PENGGUNAAN ”BIG O NOTATION” UNTUK MENGANALISA EFISIENSI ALGORITMA

Ikhsan Fanani – NIM : 13505123

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : ikhsan_fanani@yahoo.com

Abstrak

Makalah ini membahas tentang penerapan *Big O Notation* atau “Notasi O Besar” untuk menganalisa efisiensi suatu algoritma. Notasi O Besar - biasa disebut juga Notasi Landau (*Landau Notation*) atau Notasi Asimptotik (*Asymptotic Notation*) – adalah notasi matematika yang digunakan untuk menggambarkan sifat suatu fungsi asimptotik. Hal ini dilakukan untuk mengamati sifat/kecenderungan suatu fungsi untuk masukan (*input*) yang sangat besar/sangat kecil dengan cara yang simpel namun teliti sehingga dapat dilakukan perbandingan dengan fungsi-fungsi yang lain

Lebih jauh lagi, simbol O digunakan untuk mengambarkan sebuah batas atas dari asimptotik suatu jarak/ukuran dari sebuah fungsi untuk fungsi yang lebih simpel. Terdapat juga simbol-simbol seperti *o*, O, ?, dan T untuk batas-batas atas, bawah, dan rata-rata. Aplikasinya terdapat pada dua area: dalam matematika, notasi ini digunakan untuk mengetahui karakteristik dari syarat sisa untuk wilayah tak-hingga yang terpotong, terutama seri asimptotik. Dalam *computer science*,notasi ini digunakan untuk menganalisa kompleksitas dari suatu algoritma.

Biasanya, notasi O besar digunakan untuk menggambarkan batas-batas asimptotik. Namun batas asimptotik tersebut lebih umum dan tepat dinotasikan simbol dengan T (theta besar) seperti akan dijelaskan dibawah ini.

Notasi diperkenalkan pertama kali di Jerman, oleh seorang *number theorist* Paul Bachmann pada tahun 1894, pada edisi kedua dari bukunya yang berjudul *Analytische Zahlentheorie* (“analitik teori bilangan”), dari edisi pertama yang (belum membahas notasi O besar) yang diterbitkan pada 1892. Notasi ini menjadi populer oleh *number theorist* Jerman yang lain, Edmund Landau, sehingga terkadang notasi ini disebut Notasi Landau. O Besar adalah singkatan dari “order of” dalam bahasa Inggris, yang sebenarnya adalah lambang Omicron besar, belakangan digunakan huruf latin dengan bentuk yang identik yaitu huruf “O” besar, bukan 0(bilangan nol).

1. Kompleksitas Algoritma

Sebuah pertanyaan yang sering muncul adalah “Seberapa efisienkah suatu algoritma atau potongan kode?” Efisiensi tergantung dari beberapa hal, diantaranya:

- Kinerja CPU
- Kinerja Memori
- Kinerja Disk
- Kinerja Jaringan

Semua aspek tersebut sangat penting, tapi makalah ini akan membahas mengenai poin pertama, yaitu CPU. Berhati-hatilah ketika membandingkan antara

1. Performa: Berapa banyak waktu / memori / disk yang digunakan ketika program berjalan. Tergantung dari mesin, *compiler*, dan kode.
2. Kompleksitas: Apa yang akan terjadi ketika ukuran masalah semakin besar.

Kompleksitas memengaruhi performa, namun tidak sebaliknya. Waktu yang diperlukan untuk melakukan suatu baris kode / algoritma sebanding dengan operasi dasar yang dilakukan. Beberapa contoh operasi dasar :

- Satu operasi aritmatika (misal: +, *)
- Satu *assignment*
- Satu ekspresi (misal: x==0)
- Pembacaan input
- Penulisan output (primitif)

Beberapa algoritma melakukan jumlah operasi yang sama dalam setiap kali pemanggilannya.. Algoritma seperti ini dikatakan memerlukan waktu yang konstan.

Beberapa algoritma lain melakukan jumlah operasi yang berbeda, tergantung dari jumlah masukan pada parameter-nya. Misalnya, algoritma pemanggilan berurutan (*sequence*), jumlah operasi yang dilakukan tergantung dari jumlah pemanggilan. Parameter yang nilainya memengaruhi jumlah operasi yang dilakukan disebut *problem size* atau *input size*.

