yang diperoleh merupakan jumlahan operasi yang dilaksanakan oleh suatu komputer yang ideal.

Salah satu cara untuk menentukan jumlah operasi yang dilaksanakan oleh suatu algoritma adalah dengan penggunaan Notasi Big O atau O (f(n)), yang telah dibahas pada sub bab sebelumnya. Dimana penggunaan parameter n merupakan karakteristik dari masukan yang diberikan pada algoritma yang umumnya merupakan jumlah data yang diproses. Jadi bila n menyatakan jumlah data, maka penulisan O(f(n)) menunjukkan jumlah operasi yang dilakukan untuk memyelesaikan algoritma. Dalam hal ini dikatakan bahwa kompleksitas waktu algoritma tersebut adalah f(n).

Dalam melakukan analisa algoritma terdapat 3 hal yang sering diperhitungkan, yaitu :

1. Best case running time, yaitu suatu keadaan dimana algoritma diselesaikan dalam waktu tersingkat atau terbaik.
2. Worst case running time, adalah suatu keadaan terburuk dimana algoritma memerlukan waktu paling lama dalam menyelesaikan proses.
3. Average case running time, adalah analisa waktu rata-rata yang digunakan untuk melaksanakan suatu algoritma. Akan tetapi hal ini terkadang saja untuk

dilakukan, karena adanya kesulitan dalam menentukan suatu data yang secara umum dapat mewakili keadaan rata-rata data yang digunakan.

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 meyelesaikan proses.
2. Worst case running time dari suatu algoritma merupakan suatu batas atas dari suatu running time. Dengan demikian akan memberikan suatu jaminan bahwa algoritma yang digunakan akan mempunyai akhir proses.
3. Untuk beberapa algoritma, worst case running time merupakan kejadian yang sering terjadi. Sebagai contoh, dalam proses pencarian dalam 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 lainnya, maka dibutuhkan kemampuan untuk mengidentifikasi algoritma secara signifikan. Walaupun begitu, diperlukan suatu kerangka utama dalam menganalisa suatu algoritma sehingga memberi kemudahan dalam melakukan analisa, sehingga menjadi sederhana dan mudah dipelajari.

Kerangka utama yang dimaksud adalah dipandangnya running time sebagai jumlahan dari operasi yang dilakukan dalam menyelesaikan algoritma yang dimaksud. Suatu konstanta dari waktu diberikan untuk melaksanakan proses dari masing-masing instruksi dalam algoritma. Satu tahapan proses mungkin

memberikan jumlah waktu yang berbeda dari tahapan yang lain, akan tetapi diasumsikan bahwa masing-masing eksekusi tahapan ke-i memerlukan *waktu* (*times*) c_i , dimana c_i suatu konstanta.

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

2. Loop berkalang

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

3. Statement berurutan

   Apabila terdapat statement berurutan, maka penentuan running timenya menggunakan running time maksimal dari running time yang ada, sesuai dengan Teorema 2.5.1.

4. Keadaan if-then-else

Untuk suatu penggalan algoritma

1. if (kondisi) then
2. S_1
3. else
4. S_2

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

Contoh :

Diberikan suatu algoritma insertion sort, yaitu algoritma untuk memasukkan suatu data ke dalam urutan yang benar, seperti yang dilakukan dalam suatu permainan bridge atau yang lainnya sehingga diperoleh urutan data dari kecil ke besar.

```
INSERTION_SORT(A)
1.for j ← 2 to length[ A ]
2.      do key ←  A[ j ]
3.      penyisipan A[ j ] ke dalam
            barisan A[ 1 . .    j - 1 ]
4.      i ← j - 1
5.      while 1 > 0 and  A[ i ] > key
6.              do  A[ i + 1 ] ← A[ i ]
7.                  i ← i - 1
8.      A[ i + 1 ] ← key
```

Analisa :

Algoritma insertion sort diwakili oleh prosedur insertion sort yang menggunakan parametre array A[1 . . n] yang berisi bilangan dengan panjang n buah yang akan diurutkan.

Dalam menyisipkan suatu key ke tempat yang seharusnya maka harus dilakukan perbandingan-perbandingan secara bergantian. Key akan bergeser ke kiri dengan cara dibandingkan dengan A[j] berikutnya dan kemudian key disisipkan ke tempatnya. Proses dilanjutkan untuk elemen-elemen disebelah kiri A[j].

