Kompleksitas dari Algoritma-Algoritma untuk Menghitung Bilangan Fibonacci

Gregorius Ronny Kaluge – NIM : 13508019

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jalan Ganesha 10, Bandung
e-mail: if18019@students.if.itb.ac.id, miliknya_ronny@yahoo.co.id

ABSTRAK

Makalah ini akan membahas beberapa algoritma yang bisa digunakan untuk menghitung bilangan Fibonacci. Bilangan Fibonacci adalah bilangan yang terdapat pada deret Fibonacci. Deret Fibonacci biasa didefinisikan secara rekursif sebagai : f(n) = f(n-1) + f(n-2) dengan f(0) dan f(1) adalah sebuah konstanta.

Akan ada 4 algoritma dan 1 rumus yang akan dibahas pada makalah ini. Algoritma-algoritma tersebut yaitu algoritma rekursif, iteratif, dan algoritma iteratif dengan menggunakan matriks. Algoritma iteratif menggunakan matriks yang dibahas nanti akan memiliki 2 versi. Versi pertama menggunakan teknik pemangkatan secara linear sedangkan versi kedua menggunakan teknik *divide-and-conquer*. Pada masing-masing algoritma dan rumus, akan diberikan contoh program dalam bahasa PASCAL agar bisa memberi sedikit gambaran tentang implementasinya.

Masing-masing algoritma tersebut, beserta rumus untuk menghitung Fibonacci, akan dianalisis untuk dicari kompleksitasnya, kasus terbaik, kasus terburuk, dan notasi O besarnya. Selanjutnya, dari 4 algoritma dan 1 rumus tersebut akan ditentukan cara yang paling mangkus untuk menghitung bilangan Fibonacci dari segi waktu.

Kata kunci: Fibonacci, kompleksitas algoritma, *divide-and-conquer*

1. PENDAHULUAN

Bilangan Fibonacci memiliki cukup banyak aplikasi dalam dunia matematika maupun komputer. Contohnya [1]:

- Kasus terburuk untuk algoritma Euclid dalam menentukan FPB (Faktor Pembagi Terbesar) adalah jika inputnya sepasang bilangan Fibonacci yang berurutan
- Bilangan Fibonacci digunakan untuk membangkitkan bilangan *pseudorandom*.
- *Fibonacci search technique* menggunakan bilangan Fibonacci

Sebelum menggunakan bilangan Fibonacci tentunya perlu diketahui besarnya bilangan yang akan digunakan tersebut. Untuk menghitungnya, ada beberapa algoritma yang bisa digunakan. Tentu saja, pemilihan algoritma yang akan digunakan harus dilakukan sebaik-baiknya agar diperoleh algoritma yang semangkus mungkin.

2. DERET FIBONACCI [1]

Deret Fibonacci diperkenalkan pada tahun 1202 oleh Leonardo dari Pisa (yang dikenal sebagai Fibonacci). Deret ini didefinisikan secara rekursif sebagai:

f(n) = f(n-1) + f(n-2)
f(0) = 0
f(1) = 1

Sebenarnya terdapat versi lain untuk f(0) dan f(1) yaitu f(0) = 1, f(1) = 1. Namun dalam makalah ini, deret Fibonacci yang digunakan adalah deret yang nilai f(0) = 0 dan f(1) = 1.

Selain itu, deret Fibonacci yang dibahas pada makalah ini adalah deret Fibonacci untuk indeks n bilangan bulat non negatif.

3. ALGORITMA REKURSIF

3.1. Algoritma

Algoritma rekursif ini langsung mengimplementasikan definisi rekursif bilangan fibonacci dalam bentuk fungsi rekursif. Contoh fungsi fibonacci dalam bahasa PASCAL:

```
function fibo (n : integer): integer;
begin
  if n=0 then
    fibo:=0
  else if n=1 then
    fibo:=1
  else
    fibo:=fibo(n-1)+fibo(n-2);
end;
```

3.2. Analisis

Dari fungsi rekursif di atas, bisa dilihat bahwa T(0) dan T(1) bernilai 1 karena jika inputnya 0 atau 1, program akan melakukan 1 proses yaitu *assignment* (pemberian nilai). T(n) bernilai T(n-1) + T(n-2) + 1 karena di setiap tahap rekurens, program akan memanggil fibo(n-1) yang memiliki kompleksitas T(n-1) dan fibo(n-2) yang memiliki kompleksitas T(n-2), sementara itu untuk menjumlahkan keduanya dibutuhkan 1 proses.

