BAB II

MATERI PENUNJANG

2.1. Konsep Dasar Pohon

2.2.1. Graf

Teori graf merupakan salah satu bagian yang paling penting dalam matematika. Pada teori graf diberikan model matematika untuk setiap himpunan dari sejumlah objek diskret, di mana beberapa pasangan unsur dari himpunan tersebut terikat menurut suatu aturan suatu aturan tertentu.

Definisi 2.1 :

Graf G = (V,E) merupakan suatu pasangan (V,E) , dimana V adalah himpunan terbatas yang terdiri atas simpul-simpul dan E adalah himpunan sisi yang merupakan relasi biner dari V.

Pada graf berarah (*directed graph*) , sisi dinyatakan sebagai pasangan simpul yang terurut, sedangkan pada graf tidak berarah (*undirected graph*) sisi merupakan pasangan simpul yang tidak terurut.

Jika (u,v) adalah sisi pada graf G=(V,E) , maka (u,v) disebut *insident* pada simpul u dan v, dan simpul u *adjacent* dengan simpul v.

Derajat (*degree*) simpul v adalah banyaknya sisi yang insident pada simpul v.

Pada graf berarah , derajat suatu simpul dibedakan menjadi 2 yaitu derajat masuk dan derajat keluar.

Derajat masuk (*in-degree*) simpul v adalah banyaknya sisi yang masuk/menuju ke simpul v.

Derajat keluar (*out-degree*) simpul v adalah banyaknya sisi yang keluar/meninggalkan simpul v.

Lintasan (*path*) dari simpul u ke simpul u' adalah barisan $\{v_0,v_1,v_2,\ldots,v_k\}$ sedemikian rupa sehingga $u = v_0$, $u' = v_k$ dan $(v_{i-1},v_i) \in E$.

Panjang lintasan adalah banyaknya sisi yang terdapat pada lintasan.

Lintasan disebut sederhana (*simple*) jika seluruh simpul dalam lintasan tersebut tidak ada yang sama.

Suatu lintasan $\{v_0,v_1,v_2,\ldots,v_k\}$ membentuk suatu sikel (*cycle*) jika $v_0 = v_k$ dan lintasan mengandung paling sedikit satu sisi. Sikel disebut sederhana jika v_1 , v_2 ,…, v_k merupakan simpul-simpul yang berbeda.

Graf yang tidak mengandung sikel bersifat asiklik (*acyclic*).

Suatu graf disebut terhubung (*connected*), jika untuk setiap pasangan simpul yang berbeda dihubungkan oleh suatu lintasan.

2.1.2. Pohon

Struktur pohon merupakan salah satu bentuk khusus dari struktur graf.

Definisi 2.2 :

Pohon (*tree*) adalah suatu graf tidak berarah yang bersifat asiklik dan terhubung.

Hutan (*forest*) adalah suatu graf tidak berarah yang bersifat asiklik dan tidak terhubung.

Pohon berakar (*rooted tree*) adalah suatu pohon yang mempunyai sebuah simpul yang dibedakan dari simpul yang lain. Simpul tersebut disebut akar (*root*).

Misal T adalah pohon berakar dengan akar r.

Jika simpul y berada pada lintasan dari r ke simpul x, maka simpul y disebut moyang (*ancestor*) dari x.

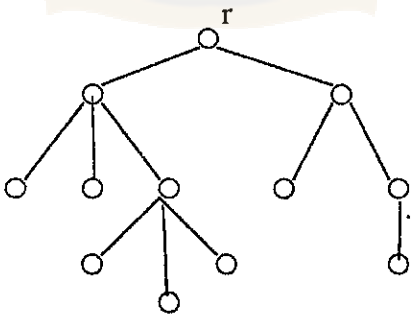
Jika simpul y adalah moyang dari simpul x, maka simpul x disebut keturunan (*descendant*) dari simpul y.

Jika sisi terakhir pada lintasan dari r ke simpul x adalah (y,x) maka simpul y disebut induk (*parent*) dari simpul x.

Jika simpul y adalah induk dari simpul x, maka simpul x disebut anak (*child*) dari simpul y.

Dua buah simpul yang mempunyai induk yang sama disebut bersaudara (*siblings*).

Contoh 2.1:



Gambar 2.1 : Pohon berakar dengan akar r

Suatu simpul yang tidak mempunyai anak disebut daun (*leaf*) atau simpul eksternal.

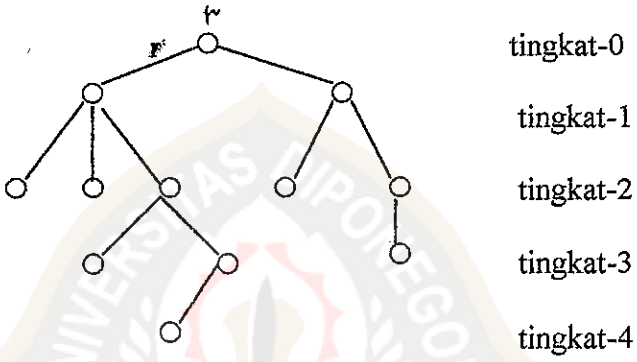
Suatu simpul yang mempunyai anak disebut cabang (*branch*) atau simpul internal.