Dimana proses akan berhenti bila salah satu keadaan berikut terpenuhi :

1. Salah satu elemen A[j] mempunyai nilai yang lebih kecil dari key.
2. Bagian ujung kiri data telah tercapai.

Perlu diingat bahwa running time dari algoritma insertion sort tergantung pada ukuran atau masukan yang diproses, dimana pengurutan dengan ukuran input yang lebih besar akan memerlukan waktu yang lebih besar dari pada input dengan ukuran yang lebih kecil. Akan tetapi untuk ukuran input yang sama juga dapat memberikan waktu yang tidak sama, karena proses pengurutan sangat tergantung dari keadaan awal input.

Secara umum diberikan cost waktu dan jumlah waktu untuk masing-masing statemen yang dijalankan untuk setiap j = 1, 2, ... n , dimana n = lenght[A], juga diberikan t_j yang merupakan jumlah waktu dari

putaran while pada baris ke lima. Diasumsikan bahwa suatu komentar diberikan cost waktu nol.

Sehingga diperoleh :

INSERTION_SORT(A)	cost	times
1.for j ← 2 to length[A]	c_1	n
2. do key ← A[j]	c_2	n - 1
3. penyisipan A[j] ke dalam barisan A[1 . . j - 1]	0	n - 1
4. i ← j - 1	c_4	n - 1
5. while 1 > 0 and A[i] > key	c_5	Σt_j
6. do A[i + 1] ¬ A[i]	c_6	$\Sigma (t_j - 1)$
7. i ← i - 1	c_7	$\Sigma (t_j - 1)$
8. A[i + 1] ← key	c_8	n - 1

Suatu statemen memberikan cost waktu c_i untuk dieksekusi yang dilakukan sebanyak n kali akan memberikan kontribusi sebesar c_i . n . Maka running time dari algoritma tersebut adalah :

$$T(n) = c_1.n + c_2.(n-1) + c_4.(n-1) + c_5.\Sigma t_j + c_6.\Sigma(t_j - 1) + c_6.\Sigma(t_j - 1) + c_7.\Sigma(t_j - 1) + c_8.(n-1)$$

Seperti pernah disinggung sebelumnya bahwa dalam menentukan running time dari algoritma insertion sort tergantung dari keadaan awal input yang digunakan, yang berpengaruh terhadap terjadinya best case atau worst case running time.

a. Keadaan best case running time.

Keadaan ini terjadi jika array input telah terurut dari kecil ke besar , sehingga untuk setiap j = 2, 3, . . . , n dapat ditemukan A[j] key pada baris ke lima. Sehingga baris ke lima dilaksanakan sekali atau $t_j = 1$. Dengan demikian diperoleh :

$$T(n) = c_1 . n + c_2.(n-1) + c_4.(n-1) + c_5.(n-1) + c_8.(n-1)$$

$$= (c_1 + c_2 + c_4 + c_5 + c_8) n - (c_2 + c_4 + c_5 + c_8)$$

$$= a.n + b$$

$$= O (n)$$

dimana konstanta a dan b tergantung dari nilai cost c_1 , maka dikatakan bahwa best case running time algoritma insertion sort mempunyai kompleksitas waktu linier, O (n).

b. Keadaan worst case running time.

Keadaan ini terjadi jika array input dalam keadaan terurut terbalik, yaitu dari besar ke kecil , dimana untuk setiap A[j] harus dibandingkan dengan A[1 . . j-1] , sehingga $t_j = j$ untuk j = 2, 3, . . , n .

Diperoleh :

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

dan

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

Maka worst case running timenya adalah

$$T(n) = c_1.n + c_2.(n - 1) + c_4.(n - 1) + c_5.(\tfrac{1}{2}(n + 1)n - 1) +$$

$$c_6.(\tfrac{1}{2}(n - 1)n) + c_7.(\tfrac{1}{2}(n - 1)n) + c_8.(n - 1)$$

$$= \tfrac{1}{2}(c_5 + c_6 + c_7)\, n^2 + \tfrac{1}{2}(2.c_1 + 2.c_2 + 2.c_4 + c_5 - c_6 - c_7 + 2.c_8)\, n - c_5$$

$$= p.n^2 + q.n + r$$

$$= O(n^2)$$

Dengan demikian worst case running timenya mempunyai kompleksitas kuadratik, $O(n^2)$.