Oleh karena itu:

$$T(n) = \begin{cases} 1 & , untuk\ 0 \le n \le 1 \\ T(n-1)+T(n-2)+1, & untuk\ n\ lainnya \end{cases}$$

Jika n=2, T(n) = T(1) + T(0) + 1 = 3, f(n+2) = 3
Jika n=3, T(n) = T(2) + T(1) + 1 = 5, f(n+2) = 5
Jika n=4, T(n) = T(3) + T(2) + 1 = 9, f(n+2) = 8
Jika n=5, T(n) = T(4) + T(3) + 1 = 15, f(n+2) = 13
Jika n=6, T(n) = T(5) + T(4) + 1 = 25, f(n+2) = 21
Jika n=7, T(n) = T(6) + T(5) + 1 = 41, f(n+2) = 34
Jika n=8, T(n) = T(7) + T(6) + 1 = 67, f(n+2) = 55

Dari situ bisa kita lihat bahwa T(n) akan berkembang secara eksponensial. Untuk mencari notasi O besarnya:

$T(n) \le T(n-1) + T(n-1) + T(n-1)$
$T(n) \le 3 \times T(n-1)$

Sehingga diperoleh $T(n) = O(3^n)$. Algoritma ini adalah algoritma yang stabil, kompleksitas untuk kasus terbaik sama dengan kompleksitas untuk kasus terburuknya.

4. ALGORITMA ITERATIF

4.1. Algoritma

Algoritma ini mirip dengan algoritma rekursif namun ada sedikit perbedaan. Pada algoritma ini, fungsi rekursif digantikan oleh sebuah iterasi. Idenya adalah sebagai berikut :

1) Menyimpan nilai f(0) dan f(1) dalam 2 variabel
2) Menghitung nilai f(2) dari nilai f(0) dan f(1) yang sudah disimpan sebelumnya, lalu menyimpan nilai f(2) dan f(1)
3) Menghitung nilai f(3) dari nilai f(1) dan f(2) yang sudah disimpan sebelumnya, lalu menyimpan nilai f(3) dan f(2)
4) Proses tersebut dilakukan berulang-ulang sampai diperoleh nilai f(n)

Contoh potongan programnya dalam bahasa PASCAL:

```
a:=0; //a diisi dengan nilai f(0)
b:=1; //b diisi dengan nilai f(1)
for i:=2 to n do
begin
  c:=b;
  b:=a+b; //b akan bernilai f(n)
  a:=c;
end;
```

4.2. Analisis

Dari potongan program di atas, langsung terlihat bahwa kompleksitas waktunya adalah:

$T(n) \le 2 + 3(n-1) = 3n - 1$

Dengan nilai T(n) tersebut, bisa ditentukan notasi O besarnya:

$T(n) \le 3n-0$
$T(n) \le 3n$

Sehingga diperoleh T(n) = O(n).

Untuk nilai n >1 T(n) pasti bernilai 3n-1. Oleh karena itu, algoritma ini adalah algoritma yang stabil. Jadi kompleksitas untuk kasus terbaik sama dengan kompleksitas untuk kasus terburuknya.

5. ALGORITMA ITERATIF MENGGUNAKAN MATRIKS

5.1. Versi I

5.1.1. Algoritma

Ide dasar algoritma ini adalah merepresentasikan bilangan Fibonacci ke-(n+1) untuk n>0 sebagai:

$$\begin{pmatrix} f(n+1) \\ f(n) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} f(n) \\ f(n-1) \end{pmatrix}$$

Dari informasi tersebut, bisa dicari rumus umumnya:

$$\begin{pmatrix} f(n+1) \\ f(n) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} f(n) \\ f(n-1) \end{pmatrix}$$

$$\begin{pmatrix} f(n+1) \\ f(n) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} f(n-1) \\ f(n-2) \end{pmatrix}$$

$$\begin{pmatrix} f(n+1) \\ f(n) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n \begin{pmatrix} f(1) \\ f(0) \end{pmatrix}$$