Derajat suatu simpul adalah banyaknya anak dari simpul tersebut.

Tingkat (*level*) simpul x adalah panjang lintasan dari akar ke simpul x.

Tinggi (*depth*) suatu pohon adalah tingkat terbesar suatu simpul dalam pohon.

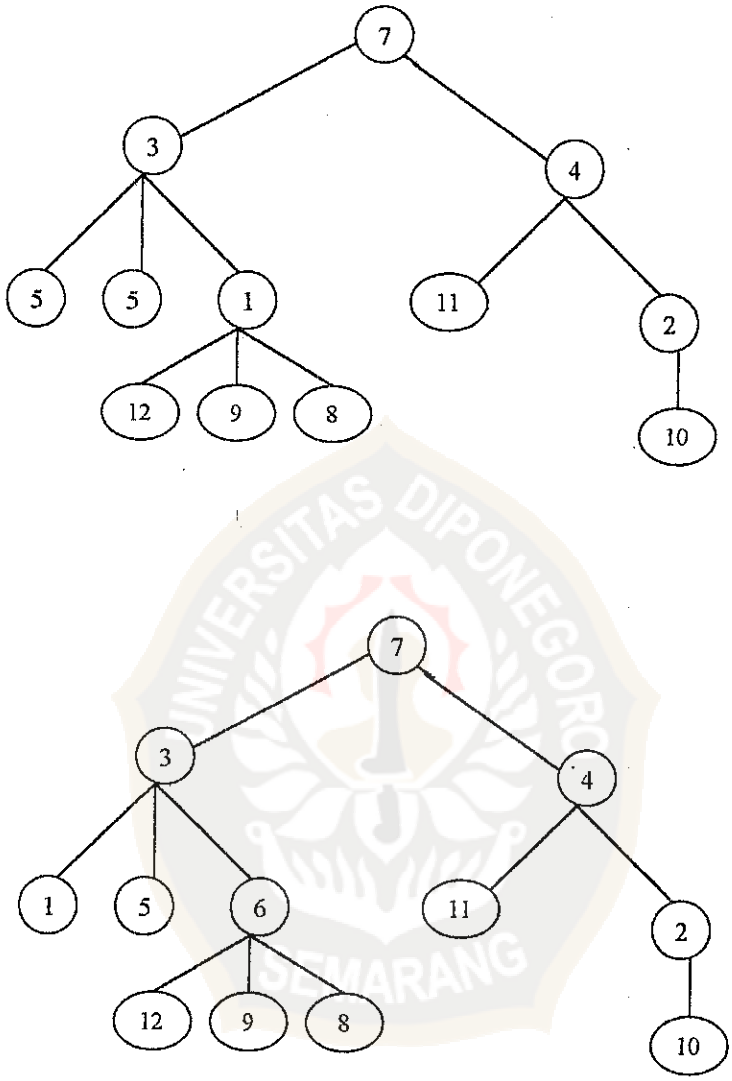
Contoh 2.2 :



Gambar 2.2 : Pohon dengan tinggi = 4

Pohon terurut (*ordered tree*) merupakan pohon berakar dengan setiap anak dari setiap simpulnya terurut, artinya jika suatu simpul v mempunyai sebanyak k buah anak, maka terdapat anak pertama, anak kedua, ... dan anak ke- k ,dimulai dari kiri ke kanan , dengan masing-masing simpul berisi satu kunci..

Contoh 2.3 :



Gambar 2.3 : Dua buah pohon terurut yang berbeda

Pada gambar di atas , kedua pohon adalah identik sebagai pohon berakar, tetapi tidak identik sebagai pohon terurut karena terdapat simpul dengan urutan berbeda, yaitu urutan anak-anak simpul 3.

Definisi 2.3 :

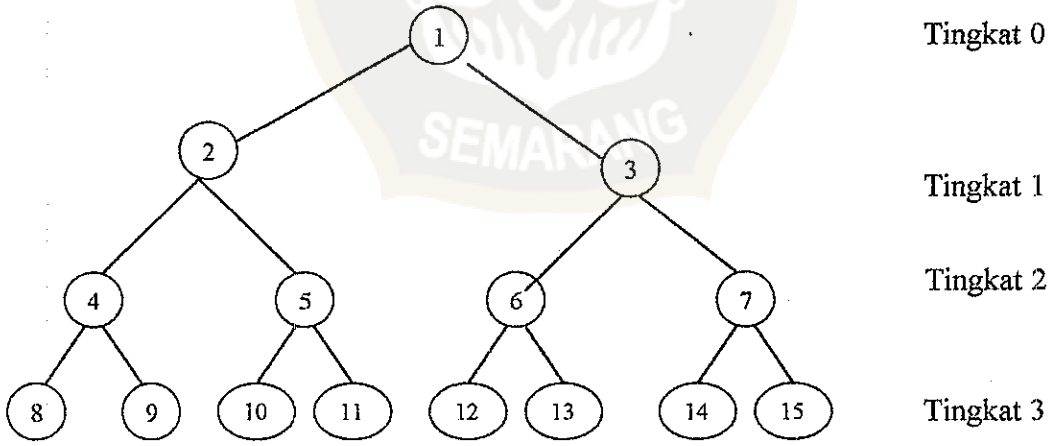
Pohon biner (*binary tree*) adalah suatu struktur pohon yang yang terdiri atas sejumlah terbatas simpul dimana simpul-simpulnya terbagi ke dalam tiga himpunan yang saling lepas, yaitu akar, sub pohon kiri (*left subtree*) dan sub pohon kanan (*right subtree*).

