Kecepatan dan Kompleksitas Waktu pada Algoritma Rekursif dan Iteratif

Binanda Smarta Aji -13512069
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13512069@stei.itb.ac.id

***Abstrak* – Makalah ini menjelaskan tentang kompleksitas waktu untuk algoritma rekursif dan algoritma iteratif. Algoritma rekursif dan algoritma iteratif terkadang diperlakukan secara berbeda oleh komputer, sehingga ada kemungkinan kedua algoritma akan memakan waktu yang berbeda di lingkungan yang sama.**

Kata kunci* – *rekursif, iteratif, komleksitas waktu, notasi O besar

I. TENTANG ALGORITMA REKURSIF DAN ITERATIF

A. Algoritma Rekursif

Algoritma rekursif adalah algoritma yang memanggil dirinya sendiri dengan parameter yang lebih kecil atau lebih sederhana. Yang dimaksud dengan memanggil diri sendiri di sini adalah memanggil subprogram yang sama dengan dirinya. Jadi penyelesaian masalahnya bergantung dari penyelesaian masalah dari "dirinya sendiri yang lebih kecil".

Algoritma rekusrif akan ters menerus memanggil dirinya dengan parameter yang lebih kecil, sampai pada akhirnya dia memanggil dirinya dengan parameter terkecil. Ketika dirinya dipanggil dengan parameter terkecil, dia akan mengeluarkan nilai sesuai dengan nilai yang diterima. Kemudian keluarannya itu akan diterima oleh yang memanggil dirinya, kemudian dia akan mengeluarkan hasil proses terhadap keluaran yang diberikan dia dengan parameter paling kecil. Demikian seterusnya hingga yang paling besar mengeluarkan hasil prosesnya.

Agar algoritma dapat bekerja sedemikian rupa, maka algoritma tersebut harus mengandung kedua unsur, yaitu basis dan rekurens.

Basis adalah bagian yang berisi nilai awal, dan nilainga tidak mengacu pada dirinya sendiri. Dengan kata lain, ini bagian yang tadi disebut sebagai "dirinya dengan parameter terkecil/paling sederhana". Bagian inilah yang menghentikan definisi rekursif, yang juga membuat algoritma ini bisa memiliki nilai yang terdefinisi.

Rekurens adalah bagian yang memanggil dirinya dengan parameter yang lebih sederhana, dan memproses nilai yang dihasilkan dari panggilannya tersebut. Setiap kali bagian ini memanggil dirinya sendiri, parameternya harus semakin dekat ke basis, dengan alasan yang sudah jelas, yaitu agar bisa menghdasilkan nilai yang terdefinisi.

Sebagai contoh, perhatikan fungsi faktorial berikut

```
int faktorial (int n){
        if (n==0){
                return 1;
        }
        else{
                return n * faktorial(n-1)
        }
}
```

Gambar 1: Fungsi faktorial dengan algoritma rekursif (bahasa C)

Bagian basis dari fungsi di atas adalah, ketika fungsi tersebut dipanggil dengan parameter 0, maka fungsi akan mengembalikan nilai 1.

Sementara bagian rekurens-nya adalah bagian ketika nilai yang menjadi parameter fungsi bukan 0, fungsi akan memanggil dirinya dengan parameter dikurangi dengan 1, kemudian keluaran dari fungsi yang dipanggil tersebut dikalikan dengan nilai parameternya.

Jika kita tuliskan tahapannya akan menjadi seperti ini:

faktorial(n)

= n*faktorial(n-1)

=n*(n-1)*faktorial(n-2)

.

.

.

=n*(n-1)*(n-2)*...*faktorial(1)

=n*(n-1)*(n-2)*...*1*faktorial(0)

=n*(n-1)*(n-2)*...*1*1

sehingga fungsi di atas memiliki hasil yang sama dengan n!.

B. Algoritma Iteratif

Algoritma ini tidak memanggil dirinya sendiri, melainkan mencatat hasil dan kemudian meju ke sekuens berikutnya, kemudian melakukan proses terhadap hasil sebelumnya.

