BAB II

LANDASAN TEORI

2.1 Pengertian Algoritma

Menurut Narsingh (NAR97) Algoritma adalah suatu petunjuk untuk menyelesaikan suatu masalah tertentu yang terdiri dari himpunan instruksi-instruksi yang diikuti langkah demi langkah yang kemudian akan menghasilkan suatu penyelesaian dari masalah tersebut. Setiap langkah harus didefinisikan dengan tepat dan jelas, dan algoritma harus berakhir setelah menemukan penyelesaian masalah dalam sejumlah langkah tertentu.

Setiap algoritma harus mempunyai lima hal penting, yaitu :

1. Ketertentuan (*finiteness*).

   Jika algoritma menjalankan sejumlah intruksi, algoritma tersebut harus mempunyai ketentuan kriteria kapan algoritma itu harus berhenti.

2. Kepastian dan kejelasan (*definiteness*).

   setiap instruksi harus jelas dan pasti setiap langkahnya dalam suatu algoritma.

3. Masukan (*input*).

   Algoritma mempunyai masukan atau kosong.

4. Keluaran (*output*).

   Algoritma harus mempunyai minimal satu data keluaran sebagai hasil pemrosesan dari suatu intruksi algoritma.

5. Efektifitas (*effectiveness*).

   Setiap instruksi harus efektif untuk dapat dijalankan oleh suatu algoritma.

Algoritma didefinisikan sebagai instruksi-instruksi yang terdefinisi dengan baik untuk menghasilkan suatu output tertentu dari suatu input tertentu dalam sejumlah berhingga langkah (CHA93).

Sehingga dapat disimpulkan bahwa algoritma adalah suatu himpunan berhingga langkah-langkah untuk menyelesaikan suatu masalah dengan aturan :

1. Setiap langkah harus jelas dan pasti (*definite*).
2. Minimal ada satu keluaran (*output*) dan masukan (*input*) boleh tidak ada.
3. Harus memiliki kriteria berhenti.

Suatu Algoritma dapat dinyatakan dalam bentuk yang berbeda-beda :

1. Langkah-langkah ditulis dalam bahasa formal.
2. Langkah-langkah ditulis dalam bentuk program komputer yang ditulis lengkap dengan bahasa pemrograman tertentu.
3. Langkah-langkah ditulis dalam bentuk diantara dua bentuk di atas, contohnya sebuah diagram alir (*flow chart*).

Setiap langkah mempunyai keuntungan masing-masing. Biasanya suatu algoritma dinyatakan dalam bahasa formal kemudian dikonversi ke dalam suatu diagram alir (*flow chart*), dan akhirnya ditulis dalam bahasa pemrograman tertentu sehingga dapat dieksekusi.

Dalam mendesain atau membuat algoritma, ada beberapa langkah yang harus diikuti, yaitu :

1. *How to devive algorithm*, yaitu menurunkan ide-ide setelah menganalisa suatu masalah.

2.2.1 Notasi "O"

Tugas yang dilakukan komputer untuk menyelesaikan suatu masalah biasanya berupa tugas yang serupa, tetapi dilakukan berulang-ulang (iterasi). Dengan banyaknya jumlah perulangan dalam suatu algoritma yang diproses oleh komputer menentukan lama waktu proses (*running time*). Jumlah perulangan yang harus dilakukan biasanya dipengaruhi oleh banyaknya masukan data yang harus diproses.

Suatu masalah tertentu dapat diproses dan diselesaikan dengan algoritma yang berbeda. Sebagai contoh adalah masalah graf yang berbobot dan terhubung dalam menentukan *minimum spanning tree*. Dengan algoritma Prim, graf tersebut dapat diselesaikan sehingga terbentuk suatu *minimum spanning tree*. Demikian pula bila menggunakan algoritma Kruskal. Dengan menggunakan kedua algoritma tersebut, masalah graf dalam menentukan *minimum spanning tree* dapat terselesaikan. Akan tetapi, waktu proses algoritma Prim dan Kruskal tersebut akan mempunyai perbedaan. Terlebih jika jumlah data masukan graf tersebut besar.

Perbedaan waktu proses sebagai fungsi jumlah data yang diproses sangat erat hubungannya dengan laju pertumbuhan (*rate of growth*) algoritma yang bersangkutan. Laju pertumbuhan menunjukkan faktor kelipatan waktu proses seiring dengan kenaikan jumlah data (SIA02).