Pada pohon biner , jika anak pohon kiri tidak kosong maka akarnya disebut sebagai anak kiri (*left child*) dari akar pohon. Jika anak pohon kanan tidak kosong maka akarnya disebut sebagai anak kanan (*right child*) dari akar pohon.

Pada pohon biner, derajat maksimum suatu simpul adalah 2.

Pohon biner komplet (*complete binary tree*) adalah pohon biner yang setiap simpul internalnya mempunyai derajat 2 dan seluruh daunnya berada pada tingkat yang sama.

Contoh 2.4 :



Gambar 2.4 : Pohon biner komplet dengan tinggi 3

2.2. Dasar-Dasar Algoritma

Kata algoritma diambil dari nama seorang penulis buku matematika dari Persia, Al Khowarizmi (825 M), yang diartikan sebagai suatu metode khusus untuk memecahkan masalah. Kemudian kata ini dimasukkan ke dalam bidang komputer dan diartikan sebagai suatu metode yang dapat digunakan oleh komputer untuk menyelesaikan suatu pemasalahan.

Algoritma merupakan suatu prosedur yang terdiri atas sejumlah terbatas langkah-langkah yang masing-masing memerlukan satu atau lebih operasi.

Ada beberapa sifat algoritma sehingga dapat dioperasikan pada komputer, yaitu :

1. Setiap langkah atau operasi harus tertentu (*definite*), yang berarti harus jelas apa yang harus dikerjakan atau tidak mengandung lebih dari satu kemungkinan yang dikerjakan. Contoh perintah yang tidak dibenarkan :

   " tambahkan 6 atau 7 pada x "

   Perintah tersebut tidak diijinkan karena tidak jelas bilangan yang akan ditambahkan pada x.
2. Setiap langkah atau operasi harus efektif (*effectiveness*), yang berarti sedapat mungkin dengan sedikit prinsip dan ditulis dalam waktu terbatas, sehingga setiap langkah memang dibutuhkan untuk memperoleh hasil.
3. Algoritma memiliki input, artinya suatu algoritma berasal dari masalah kemudian diberi satu atau lebih masukan untuk diproses.

4. Algoritma memiliki output, artinya suatu algoritma menghasilkan satu atau lebih keluaran / hasil.
5. Algoritma bersifat *finite*, artinya algoritma berhenti setelah sejumlah terbatas langkah.

Suatu algoritma ditulis ke dalam bahasa pemrograman. Banyak bahasa pemrograman didesain sehingga setiap kata memiliki arti yang unik. Kadang terdapat kata yang digunakan sinonim dalam program seperti *procedure* atau *subroutine*.

2.3. Kompleksitas Algoritma

Analisa algoritma berhubungan dengan proses memperoleh perkiraan dari waktu dan ruang yang diperlukan untuk mengeksekusi algoritma. Kompleksitas algoritma mengacu pada jumlah waktu dan ruang yang dibutuhkan untuk mengeksekusi algoritma.

Beberapa terminologi yang digunakan untuk mendeskripsikan kompleksitas waktu algoritma tercantum dalam tabel 2.1.

Tabel 2.1 Terminologi Kompleksitas Algoritma

Kompleksitas	Terminolgi
O(1)	Kompleksitas konstanta
O(log n)	Kompleksitas logaritma
O(n)	Kompleksitas linier
O(n log n)	Kompleksitas n log n
$O(n^b)$	Kompleksitas polinomial
$O(b^n)$	Kompleksitas eksponensial
O(n!)	Kompleksitas faktorial

Setiap running time dengan kompleksitas yang berbeda membutuhkan waktu yang berbeda untuk menyelesaikan suatu program. Tabel 2.2 menunjukkan waktu eksekusi untuk beberapa jumlah input pada beberapa running time dengan kompleksitas yang berbeda.

