BAB III

A L G O R I T M A

3.1. PENGERTIAN ALGORITMA

Komputer adalah alat bantu yang dapat dipakai untuk memecahkan pelbagai masalah. Masalah yang akan dipecahkan tersebut harus diintrodusir ke komputer sehingga dimengertinya. Untuk itu diperlukan komunikasi antara manusia dengan komputer. Salah satu cara untuk berkomunikasi dengan komputer adalah melalui bahasa yang dimengertinya. Pada penyelesaian suatu masalah, seseorang memusatkan perhatian pada masalahnya dan membuat desain yang dapat mewakili pemecahan masalahnya. Desain tersebut diwujudkan dalam notasi-notasi khusus yang disebut algoritma.

Menurut Goodman dalam buku "Introduction To The Design Analysis Of Algorithms" yang dikutip dari "Oxford English Dictionary" edisi 1971, "Algorithm" adalah "Erroneous refashioning of algorism", algorism dipandang kurang lebihnya sinonim dengan aljabar dan aritmetika, definisi ini berasal dari abad ke 9.

Secara mudahnya algoritma adalah suatu konsep dalam pemecahan masalah matematik tertentu, yang memuat intruksi-intruksi dan harus diikuti langkah demi langkah menuju solusinya.

Suatu algoritma dapat dinyatakan dalam beberapa bentuk yaitu :

1. Langkah-langkahnya dinyatakan dalam bahasa sehari-hari.
2. Boleh juga dituliskan dalam suatu bahasa komputer tertentu.
3. Atau gabungan dari cara 1 dan 2 yang disebut di-

agram alur (flowchart).

Biasanya suatu algoritma dituliskan dalam sekaligus tiga cara tersebut. Mula-mula dalam bahasa sehari-hari kemudian diubah dalam diagram alur, dan seterusnya dikembangkan dalam suatu bahasa pemrograman tertentu.

Permasalahan yang diselesaikan dengan algoritma harus berhingga dan dapat diselesaikan dalam waktu yang terbatas pula, dan pemecahannya harus mempunyai tujuan yang jelas sehingga dapat dicapai. Sebagai contoh sederhana misalnya, pergi keseberang jalan adalah suatu tindakan yang tujuannya dapat dicapai, tetapi pergi keneraka jelas bukan masalah yang dapat diselesaikan dengan algoritma.

Jadi untuk menyusun suatu algoritma, pertama-tama harus memahami secara terperinci dari permasalahannya, karena algoritma tidak hanya memuat perintah-perintah apa yang harus dikerjakan, tetapi juga harus diperhatikan efisiensinya, agar setiap langkahnya tepat dan benar-benar menuju solusinya.

Setelah algoritma dituliskan dalam bahasa biasa, kemudian diubah dalam bahasa perantara antara bahasa biasa dengan bahasa pemrograman, yaitu diagram alur. Diagram alur yang baik, tentunya dapat menyatakan suatu algoritma sedemikian hingga tidak hanya tergantung pada bahasa pemrograman dan komputer tertentu saja. Algoritma yang berbentuk diagram alur selanjutnya dikembangkan menjadi suatu program dalam bahasa pemrograman tertentu, agar algoritma tersebut dapat diproses dengan komputer.

Program adalah algoritma yang mengontrol pola tingkah laku suatu komputer. Dengan kata lain, program adalah algoritma yang dimaksudkan untuk dieksekusi oleh komputer. Dalam pengertian algoritma secara umum, hanya di-

titikberatkan pada urut-urutan pelaksanaan yang harus dimengerti dengan baik dan benar, tanpa menghiraukan bagaimana pengertian tersebut diwujudkan. Karena suatu komputer terdiri dari sekumpulan peralatan, dan berkat konstruksinya maka komputer hanya dapat mengerjakan sekumpulan instruksi-instruksi tertentu yang terbatas jumlahnya dan yang telah terdefinisi. Jika kepada komputer diberikan suatu program, maka dengan patuh komputer akan mengerjakannya persis seperti yang dinyatakan dalam algoritmanya. Komputer sangat patuh terhadap semua instruksi yang diberikan padanya, sehingga dalam memberikan instruksi-instruksi harus terdefinisi dengan terperinci dan teliti, hal ini menuntut pemahaman bahasa pemrograman yang digunakan dan pengalaman membuat program.
Dalam buku ini program-programnya dituliskan dalam bahasa basic, tetapi bahasa tersebut tidak dibahas karena pembaca dianggap sudah memahaminya.

