Kompleksitas Waktu
dan
Effisiensi Algoritma

wijanarto

Review RAM

- RAM
  - Model Matematika untuk komputer, yang memudahkan pengukuran suatu algoritma
  - Sebagai dasar/pengantar untuk menganalisa algoritma
  - Semua aturan RAM di pakai dalam menentukan kompleksitas waktu dan efisiensi algoritma
- Dalam pertemuan selanjutnya, teknik analisis algoritma menggunakan rumusan yang lebih baku, tidak seperti dalam RAM.

Pengantar

- Pada suatu algoritma umumnya yang di perlukan adalah :

1. **Space**, yaitu alokasi yang bersifat statis
2. **Struktur Program**, disini menyangkut pada berapa banyak langkah yang di perlukan untuk menjalankan algoritma tersebut
3. **Rekursif**, pemakaian fungsi rekursif pada suatu algoritma.

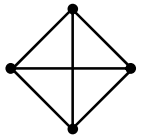
Ukuran Efisiensi Waktu

- Efisiensi untuk suatu algoritma tidak diukur dengan satuan waktu (detik, milidetik, dsb), karena waktu tempuh suatu algoritma sangat bergantung pada :
  - banyaknya data → problem size
  - spesifikasi komputer → Hardware (RAM, processor, dll)
  - compiler → software
  - tegangan listrik → contoh kasusnya, pemakaian notebook menggunakan daya baterai juga berpengaruh pada waktu tempuhnya karena kerja processor dapat dikatakan kurang normal.
  - Dll. → programmer

Efisiensi Waktu

- Efisiensi waktu algoritma diukur dalam satuan n (problem size).
- 4 langkah untuk menentukan ukuran efisiensi waktu, antara lain :
  - Menentukan problem size (n)
  - Menentukan operasi dominan
  - Menentukan fungsi langkah → g(n)
  - Menentukan kompleksitas waktu O(f(n)) (Big Oh function)

Menentukan problem size (n)

Problem	n
Max/min/rata-rata Sorting/Searching	n = banyaknya data
Perkalian 2 matriks A x B p x q q x r	n = max (p, q, r)
Graf |v| = 4 |E| = 6 	n { banyaknya titik |v| ; banyaknya garis |E|
Pengolahan citra w 800 x 600 h	n = banyaknya titik (w, h) n { w x h ; h ; w x h ; 8 x 6

Menentukan operasi dominan

- Operasi dominan yang dimaksudkan di sini sangat bergantung pada permasalahan, dan operasi yang dilakukan yang banyaknya bergantung pada n, dengan kata lain operasi dominan merupakan operasi yang paling banyak dilakukan, sesuai konteks permasalahan.

contoh

- pada algoritma menentukan max/min → operasi dominannya adalah operasi perbandingan “<” atau “>”
- pada algoritma searching → operasi dominannya adalah operasi “=”
- pada algoritma sorting → operasi dominannya adalah operasi “<” atau “>” operasi yang lain seperti “←” tidak dominan, karena belum tentu terjadi penukaran atau perpindahan (contoh kasus : jika data yang diinputkan sudah dalam keadaan terurut)

contoh

- pada algoritma menentukan rata-rata → operasi dominannya adalah penjumlahan (+)
- pada algoritma perkalian 2 matriks → operasi dominannya adalah perkalian, sedangkan operasi dominan yang keduanya (2^{nd} dominant operation) adalah penjumlahan atau pengurangan

contoh

- pada algoritma menentukan modus → operasi dominannya adalah pembandingan “<” atau “>” yang terjadi dalam proses pengurutan, lalu diikuti dengan operasi dominan yang keduanya (2^{nd} dominant operation) adalah pembandingan “=” yang terjadi dalam proses menghitung frekuensi dari masing-masing data

Menentukan fungsi langkah → g(n)

- g(n) = banyak kali operasi dominan dilakukan (dalam n)

```
max ← x(1) ; min ← x(1) ;
for i = 2 to n do
        if x(i) > max then max ← x(i)
```

```
max ← x(1) ; min ← x(1) ;
for i = 2 to n do
if x(i) > max then max ← x(i)
        if x(i) < min then min ← x(i)
```

```
max ← x(1) ; min ← x(1) ;
for i = 2 to n do
if x(i) > max then max ← x(i) else . . .
        if x(i) < min then min ← x(i)
```

pada contoh algoritma ini diperoleh keadaan ***best case***, yakni ketika data terurut *ascending* ; dan sebaliknya akan diperoleh keadaan ***worst case***, yakni ketika data terurut *descending* atau data terbesar berada di X(1).