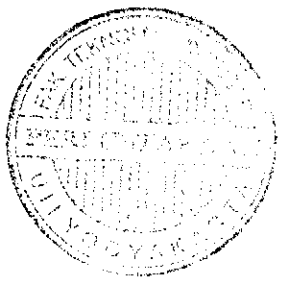
Dalam komputer, laju pertumbuhan dinyatakan dalam notasi-O (dibaca, notasi big-oh / O besar). Notasi-O memberikan cara untuk menyatakan laju pertumbuhan algoritma secara global/aproksimasi dan tidak memperhatikan

perbedaan faktor konstanta serta perbedaan-perbedaan lain yang tidak begitu berpengaruh.

Teorema 2.1

Jika $a_0, a_1, \ldots, a_n$ adalah bilangan riil dengan $a_n \neq 0$ maka

$f(x) = a_n x^n + .. + a_1 x + a_0$ adalah $O(x^n)$



contoh :

deret $1 + 2 + 3 + \ldots + n$

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

sehingga, $1 + 2 + 3 + \ldots + n$ adalah $O(n^2)$

Teorema 2.2

Jika b adalah bilangan riil > 1 maka :

$^b\log x$ adalah $O(x^n)$ untuk semua bilangan bulat $n \geq 1$

x^n adalah $O(b^x)$ untuk semua bilangan bulat $n \geq 0$

$x\ ^b\log x$ adalah $O(x^2)\ \forall x \geq b$

Teorema 2.3

Hirarki fungsi yang sering dipakai untuk menyatakanorder adalah sebagai berikut:

(Setiap fungsi merupakan big-oh dari fungsi di kanannya) :

$$1, {}^2\log(n), \ldots, \sqrt[4]{n}, \sqrt[3]{n}, \sqrt{n}, n, n\left({}^2\log(n)\right), n\sqrt{n}, n^2, n^3, 2^n, n!, n^n$$

Teorema 2.4

a. Jika f(n) = O(g(n)) dan c adalah konstanta, maka c f(n) = O(g(n))

b. Jika f(n) = O(g(n)) dan Jika h(n) = O(g(n)), maka f(n) + h(n) = O(g(n))

c. Jika f(n) = O(a(n)) dan Jika g(n) = O(b(n)), maka f(n) g(n) = O(a(n) b(n))

d. Jika a(n) = O(b(n)) dan b(n) = O(c(n)), maka a(n) = O(c(n))

e. Jika f(n) = O(a(n)) dan Jika g(n) = O(b(n)), maka f(n) + g(n) = O(max $\{|a(n)|,|b(n)|\}$)

2.2.2 Efisiensi Algoritma

Analisa yang paling sering dilakukan pada suatu algoritma adalah waktu proses. Menentukan waktu proses secara tepat merupakan pekerjaan yang sulit, karena waktu proses secara eksak sangat tergantung pada implementasi algoritma dan perangkat keras komputer yang dipakai. Analisa yang diinginkan untuk menyatakan efisiensi algoritma haruslah dibuat seumum mungkin sehingga bisa dipakai pada semua algoritma, terlepas dari implementasi(juga kompiler yang dipakai) maupun perangkat keras komputer yang digunakan. Akibatnya, analisa tidak akan dilakukan dalam konteks waktu proses secara eksak. Analisa efisiensi algoritma biasanya dinyatakan sebagai kompeksitas waktu. Kompleksitas waktu suatu algoritma menyatakan sejumlah langkah-langkah yang dikerjakan oleh algoritma untuk menghitung fungsi-fungsi yang telah dituliskan. Kompleksitas waktu dinyatakan dalam order waktu proses (Big-Oh) sebagai sejumlah fungsi jumlah data masukan yang diberikan. Dalam analisa tersebut, kita memfokuskan diri pada operasi aktif, yang merupakan pusat algoritma, yaitu bagian algoritma

yang dieksekusi paling sering. Bagian-bagian lain seperti pemasukan data, penugasan (*assigment*), dan lain-lain yang tidak sesering bagian operasi aktif dapat diabaikan. Karena bagian-bagian tersebut tidak dieksekusi sesering operasi aktif (SIA02). Jumlah eksekusi operasi aktif inilah yang selanjutnya akan dihitung kompleksitas waktunya.

Sebagai contoh, potongan program untung menghitung jumlah n buah bilangan riil yang disimpan dalam suatu vektor V :

```
sum  = 0                          bagian (a)
for i = 1 to n
     sum = sum + V[i]             bagian (b)
end for i
write (sum)                       bagian (c)
```

Untuk mencari operasi aktif, haruslah ditentukan berapa kali program dieksekusi pada tiap-tiap bagian.

