PENERAPAN PROGRAM DINAMIS UNTUK MENGHITUNG ANGKA FIBONACCI DAN KOEFISIEN BINOMIAL

Reisha Humaira – NIM 13505047

Program Studi Teknik Informatika Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if15047@students.if.itb.ac.id

ABSTRAK

Makalah ini membahas tentang implementasi program dinamis (*dynamic programming*). Pembahasan pada makalah ini ditekankan pada penghitungan angka Fibonacci dan koefisien binomial dengan program dinamis.

Angka Fibonacci dan koefisien binomial merupakan bilangan yang mengandung unsur perulangan (rekursif). Secara naif, kita bisa menghitung angka Fibonacci dan koefisien binomial dengan algoritma rekursif, sesuai dengan definisi masing-masing bilangan itu.

Namun, jika kita perhatikan lebih jauh, penggunaan algoritma rekursif untuk kedua persoalan ini justru sangat tidak efisien. Kalkulasi yang sama kerap dilakukan berulang kali. Hal ini akan memperlambat waktu komputasi untuk nilai-nilai yang besar.

Algoritma rekursif yang dipakai untuk menghitung angka Fibonacci dan koefisien binomial memiliki kompleksitas yang eksponensial. Dengan program dinamis, kita bisa mereduksi hal ini sehingga akhirnya diperoleh kompleksitas algoritma penyelesaian yang linear. Kalkulasi berulang seperti disebutkan tadi bisa dihilangkan.

Kata kunci: *dynamic programming, Fibonacci number, binomial coefficient*.

1. PENDAHULUAN

Program dinamis (*dynamic programming*) yang ditemukan oleh Richard Bellman pada tahun 1953 merupakan suatu metode penyelesaian masalah di mana solusi persoalan dapat dipandang sebagai serangkaian keputusan yang saling berkaitan. Program dinamis merupakan salah satu metode yang mangkus yang biasanya digunakan untuk menyederhanakan persoalan-persoalan rekursif.

Seperti halnya algoritma *greedy*, program dinamis juga merupakan suatu ancangan untuk menyelesaikan masalah optimasi. Hanya saja, pada metode *greedy* hanya satu rangkaian keputusan yang pernah dihasilkan, sedangkan dengan program dinamis lebih dari satu rangkaian keputusan.

Program dinamis fokus pada bagian permasalahan yang tumpang-tindih (*overlapping subproblems*). Rangkaian keputusan dibuat dengan prinsip optimalitas (*optimal substructure*), di mana solusi optimal dari bagian solusi permasalahan bisa digunakan untuk menemukan solusi optimal untuk masalah secara keseluruhan.

Penerapan program dinamis ini sangat luas. Di antaranya, yang sederhana, adalah untuk menghitung angka Fibonacci dan koefisien binomial. Juga terdapat sejumlah algoritma lain yang didasarkan pada program dinamis, seperti algoritma Cocke-Younger-Kasami (CYK) untuk menentukan apakah suatu *string* dapat diterima suatu *context-free grammar*, algoritma Viterbi untuk model *hidden Markov*, algoritma Earley untuk *chart parser*, algoritma Needleman-Wunsch yang dipakai dalam bioinformatik, algoritma Floyd's All-Pairs untuk mencari lintasan terpendek, algoritma Selinger untuk optimasi *query* suatu basis data, algoritma De Boor untuk evaluasi kurva *B-spline*, dan lain-lain.

2. PROGRAM DINAMIS UNTUK MENGHITUNG ANGKA FIBONACCI

Salah satu bentuk aplikasi nyata dari program dinamis adalah pada penghitungan angka Fibonacci. Program dinamis adalah suatu alternatif untuk menghitung angka Fibonacci dengan lebih cepat, menggantikan algoritma rekursif yang sangat tidak efisien.

2.1 Angka Fibonacci

Dalam matematika, angka Fibonacci didefinisikan sebagai hubungan perulangan sebagai berikut.

$$F(n) = \begin{cases} 0 & , n = 0 \\ 1 & , n = 1 \\ F(n-1) + F(n-2) & , n > 1 \end{cases}$$

Jadi, suatu angka adalah penjumlahan dari dua angka sebelumnya.