Jika algoritma rekursif memerlukan basis sebagai nilai akhir, algoritma iteratif memerlukan inisialisasi nilai awal sebelum melakukan iterasi. Dan tentu saja harus menentukan batas akhir dari iterasi , utnuk menentukan waktu iterasi berakhir.

Algoritma iteratif mengulangi tindakan yang sama di setiap sekuens, hanya yang menjadi bahan pekerjaan berbeda di setiap sekuens.

```
int faktorial(int n){
        res = 1;
        for(int i = 1; i <=n; i++){
                res *= i;
        }
        return res;
}
```

Gambar 3: Fungsi faktorial dengan algoritma iteratif (Bahasa C)

Di fungsi itu dapat dilihat bahwa inisialisasi nilai keluaran (res) dilakukan di awal. Kemudian memasuki bagian iterasi, hingga i = n. Jika ditelusuri tahapannya maka:

res = 1

res = 1*1 i++ (i=2)

res = 1*1*2 i++ (i=3)

res = 1*2*3 i++ (i=4)

.

.

.

res = 1*2*3*...*(n-1) i++ (i=n)

res = 1*2*3*...*(n-1)*n i++ (i=n+1) (keluar loop)

dan akhirnya fungsi mengeluarkan res yang nilainya sama dengan n!.

II. KOMPLEKSITAS WAKTU

Kompleksitas waktu adalah besaran yang digunakan sebagai ukuran waktu suatu algoritma. Biasanya kompleksitas waktu digunakan untuk menetukan waktu yang diperlukan suatu algoritma menyelesaikan masalah, dengan tidak terpengaruh kondisi lingkungannya.

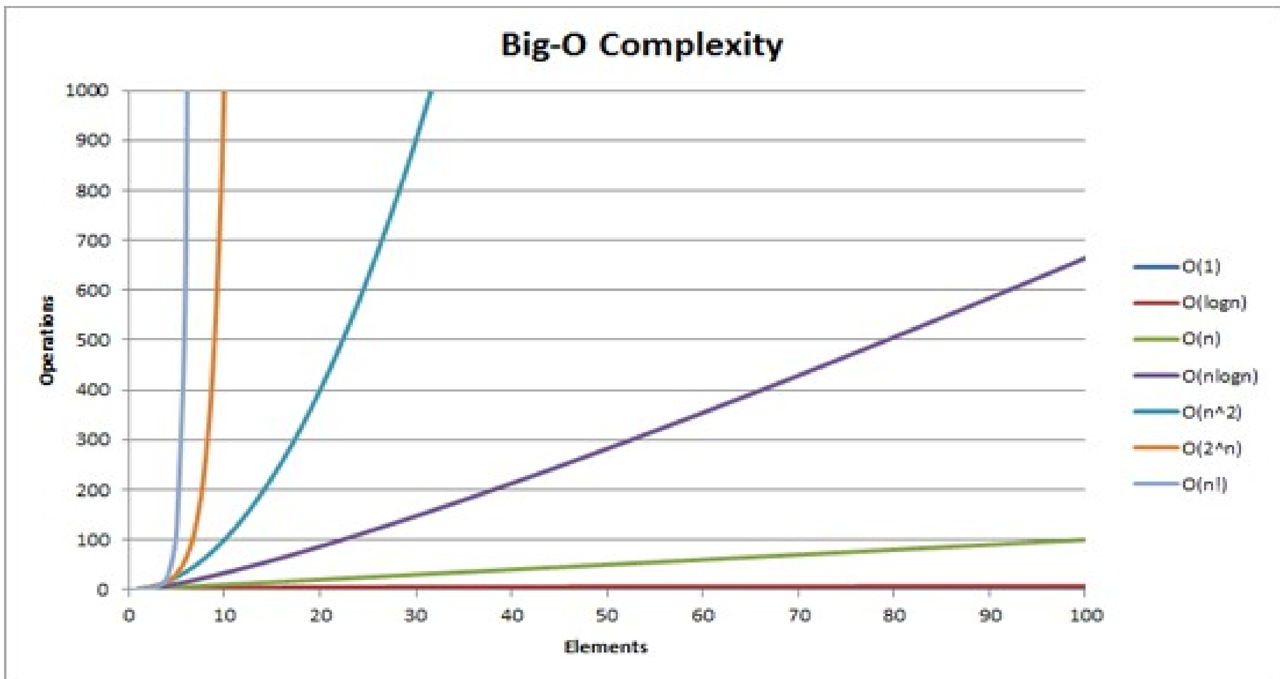
Kompleksitas waktu suatu algoritma diukur dari jumlah tahapan komputasi yang dilakukan suatu algoritma untuk menyelesaikan masalah dalam bentuk fungsi dengan ukuran masukan n, ditulis T(n). Idealnya, semua operasi dalam suatu algortitma diperhitungkan, tetapi karena alasan praktis, yang menjadi perhitungan adalah operas-operasi yang dinilai mendasar. Contohnya untuk algoritma pengurutan isi array, operasi yang mendasar adalah perbandingan antar variabel untuk dasar perulangan / loop, yaitu operasi <, >, =, !=, >=, <=., kemudian juga operasi *assignment* yang mengurutkan isi dari tabel/array.