- Bagian (a) dieksekusi 1 kali
- Bagian (b) merupakan suatu kalang (loop). Kalang ini akan diproses berdasarkan kenaikan harga i, dari i = 1 hingga i = n. Jadi statemen `sum = sum + V[i]` akan diproses sebanyak n kali sesuai dengan kenaikan nilai i.
- Bagian (c) akan diproses 1 kali

Karena bagian (b) merupakan bagian yang paling sering diproses, maka bagian (b) merupakan operasi aktifnya. Bagian (a) dan (c) dapat diabaikan, karena bagian-bagian tersebut tidak diproses sesering bagian (b).

Bagian (b) akan diproses oleh algoritma sebanyak data masukian (= n). Maka program penjumlahan bilangan riil tersebut mempunyai order sebanding dengan n. Dengan kata lain, program tersebut mempunyai kompleksitas waktu = O(n).

2.3 Konsep Dasar *Minimum Spanning Tree*

Tree adalah graf yang terhubung yang tidak memuat *cycle* (graf yang acyclic). Spanning tree dari graf adalah tree yang memuat semua titik dari G (CHA93).

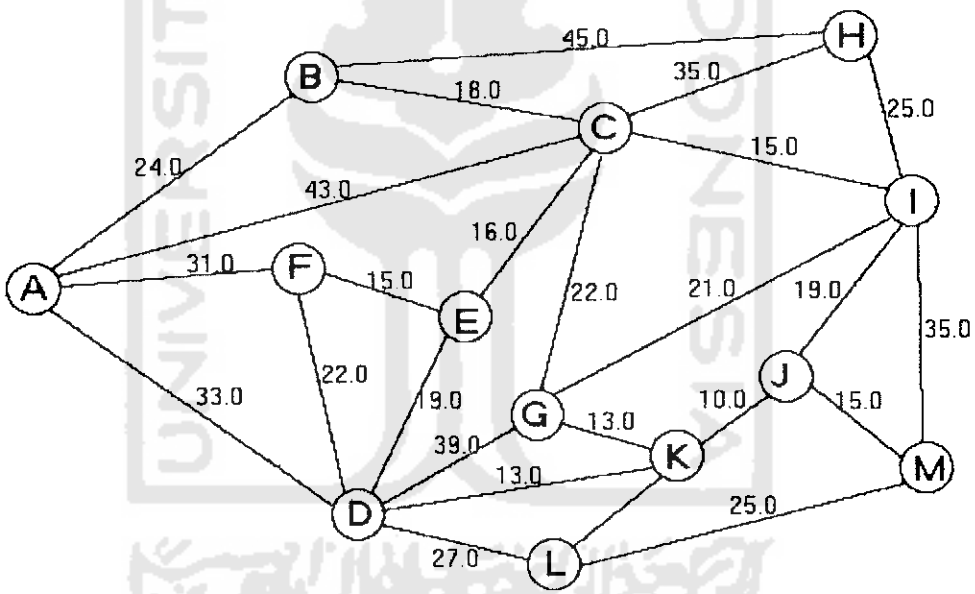
Suatu graf yang terhubung mempunyai minimum satu *spanning tree*, karena graf terhubung, *acyclic*, dan memuat semua titiknya untuk graf terhubung dengan ≥ 2 titik. Karena graf terhubung, maka untuk setiap 2 titik dihubungkan semua titiknya. Jika path tidak membentuk *cycle*, maka terdapat spanning tree. Jika path membentuk *cycle*, dihapus salah satu sembarang garisnya sehingga tidak terdapat *cycle* lagi, tetapi pasti memuat semua titiknya. Jadi terdapat *spanning tree*. Dengan demikian pohon rentang minimum atau *minimum spanning tree* dari suatu graf mempunyai total bobot minimum dari sisi-sisinya. (DEO97). Pohon rentangan dalam suatu graph bisa tidak bersifat unik karena adanya beberapa sisi yang bisa berbobot sama, namun jika lebih dari *minimum spanning tree* maka total bobotnya adalah sama.

Suatu masalah pohon rentang minimum (*minimum spanning tree*) menyangkut suatu himpunan nodes dan suatu himpunan cabang yang diusulkan, yang satupun tidak ada yang terorientasi. Tiap-tiap cabang yang diusulkan

memiliki suatu biaya tak negatif yang berkaitan dengannya. Tujuannnya adalah menyusun suatu jaringan tersambung yang mengandung semua nodes dan sedemikian rupa sehingga jumlah biaya yang berkaitan dengan cabang-cabang yang digunakan adalah minimum.