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, 17711, 28657, 46368, 75025, 121393, 196418, 317811…

Angka Fibonacci memiliki satu sifat menarik. Jika kita membagi satu angka dalam deret tersebut dengan angka sebelumnya, akan didapat sebuah angka hasil pembagian yang besarnya sangat mendekati satu sama lain. Nyatanya, angka ini bernilai tetap setelah angka ke-13 dalam deret tersebut. Angka ini dikenal sebagai *golden ratio* φ (phi) di mana

$$\varphi = \frac{1+\sqrt{5}}{2} \approx 1.618\,033\,988\,749\,894\,848\,204$$

Angka Fibonacci juga memiliki sifat identitas:

1. $F(n+1) = F(n) + F(n-1)$
2. $F(0) + F(1) + F(2) + \ldots + F(n) = F(n+2) - 1$
3. $F(1) + 2\,F(2) + 3\,F(3) + \ldots + n\,F(n) = n\,F(n+2) - F(n+3) + 2$
4. $F(0)^2 + F(1)^2 + F(2)^2 + \ldots + F(n)^2 = F(n)\,F(n+1)$

2.2 Penerapan Program Dinamis

Dari definisi angka Fibonacci sendiri, kita bisa membuat suatu algoritma rekursif untuk menghitung angka Fibonacci seperti berikut.

```
function Fibonacci(input n:integer)→integer
{ Menghitung angka Fibonacci yang ke-n secara
  rekursif. Untuk angka yang lebih besar,
  tipe data bisa diganti dengan LongInteger
  atau yang lain.
  Masukan: n
  Keluaran: angka Fibonacci ke-n
}
Deklarasi

Algoritma
    if n=0 then
        return 0
    else if n=1 then
        return 1
    else
        return Fibonacci(n-1)+Fibonacci(n-2)
    endif
```

Jika $T(n)$ adalah waktu yang untuk menghitung $F(n)$ di mana $F(n)$ adalah angka Fibonacci yang ke-n, maka dengan memakai fungsi di atas jelaslah bahwa

$$T(n) = T(n-1) + T(n-2)$$

dan

$$T(1) = T(2) = c$$

di mana c adalah suatu konstanta. Karena itu,

$$T(n) = c\,F(n)$$

Johannes Kepler memperoleh bahwa

$$\lim_{n\to\infty} \frac{F(n+1)}{F(n)} = \varphi,$$

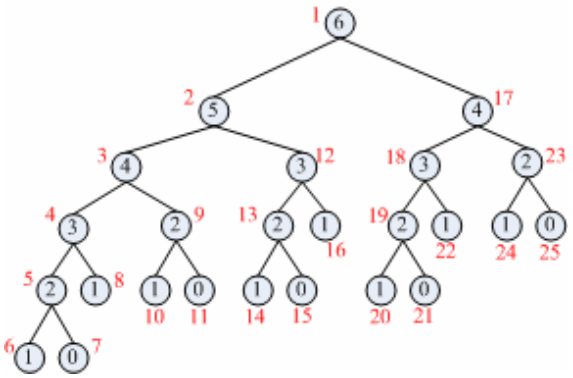
sehingga

$$T(n) = O(F(n)) = O(1{,}618\ldots^n)$$

Fungsi yang sederhana ini ternyata mempunyai kompleksitas waktu yang eksponensial. Tentunya algoritma seperti di atas dapat dipakai, hanya saja sangat tidak mangkus untuk nilai n yang besar.

Bisa kita lihat, bahwa Fibonacci(n+1) harus melakukan seluruh perhitungan dari Fibonacci(n) dan Fibonacci(n–1). Dan di dalamnya ada komputasi yang sama yang dilakukan secara berulang-ulang.

Sebagai gambaran, pemanggilan fungsi Fibbonacci untuk menghitung angka Fibonacci keenam bisa digambarkan dengan pohon seperti berikut.



Gambar 1 Pohon pemanggilan fungsi Fibonacci
Simpul pohon menyatakan *n*, sedangkan angka yang berwarna merah menyatakan urutan pemanggilan fungsi

Dari gambar di atas terlihat bahwa semua daun dari pohon memiliki nilai 0 atau 1, dan penjumlahan dari nilai pada semua daun ini adalah hasil akhir (nilai yang dikembalikan fungsi). Jadi, untuk setiap n, jumlah daun dari pohon pemanggilan fungsi Fibonacci itu adalah nilai angka Fibonacci itu sendiri. Tentu saja hal ini membutuhkan waktu komputasi yang luar biasa besar. Sebagai contoh, untuk n = 50, akan dilakukan Fibonacci(50) = 12.586.269.025 pemanggilan fungsi rekursif Fibonacci.