Tabel 2.2. Waktu eksekusi untuk beberapa jumlah data pada beberapa running time

T(n)	Jumlah Data			
	n = 10	n = 100	n = 1000	n = 10000
33n	0.00033 sec	0.003 sec	0.033 sec	0.33 sec
46 n log n	0.0015 sec	0.03 sec	0.45 sec	6.1 sec
13 n^2	0.0013 sec	0.13 sec	13 sec	22 menit
3,4 n^3	0.0034 sec	3.4 sec	0.94 jam	39 hari
2^n	0.001 sec	4×10^{14} abad		

2.4. Pengurutan

Pengurutan (*Sorting*) diartikan sebagai suatu proses penyusunan sekumpulan objek ke dalam suatu urutan tertentu.

Tujuan pengurutan adalah untuk mendapatkan kemudahan dalam pencarian anggota dari suatu himpunan.

Proses ini merupakan aktivitas yang penting, khususnya dalam pengolahan data. Ketertarikan utama pada pengurutan adalah hal teknik dasar yang digunakan untuk membentuk sejumlah algoritma.

Pengurutan merupakan subjek yang ideal untuk menunjukkan adanya sejumlah variasi algoritma dengan tujuan yang sama. Contoh-contoh objek yang terurutkan dapat kita jumpai dalam buku telepon, kamus, katalog buku perpustakaan dan sebagainya.

Dimisalkan rinci data : a_1, a_2, a_3, ... , a_n, maka pengurutan akan menukar posisi setiap rinci data ke dalam suatu urutan a_{k1}, a_{k2}, ... a_{kn} sedemikian rupa sehingga terdapat fungsi pengurutan $f(a_{k1}) < f(a_{k2}) < \ldots < f(a_{kn})$

Metode pengurutan dapat dimasukkan kedalam 2 kategori yaitu pengurutan larik (*array*) dan pengurutan berkas (*sequential*). Pengurutan larik disebut juga pengurutan internal, dimana larik diurutkan kedalam suatu penyimpanan internal, berkecepatan tinggi yaitu *Random Access Memory*.

Sedangkan pengurutan berkas disebut juga pengurutan eksternal dimana berkas diletakkan dalam suatu media penyimpanan eksternal yang lebih besar.

Beberapa contoh pengurutan larik antara lain : heapsort, quicksort, bubblesort, shellsort, insertionsort, selectionsort dan lain-lain.

Contoh pengurutan berkas antara lain : polyphasesort dan mergesort.

Dalam pengurutan harus diperhatikan bahwa metode yang digunakan harus ekonomis dalam penggunaan media penyimpan yang tersedia dan pertimbangan waktu.

2.5 Menghitung Running Time

2.5.1. Running Time Program

Saat menyelesaikan suatu masalah, sering dihadapkan pada beberapa pilihan algoritma. Terdapat dua hal yang dijadikan dasar memilih algoritma yaitu :

1. Algoritma yang mudah dimengerti serta mudah dalam pengkodean dan pengecekan kesalahan, terutama untuk program yang digunakan lebih dari sekali atau program yang rumit.
2. Algoritma yang efisien dalam penggunaan komputer atau dapat dijalankan secepat mungkin . Ukuran untuk mengetahui efisiensi program adalah running time program.

Definisi 2.4 :

Running time adalah ukuran waktu sebagai fungsi dari besar data masukan untuk menjalankan suatu program sehingga menghasilkan output.

Running time dinotasikan dengan notasi T(n) , dengan n adalah besar data masukan / input. Running Time program tergantung pada faktor-faktor seperti :

1. Input program
2. Kualitas kompiler (kecepatan mesin) dalam menjalankan program
3. Kompleksitas waktu.

Untuk menghitung running time program digunakan fungsi pertumbuhan yang merupakan dasar dari running time.

2.5.2. Fungsi Pertumbuhan

Diberikan sebuah program komputer dengan input n bilangan bulat. Pertimbangan penting yang berkaitan dengan program tersebut adalah bagaimana kecepatan fungsi pertumbuhan pada n. Notasi yang digunakan untuk menganalisa fungsi pertumbuhan disebut sebagai "big-Oh".

Definisi 2.5 :

T(n) adalah O (f (n)) , jika terdapat konstanta c dan n_0 sedemikian sehingga

$T(n) \leq c.(f(n))$, untuk setiap $n \geq n_0$.

Program yang mempunyai running time O(f(n)) dikatakan mempunyai fungsi pertumbuhan f (n).

Contoh 2.5 :

Tunjukkan bahwa $T(n) = (n+1)^2$ adalah $O(n^2)$.

Penyelesaian :

Dengan mengambil $n_0 = 1$ dan $c = 4$ maka berdasarkan definisi 2.5 berlaku :

$(n+1)^2 \leq 4n^2$, untuk setiap $n \geq 1$

Contoh 2.6 :

Apakah fungsi $T(n) = 3^n$ adalah $O(2^n)$

Penyelesaian:

Andaikan $T(n) = 3^n$ adalah $O(2^n)$, maka terdapat konstanta bulat c dan n_0 sedemikian sehingga berlaku $3^n \leq c.2^n$ untuk setiap $n \geq n_0$.