Pada tahun 1956, Prim dan Kruskal yang berkerja secara terpisah yang masing-masing berhasil menyusun algoritma untuk membuat *minimum spanning tree* secara efisien.

Contoh suatu graf dari *Minimum Spanning Tree.*



Gambar 2.10. Contoh Graf

Untuk mencari total bobot tiap-tiap pohon rentang tersebut dengan cara yang paling sederhana adalah dengan mendaftarkan semua pohon rentang yang mungkin dibuat. Selanjutnya dipilih pohon rentang dengan total bobot yang paling

Dari graph di atas, misalnya dilakukan pencarian mulai dari vertek A (tidak harus dimulai dari vertek A). Maka algoritma ini menghasilkan tahapan-tahapan iterasi pencarian sebagai berikut :

Tabel 2.1. Iterasi algoritma Prim

No Iteasi	W	V-W	Sisi-sisi yang dievaluasi	Vertek berikutnya
1	{A}	{B, C, D, E, F, G, H, I, J, K, L, M}	{A,B}=24, {A,C}=43, {A,D}=33, {A,F}=31	B, karena {A,B} adalah minimum
2	{A, B}	{C, D, E, F, G, H, I, J, K, L, M}	{A,C}=43, {A,D}=33, {A,F}=31, {B,C}=18, {B,H}=45	C, karena {B,C} adalah minimum
3	{A, B, C}	{D, E, F, G, H, I, J, K, L, M}	{A,D}=33, {A,F}=31, {B,H}=45, {C,E}=16, {C,G}=22, {C,H}=35, {C,I}=15	I, karena {C,I} adalah minimum
4	{A, B,C, I}	{D, E, F, G, H, J, K, L, M}	{A,D}=33, {A,F}=31, {B,H}=45, {C,E}=16, {C,G}=22, {C,H}=35, {G,I}=21, {II,I}=25, {I,J}=19, {I,M}=35	E, karena {C,E} adalah minimum
5	{A, B,C, E, F, I}	{D, G, H, J, K, L, M}	{A,D}=33, {B,H}=45, {C,G}=22, {C,H}=35, {E,D}= 19, {G,I}=21, {H,I}=25, {I,J}=19, {I,M}=35	F, karena {E,F} adalah minimum

6	{A, B,C, D, E, F, I}	{D, G, H, J, K, L, M}	{A,D}=33, {B,H}=45, {C,G}=22, {C,H}=35, {F,D}= 19, {F,D}=22, {G,I}=21, {H,I}=25, {I,J}=19, {I,M}=35	D, karena {D,E} adalah minimum
7	{A, B,C, D, E, F, I, K}	{G, H, J, K, L, M}	{B,H}=45, {C,G}=22, {C,H}=35, {D,G}=39, {D,K}=13, {D,L}= 27, {G,I}=21, {H,I}=25, {I,J}=19, {I,M}=35	K, karena {D,K} adalah minimum
8	{A, B,C, D, E, F, I, J, K}	{G, H, J, L, M}	{B,H}=45, {C,G}=22, {C,H}=35, {D,G}=39, {D,L}= 27, {K,G}=13, {K,J}=10, {K,L}=19, {G,I}=21, {H,I}=25, {I,J}=19, {I,M}=35	J, karena {K,J} adalah minimum
9	{A, B,C, D, E, F, I, J, K}	{G, H, L, M}	{B,H}=45, {C,G}=22, {C,H}=35, {D,G}=39, {D,L}= 27, {G,K}=13, {G,I}=21, {H,I}=25, {I,M}=35, {J,M}=15, {K,L}=19	G, karena {K,G} adalah minimum
10	{A, B,C, D, E, F, G, I, J, K}	{H, L, M}	{B,H}=45, {C,H}=35, {D,L}= 27, {H,I}=25, {I,M}=35, {J,M}=15, {K,L}=19	M, karena {J,M} adalah minimum
11	{A, B,C, D, E, F, G, I, J, K, M}	{H, L}	{B,H}=45, {C,H}=35, {D,L}= 27, {H,I}=25, {K,L}=19, {L,M}=25	L, karena {K,L} adalah minimum

Yang menjadi masalah dalam implementasinya adalah keperluan adanya pemeriksaan kondisi siklik tersebut. Salah satu pemecahannya adalah dengan subsetting, yaitu pembentukan subset-subset yang disjoint dan secara bertahap dilakukan penggabungan atas dua subset yang berhubungan dengan suatu sisi dengan bobot terpendek. Algoritma lengkapnya :

