Penerapan Kompleksitas Algoritma untuk Mengetahui Keefektifan Algoritma Baca File dengan File Dummy

Sonny Fitra Arfian (13510009)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
sonny.fa@students.itb.ac.id

***Abstrak*—Pembuatan algoritma untuk membaca sebuah file agar dimengerti oleh program dapat dibuat melalui berbagai cara, salah satunya memakai file dummy. Dalam pembahasan kali ini akan dilihat sebarapa kompleks algoritma tersebut dan apakah algoritma tersen=but cukup mangkus untuk digunakan dalam aplikasi berskala besar.**

***Index Terms*—algoritma, C, dummy, kompleksitas, O-besar**

I. PENDAHULUAN

Algoritma adalah suatu urutan langkah-langkah penyelesaian masalah secara matematis. Algoritma yang baik adalah algoritma yang baik dan mangkus (efisien). Definisi mangkus disini adalah seberapa cepat algoritma tersebut dilakukan. Algoritma yang benar tetapi memiliki waktu penyelesaian yang terlalu lama tentu akan mengganggu kinerja algoritma tersebut.

Kemangkusan suatu algoritma dapat diukur dari jumlah waktu dan ruang (space) memori yang dibutuhkan untuk menjalankannya. Besarnya waktu dan ruang yang dibutuhkan suatu algoritma sangat bergantung pada ukuran masukan (input) yang diproses. Seiring dengan bertambahnya masukan maka waktu dan ruang yang dibutuhkan untuk memproses algoritma tersebut akan bertambah. Hal inilah yang menyebakan kemangkusan algoritma penting.

Kemangkusan suatu algoritma juga dapat digunakan untuk membandingkan berbagai algoritma yang akan digunakan. Dalam penyelesaian masalah, dapat digunakan berbagai algortima tertentu yang memiliki cara kerja yang berbeda. Untuk itu haruslah diketahui algoritma mana yang paling sesuai untuk digunakan dalam program.

I.I KOMPLEKSITAS WAKTU DAN RUANG

Kompleksitas suatu algoritma dapat dibagi menjadi 2 jenis, yaitu kompleksitas waktu dan komoleksitas ruang. Kompleksitas waktu, T(n), merupakan kompleksitas algoritma yang diukur dari banyaknya operasi atau instruksi yang dieksekusi. Sedangkan kompleksitas ruang, S(n), adalah kompleksitas algoritma yang diukur dari jumlah memori yang digunakan.

Kedua jenis kompleksitas ini sukar untuk dilihat secara langsung. Hal ini deisebabkan oleh beberapa factor berikut :

1. Pada kenyataannya waktu eksekusi suatu operasi sukar diamati.
2. Komputer dengan arsitektur yang berbeda akan menghasilkan waktu eksekusi operasi yang berbeda pula.
3. *Compiler* suatu bahasa pemrograman menerjemahkan algoritma dengan cara yang berbeda-beda.

Maka dari itu, harus ditetapkan suatu model pengukuran abstrak yang bersifat independen dari jenis/spesifikasi komputer dan *compiler* yang digunakan. Besaran yang dipakai untuk menerangkan model abstrak pengukuran ruang/waktu ini adalah kompleksitas algoritma. Terminologi yang diperlukan dalam membahas kompleksitas waktu dan kompleksitas ruang adalah :

1. Ukuran besar masukan data suatu algoritma adalah *n*.
2. Kompleksitas waktu, *T(n)*, adalah waktu yang dibutuhkan untuk melaksanakan algoritma untuk ukuran masukan *n*.
3. Kompleksitas ruang, *S(n)*, adalah ruang memori yang dibutuhkan algoritmasebagai fungsi dari ukuran masukan *n*.

Dikarenakan memory pada komputer sekarang telah cukup besar, maka hal yang akan lebih diperhatikan adalah kompleksitas waktu dari suatu algoritma.

Terdapat tiga macam kompleksitas waktu, yaitu :

1. $T_{max}(n)$: kompleksitas waktu untuk kasus terburuk (*worst case*).
2. $T_{min}(n)$: kompleksitas waktu untuk kasus terbaik (*best case*).
3. $T_{avg}(n)$: kompleksitas waktu untuk kasus rata-rata (*average case*).