Sehingga jika kita ingin menghitung Fibonacci ke-10:

$$\begin{pmatrix} f(10) \\ f(9) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^9 \begin{pmatrix} f(1) \\ f(0) \end{pmatrix}$$

$$\begin{pmatrix} f(10) \\ f(9) \end{pmatrix} = \begin{pmatrix} 55 & 34 \\ 34 & 21 \end{pmatrix} \begin{pmatrix} f(1) \\ f(0) \end{pmatrix}$$

Dari hasil perkalian matriks diperoleh:

$f(10) = 55 \times f(1) + 34 \times f(0)$
$= 55 \times 1 + 34 \times 0$
$= 55$

Algoritmanya bisa dirumuskan sebagai berikut:

Algoritma Menghitung Fibonacci
Kamus
X : matriks satuan berukuran 2×2
A : matriks 2×2 dengan $a_{11}=a_{12}=a_{12}= 1$ dan $a_{22}= 0$
Hasil : integer

Algoritma
{perkalian (a,b) mengembalikan hasil perkalian matriks a dan b}
for i ← 1 to n-1
    X ← perkalian(X, A)
Hasil ← $a_{11} \times f(1) + a_{12} \times f(0)$

Contoh program dalam bahasa PASCAL:

```
type matriks= array[1..2,1..2]of int64;
var f: array[0..1]of integer;
    a,hasil: matriks;
    i,n: integer;
    fib: int64;
function kali (a, b: matriks): matriks;
//menghasilkan perkalian a dan b
begin
  kali[1,1]:=a[1,1]*b[1,1]+a[1,2]*b[2,1];
  kali[1,2]:=a[1,1]*b[1,2]+a[1,2]*b[2,2];
  kali[2,1]:=a[2,1]*b[1,1]+a[2,2]*b[2,1];
  kali[2,2]:=a[2,1]*b[1,2]+a[2,2]*b[2,2];
end;

begin
  //-----membaca data-----
  readln(n); //yg dihitung : fibonacci(n)
  //---inisialisasi f(0) dan f(1)---
  f[0]:=0;   f[1]:=1;
  //---inisialisasi matriks fibonacci---
  a[1,1]:=1;  a[1,2]:=1;
  a[2,1]:=1;  a[2,2]:=0;
  //---inisialisasi matriks hasil---
  hasil[1,1]:=1;  hasil[1,2]:=0;
  hasil[2,1]:=0;  hasil[2,2]:=1;
  //---menentukan output---
  if (n<2) then
    fib:=f[n]
  else begin
    //---mencari nilai a^(n-1)---
    for i:=1 to n-1 do
      hasil:=kali(hasil,a);
    //---menghitung output---
    fib:=hasil[1,1]*f[1]+hasil[1,2]*f[0];
  end;
  writeln(fib);
end.
```

5.1.2. Analisis

Jika dilihat dari program di samping, proses yang dilakukan selain membaca data adalah proses inisialisasi, perkalian bilangan, perkalian matriks, dan penjumlahan. Inisialisasi dilakukan sebanyak 10 kali. Proses perkalian matriks dilakukan sebanyak n-1 kali, setiap mengalikan matriks dilakukan 8 kali perkalian dan 4 kali penjumlahan. Yang terakhir, untuk menghitung outputnya, dilakukan 1 proses penjumlahan dan 2 proses perkalian. Dari informasi tersebut, dapat diperoleh:

$T(n) = 10 + (n-1)(8+4) + (2+1)$
$T(n) = 13 + 12n – 12$
$T(n) = 12n -1$

Selanjutnya, setelah memperoleh T(n), bisa diperoleh pula notasi O besarnya:

$T(n) \leq 12n -0$
$T(n) \leq 12n$

Jadi $T(n) = O(n)$.

Jika diperhatikan, algoritma ini adalah algoritma yang stabil sehingga kompleksitas untuk kasus terbaik sama dengan kompleksitas untuk kasus terburuknya.

5.2. Versi II

Sebelum membahas algoritma ini, ada baiknya kita membahas *divide-and-conquer*.