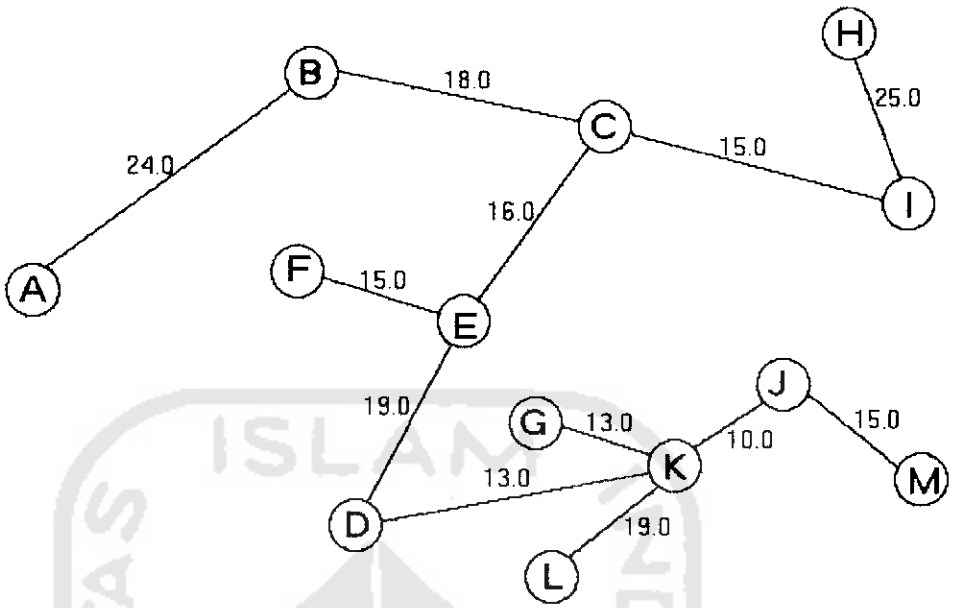
1. Tahap pertama, jika dalam V terdapat n verteks maka diinisialisasi n buah subset yang disjoint, masing-masing berisi satu verteks, sebagai subset-subset awal.
2. Tahap berikutnya, urutkan sisi-sisi dengan bobot yang terkecil hingga yang terbesar.
3. Mulai dari sisi dengan bobot terkecil hingga yang terbesar lakukan dalam iterasi :
   a. Jika sisi tersebut menghubungkan dua vertek dalan satu subset (berarti membentukj siklik) maka *skip* (abaikan) sisi tersebut dan perikasa sisi berikutnya.
   b. Jika tidak (berarti tidak membentuk siklik) maka kedua subset dari vertek-vertek yang bersangkutan digabungkan menjadi satu subset yang lebih besar.
   c. Iterasi akan berlangsung hingga semua sisi terproses.

Dari contoh di atas, maka berturut-turut diperiksa sisi-sisi yang telah diurutkan dari yang terkecil hingga yang terbesar sebagai berikut :

1. {J,K}=10, OK karena tidak membentuk siklik.
2. {D,K}=13, OK karena tidak membentuk siklik.

3. {G,K}=13, OK karena tidak membentuk siklik.
4. {C,I}=15, OK karena tidak membentuk siklik.
5. {E,F}=15, OK karena tidak membentuk siklik.
6. {J,M}=15, OK karena tidak membentuk siklik.
7. {C,E}=16, OK karena tidak membentuk siklik.
8. {B,C}=18, OK karena tidak membentuk siklik.
9. {D,E}=19, OK karena tidak membentuk siklik.
10. {I,J}=19, membentuk siklik.
11. {L,K}=19, OK karena tidak membentuk siklik.
12. {G,I}=21, membentuk siklik.
13. {C,G}=22, membentuk siklik.
14. {D,F}=22, membentuk siklik.
15. {A,B}=24, OK karena tidak membentuk siklik.
16. {H,I}=25, OK karena tidak membentuk siklik.
17. Selesai, karena semua verteks telah tergabung dalam MST.

Setelah semua sisi-sisi tersebut diperiksa dengan tahapan-tahapan diatas dengan metode Kruskal, maka pohon rentang minimum yang dihasilkan sama dengan pohon rentang minimum yang dengan metode Prim, yaitu sebagai berikut:



Gambar 2.12. Graf hasil Metode Kruskal