Ketika kita mengukur kompleksitas suatu algoritma, kita bukan memerlukan jumlah operasi yang tepat, kita memerlukan bagaimana hubungan antara jumlah operasi tersebut dengan ukuran masalah (*problem size*). Jika *problem size*-nya *double*, apakah jumlah operasi akan tetap dua kali lipat? Ataukah bekembang dengan cara tertentu? Untuk algoritma yang memerlukan waktu yang konstan, melipatgandakan ukuran masalah tidak memengaruhi jumlah operasi yang dilakukan.

Lebih jauh lagi, biasanya kita tertarik dengan *worst case*, atau kasus terburuk; yaitu jumlah operasi terbanyak yang mungkin dilakukan sebuah algoritma untuk ukuran masalah yang diberikan (selain itu, terdapat juga kasus rata-rata –*average case*- dan kasus terbaik –*best case*-).

2. Big O Notation

Terdapat dua macam penggunaan notasi ini: Asimptotik Tak Hingga (Infinite Asymptotik) dan Asimptotik Sangat Kecil (Inifinitesimal Asymptotik). Perbedaan keduanya hanya terdapat pada aplikasi, bukan pada konsep dasarnya. Definsi umum dari "O Besar" adalah sama untuk kedua kasus, namun dengan batas-batas (*limits*) yang berbeda untuk argumen fungsi-nya.

Infinite Asymptotic

Notasi O Besar sangat berguna untuk mengnalisa efisiensi suatu algoritma. Misalnya, waktu yang diperlukan untuk menyelesaikan sebuah masalah dengan parameter n bisa jadi

$$T(n) = 4n^2 - 2n + 2$$

Semakin n bertambah besar, suku n^2 akan mendominasi pertumbuhan, sehingga suku-suku yang lain dapat diabaikan – sebagai perumpamaan, jika n = 500, suku $4n^2$ akan 1000 kali lebih besar daripada suku 2n, sehingga mengabaikan suku-suku yang lebih kecil tidak akan memberikan efek yang cukup signifikan.

Lebih jauh lagi, koefisien dari suku tersebut menjadi tidak relevan / dapat diabaikan. Misalnya ekspresi yang mengandung suku n^2 atau 2^n. Meskipun $T(n) = 1.000.000n^2$, jika $U(n) = n^x$, koefisien selalu dapat diabaikan, karena n berkembang lebih besar dari 1.000.000. ($T(1.000.000)) = 1.000.000^x = U(1.000.000)$).

Notasi O Besar mengambil yang tersisa, atau dapat ditulis:

$$T(n) \in O(n^2)$$

(dibaca "O Besar untuk n kuadrat") dan berarti algoritma tersebut memiliki kompleksitas waktu n^2.

Infinitesimal Asymptotic

Notasi O Besar dapat pula digunakan untuk menggambarkan kesalahan dalam sebuah aproksimasi dalam fungsi matematika. Seperti misalnya :

$$e^x = 1 + x + \frac{x^2}{2} + \mathrm{O}(x^3) \qquad \text{as } x \to 0$$

Mengekspresikan bahwa kesalahannya, memiliki selisih

$$e^x - \left(1 + x + \frac{x^2}{2}\right)$$

dan lebih kecil dalam nilai absolut dibandingkan waktu konstan $|n|^3$ untuk x mendekati 0.

Dalam analisa asimptotik, kita lebih mementingkan kecenderungan ukuran (*order of magnitude*) daripada nilai aktual/sebenarnya dari sebuah fungsi. Dan lagi, kita tidak perlu mengetahui berapa lama waktu yang dibutuhkan untuk melakukan suatu operasi pada jenis kompter tertentu. Yang jelas, kita dengan mudah dapat melihat bahwa "n^2" adalah fungsi yang berkembang jauh lebih cepat dibandingkan fungsi linear seperi "n".

Untuk menggambarkan kecenderungan ukuran dari suatu operasi, kita menggunakan "Notasi O Besar". Jika kita memiliki algoritma yang melakukan $7n^4 + 35n^3 - 19n^2 + 3$ operasi, Notasi O Besar-nya adalah $O(n^4)$. Jika kita memiliki algoritma yang melakukan $2n + 5$ operasi, Notasi O Besarnya adalah $O(n)$. Perhitungan tersebut cukup sederhana.

Contoh
Ambil sebuah polinom

$$f(x) = 6x^4 - 2x^3 + 5$$
$$g(x) = x^4.$$

Dikatakan f(x) memiliki kecenderungan O(g(x)) atau $O(x^4)$. Dari definisi kecenderungan,

$$|f(x)| < C\,|g(x)|$$

dimana C konstan.
Pembuktian:

$$|6x^4 - 2x^3 + 5| \leq 6x^4 + 2x^3 + 5$$

Untuk $x > 1$

$$|6x^4 - 2x^3 + 5| \leq 6x^4 + 2x^4 + 5x^4$$

Karena $x^4 > x^3$ dan seterusnya..

$$|6x^4 - 2x^3 + 5| \leq 13x^4$$

$$|6x^4 - 2x^3 + 5| \leq 13\,|x^4|.$$

dengan C = 13.