5.2.1. Divide-and-Conquer[2]

Divide-and-conquer adalah sebuah algoritma yang berkerja sebagai berikut:

1) Sebuah masalah dibagi menjadi beberapa bagian yang lebih kecil, idealnya dengan ukuran yang kurang lebih sama
2) Masalah-masalah yang kecil tersebut diselesaikan
3) Jika diperlukan, solusi yang diperoleh dari masalah-masalah yang kecil tersebut digabungkan untuk memecahkan masalah yang lebih besar

Contoh algoritma yang menggunakan teknik *divide-and-conquer* adalah *binary search*, *mergesort*, dan *quicksort*.

5.2.2. Algoritma

Algoritma ini sebenarnya memiliki ide dasar yang sama dengan versi I, yaitu menghitung nilai Fibonacci menggunakan pemangkatan matriks. Yang berbeda adalah cara menghitung pangkatnya. Seandainya, kita ingin menghitung f(20), kita bisa menggunakan rumus:

$$\begin{pmatrix} f(20) \\ f(19) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{19} \begin{pmatrix} f(1) \\ f(0) \end{pmatrix}$$

Selanjutnya, untuk menghitung pemangkatan matriks, bisa digunakan teknik *divide-and-conquer* sebagai berikut:

$$\begin{pmatrix}1 & 1\\ 1 & 0\end{pmatrix}^{19} = \begin{pmatrix}1 & 1\\ 1 & 0\end{pmatrix}^{9}\begin{pmatrix}1 & 1\\ 1 & 0\end{pmatrix}^{9}\begin{pmatrix}1 & 1\\ 1 & 0\end{pmatrix}^{1}$$

sedangkan

$$\begin{pmatrix}1 & 1\\ 1 & 0\end{pmatrix}^{9} = \begin{pmatrix}1 & 1\\ 1 & 0\end{pmatrix}^{4}\begin{pmatrix}1 & 1\\ 1 & 0\end{pmatrix}^{4}\begin{pmatrix}1 & 1\\ 1 & 0\end{pmatrix}^{1},$$

$$\begin{pmatrix}1 & 1\\ 1 & 0\end{pmatrix}^{4} = \begin{pmatrix}1 & 1\\ 1 & 0\end{pmatrix}^{2}\begin{pmatrix}1 & 1\\ 1 & 0\end{pmatrix}^{2},$$

dan

$$\begin{pmatrix}1 & 1\\ 1 & 0\end{pmatrix}^{2} = \begin{pmatrix}1 & 1\\ 1 & 0\end{pmatrix}^{1}\begin{pmatrix}1 & 1\\ 1 & 0\end{pmatrix}^{1}$$

Dari perhitungan di atas, bisa dirumuskan algoritma pemangkatannya sebagai berikut:

Function pangkat (n:integer, A:matriks) → matriks
{mengembalikan nilai A^n dengan prekondisi : n>0}
Kamus
temp : matriks

Algoritma
{perkalian (a,b) mengembalikan hasil perkalian matriks a dan b}
if (n=1)
  → A
else
  temp ← pangkat(n/2,A)
  if (n mod 2 = 1)
    → perkalian(perkalian(temp,temp), A)
  else
    → perkalian(temp,temp)