Berarti $(3/2)^n \leq c$ untuk setiap $n \geq n_0$. Tetapi $(3/2)^n$ akan terus bertambah besar untuk n besar. Sehingga tidak terdapat konstanta c yang memenuhi $(3/2)^n \leq c$,untuk semua n. Jadi $T(n) = 3^n$ bukan $O(2^n)$.

Suatu program biasanya dibuat dalam beberapa subprogram / prosedur, maka untuk menghitung running time program secara keseluruhan digunakan sedikit prinsip dasar. Prinsip tersebut didasarkan bagaimana menjumlahkan dan mengalikan dalam notasi big -Oh.

Theorema 2.1 :

Misal T_1 (n) dan T_2 (n) adalah running time dari subprogram P yaitu P_1 an P_2 dengan T_1 (n) adalah O (f (n)) dan T_2 (n) adalah O (g(n)) maka running time subprogram P_1 diikuti subprogram P_2 yang dinotasikan T_1 (n) + T_2 (n) adalah O (max (f (n), g (n))).

Bukti :

Untuk menunjukkan T_1 (n) + T_2 (n) adalah O (max (f (n) , g (n))), ambil konstanta c_1 , c_2 , n_1 dan n_2 sedemikian sehingga

T_1 (n) $\leq c_1$.f (n) , untuk $n \geq n_1$

T_2 (n) $\leq c_2$.g(n) , untuk $n \geq n_2$

Misalkan no = max (n_1 , n_2), maka berlaku

T_1 (n) $\leq c_1$.f (n) , untuk $n \geq n_0$

T_2 (n) $\leq c_2$.g(n) , untuk $n \geq n_0$

sehingga berlaku :

$T_1 (n) + T_2 (n) \leq c_1.f (n) + c_2.g(n)$

Misalkan $c_3 = \max(c_1,c_2)$, maka

$T_1 (n) + T_2 (n) \leq c_3.f (n) + c_3.g(n)$

$\leq c_3 (f (n) + g(n))$

$\leq c_3 (\max (f(n),g(n))+\max(f(n),g(n)))$

$\leq 2c_3 (\max (f(n),g(n)))$

untuk $c = 2c_3$ dan $n \geq n_0$.

Jadi running time dari $T_1 (n) + T_2 (n)$ adalah O (max (f (n) , g (n))) .□

Teorema 2.1 di atas disebut aturan penjumlahan (*rule of sum*).

Contoh 2.7 :

Aturan penjumlahan diatas dapat digunakan untuk menghitung running time dari barisan langkah program, dimana setiap langkah kemungkinan adalah bagian program dengan loop atau cabang.

Misal terdapat tiga langkah yang mempunyai running time $O(n^2)$, $O(n^3)$ dan (n log n). Maka running time dari 2 langkah pertama tersebut adalah O (max (n^2 , n^3)) yaitu O (n^3) dan running time dari ketiga langkah tersbut adalah O (max (n^3, n log n)) yaitu n^3.

Secara umum , running time dari barisan langkah tersebut adalah running time dari langkah yang mempunyai running time terbesar.

Theorema 2.2 :

Misa T_1 (n) dan T_2 (n) adalah running time dari sub program P yaitu P_1 dan P_2, dengan T_1 (n) adalah O (f (n)) dan T_2 (n) adalah O (g(n)) maka running time T_1 (n) . T_2 (n) adalah O (f (n) . g (n))

Bukti :

Untuk menunjukkan T_1 (n) . T_2 (n) adalah O (f (n) . g (n)), ambil konstanta c_1 , c_2 , n_1 dan n_2 sedemikian sehingga

$T_1 (n) \leq c_1.f(n)$, untuk $n \geq n_1$

$T_2 (n) \leq c_2.g(n)$, untuk $n \geq n_2$

Misalkan no = max (n_1 , n_2), maka berlaku

$T_1 (n) \leq c_1.f(n)$, untuk $n \geq n_0$

$T_2 (n) \leq c_2.g(n)$, untuk $n \geq n_0$

sehingga berlaku :

$T_1 (n) . T_2 (n) \leq c_1..f(n) . c_2.g(n)$

$\leq c_1.c_2 (f(n).g(n))$

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

untuk $c = c_1.c_2$ dan $n \geq n_0$.

Jadi running time dari T_1 (n) . T_2 (n) adalah O (f (n) . g (n)) .□

Teorema 2.2 di atas disebut aturan perkalian (*rule of product*).