I.II KOMPLEKSITAS WAKTU ASIMPTOTIK

Perbandingan pertumbuhan $T(n)$ dengan n^2 pada $T(n) = 2n^2 + 6n + 1$.

n	$T(n) = 2n^2 + 6n + 1$	n^2
10	261	100
100	2061	1000
1000	2.006.001	1.000.000
10.000	2.000.060.001	1.000.000.000

- Untuk n yang besar, pertumbuhan $T(n)$ sebanding dengan n^2. Pada kasus ini, $T(n)$ tumbuh seperti n^2 tumbuh.

- $T(n)$ tumbuh seperti n^2 tumbuh saat n bertambah. Kita katakan bahwa $T(n)$ berorde n^2 dan kita tuliskan

$$T(n) = O(n^2)$$

Notasi "O" diatas adalah notasi "O-Besar" yang merupakan notasi kompleksitas waktu asimptotik.

Pengelompokan Algoritma Berdasarkan Notasi *O*-Besar

Kelompok Algoritma	Nama
$O(1)$	konstan
$O(\log n)$	logaritmik
$O(n)$	lanjar
$O(n \log n)$	$n \log n$
$O(n^2)$	kuadratik
$O(n^3)$	kubik
$O(2^n)$	eksponensial
$O(n!)$	faktorial

Dalam analisis algoritma, terdapat beberapa acuan penting yang diikuti, yaitu :

1. Pengisian nilai (*assignment*), perbandingan, operasi aritmatik, *read*, *write* membutuhkan waktu O(1).
2. Pengaksesan elemen larik atau memilih *field* dari sebuah *record* membutuhkan waktu O(1).
3. if C then S1 else S2 membutuhkan waktu pengerjaan $T_C + \max(T_{S1}, T_{S2})$.
4. Kompleksitas untuk for adalah jumlah pengulangan dikali dengan kompleksitas waktu badan.
5. while S do C; dan repeat C until S; Waktunya adalah jumlah pengulangan dikali waktu badan C dan S. Jika S tidak diketahui maka waktunya tidak dapat dihitung.
6. Waktu untuk pemanggilan prosedur dan fungsi adalah O(1).
7. Prosedur/fungsi rekursif dihitung dengan relasi rekurens.

II. ANALISIS ALGORITMA

Algoritma baca file eksternal dapat dibuat dengan berbagai jenis. Disini penulis menggunakan algoritma baca file eksternal menggunakan file dummy. Algoritma ini digunakan dalam program "*Facepalmer*" untuk Tugas Besar kuliah IF2030 tahun 2011/2012. Algoritma tersebut ditulisa dalam bahasa C, sehingga seluruh algoritma yang ada dalam tulisan ini akan berupa source code bahasa C. Diharapkan pembaca telah memahami bagaimana membuat source code dalam bahasa C.

File yang akan dibaca adalah file pada tabel 1 dibawah. File tersebut akan dibaca oleh program dan menyimpannya pada dua buah array. Array pertama adalah array untuk menampuh data user. Data ditandai dengan sebuah string "*@user*" pada awal string pada teks. Data user sendiri terdiri dari *email*, *nama*, *tanggal lahir*, *kota*, *universitas*, dan *SMU*. Array kedua adalah array untuk menampung data friend yang terdiri dari *from* dan *to*. Data friend diawali oleh string "*@friend*" pada file. Untuk lebih jelasnya dapat dilihat tabel 2 dibawah.

```
@user
amin@gmail.com "Amin Badrun" 1-8-1990 "Bandung" "ITB" "SMUN 3 Bandung"#
cici@email.com "Cici Dadan" 12-7-1990 "Bandung" "UNPAD" "SMUN 8 Jakarta"#
emi_fitria@gmx.de "Emi Fitria" 23-6-1990 "Surabaya" "ITB" "SMUN 5 Surabaya"#
hamdan@hotmail.com "Gilang Hamdan" 30-5-1990 "Jakarta" NIL "SMUN 8 Jakarta"#
dana@yahoo.com "Dana Windar" 12-12-1990 "Semarang" "UNS" "SMUN 1 Semarang"#
ja2ng@yahoo.com "Iwan Jajang" 4-11-1990 "Jakarta" "UNIKOM" "SMUN 70 Jakarta"#
@friend
amin@gmail.com ->cici@email.com
amin@gmail.com ->ja2ng@yahoo.com
amin@gmail.com ->dana@yahoo.com
cici@email.com ->amin@gmail.com
cici@email.com ->emi_fitria@gmx.de
cici@email.com ->dana@yahoo.com
ja2ng@yahoo.com ->cici@email.com
ja2ng@yahoo.com ->hamdan@hotmail.com
hamdan@hotmail.com ->ja2ng@yahoo.com
hamdan@hotmail.com ->emi_fitria@gmx.de
@end
```