Contoh program dalam bahasa PASCAL:

```
type matriks= array[1..2,1..2]of int64;
var f: array[0..1]of integer;
    a,hasil: matriks;
    n:integer;
    fib: int64;

function kali (a, b: matriks): matriks;
//menghasilkan perkalian a dan b
begin
  kali[1,1]:=a[1,1]*b[1,1]+a[1,2]*b[2,1];
  kali[1,2]:=a[1,1]*b[1,2]+a[1,2]*b[2,2];
  kali[2,1]:=a[2,1]*b[1,1]+a[2,2]*b[2,1];
  kali[2,2]:=a[2,1]*b[1,2]+a[2,2]*b[2,2];
end;

function pangkat (n: integer; a:matriks):
 matriks;
//menghasilkan a^n dengan prekondisi n>0
var temp:matriks;
begin
  if (n=1) then //basis-1
    pangkat:=a
  else begin //rekurens
    temp:=pangkat(n div 2,a);
    if (n mod 2=1) then
      pangkat:=kali(kali(temp,temp),a)
    else
      pangkat:=kali(temp,temp);
  end;
end;

begin
  //-----membaca data-----
  readln(n); //yg dihitung : fibonacci(n)
  //---inisialisasi f(0) dan f(1)---
  f[0]:=0;   f[1]:=1;
  //---inisialisasi matriks fibonacci---
  a[1,1]:=1;  a[1,2]:=1;
  a[2,1]:=1;  a[2,2]:=0;
  //---menentukan output---
  if (n<2) then
    fib:=f[n]
  else begin
    //---mencari nilai a^(n-1)---
    hasil:=pangkat(n-1,a);
    //---menghitung output---
    fib:=hasil[1,1]*f[1]+hasil[1,2]*f[0];
  end;
  writeln(fib);
end.
```

5.2.3. Analisis

Dari program di atas, bisa dilihat bahwa ada proses inisalisasi, perkalian bilangan, perkalian matriks, pemangkatan, dan penjumlahan. Proses insialisasi ada sebanyak 6 proses. Pada setiap perkalian matriks ada 8 proses perkalian dan 4 proses penjumlahan. Pada setiap proses pemangkatan, ada maksimal 2 perkalian matriks dan 1 proses *assignment*. Sedangkan pada bagian akhir dibutuhkan 2 proses perkalian dan 1 proses penjumlahan.

Untuk menghitung proses pemangkatan:

$g(1) = 4$

Pada kasus terburuk :

$g(n) = g(n/2) + 2\times(8+4)$

$g(n) = g(n/2) + 24$

sehingga :

$g(n) = 24\times\lfloor {}^2\log n\rfloor + 4$

g(1) bernilai 4 karena untuk *assignment* nilai ke matriks dibutuhkan 4 proses, yaitu *assignment* ke masing-masing elemen matriks. Setelah g(n) diperoleh:

$T(n) = 6 + g(n) + 2 + 1$

$T(n) = 9 + 24\times\lfloor {}^2\log n\rfloor + 4$

$T(n) = 24\times\lfloor {}^2\log n\rfloor + 13$

Setelah memperoleh T(n), kita bisa mencari notasi O besarnya:

$T(n) \le 24\times4\times \log n + 13\times4\times \log n$

$T(n) \le 148\times \log n$

Sehingga T(n) = O(log n).

Kasus terbaik algoritma ini adalah jika $n=2^k+1$ untuk setiap k bilangan bulat positif. Alasannya, saat n bernilai 2^k+1, pada setiap tahap rekursi proses perkalian matriks yang dilakukan hanya 1 kali saja. Dengan kata lain, kasus terbaiknya terjadi saat banyaknya bit bernilai 1 dari representasi biner n-1 hanya 1.

Sementara itu, kasus terburuk untuk algoritma ini adalah saat $n = 2^k$ untuk setiap k bilangan bulat positif. Alasannya, saat n bernilai 2^k, pada setiap tahap rekursi akan dilakukan 2 kali perkalian matriks. Dengan kata lain, kasus terburuknya terjadi saat semua bit dari representasi biner n-1 bernilai 1.

6. RUMUS UNTUK MENGHITUNG FIBONACCI

6.1. Penurunan Rumus[3]

Deret Fibonacci memiliki persamaan f(n) = f(n-1) + f(n-2) untuk f(0) = 0 dan f(1) = 1. Sedangkan persamaan karakteristiknya adalah $r^2 - r - 1 = 0$. Persamaan karakteristik tersebut memiliki akar r1 = $\frac{(1+\sqrt{5})}{2}$ dan r2 = $\frac{(1-\sqrt{5})}{2}$. Menurut teorema 1 [3] di halaman 414, deret Fibonacci bisa dituliskan sebagai:

$$f(n) = \alpha_1\left(\frac{1+\sqrt{5}}{2}\right)^n + \alpha_2\left(\frac{1-\sqrt{5}}{2}\right)^n \quad (1)$$

dengan α_1 dan α_2 adalah konstanta. Untuk mencari nilai α_1 dan α_2, kita gunakan f(0) = 0 dan f(1) = 1.