Contoh 2.8 :

Berikut ini adalah contoh sub program / prosedur yang akan dihitung running time-nya yaitu *bubblesort*, yang mengurutkan larik bilangan bulat ke dalam suatu urutan menaik.

```
Procedure bubble ( var A : array [ 1 ... n ] of integer ) ;
var   i , j , tukar : integer ;
    begin
        for  i : = 1   to   n -1   do                          ( 1 )
            begin
                for j : = n  downto i + 1 do                   ( 2 )
                    if ( A [ j - 1]  > A [ j ] than begin      ( 3 )
                        tukar : = A [ j - 1 ] ;                ( 4 )
                        A [ j - 1] = a [ j ] ;                 ( 5 )
                        A [ j ] : = tukar ;                    ( 6 )
            end ;
    end ;
```

Penjelasan :

Banyaknya elemen yang akan diurutkan adalah n. Pertama diteliti setiap statemen penugasan yang memerlukan waktu yang konstan tidak bergantung dari besar masukan, yaitu statemen (4) , (5) dan (6). Ketiga statemen tersebut masing-masing mempunyai O (1). Dengan aturan penjumlahan maka running time dari ketiga statemen tersebut adalah O (max (1, 1, 1) = O (1)

Kemudian dilanjutkan pada statemen (2). Statemen (2) merupakan statemen looping yang akan menjalankan statemen (3) – (6) . Akan tetapi di dalam looping ada statemen if yaitu di statemen (3) , sehingga statemen (4) – (6) hanya akan dijalankan jika statemen (3) dipenuhi. Untuk menghitung berapa kali statemen (4) – (6) dijalankan , diambil kondisi terburuk yaitu statemen (3) – (6) dijalankan mulai j = n sampai j = i +1. Setiap kali (3) – (4) dijalankan memerlukan O(1).

Banyaknya iterasi pada loop ini adalah dari n menurun sampai i+1, yaitu sebanyak n - i kali. Dengan aturan perkalian maka waktu yang diperlukan oleh statemen (2) - (6) adalah O ((n - i) x 1) yaitu O (n - i)

Kemudian loop paling luar yaitu statemen (1) dijalankan sebanyak n - 1 kali (dari 1 sampai n - 1), sehingga total running time prosedur terbatas ke atas oleh :

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

Berarti prosedur tersebut mempunyai O (n^2). Jadi prosedur pengurutan Bubblesort mempunyai O (n^2)

Secara umum, running time dari statemen kelompok statemen dipengaruhi besar masukan dan/atau satu atau lebih variabel .

1. Running time setiap penugasan, statemen membaca data dan mencetak mempunyai O(1)..
2. Running time barisan statemen ditentukan dengan aturan penjumlahan

3. Running time statemen if adalah waktu menjalankan statemen ditambah waktu test kondisi. Waktu tes kondisi adalah O (1). Waktu yang diperlukan if-then-else adalah waktu terbesar di antara kedua kondisi berikut ini :

   Jika kondisi bernilai benar maka waktu yang diperlukan adalah waktu tes kondisi ditambah waktu yang diperlukan untuk menjalankan statemen sesudah *then*, sedangkan jika kondisi bernilai salah maka waktu yang diperlukan adalah waktu tes kondisi ditambah waktu untuk menjalankan statemen sesudah *else*.

4. Waktu untuk menjalankan loop adalah penjumlahan waktu untuk menjalankan isi dan waktu untuk tes kondisi.

2.6. Bahasa Pemrograman Pascal

Bahasa pemrograman Pascal dibuat oleh Profesor Niklaus Wirth di Eidgenossische Technische Hochschule (ETH), Swiss. Pengerjaannya dimulai sekitar tahun 1970 dan menjadi populer sebagai bahasa pemrograman di universitas-universitas, juga sebagai bahasa pemrograman di bidang industri dan pemerintahan. Dengan mempelajari Pascal, seseorang akan mudah untuk mempelajari bahasa pemrograman yang lain seperti FORTRAN, BASIC atau COBOL.

Keunggulan yang dimiliki Pascal antara lain :

1. Tersedianya struktur kontrol yang lebih banyak.

   Pascal menyediakan jangkauan yang lebar dari statemen kontrol iteratif dan kondisi : repeat - until, while -do, if-then-else, for-do, case- dan lain-lain.

2. Tersedianya struktur data yang banyak

   Pascal menyediakan tipe data record, set, file, pointer, subrange dan yang didefinisikan sendiri oleh pemakai.

Secara garis besar program Pascal tersusun atas :

a. Kepala program (*heading*)
b. Bagian deklarasi (*declaration*)
c. Bagian penyatuan (*statement*)

Beberapa konsep dasar pada deklarasi Pascal yang harus diketahui antara lain tipe data, operator, variabel, konstanta.

1. Tipe data skalar, merupakan bentuk umum klas dari item data.

   Beberapa tipe data sederhana dalam Pascal antara lain : integer, byte, real, char, boolean, string.

2. Operator, merupakan himpunan tanda yang mempunyai arti untuk mengerjakan operasi.
3. Variabel, merupakan suatu lokasi dimemori dimana nilai data disimpan. Setiap variabel yang akan digunakan harus dideklarasikan terlebih dahulu pada bagian deklarasi.
4. Konstanta merupakan suatu nilai yang didefininisikan, berupa bilangan integer , real, karakter atau string.