Tabel 1. File yang dibaca

Selain itu, demi tercapainya sasaran makalah ini dengan tepat maka akan digunakan kompleksitas waktu asimtotik.

```
typedef struct {
        char email[64];
```

```
        char nama[64];
        tanggal ttl;
        char kota[64];
        char univ[64];
        char smu[64];
} ElmtUser;

typedef struct {
        char from[64];
        char to[64];
} ElmtFriend;
```

Tabel 2. Definisi array user dan friend

II.I KOMPLEKSITAS BACA USER

Setalah kita mengetahui bagaimana cara membaca file yang diharapkan, kita akan melihat algoritma yang digunakan untuk membaca file pada bagian user.

```
void RetUser (char line[], ElmtUser
*Usr){
/* Mengambil data dari line dan
memasukkannya ke dalam database user
*/
//kamus
    char Kata[100];
//algoritma
    DSTART();
    ReadyDummy(line);
    GetUserD();
    CopyStrng(CKata,(*Usr).email);
    GetUserD();
    CopyStrng(CKata,(*Usr).nama);
    GetUserD();
    GetTanggal(CKata,&(*Usr).ttl);
    GetUserD();
    CopyStrng(CKata,(*Usr).kota);
    GetUserD();
    CopyStrng(CKata,(*Usr).univ);
    GetUserD();
    CopyStrng(CKata,(*Usr).smu);
    fclose(dummy);
}
```

Tabel 3. Algoritma Baca User

Dilihat secara sekilas, algoritma diatas cukup linear. Harus dilihat lagi apa algoritma yang menyusun algoritma baca user diatas.

```
void DSTART () {
    dummy = fopen("dummy.txt","w");
    DADV();
}
void ReadyDummy(char line[]){
    retval=fprintf(dummy,"%s^",line);
    fclose(dummy);
    fopen("dummy.txt","r");
    DADV();
}
```

Tabel 4. Algoritma penyusun baca file

Algoritma diatas merupakan salah satu penyusun algoritma baca file. Demi ketercapaian tujuan utama tulisan ini, penulis tidak akan membahas secara detail algoritma pelengkap yang ada pada makalah ini.

Pada prosedur DSTART, O(n) merupakan kompleksitas linear karena tidak ada pengaruh dari masukan. O(n) dari prosedur ini dinyatakan sebagai berikut :

$$Kompleksitas\ total = O(1) + O(1)$$
$$= O(1)$$

Prosedur ReadyDummy juga memiliki O(n) yang linear sehingga dapat disimpulkan bahwa kompleksitas total dari prosedur tersebut adalah :

$$O(1) + O(1) + O(1) + O(1) = O(1)$$

```
void CopyStrng(char cpy1[],char
cpy2[]){
/**/
//kamus
    int i;
//algoritma
    for (i=0;i<=NMax;i++){
        cpy2[i]=cpy1[i];
        if
(cpy1[i]==SNil){cpy2[i]=SNil; break;}
    }
                    }
```