$f(0) = \alpha_1 + \alpha_2 = 0$ (2)

$f(1) = \alpha_1\frac{(1+\sqrt{5})}{2} + \alpha_2\frac{(1-\sqrt{5})}{2} = 1$ (3)

Solusi dari persamaan (2) dan (3) di atas adalah:

$\alpha_1 = \frac{1}{\sqrt{5}}$

$\alpha_2 = \frac{-1}{\sqrt{5}}$

Setelah memperoleh nilai α_1 dan α_2, nilai tersebut bisa dimasukkan ke persamaan (1) sehingga diperoleh:

$$f(n) = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n$$

Rumus di atas menunjukkan bahwa deret Fibonacci berkembang secara eksponensial. Artinya, nilai $O(3^n)$ yang diperoleh pada Analisis bab 3.2 sudah benar.

6.2. Penggunaan

Contoh penggunaannya pada program dalam bahasa PASCAL:

```
const x= sqrt(5);
var n: integer;
    fib,a,b,c,d: double;

begin
  readln(n);
  if (n<2) then
    writeln(n)
  else begin
    c:=exp(ln((1+x)/2)*n);
    //menghitung ((1+x)/2)^n
    a:=1/x*c; //alpha1 kali c
    d:=exp(ln(abs(1-x)/2)*n);
    //menghitung abs(((1-x)/2)^n)
    if (n mod 2=1) then
      d:=d*-1; //kalau n ganjil, berarti d
seharusnya bernilai negatif
    b:=1/x*d; //-alpha2 kali d

    fib:=a-b;
    writeln(fib:0:0); //untuk pembulatan
  end;
end.
```

6.3. Analisis

Dari contoh program di atas, kita bisa memperkirakan kompleksitas untuk algoritma ini. Proses-proses yang dilakukan pada program adalah proses penjumlahan, pengurangan, perbandingan, *if*, absolut, modulo, perkalian, pembagian, logaritma, dan eksponen. Jika setiap proses tersebut dianggap sebagai 1 proses saja (tidak mengandung proses-proses lain), maka:

$T(n) = 5 + 2 + 6 + 3 + 1 + 2 + 1 = 20$

Dari T(n) yang diperoleh, bisa dicari notasi O besarnya:

$T(n) \leq 21$

Sehingga T(n) = O(1). Karena algoritma ini adalah algoritma yang stabil, kompleksitas untuk kasus terbaik sama dengan kompleksitas untuk kasus terburuknya.

Namun, jika proses eksponen dan logaritma tidak dianggap 1 proses sendiri, bisa jadi T(n) yang diperoleh menghasilkan O(log n). Penulis kurang tahu mengenai kompleksitas dari fungsi exp dan ln sehingga mengasumsikan bahwa kedua proses tersebut O(1).

7. KESIMPULAN

Dari hasil analisis dari tiap algoritma dapat disimpulkan:

1) Algoritma rekursif adalah algoritma yang paling tidak mangkus. Fungsi T(n) dari algoritma ini berkembang secara eksponensial.
2) Algoritma iteratif menggunakan matriks versi 1 berada di urutan ke-2 algoritma yang paling tidak mangkus. Fungsi T(n) algoritma ini berkembang secara linear namun konstanta pengali n-nya 12.
3) Algoritma iteratif memiliki kompleksitas yang cukup baik. Fungsi T(n) algoritma ini berkembang secara linear dengan konstanta pengali n-nya 3.
4) Algoritma iteratif menggunakan matriks versi 2 memiliki kompleksitas O(log n). Algoritma ini sangat mangkus untuk menghitung bilangan Fibonacci.
5) Penggunaan rumus eksplisit adalah cara paling mangkus untuk menghitung bilangan Fibonacci.

REFERENSI

[1] http://en.wikipedia.org/wiki/Fibonacci_number; Tanggal akses 19 Desember 2009.

[2] Levitin, Anany, "Introduction to The Design & Analysis of Algorithms", Addison-Wesley's, 2003.

[3] Rosen, Kenneth H. , "Discrete Mathematics and Its Applications Fifth Edition", McGrawHill, 2003.