Suatu algoritma harus bersifat mudah difahami, demikian juga bila sudah dibuat dalam suatu program, sehingga bila dalam program tersebut terdapat simbol-simbol yang terasa asing perlu diberikan keterangan secukupnya, baik disisipkan dalam programnya atau ditulis dalam lembar tersendiri.

Algoritma ditulis dalam suatu program perlu diuji kebenarannya dalam proses komputer. Untuk mengujinya diberikan permasalahan yang sederhana tetapi mewakili permasalahan yang sebenarnya. Setelah proses selesai hasilnya ditinjau kembali, apakah sudah sesuai dengan yang diharapkan. Adakalanya hasil proses berupa kode-kode, sehingga untuk memahaminya diperlukan keterangan tambahan. Input maupun output dari algoritma teori graph biasanya

berupa angka-angka, yaitu bentuk lain dari suatu graph , sehingga untuk memahami permasalahan sebenarnya secara umum diperlukan penjelasan tambahan. Pengubahan kebentuk lain dari suatu graph terpaksa dilakukan supaya graph tersebut dimengerti oleh komputer. Demikian juga dengan output nya harus diubah dalam bentuk aslinya. Sekilas pandang cara tersebut membuang-buang waktu dan memerlukan kerja ganda. Hal itu memang benar bila masalah yang diajukan berupa graph yang kecil dan sederhana, tetapi untuk graph yang besar, sumbangan dari suatu algoritma tentunya bermanfaat, karena akan memberikan jawaban yang tepat dan dalam waktu yang singkat.

Sebagai contoh sederhana, misalnya mencari lintasan minimal setiap pasangan titik yang ada dalam graph berarah yang terhubung dengan 20 titik.

Dari graph tersebut dapat ditemukan 20(20-1)=380 pasangan titik, jika seseorang dapat mencari lintasan minimal sepasang titik rata-rata memerlukan waktu 1 menit, maka untuk menyelesaikan masalahnya diperlukan waktu 380 menit, belum lagi adanya resiko kesalahan.

3.2. BEBERA BENTUK INPUT

Algoritma dalam teori graph yang proses pengolahannya menggunakan komputer, diperlukan input dalam bentuk tertentu.

Berikut ini diberikan beberapa bentuk input untuk algoritma dalam teori graph yang masing-masing memiliki kelebihan dan kekurangan, untuk kemudian pemilihan bentuk input dari suatu algoritma tergantung dari bentuk graph dan permasalahannya.