Dengan program dinamis, kita bisa memperbaiki algoritma rekursif di atas, sehingga komputasi menjadi lebih efisien. Kita bisa menggunakan kedua ancangan

(*approach*) yang ada pada program dinamis: *top-down* dan *bottom-up*. Penyelesaian dengan kedua ancangan itu dijelaskan sebagai berikut.

1. Dengan *top-down approach*

   Di sini kita menggunakan sebuah *array*, misalkan *array* Fib yang berukuran *n*, di mana Fib[*i*] berisi angka Fibonacci yang ke-*(i+1)*. Algoritmanya seperti berikut.

```
function Fibonacci(input n:integer)→integer
{ Menghitung angka Fibonacci yang ke-n dengan
  program dinamis maju.
  Masukan: n
  Keluaran: angka Fibonacci ke-n
}
Deklarasi
    i : integer
    Fib : array [1..n] of integer

Algoritma
    if n=0 then
        return 0
    else if n=1 or n=2 then
        return 1
    else
        Fib[1] ← 1
        Fib[2] ← 1
        for i←3 to n do
            Fib[i] ← Fib[i-1] + Fib[i-2]
        endfor
        return Fib[i]
    endif
```

   Dengan cara ini, kita bisa memperoleh nilai Fib[3], Fib[4], dan seterusnya; kita senantiasa menyimpan nilai-nilai itu ke dalam *array*. Ketika *loop* berakhir, fungsi akan mengembalikan elemen terakhir *array* yang merupakan angka Fibonacci ke-*n* yang dihitung.

2. Dengan *bottom-up approach*

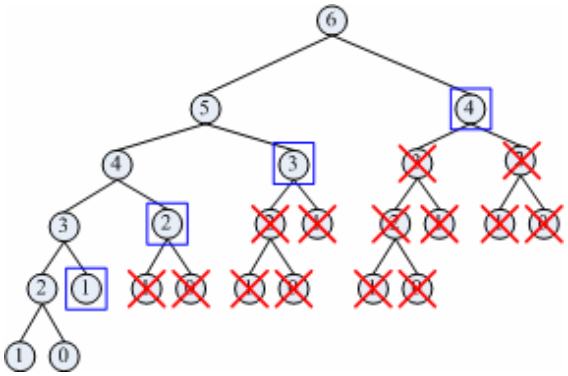
   Di sini kita menghitung nilai Fibonacci yang lebih kecil terlebih dahulu, lalu mengkalkulasikan nilai yang lebih besar dari nilai-nilai yang diperoleh sebelumnya. Ini perbaikan dari ancangan sebelumnya, di mana *array*-nya kita hilangkan. Algoritmanya seperti berikut.

```
function Fibonacci(input n:integer)→integer
{ Menghitung angka Fibonacci yang ke-n dengan
  program dinamis mundur.
  Masukan: n
  Keluaran: angka Fibonacci ke-n
}
Deklarasi
    i : integer
    prevF, currF, newF : integer

Algoritma
    prevF ← 0
    currF ← 1
    for i←2 to n do
        newF ← prevF + currF
        prevF ← currF
        currF ← newF
    endfor
    return currF
```

Kedua pendekatan tersebut memiliki kompleksitas yang sama, yaitu *O(n)*. Kompleksitas yang linear dan tentunya ini jauh lebih baik dibandingkan dengan algoritma rekursif. Bedanya hanya terletak pada memori yang digunakan, di mana dengan pendekatan *top-down*, kita butuh memori yang lebih banyak untuk *array*.

Dengan program dinamis, akhirnya kita bisa menghilangkan kalkulasi yang ada pada subpohon pemanggilan fungsi. Hal ini digambarkan seperti berikut.



Gambar 2 Pohon pemanggilan fungsi Fibonacci
Subpohon 'dipangkas'

Simpul-simpul yang ditandai dengan kotak biru adalah nilai-nilai yang sudah dihitung sebelumnya, sehingga pemanggilan fungsi pada bagian tersebut bisa di-*skip*.

3. PROGRAM DINAMIS UNTUK MENGHITUNG KOEFISIEN BINOMIAL

3.1 Koefisien Binomial

Dalam matematika, terutama kombinatorial, koefisien binomial dari suatu bilangan asli *n* dan bilangan integer *k* adalah jumlah kombinasi yang ada. Dengan kata lain, jika diberikan *n* buah benda, jumlah cara memilih *k* buah bola yang berbeda sama dengan koefisien binomial.