Selanjutnya akan dibahas mengenai beberapa hal yang akan sering digunakan dalam bagian pernyatan program antara lain : prosedur , fungsi, larik, looping dan grafik.

2.6.1. Prosedur dan Fungsi

Pascal menyediakan dua buah subprogram yaitu prosedur dan fungsi. Dengan adanya fasilitas prosedur dapat dibuat subprogram-subprogram yang seolah-olah satu dengan yang lain tidak saling bergantung, sehingga masing-masing subprogram dapat dibuat dan diuji secara independen. Kemudian setelah masing-masing subprogram benar, dilakukan penggabungan didalam program utama. Hal ini menguntungkan karena setiap subprogram hanya ditulis satu kali saja, meskipun dipanggil beberapa kali.

Untuk membuat prosedur dan fungsi sama dengan membuat program kecil dengan mengganti kata 'program' dengan kata 'procedure' atau 'function' dan setelah akhir prosedur atau fungsi ditutup dengan 'end'.

Pengertian yang harus dipahami dalam membuat prosedur atau fungsi adalah parameter.

Parameter merupakan suatu cara program , prosedur atau fungsi untuk saling melakukan komunikasi. Melalui parameter ini program utama dapat mengirimkan nilai kedalam prosedur atau fungsi untuk digunakan dalam proses dan mengirimkan kembali hasilnya jika diperlukan. Parameter formal adalah parameter yang terdapat pada kepala prosedur atau fungsi setelah nama prosedur atau fungsi dan berada dalam tanda kurung.

Parameter aktual adalah parameter yang digunakan untuk memanggil prosedur atau fungsi.

Parameter formal dapat berupa parameter nilai atau parameter variabel.

Parameter nilai adalah parameter yang nilainya tidak berubah ketika keluar dari prosedur atau fungsi.

Parameter variabel adalah parameter yang nilainya berubah ketika keluar dari prosedur atau fungsi, sehingga pada deklarasinya harus diberi kata 'var'.

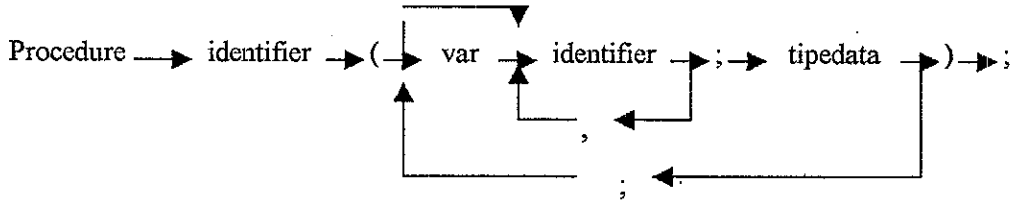
Dalam pemberian nama prosedur atau fungsi dapat digunakan sembarang identifier, selain kata-kata baku yang sudah didefinisikan tersendiri oleh Pascal (reserved words).Contoh reserved word antara lain : round, trunc, sin dan lain-lain

Identifier dibedakan menjadi identifier lokal dan identifier global. Suatu identifier dikatakan bersifat lokal terhadap suatu prosedur atau fungsi, apabila identifier tersebut hanya dideklarasikan dalam prosedur atau fungsi itu sendiri dan tidak dapat digunakan selain oleh prosedur atau fungsi dimana identifier dideklarasikan.

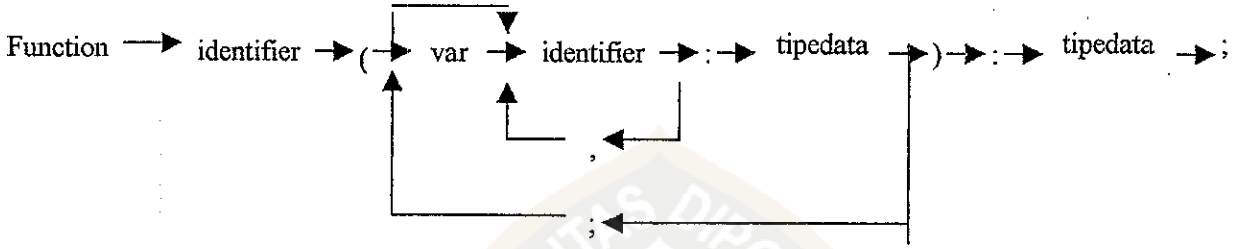
Suatu identifier dikatakan bersifat global terhadap suatu prosedur atau fungsi jika dideklarasikan pada tingkat yang lebih tinggi yang melingkupi prosedur atau fungsi tersebut, atau identifier yang dideklarasikan didalam program utama.

Yang membedakan fungsi dengan prosedur adalah pada parameter formal fungsi disertakan tipe data hasil proses yang akan dikembalikan oleh fungsi.