Tabel 5. Algoritma CopyStrng

Algoritma CopyStrng mepunyai kompleksitas yang menyesuaikan dengan ukuran string yang akan disalin. Walaupun terlihat bahwa kalang for akan berulang hingga *n* tetapi kondisi pada if akan menghentikan proses ketika string telah disalin. Kondisi if sendiri mempunyai kompleksitas yang linear, $O(1) + O(1) = O(1)$. Maka kompleksitas dari prosedure diatas adalah :

$$Komplesitas\ total$$
$$= n.\left(O(1) + \left(O(1) + O(1)\right)\right)$$
$$= n.O(1) = O(n)$$

Algoritma berikutnya yang akan kita lihat adalah algoritma GetUserD.

```
void GetUserD()
/* Mengakuisisi kata, menyimpan dalam
CKata
I.S. : CC adalah karakter pertama dari
kata
F.S. : CKata berisi kata yang sudah
diakuisisi;
CC = BLANK atau CC = MARK;
CC adalah karakter sesudah karakter
terakhir yang
diakuisisi */
{
/* Kamus Lokal */
    int i; //= 1; /* inisialisasi */
```

```
/* Algoritma*/
    DIgnoreBlank();
    if
((DCC=='"')||(DCC=='”')||(DCC=='`')) {
        DADV();
        for (i=0;i<=NMax;i++) {
            CKata[i] = DCC;
            DADV();
            if
((DCC=='"')||(DCC=='”')||(DCC=='`')) {
                DADV();
                break;
            }
        } /* CC = MARK or CC = BLANK
*/
    } else {
        for (i=0;i<=NMax;i++) {
            CKata[i]=DCC;
            DADV();
            if (DCC==BLANK) {
                break;
            }
        }
    }
    CKata[i+1]=SNil;
}
```

Tabel 6. Algoritma GetUserD

DIgnoreBlank adalah algoritma untuk mengabaikan string blank atau berisi spasi. Algoritma ini memiliki kompleksitas $O(n)$karena berjalan tergantung jumlanh blank yang ditemukan. Untuk kondisi if sendiri mempunyai kompleksitas sebagai berikut :

$C = O(1) + O(1)$

$S1 = O(n); karena\ cara\ kerjanya\ mirip\ CopyStrng$

$S2 = O(n); sama\ dengan\ S1$

Sehingga kompleksitas totalnya

$= O(1) + \max(O(n), O(n))$

$= O(n)$

Untuk nilai kompleksitas total dari GetUserD sendiri adalah sebagai berikut :

$Kompleksitas\ total = O(n) + O(n) + O(1)$

$= O(n)$

Dengan menganalisa bagian-bagian tersebut, maka dapat dicari kompleksitas dari algoritma Baca User itu sendiri. Jika fungsi fclose mempunyai kompleksitas $O(1)$, maka kompleksitas Baca User (dengan menghilangkan prosedur yang sama karena $O(x) + O(x) = O(x)$) adalah

$Kompleksitas\ Baca\ User$

$= O(1) + O(1) + O(n) + O(n) + O(1)$

$= O(n)$

Perlu diperhatikan bahwa GetTanggal tidak diperhitungkan karena merupakan suatu pengembangan dari GetUserD dan memiliki kompleksitas yang sama yaitu $O(n)$.

II.II Kompleksitas Baca Friend

Baca Friend tidak serumit Baca User, tetapi memiliki prosedur yang relatif sama.

```
void RetFriend (char line[],
ElmtFriend *Frd){
/**/
    DSTART();
    ReadyDummy(line);
    GetFriendD();
    CopyStrng(CKata,(*Frd).from);
    GetFriendD();
    CopyStrng(CKata,(*Frd).to);
                }
```

Tabel 7. Algoritma Baca Friend

Algoritma baru yang perlu diketahui adalah GetFriendD. Algoritma ini setara dengan GetUserD pada algoritma Baca User.

```
void GetFriendD()
/* Mengakuisisi kata, menyimpan dalam
CKata
I.S. : CC adalah karakter pertama dari
kata
F.S. : CKata berisi kata yang sudah
diakuisisi;
CC = BLANK atau CC = MARK;
CC adalah karakter sesudah karakter
terakhir yang
diakuisisi */
{
/* Kamus Lokal */
    int i; //= 1; /* inisialisasi */
/* Algoritma*/
    DIgnoreBlank();
    DIgnoreArrow();
    for (i=0;i<=NMax;i++) {
        CKata[i] = DCC;
        DADV();
        printf("%c<",DCC);
        if (DCC==SNil) {
            printf("\nNULLL");
        }
        if ((DCC==BLANK)||(DCC=='^'))
{
            break;
        }
    }
    CKata[i+1]=SNil;
}
```