Kompleksitas waktu juga pada umumnya digunakan untuk melakukan perkiraan terhadap waktu yang diperlukan suatu algoritma pada sebuah komputer. Tiap komputer memiliki kece[atan yang berbeda-beda, tetapi kompleksitas waktu dibuat sedemikian rupa sehingga dapat menyesuaikan dengan kondisi komputer yang berbeda-beda.

Misalkan ada sebuah algortima yang memiliki jumlah operasi n untuk besar masukan n, sehingga memiliki kompleksitas waktu T(n)=n. Jika kita mengetahui bahwa waktu yang diperlukan komputer A untuk melakukan satu kali operasi tersebut sebesar x detik, maka kira-kira waktu yang diperlukan algoritma tersebut di Komputer A adalah nx detik. Kemudian jika komputer B memerlukan waktu y detik untuk melakukan satu operasi tersebut, maka waktu yang diperlukan algoritma tersebut di komputer b adalah xb detik.

Hal lain yang juga berhubungan dengan kompleksitas waktu adalah parameter yang mencirikan ukuran masukan. Contohnya adalah algoritma pencarian suatu elemen pada array. Jumlah operasi yang dilakukan tidak selalu sama, bergantung pada posisi elemen tersebut dalam array. Karena itu, kompleksitas waktu dibedakan atas tiga macam, kompleksitas waktu untuk kasus terburuk ($T_{max}(n)$), kompleksitas waktu untuk kasus terbaik ($T_{min}(n)$), dan kompeksitas waktu rata-rata ($T_{avg}(n)$).

Hubungan pertambahan waktu dan bretambahnya besar ukuran dapat dilihat dari notasi kompleksitas waktu asimtotiknya, salah satunya adal notasi O besar yang mengabaikan koefisien dan variabel berorede rendah dari T(n). Dengan notasi O besar kita bisa tahu batas atas dari pertumbuhan suatu algoritma seiring dengan bertambahnya ukuran masukan. Contohnya, untu T(n)= n^2+1, notasi O besarnya adalah O(n^2). Kaena berorde 2 maka pertumbuhannya cenderung kuadratik.



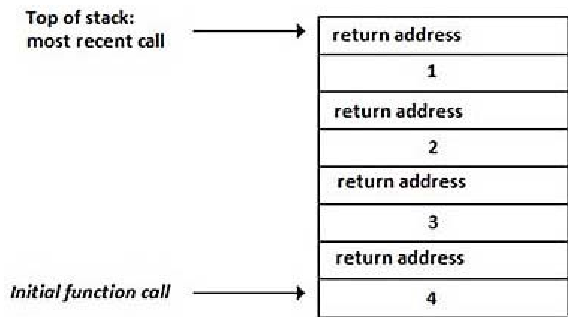
Gambar 4: Grafik yang menggambarkan pertumbuhan fungsi berdasarkan notasi O besar (Sumber: http://www.objc.io/issue-7/collections.html)

Selain itu ada juga notasi Omega besar dan Tetha besar. Berbeda dengan O besar, Omega besar menyediakan batas bawah untuk fungsi T(n). Tetha besar ditentukan oleh orde dari O besar dan Omega besar.