Untuk lebih jelasnya, diagram syntax kepala prosedur dan fungsi disajikan dalam gambar 2.5 (a) dan (b)



(a)



(b)

Gambar 2.5 (a) Diagram syntax kepala prosedur

(b) Diagram syntax kepala fungsi

Contoh kepala prosedur dan fungsi :

```
procedure sample ( x : real ; y : integer ) ;
procedure cari ( var x : list ; n : integer ) ;
function ratarata (var nilai : array [1…50] of integer; size :integer): real;
function legal (cek : integer ) : boolean ;
```

Contoh pemanggilan prosedur atau fungsi

```
sample ( p, q ) ;
cari ( L[i] , i ) ;
```

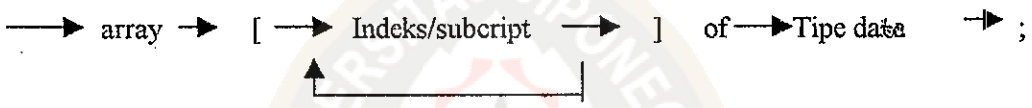
ratarata (A[z] , z) ;

legal (nilai) ;

2.6.2. Larik

Larik merupakan sekumpulan tipe data yang identik obyek yang digantikan dengan nama yang sama. Masing-masing objek dalam larik dikenali dengan menggunakan indeks atau subcript larik.

Diagram syntax untuk mendeklarasikan larik disajikan dalam gambar 2.6



Gambar 2.6 Diagram syntax tipe larik

Contoh :

Type example 1 = array [1 ... 10] of integer ;

example 2 = array [colors] of char ;

Var example 3 : array [1 ... 100] of integer ;

example 4 : array [1 ... 10 , 1 ... 10] of integer ;

2.6.3. Lopping

Looping merupakan suatu proses perulangan dalam mengerjakan sejumlah operasi. Terdapat tiga statemen dalam Pascal untuk membentuk looping, yaitu while repeat dan for .

2.6.3.1. While

Pernyataan while merupakan suatu bentuk loop berdasarkan suatu tes kondisi.

Bentuk : *While* → ekspresi boolean → *do* → statemen →

Bagian isi dari loop while akan dijalankan jika ekspresi boolean tersebut bernilai benar. Setelah seluruh statemen pada bagian isi dilaksanakan, maka akan kembali dan tes ekspresi boolean lagi.

Proses tersebut terus dijalankan hingga ekspresi boolean salah.

Ketika ekspresi boolean salah maka eksekusi diteruskan ke statemen setelah loop while tersebut.

2.6.3.2. Repeat

Pernyataan repeat merupakan suatu bentuk loop dengan eksekusi isi dari loop tersebut minimal sekali, sehingga repeat pasti dijalankan.

Bentuk : *Repeat* ⟶ statemen ⟶ *until* ⟶ ekspresi boolean

Tes kondisi repeat dilakukan setelah menjalankan isi loop, yaitu dengan statemen ''until'. Jika pada tes kondisi tersebut ekspresi boolean bernilai salah maka loop repeat akan dijalankan terus, hingga pada tes kondisi ekspresi boolean bernilai benar.

2.6.3.3.For

Pernyataan for merupakan suatu bentuk loop, dengan banyaknya isi loop tersebut telah ditentukan oleh suatu nilai variabel ordinal.

Bentuk: *For* ⟶ variabel ordinal ⟶ := ⟶ ekspresi 1 ⟶ *to* ⟶ ekspresi 2 ⟶ *do*

⟶ Statemen

2.6.4 Grafik

Selain digunakan untuk melakukan perhitungan-perhitungan dan menampilkan karakter-karakter di mana komputer bekerja dalam lingkungan *mode teks* , komputer juga dapat dimanfaatkan untuk membuat grafik. Di dalam Pascal, membuat grafik dilakukan dengan menggunakan *mode grafik*, yang berbeda dengan mode teks. Dengan mode teks, layar monitor dibagi menjadi 80x 20 . Artinya sumbu X menjadi 80 kolom dan sumbu Y menjadi 20 baris. Sedangkan pada mode grafik layar monitor dibagi menjadi bermacam-macam, tergantung mode driver yang digunakan.

Untuk dapat bekerja di dalam mode grafik, harus menggunakan *unit graph.* Untuk membuat grafik maka unit standar graph harus disebutkan terlebih dahulu sebelum program utama . File yag harus ada dalam unit tersebut antara lain : GRAPH.TPU , *.CHR dan EGAVGA.BGI atau CGA.BGI. Untuk memulai menggunakan grafik maka prosedur standar *Initgraph* harus disebutkan terlebih dahulu :

Initgraph(graph_driver,graph_mode,driver_path);

Graph driver merupakan driver yang digunakan pada komputer untuk memberitahu komputer agar bekerja dengan monitor tertentu. Komputer dapat menentukan sendiri drivernya yaitu dengan adanya perintah "*detect*".

Graph mode merupakan mode grafik yang dipakai pada driver yang bersangkutan. Driver path menunjukkan direktori letak dari file-file yang berekstensi BGI. Untuk mengakhiri grafik dan kembali ke mode layar semula digunakan prosedur standar "*Closegraph*".