3.2.1. Matriks Adjacent (Adjacency Matrix)

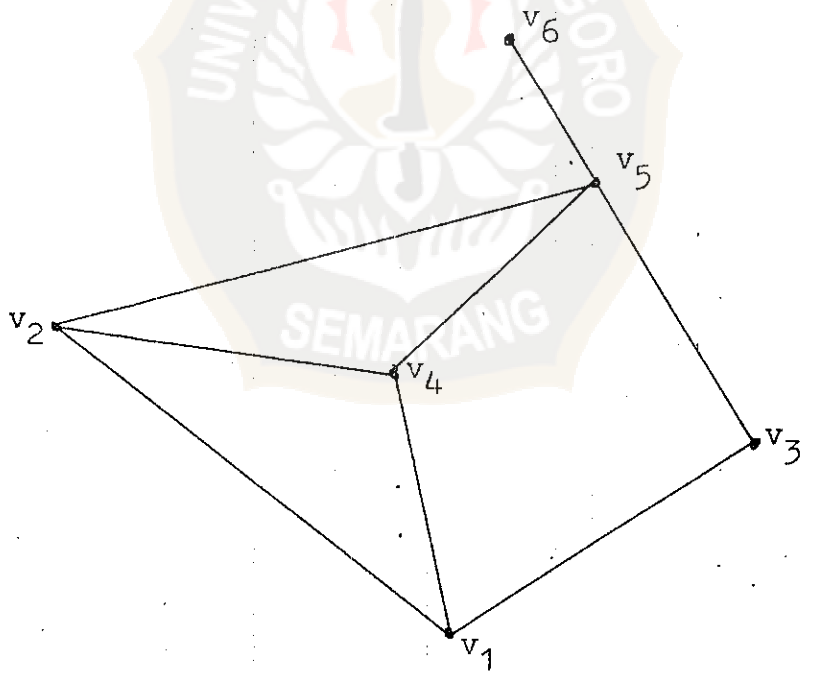
3.2.1. Matriks Adjacent (Adjacency Matrix)

Bila diberikan graph G dengan n titik dan tidak terdapat garis pararel, maka dapat disusun matriks adjacent $X = [x_{ij}]$, berupa matriks biner dengan ukuran n x n, dengan aturan penulisan sebagai berikut :

$x_{ij} = 1$, jika titik i adjacent dengan titik j.

$x_{ij} = 0$, jika titik i tidak adjacent dengan titik j.

Untuk lebih jelasnya diberikan contoh matriks adjacent dari graph dengan 6 titik seperti pada gambar 3.1.



$$X = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Gambar 3.1. Graph dengan 6 titik & matriks adjacentnya.

Yang perlu diperhatikan dalam matriks adjacent X dari suatu graph G adalah :

1. Diagonal dari matriks adjacent X selalu berharga 0 jika dan hanya jika graph G tidak mempunyai loop. Jika titik ke i ($1 \leq i \leq n$) dari graph G mempunyai loop maka $x_{ii} = 1$.
2. Jika graph G tidak mempunyai loop, maka derajat titik i dapat ditemukan dengan menghitung banyaknya angka 1 dalam baris ke i atau kolom ke i.
3. Graph yang mempunyai garis pararel, inputnya tidak dapat dituliskan dalam bentuk matriks adjacent.

Untuk menyimpan input dalam bentuk matriks adjacent dari graph berarah dengan n titik, dalam memori komputer harus disediakan tempat sebesar n^2 byte, sedangkan untuk graph tidak berarah dengan n titik dan tanpa loop, dalam memori harus disediakan tempat sebesar n(n-1)/2 byte, sebab matriksnya simetris terhadap diagonalnya, maka input yang perlu disimpan cukup elemen/unsur yang terletak disegitiga atas saja.

3.2.2. Matriks Insiden (Incidence Matrix)

Bila diberikan graph dengan n titik, e garis dan tanpa loop, maka dapat disusun matriks insiden $B=[b_{ij}]$, yaitu matriks biner dengan ukuran n x e dengan aturan penulisan sebagai berikut :

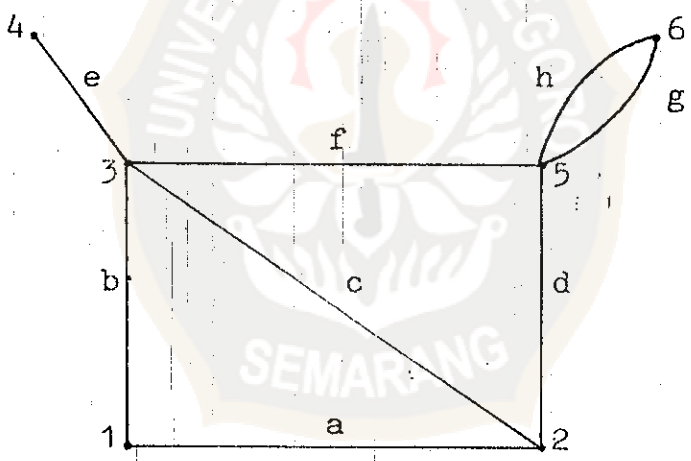
$b_{ij} = 1$, jika garis j melalui titik i.