III. KOMPLEKSITAS WAKTU DAN KECEPATAN PROGRAM DENGAN ALGORITMA REKURSIF DAN ITERATIF

Jika kita melakukan perhitungan untuk fungsi yang melakukan algoritma rekursif, kemudian dibandingkan dengan fungsi yang menggunakan algoritma iteratif untuk tujuan yang sama, dan dengan jumlah langkah yang sama, akan menghasilkan persamaan kompleksitas waktu yang sama pula. Kenyataannya, jika kita menjalankan kedua fungsi tersebut pada lingkungan yang sama, waktu yang diperlukan bisa berbeda. Hal ini disebabkan karena perlakuan komputer terhadap algoritma rekursif dan iteratif itu bisa berbeda.

Jika komputer melakukan eksekusi fungsi dengan algoritma rekursif sebagaimana fungsi rekursif dikerjakan secara manual, maka setiap kali melakukan pemanggilan fungsi, komputer akan melakukan alokasi ke stack memory dan melakukan penyimpanan untuk keluaran dari masing-masing fungsi yang terpanggil. Hal ini menyebabkan adanya waktu tambahan yang diperlukan selain untuk melakukan proses *assignment*.



Gambar 5: penggambaran kondisi stack untuk algoritma rekursif
(Sumber: http://www.informit.com/articles/article.aspx?p=1874865)

Pada kasus lain, jika program tersebut ditulis dengan bahasa pemrograman fungsional, atau di-*compile* dengan *compiler* yang bisa melakukan optimasi untuk algoritma rekursif, maka akan ada optimasi untuk jenis algoritma rekursi yang menyebabkan perlakuan terhadap program tersebut kurang lebih sama dengan fungsi yang menggunakan algoritma iteratif yang ditulis dengan bahasa pemrograman imperatif.

Karena itu, jika kita melakukan perhitungan kompleksitas waktu antara kedua fungsi itu (rekursif dan iteratif), persamaan kompleksitas waktu yang dihasilkan bisa tak berlaku, dalam hal bahwa kedua fungsi itu memiliki persamaan kompleksitas waktu yang sama. Dan

jika perhitungan yang sama dilakukan kepada fungsi yang mengimplementasikan keduanya, hasilnya bisa benar-benar tak sesuai dengan kenyataannya.

Lantas, apa yang sebaiknya dilakukan untuk menetukan kompleksitas waktu? Sebenarnya masalahnya disebabkan karena perhitungan kompleksitas waktu hanya ditentukan oleh jumlah *assignment* yang dilakukan oleh suatu algoritma. Jadi jika kita melakukannya dengan memperhitungkan jumlah pemanggilan fungsi, nilainya juga bisa berbeda.

Tetapi yang menjadi masalah jika kita memperhitungkan jumlah pemanggilan fungsi, adalah nilainya. Misalkan kita menentukan besarnya waktu yang diperlukan untuk melakukan *assignment* dengan n, kita tidak mungkin menyatakan bahwa waktu yang diperlukan untuk memanggil fungsi juga sebesar n, sebab waktu yang diperlukan oleh komputer untuk memanggil fungsi tidak selalu sama dengan n.

Jadi, coba kita misalkan waktu yang diperlukan untuk memanggil fungsi adalah f. Kemudian kita mencoba melakukan perhitungan kompleksitas waktu dengan peratruna tersebut untuk fungsi berikut.

```
 int func (int n){
int func (int n){
        int res = n;
        for (i=n-1;  n>=0; i--){
                rest += i;
        }                                 -1);
        return res;
}
```

Gambar 3: Fungsi func dengan algoritma rekursif (Bahasa C)

Gambar 6: Fungsi func dengan algoritma iteratif (Bahasa C)

Kita lihat bahwa dalam melakukan perhitungan, dalam fungsi dengan algoritma rekursif, dilakukan operasi yang memanggil fungsi. Kita tahu bahwa setiap fungsi pasti melakukan 1 operasi, maka nilainya sebesar f+1, untuk setiap funsi dipanggil. Lantas, karena fungsi tersebut dipanggil sebesar n+1 kali maka nilainya (n+1)(f+1).