Tabel 8. Algoritma GetFriendD

Algoritma diatas mempunyai kompleksitas yang mirip dengan GetUserD, sehingga O(n) dari GetFriendD sama dengan GetUserD. Kompleksitas GetFriendD terdiri dari 3 code linear, dengan kompleksitas $O(1)$, dan sebuah kalang for. Kalang ini mempunyai kompleksitas $n.\big(O(1) + O(1) + O(1) + O(1) + O(1)\big) = O(n)$ dengan kompleksitas pada bagian if sama-sama $O(1) + O(1) = O(1)$. Sehingga kompleksitas total dari algoritma ini adalah

$$kompleksitas\ total = O(1) + O(1) + O(n) + O(1)$$
$$= O(n)$$

Dengan demikian, kompleksitas dari baca friend adalah sebagai berikut : (dengan alasan yang sama, code yang sama dihilangkan)

$$kompleksitas\ baca\ friend$$
$$= O(1) + O(1) + O(n) + O(n)$$
$$= O(n)$$

II.III KOMPLEKSITAS BACA FILE DENGAN DUMMY

Dengan diketahuinya dua komponen utama baca file dengan menggunakan file dummy, maka kompleksitas keseluruhan algoritma ini dapat ditemukan. Kompleksitas tersebut dihitung sebagai berikut :

$$kompleksitas\ baca\ file$$
$$= kompleksitas\ baca\ user + kompleksitas\ baca\ friend$$
$$= O(n) + O(n)$$
$$= O(n)$$

Yang perlu diperhatikan adalah bahwa kompleksitas tersebut dipakai untuk satu buah masukan atau jika dilihat dari file inputnya adalah satu baris string. Untuk kompleksitas keseluruhan dari algoritma tersebut, perlu dikalikan n buah baris yang ada. Kompleksitasnya menjadi :

$$Kompleksitas\ baca\ file = n.O(n)$$
$$= O(n^2)$$

Algoritma dengan kompleksitas waktu asimtotik $O(n^2)$ adalah algoritma yang tidak cocok untuk menangani persoalan yang mempunyai input terlalu besar. Walaupun pada program “Facepalmer” algoritma berjalan dengan baik, input yang digunakan belum begitu besar. Hal ini sejalan dengan kemampuan komputer jaman sekarang yang sudah cukup cepat untuk penggunaan personal.

III. KESIMPULAN

Dari hasil penelusuran kompleksitas waktu asimtotik diatas dapat disimpulkan bahwa algoritma Baca File menggunakan File Dummy mempunyai kompleksitas asimtotik $O(n^2)$. Hal ini mempunyai arti bahwa algoritma ini mempunyai waktu pelaksanaan yang kuadratik, sehingga kurang cocok untuk persoalan dengan input yang cukup besar.

Definisi masukan besar sendiri relatif dengan perkembangan teknologi. Semakin maju teknologi komputer, definisi masukan besar akan bergeser ke arah data yang lebih besar.

Untuk persoalan yang kecil, algoritma ini bisa digunakan dan dapat memberi kelebihan dalam pencarian masalah pada program. Hal ini dilihat dari cara kerja program “Facepalmer”.

REFERENSI

[1] Munir Renaldi, *Diktat Kuliah IF2091: Struktur Diskrit*, Edisi Keempat, Bandung: Program Studi Teknik Informatika STEI, 2008, hlm. X-1 – X-28.

[2] http://kuliah.itb.ac.id/mod/resource/view.php?id=7137. Tanggal 12/12/2011 pukul 08.00

[3] Facepalmer, Tugas Besar IF2030 2011/2012, Kelas 4, Kelompok 8

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, 12 Desember 2011

ttd

Sonny Fitra Arfian (13510009)