$b_{ij} = 0$, jika garis j tidak melalui titik i.

Untuk lebih jelasnya diberikan contoh matriks insiden dari graph dengan 6 titik dan 8 garis seperti pada gambar 3.2. Yang perlu diperhatikan dalam matriks insiden B dari suatu graph G adalah :

1. Karena setiap garis pasti menghubungkan dua titik, maka setiap kolom pasti terdapat dua angka 1.
2. Derajat dari titik i ($1 \leq i \leq n$) dapat diketahui dengan menghitung banyaknya angka 1 dalam baris ke i.
3. Jika dalam baris ke i hanya berisi angka 0, maka titik ke i adalah titik terasing.
4. Graph yang mempunyai loop, inputnya tidak dapat dituliskan dalam bentuk matriks insiden.

Untuk menyimpan input dalam bentuk matriks insiden dari graph G dengan n titik dan e garis, dalam memori harus disediakan n x e byte.



$$B = \begin{array}{c} \begin{matrix} a & b & c & d & e & f & g & h \end{matrix} \\ \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{matrix} \end{array}$$

Gambar 3.2. Graph dengan 6 titik dan 8 garis dan matriks insidennya.

3.2.3. Edge Listing

Bentuk input ini merupakan barisan dari semua pasa-

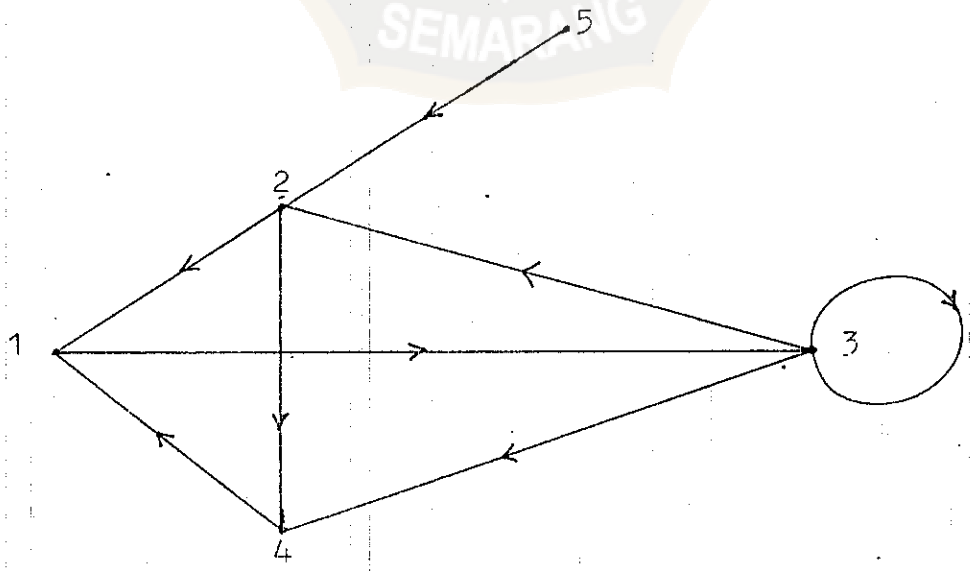
ngan titik yang adjacent. Sebagai contoh, bila diberikan graph berarah dengan 5 titik dan 9 garis seperti pada gambar 3.3, maka dapat disusun edge listing sebagai berikut :

(1,3), (2,1), (2,4), (3,2), (3,3), (4,1), (3,4), (5,2).

Untuk graph tak berarah tidak memperhatikan urutan dalam menuliskan pasangan titik-titiknya.