Jadi jika kita bandingkan, fungsi dengan algoritma rekursif memiliki kompleksitas waktu T(n)=(n+1)(f+1) sementara untuk fungsi dengan algoritma iteratif memiliki kompleksitas waktu T(n)=n+1.

Kenyataanya, ada kalanya *compiler* menerjemahkan suatu algoritma rekursif sebagaimana algoritma iteratif, yang berarti kompleksitas waktu yang dihasilkan kurang lebih sama. Hal ini berarti tidak ada pemanggilan fungsi dalam eksekusi fungsi, sehingga kita bisa menganggap nilai f = 0. Sehingga jika diterapkan ke fungsi yang telah disebutkan, akan ditemukan bahwa kompleksitas waktu kedua fungsi adalah T(n)=n+1.

Tetapi cara di atas juga masih belum bisa menunjukkan kompleksitas waktu secara umum, karena ada kalanya program dijalankan dalam lingkugan fungsional, yang membuat komputer melakukan optimasi terhadap fungsi rekursif, tetapi tidak untuk fungsi iteratif, dan menyebabkan fungsi iteratif menjadi lebih lambat daripada fungsi rekursif.

Jadi, bagaimana cara menggambarkan kompleksitas waktu secara umum yang berlaku untuk semua kasus? Di sinilah notasi O besar berperan. Memang tidak bisa menentukan secara persis waktu atau jumlah langkah yang dilakukan suatu algoritma, tetapi menjelaskan hubungan besaran input terhadap waktu yang diperlukan fungsi tersebut. Fungsi dengan kompleksitas waktu yang linier (O(n)) tentu lebih baik daripada fungsi dengan kompleksitas waktu kuadratik ($O(n^2)$) jika melihat kompleksitas waktunya.

Jika kita tentukan notasi O besar dari kedua fungsi yang digunakan sebelumnya, kita akan temukan bahwa notasi O besar dari keduanya adalah O(n). Karena berorede satu, maka pertumbuhan dari seiring dengan bertambahnya ukuran n bersifat linier. Tetapi tetap saja tak bisa menentukan algoritma yang lebih baik di antara dua fungsi tersebut.

IV KESIMPULAN

Pada akhirnya, yang dapat kita temukan aalah bahwa kompleksitas waktu suatu algoritma tidak bisa menjadi satu-satunya acuan untuk menentukan kecepatan kerja program. Untuk dua program yang secara teori memiliki kompleksitas waktu yang sama, dapat memiliki kecepatan yang berbeda meskipun dijalankan di lingkungan yang sama, karena perbedaan perlakuan antara algoritma rekursif dan algoritma iteratif.

DAFTAR PUSTAKA

(1) 1. http://www.informit.com/articles/article.aspx?p=1874865, diakses tanggal 10 Desember 2014

(2) 2. http://www.informit.com/articles/article.aspx?p=1390173&seqNum=3 diakses tanggal 10 Desember 2014

(3) 3.http://www.cs.odu.edu/~toida/nerzic/content/recursive_alg/rec_alg.html diakses tanggal 11 Desember 2014

(4) 4. http://www.objc.io/issue-7/collections.html diakses tanggal 11 Desember 2014

(5) 5. Rinaldi Munir, *Diktat Kuliah IF2120 Matematika*

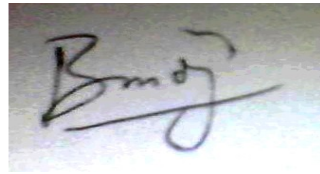
(6) *Diskrit.Program Studi Teknik Informatika*, 2006.

(7) 6. Rosen, Kenneth H. *Discrete Mathematics and Its Applications 7th Edition*, McGraw-Hill, 2012.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 11 Desember 2014



Binanda Smarta Aji - 13512069