Koefisien binomial didefinisikan sebagai berikut.

$$\binom{n}{k} = \frac{n \cdot (n-1) \cdots (n-k+1)}{k \cdot (k-1) \cdots 1} = \frac{n!}{k!(n-k)!}$$

untuk $n \geq k \geq 0$ dan

$$\binom{n}{k} = 0$$

untuk $k < 0$ atau $k > n$.

Untuk nilai n dan k non-negatif, diperoleh

$$\begin{Bmatrix} n \\ k \end{Bmatrix} = \begin{cases} \begin{Bmatrix} n-1 \\ k-1 \end{Bmatrix} + \begin{Bmatrix} n-1 \\ k \end{Bmatrix} & , 0 < k < n \\ 1 & , k=0 \text{ atau } k=n \end{cases}$$

Koefisien binomial dituliskan dengan notasi $\binom{n}{k}$. Notasi ini diperkenalkan oleh Albert von Ettinghausen pada tahun 1826, meskipun bilangan ini telah dikenal berabad-abad sebelumnya. Notasi lainnya yang biasa digunakan seperti $C(n, k)$, ${}_nC_k$ atau C_n^k (C untuk *combination*).

3.2 Penerapan Program Dinamis

Dari definisi koefisien binomial, kita bisa membuat suatu algoritma seperti berikut.

```
function Binomial(input n,k:integer)→integer
{ Menghitung koefisien Binomial dari (n,k)
  secara rekursif. Untuk angka yang lebih
  besar, tipe data bisa diganti dengan
  LongInteger atau yang lain.
  Masukan: n,k
  Keluaran: koefisien binomial (n,k)
}
Deklarasi

Algoritma
    if n<k or k<0 then
        return 0
    else if k=0 or k=n then
        return 1
    else
        return Binomial(n-1,k-1) +
               Binomial(n-1,k)
    endif
```

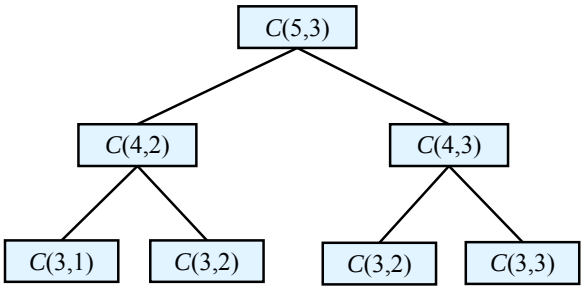
Kompleksitas algoritma di atas adalah

$$T(n,k) = T(n-1, k-1) + T(n-k, k)$$

atau

$$T(n,k) = O(n^k)$$

Dengan algoritma di atas, banyak komputasi yang sama yang dilakukan berulang-ulang. Sebagai contoh, perhitungan $C(5,3)$ digambarkan dengan pohon seperti berikut.



Gambar 3 Pohon pemanggilan fungsi Binomial

Seperti halnya pada penghitungan angka Fibonacci, kita bisa memperbaiki algoritma di atas dengan program dinamis. Prinsip yang dipakai yaitu:

a. Buat matriks berukuran *(n+1)* × *(k+1)* untuk menyimpan hasil perhitungan.

b. Inisialisasi matriks dengan terkecil dari instansiasi persoalan

c. Set elemen matriks (tentunya dengan urutan yang tepat) menggunakan hasil yang telah dihitung sebelumnya. Setiap elemen itu tepat dihitung satu kali saja.

d. Nilai akhir hasil perhitungan adalah solusi dari persoalan.

e. Di sini kita mengimplementasikan iteratif, bukan rekursif.

Jelasnya sebagai berikut.

1. Dengan *top-down approach*

   Algoritmanya seperti berikut.

```
function Binomial(input n,k:integer)→integer
{ Menghitung koefisien Binomial dari (n,k)
  dengan program dinamis maju.
  Masukan: n,k
  Keluaran: koefisien binomial (n,k)
}
Deklarasi
    i,j : integer
    Bin : array [1..n+1][1..k+1] of integer

Algoritma
    {inisialisasi matriksa}
    for i←1 to n+1 do
        for j←1 to k+1 do
            Bin[i][j] ← 0
        endfor
    endfor
```

```
    {menghitung koefisien binomial}
    if n=k or k=0 then
        return 1
    else if Bin[n+1][k+1]>0 then
        return Bin[n+1][k+1]
    else
        Bin[n+1][k+1] ← Binomial[n][k] +
                        Binomial[n][k-1]
        return Bin[n+1][k+1]
```