Menentukan kompleksitas waktu O(f(n)) (Big Oh function)

- Suatu algoritma dengan fungsi langkah g(n) dikatakan mempunyai kompleksitas waktu O(f(n)) jika terdapat konstanta c>0 sedemikian hingga : $g(n) \leq c.f(n)$ untuk $n > n_0$

- Algoritma MaxMin, CountingSort → g(n) = 2n-2 → O(n) Linear
- Algoritma BubbleSort → $g(n) = n^2/2 - n/2$ → $O(n^2)$ Kwadratik
- Algoritma Perkalian 2 matrix $_{nxn}$ → $g(n) = n^3 + kn$ → $O(n^3)$ Kubik
- Algoritma MergeSort, QuickSort → g(n) =(n log n) →O(nlogn) Logaritmik

contoh

Tentukan g(n) dan Big Oh function dari algoritma di bawah ini ?

```
k = n
while k > 0 do begin
        for i = 1 to n do
                if (x > 0) then . . . .
        k = k div 2
end
```

Jawaban : g(n) = n log n + 1 → O(n log n)

Memahami kompleksitas waktu O(f(n))

- Ketika diadakan percobaan untuk mengetahui waktu tempuh beberapa algoritma dengan berbagai jumlah data yang bervariasi, diperoleh data sebagai berikut :
- Waktu tempuh algoritma menentukan max/min berbanding lurus dengan banyaknya data.

Contoh kasus

N	waktu tempuh (milidetik)
1.000.000	20
2.000.000	40
4.000.000	80

pada O(n), yang bersifat linear, diketahui semakin besar jumlah datanya, akan semakin stabil linearitasnya.

Contoh kasus

- Algoritma menentukan max/min

N	Waktu tempuh (menggunakan else)	Waktu tempuh (tanpa else)
1.000.000	10	10
10.000.000	120	110
20.000.000	150	120
30.000.000	220	170
40.000.000	290	220
50.000.000	360	290
100.000.000	720	560
Prediksi 1 Milyar	7200	5600

Contoh Kasus

- Algoritma kuadratik

n	Sekuensial	Bubble
10.000	580	460
20.000	1.890	1.800
30.000	4.000	4.095
40.000	6.900	7.260
50.000	10.435	11.325
100.000	36.200	45.415
Prediksi 1.000.000	3.620.000	4.541.500

Contoh Kasus

- Pada algoritma perkalian 2 matriks → g(n) = n pangkat 3

N	waktu tempuh
100	40
200	320
300	730
400	1.762
500	3.425
600	5.850
700	9.160
800	13.760
900	19.360
1.000	26.380

Efisiensi Memory/Space

- **Yaitu menentukan besar memory yang diperlukan oleh suatu algoritma.**
- Kebutuhan memory (*space*) suatu algoritma juga tidak bisa diukur dalam satuan memory (byte, KB), karena kebutuhan memory yang sebenarnya bergantung dari banyak data dan struktur datanya. Kebutuhan memory dari suatu algoritma diukur dalam **satuan problem size n**.

Kemudahan Implementasi

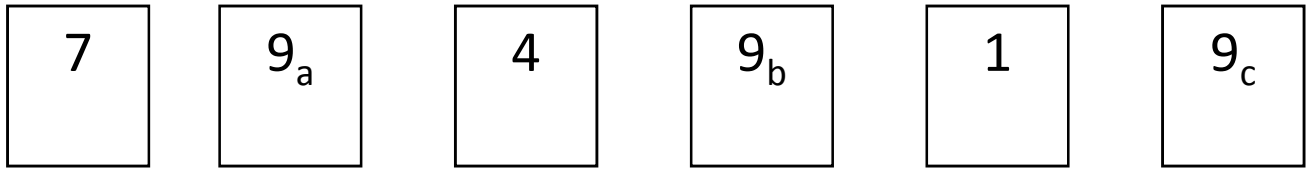
- Maksud dari kemudahan implementasi di sini adalah mengukur seberapa mudah/sederhana algoritma tersebut dibuat programnya, hal ini bisa dilihat dari teknik perancangannya atau struktur data yang digunakan. Biasanya sering digunakan dalam membandingkan suatu algoritma dengan algoritma lainnya, dan bukan diukur dengan tingkatan seperti sulit, mudah, atau pun sedang. Misalnya, bila kita membandingkan algoritma sekuensial sort dengan quick sort, ternyata algoritma sekuensial sort lebih mudah , karena quicksort menggunakan teknik devide & conquer.
- Pigeonhole Sort lebih mudah dibandingkan dengan Radix Sort, karena Radix Sort menggunakan queue.

Data Movement (sorting)

- Unsur ini berusaha mencari tahu banyaknya peristiwa perpindahan atau penukaran yang terjadi pada suatu algoritma sorting. Untuk mengetahui data movement ini kadang-kadang menemui kesulitan, karena mungkin tidak pasti atau mungkin diperlukan perhitungan dan penyelidikan yang lebih lanjut.