Kelebihan dari bentuk input edge listing dari bentuk input sebelumnya adalah kemampuannya dalam menuliskan garis paralel dan loop. Untuk menyimpan input dalam bentuk edge listing dari graph dengan n titik dan e garis, dalam memori harus tersedia 2 x e x b byte. Harga b diperoleh dari :

$$2^{b-1} \leq n \leq 2^{b}$$



Gambar 3.3. Graph berarah dengan 5 titik dan 9 garis.

3.2.4. Two Linear Arrays

Bentuk input dari two linear arrays berupa dua himpunan titik-titik, yaitu $F = (f_1, f_2, \ldots\ldots, f_e)$ dan H =

$(h_1,h_2,.....,h_e)$, dengan e adalah banyaknya garis dalam graph, dan setiap garisnya dituliskan dengan aturan :

Misalnya pada garis ke i $(1 \leq i \leq e)$, maka :

Untuk graph berarah : garis yang berasal dari titik f_i menuju h_i.

Untuk graph tidak berarah : garis yang menghubungkan antara titik f_i dan h_i.

Sebagai contoh, untuk input dari graph berarah seperti pada gambar 3.3, maka dapat disajikan dalam bentuk input two linear arrays sebagai berikut :

F = (1,2,2,3,3,3,4,5)

H = (3,1,4,2,3,4,1,2)

Kebutuhan tempat dalam memori yang harus tersedia untuk bentuk input ini sama dengan dalam edge listing.

3.2.5. Successor Listing

Bila diberikan graph dengan n titik, maka dapat disusun successor listing dengan aturan penulisan sebagai berikut :

Ambil titik ke 1 sebagai awal penulisan, kemudian :

Untuk graph berarah : semua titik yang dituju oleh titik satu, dituliskan dibelakangnya.

Untuk graph tidak berarah : semua titik yang adjacent dengan titik 1, dituliskan dibelakangnya.

Setelah titik ke 1 selesai, ambil titik ke 2 sebagai awal penulisan, kemudian mengikuti seperti cara diatas. Demikian seterusnya hingga titik ke n.

Untuk lebih jelasnya diberikan contoh successor listing dari graph seperti pada gambar 3.3 sebagai berikut :

1 : 3

2 : 1,4

3 : 2,3,4

4 : 1

5 : 2

Untuk graph tidak berarah , garis (i,j) dituliskan dua kali, yaitu pada titik ke i dan pada titik ke j, sehingga tidak menghemat tempat dalam memori, oleh karena itu bentuk input ini tidak cocok untuk graph tidak berarah.

Jika diberikan graph berarah dengan n titik, dan dimisalkan d adalah rata-rata dari derajat titik-titiknya, derajat dari suatu titik adalah banyaknya garis yang meninggalkan titik tersebut (untuk graph berarah), maka penulisan input dalam bentuk successor listing, dalam memori harus tersedia tempat sebanyak n(d+1) byte.

Bila diperhatikan, penulisan input dari bentuk yang satu ke bentuk yang lain dari suatu graph, tidak jauh berbeda. Yang diperhatikan dalam setiap penulisan dari berbagai bentuk input adalah, dihubungkan atau tidaknya dua titik oleh suatu garis.

Dari dasar pemikiran tersebut, suatu bentuk input dapat dikembangkan menjadi bentuk input yang lain, yang sesuai dengan kebutuhannya.

Sebagai contoh misalnya, dari bentuk input matriks adjacent, dapat dibentuk pengembangannya, misalnya $A = [a_{ij}]$, yaitu apabila diberikan graph yang setiap garisnya diberi harga lintasan, maka untuk $x_{ij}=1$ dapat digantikan dengan a_{ij}=harga lintasannya, sedangkan untuk $x_{ij} = 0$ dapat diganti dengan vang lain, sesuai dengan permasalahan yang akan diselesaikan, misalnya, masalahnya mencari lintasan minimal, maka $x_{ij}=0$ dapat digantikan oleh $a_{ij}= \infty$.