2. Dengan *bottom-up approach*

   Algoritmanya seperti berikut.

```
function Binomial(input n,k:integer)→integer
{ Menghitung koefisien Binomial dari (n,k)
  dengan program dinamis mundur.
  Masukan: n,k
  Keluaran: koefisien binomial (n,k)
}
Deklarasi
    i,j : integer
    Bin : array [1..n+1][1..k+1] of integer

Algoritma
    {inisialisasi matriksa}
    for i←1 to n+1 do
        for j←1 to k+1 do
            Bin [i][j] ← 0
        endfor
    endfor

    {menghitung koefisien binomial}
    for i←1 to n+1 do
        for j←1 to (Min(i,k)+1) do
            if j=0 or j=k then
                Bin[i][j] ← 1
            else
                Bin[i][j] ← Bin[i-1][j-1] +
                            Bin[i-1][j]
            endif
        endfor
    endfor

    return Bin[n+1][k+1]
```

Kedua pendekatan tersebut memiliki kompleksitas yang sama yaitu $O(nk)$, atau pada kasus terburuk ($k=n/2$) kompleksitasnya $O(n^2)$. Kompleksitas ini tentu lebih baik dibandingkan dengan algoritma rekursif.

4. KESIMPULAN

Kesimpulan yang dapat dimbil dari pembahasan di atas adalah:

1. Algoritma rekursif ternyata sangat tidak mangkus dipakai untuk menghitung angka Fibonacci dan koefisien binomial, karena ada bagian dari persoalan yang rangkap. Maksudnya sejumlah kalkulasi yang sama dilakukan berulang-ulang. Ini tentunya berbeda dengan penggunaan rekursif untuk menghitung faktorial, karena pada faktorial kalkulasi tepat dilakukan sekali untuk setiap bagian persoalan.
2. Program dinamis mampu menghindari terjadinya kalkulasi berulang itu dengan membangun solusi dari bagian persoalan tepat sekali saja.
3. Penggunaan program dinamis untuk menghitung angka Fibonacci dan koefisien binomial ternyata jauh lebih mangkus.

REFERENSI

[1] Algorithmist. (2007). Dynamic Programming. http://www.algorithmist.com/index.php/Dynamic_Programming. Tanggal akses: 22 Mei 2007 pukul 10:57.

[2] Chen, Jeff; Morgan, Tom. (2006). The Citizen's Guide to Dynamic Programming. http://activities.tjhsst.edu/sct/lectures/dpfinal.pdf. Tanggal akses: 22 Mei 2007 pukul 12:16.

[3] Morris, John. (2007). Data Structures and Algorithms: Dynamic Algorithms. http://www.cs.auckland.ac.nz/software/AlgAnim/dynamic.html. Tanggal akses: 22 Mei 2007 pukul 14:28.

[4] Munir, Rinaldi. (2007). Diktat Kuliah IF2251 Strategi Algoritmik. Program Studi Teknik Informatika, Institut Teknologi Bandung.

[5] PED. (2007). Dynamic Programming. http://www.csc.liv.ac.uk/~ped/teachadmin/algor/dyprog.html. Tanggal akses: 22 Mei 2007 pukul 14:04.

[6] Skiena, Steven. (2007). Introduction to Dynamic Programming. http://www.cs.sunysb.edu/~skiena/373/newlectures/lecture16.pdf. Tanggal akses: 22 Mei 2007 pukul 14:06.

[7] Steenbergen, Martijn van. (2005). Dynamic Programming. http://martijn.van.steenbergen.nl/projects/DynProg.pdf. Tanggal akses: 22 Mei 2007 pukul 12:11.

[8] Wikibooks. (2007). Algorithms/Dynamic Programming. http://en.wikibooks.org/wiki/Algorithms/Chapter_6. Tanggal akses: 22 Mei 2007 pukul 10:52.

[9] Wikipedia. (2007). Binomial Coefficient. http://en.wikipedia.org/wiki/Binomial_coefficient. Tanggal akses: 22 Mei 2007 pukul 14:08.

[10] Wikipedia. (2007). Dynamic Programming. http://en.wikipedia.org/wiki/Dynamic_programming. Tanggal akses: 22 Mei 2007 pukul 11:10.

[11] Wikipedia. (2007). Fibonacci Number. http://en.wikipedia.org/wiki/Fibonacci_number. Tanggal akses: 22 Mei 2007 pukul 10:46.