Notasi
Pernyataan "f(x) adalah O(g(x))" sebagaimana didefinisikan sebelumnya, biasa ditulis

$$f(x) = O(g(x))$$

Pernyataan ini adalah penyalahgunaan notasi. Persamaan dari dua buah fungsi tidak dinyatakan. Properti O(g(x)) tidaklah simetrik:

$$O(x) = O(x^2) \text{ but } O(x^2) \neq O(x)$$

Karena alasan ini, beberapa penulis lebih memilih menggunakan notasi himpunan dan menulis

$$f \in O(g)$$

Menganggap O(g) sebagai himpunan dari fungsi-fungsi yang didominasi oleh g. Dalam penggunaan yang lebih rumit, , O( ) dapat muncul pada tempat yang berbeda di dalam sebuah persamaan, bahkan beberapa kali untuk masing-masing sisi. Misalnya, pernyataan berikut benar untuk $n \to \infty$

$(n + 1)^2 = n^2 + O(n)$
$(n + O(n^{1/2}))(n + O(\log n))^2 = n^3 + O(n^{5/2})$
$n^{O(1)} = O(e^n)$

Maksud dari pernyataan diatas adalah :
Untuk setiap fungsi yang memenuhi untuk setiap O( ) pada sisi kiri, terdapat fungsi-fungsi yang memenuhi masing-masing O( ) pada sisi kanan, melakukan substitusi untuk semua fungsi-fungsi ini ke dalam persamaan menyebabkan kedua sisi menjadi sama. Misalnya, persamaan ke-3 diatas berarti: "Untuk setiap fungsi $f(n) = O(1)$, terdapat fungsi-fungsi $g(n) = O(e^n)$ sehingga $n^{f(n)} = g(n)$"

Kecenderungan Fungsi Umum
Di bawah ini adalah klasifikasi dari fungsi-fungsi yang biasa ditemukan dalam menganalisa sebuah algoritma. Semuanya menggunakan n yang bertambah hingga tak-hingga. Daftar berikut diurutkan berdasarkan kecepatan pertumbuhan dari fungsi. Fungsi yang paling lambat berkembang ditulis lebih dahulu. C adalah konstanta yang dapat berubah.

Notasi	Nama	Contoh
$O(1)$	Konstan	Menentukan apakah suatu bilangan ganjil atau genap
$O(\log * n)$	Iterasi logaritmik	Algoritma pencarian Hopcraft dan Ullman untuk himpunan disjoint
$O(\log n)$	Logaritmik	Pencarian dalam list terurut dengan *Binary Search Algorithm*
$O((\log n)^c)$	Polilogaritmik	Menentukan bilangan prima dengan *AKS primality test*
$O(n)$	Linear	Pencarian dalam list tidak terurut
$O(n \log n)$	Linearitmik	Mengurutkan list dengan Heapsort
$O(n^2)$	Kuadratik	Mengurutkan list dengan Insertion Sort
$O(n^c)$, $c > 1$	Poliomial	Pencarian *shortest path* dengan algoritma *Floyd-Warshall*
$O(c^n)$	Eksponensial	Pencarian solusi untuk *traveling salesman problem*
$O(n!)$	Faktorial	Menyelesaikan *traveling salesman problem* dengan menggunakan *brute force*
$O(2^{cn})$	Dobel Eksponensial	Pencarian himpunan lengkap dari AC-unifiers (associative-commutative unifiers)

Tidak biasa, namun pertumbuhan yang jauh lebih cepat masih mungkin terjadi, seperti versi satu nilai dari *Ackermann function*, A(n,n). Sebaliknya, fungsi dengan pertumbuhan sangat lambat pun dimungkinkan, seperti misalnya invers dari fungsi diatas. Meskipun tidak memiliki batas, fungsi-fungsi tersebut biasa dianggap konstan dalam praktik umum.

Daftar Pustaka

[1] *http://en.wikipedia.org/wiki/Big_O_notation/*.
Tanggal akses: 3 Januari 2007 pukul 21.00

[2] *http://www.cs.wisc.edu/~hasti/cs367-common/COMPLEXITY.html*. Tanggal akses 3 Januari 2007 pukul 21.00