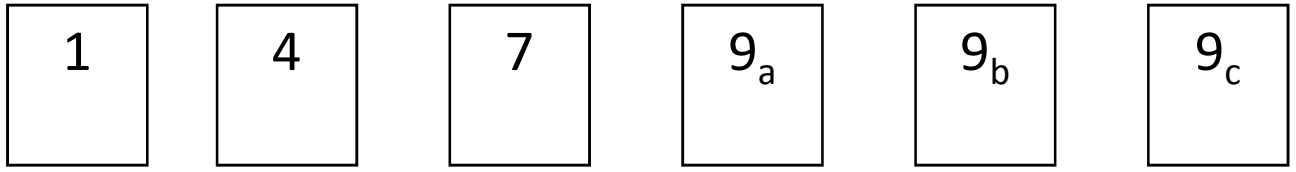
Stability (sorting)

- Algoritma dapat bersifat stabil atau tidak stabil. Stabilitas suatu algoritma dapat dilihat dari kestabilan index data untuk data yang sama.



Index : 1 2 3 4 5 6

Setelah mengalami proses pengurutan, algoritma tersebut akan disebut stabil, jika data menjadi :



Index : 5 3 1 **2** **4** **6**

EFISIENSI ALGORITMA

- Dari kuliah sebelumnya sudah di bahas mengenai konsep dasar efisiensi pada umumnya, dan pada bagian ini akan di berikan contoh perhitungan waktu proses pada suatu algoritma. Contoh

```
s←0                          (a)
for i←1 to n do
  s←s+i                      (b)
write(s)                     (c)
```

analisa

- Bagian (a) di eksekusi 1 kali
- Bagian (b) merupakan suatu loop, yang didasarkan atas kenaikan harga i dari i=1 hingga i=n. Jadi statemen s=s+1 akan diproses sebanyak n kali sesuai kenaikan harga i
- Bagian c akan diproses 1 kali
- Karena bagian (b) merupakan bagian paling yang paling sering diproses, maka bagian (b) merupakan operasi aktif, sedang (a) dan (c) dapat di abaikan karena bagian tersebut tidak sering diproses.
- Bagian (b) diproses sama dengan banyak data yang di masukan (n).
- Maka program penjumlahan bilangan riil tersebut mempunyai order sebanding dengan n atau O(n).

contoh

```
for i←2 to n do
    A←2*n+i*n
```

- Analisis :

Jumlah pemrosesan A=2*n+i*n mengikuti iterasi dalam i, yaitu dari i=2 hingga i=n, jadi sebanyak (n-2)+1=(n-1) kali. Perhatikan disini yang penting bukanlah berapa nilai variable A (yang merupakan fungsi dari i dan n), tapi keseringan pemrosesan A. Sehingga algoritma tadi berorder O(n)

contoh

```
for i←1 to n do
    for j←1 to i do
        A←n+i*j
```

- Analisis :

Pada i=1, j berjalan dari 1 hingga 1 sehingga A diproses 1 kali
Pada i=2, j berjalan dari 1 hingga 2 sehingga A diproses 2 kali
Pada i=3, j berjalan dari 1 hingga 3 sehingga A diproses 3 kali
… dan seterusnya
Pada i=n, j berjalan dari 1 hingga n sehingga A diproses n kali
Secara keseluruhan A akan dip roses sebanyak (1+2+3+…+n)=

$\frac{n(n-1)}{2} = \frac{1}{2}n^2 + \frac{1}{2}n$ kali, dan

algoritma tersebut mempunyai order $O(n^2)$.

SPACE

- Persoalan yang mendasar adalah bagaimana mengkonversi space (dalam satuan byte) ke langkah.
- Salah satu cara yang paling mudah adalah dengan menghilangkan satuannya (byte), selain itu juga dengan asumsi tipe data dinamis di perhitungkan pada alokasi awal.
- Dengan cara ini space dapat di tentukan bersama komponen lain.

SPACE

- Misal satuan variable dan konstanta yang bertipe :
  - int,real/float besarnya di anggap sama (misal 4 byte atau 8 byte)
  - char di anggap 1 byte
  - float array [2][2]= 4*4 byte=16 byte
- Jadi pada prinsipnya space itu statis awalnya dan besarnya tertentu, sehingga seringkali space di nyatakan ke suatu konstanta (C) tertentu (di abaikan).
- Dengan demikian dapat di katakan bahwa pada awalnya tergantung pada hasil analisa struktur program dan bentuk rekursifnya saja.

Ringkasan

- Kompleksitas dan Efisiensi algoritma akan di tentukan dengan model matematika dalam bentuk fungsi g(n).
- Klasifikasi fungsi yang sudah di pelajasi pada bab sebelumnya (Notasi asimtotik), merupakan model kompleksitas yang di dasarkan pada ukuran tertentu suatu algoritma
- Pada pertemuan selanjutnya akan di bahas bagaimana menentukan kompleksitas waktu tempuh dan efisiensi secara sistematis