Perbandingan antara Metode Rekursif terhadap Dynamic Programming pada Penyelesaian Persoalan Bilangan Fibonacci

Geraldi Dzakwan 13514065
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
geraldzakwan@students.itb.ac.id

***Abstrak*— Masalah bilangan Fibonacci merupakan salah satu masalah klasik pada bidang matematika dan ilmu komputer. Masalah ini penting karena mempunyai penerapan yang luas pada berbagai bidang seperti matematika, biologi, ekonomi, dan juga ilmu komputer itu sendiri. Masalah bilangan Fibonacci dapat diselesaikan dengan berbagai algoritma, dua diantaranya adalah dengan prinsip rekursif dan dengan prinsip program dinamis. Pada bagian awal makalah, akan dibahas mengenai program dinamis dan sedikit mengulas barisan Fibonacci. Kemudian, dibahas implementasi penggunaan kedua algoritma untuk menyelesaikan persoalan bilangan Fibonacci. Lalu, dari hasil yang didapat, akan dibandingkan kinerja kedua algoritma. Aplikasi barisan Fibonacci juga akan dibahas pada bagian akhir makalah.**

***Kata kunci*—rekursif, program dinamis, barisan Fibonacci, bilangan Fibonacci**

I. PENDAHULUAN

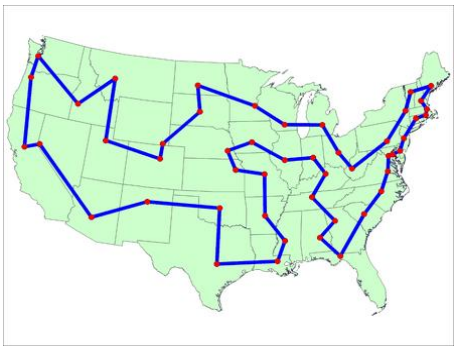
Program dinamis adalah cara untuk menyelesaikan persoalan dengan cara membagi persoalan menjadi beberapa subpersoalan yang lebih sederhana. Subpersoalan harus mengarah kepada persoalan sesungguhnya. Sehingga, solusi subpersoalan yang didapat nantinya akan digunakan sebagai bantuan untuk memeroleh solusi sesungguhnya.

Secara umum, penyelesaian masalah dengan menggunakan program dinamis memiliki kompleksitas yang lebih baik dibandingkan dengan penyelesaian masalah dengan menggunakan algoritma lain, seperti algoritma brute force, greedy, dan lain-lain.

Namun, tidak semua masalah dapat diselesaikan dengan menggunakan prinsip program dinamis. Masalah yang dapat diselesaikan dengan menggunakan program dinamis adalah masalah yang memiliki karakteristik tertentu yang akan dijelaskan kemudian pada bagian dasar teori program dinamis.

Program dinamis dapat digunakan untuk memecahkan berbagai macam masalah yang cukup kompleks, seperti mencari lintasan terpendek *(shortest path*) dari sebuah graf, persoalan Travelling Salesman Person (TSP), persoalan penganggaran modal (Capital Budgeting), persoalan knapsack, dan lain-lain.

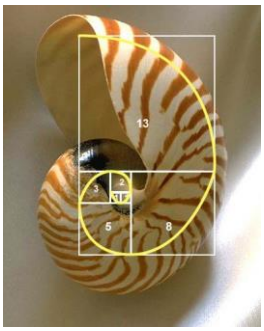
Namun, saya memilih satu masalah yang penyelesaiannya dengan algoritma program dinamis belum pernah dibahas pada saat kuliah Strategi Algoritma, yaitu masalah bilangan Fibonacci.



Gambar 1.1 Masalah TSP pada peta Amerika yang lazim dicari solusi optimalnya menggunakan program dinamis
Source : https://www.cs.rit.edu/~ark/351/dp/fig04.png

Masalah bilangan Fibonacci merupakan salah satu masalah yang cukup sederhana namun menarik dan penerapannya banyak digunakan dalam berbagai macam bidang, seperti matematika, biologi, ekonomi, dan termasuk di dalamnya ilmu komputer.

Masalah ini menarik karena memiliki keterkaitan yang tinggi dengan alam semesta. Beberapa hal natural yang terkait dengan barisan Fibonacci adalah pola pada cangkang keong, jumlah daun pada bunga, pola bunga, perbandingan panjang jari terhadap lekuk jari, dan lain-lain.



Gambar 1.2 Pola pada cangkang keong yang menampakkan rasio Fibonacci atau yang biasa dikenal sebagai *golden ratio*
Source : http://alewoh.com/image/asal-usul-barisan-fibonacci-2.jpg

II. Dasar Teori Program Dinamis

Program Dinamis (*dynamic programming*) merupakan salah satu strategi atau metode untuk menyelesaikan suatu permasalahan yang penggunaannya sudah sangat luas.

Strategi dalam memecahkan masalah dengan menggunakan program dinamis terdiri dari dua poin utama berikut :

- Metode pemecahan masalah dengan cara menguraikan solusi menjadi sekumpulan tahapan (*stage*).
- Tahapan tersebut disusun sedemikian sehingga solusi dari persoalan dapat dipandang dari serangkaian keputusan yang saling berkaitan.

Istilah "program dinamis" muncul karena perhitungan solusi menggunakan tabel yang ukurannya dapat berubah. Kata "program" pada program dinamis tidak berhubungan dengan *source code*, melainkan makna kata "program" dalam program dinamis adalah perencanaan.

Karakteristik dari penyelesaian persoalan dengan menggunakan program dinamis adalah sebagai berikut :

1. Terdapat sejumlah berhingga pilihan yang mungkin.
2. Solusi dibangun secara bertahap.
3. Solusi pada setiap tahap dibangun dari hasil solusi tahap sebelumnya.
4. Menggunakan persyaratan optimasi dan kendala untuk membatasi sejumlah pilihan yang harus dipertimbangkan pada suatu tahap.

Penyelesaian suatu persoalan dengan program dinamis memiliki kemiripan dengan algoritma greedy. Perbedaannya adalah sebagai berikut :

→ Greedy : hanya satu rangkaian keputusan yang dihasilkan.

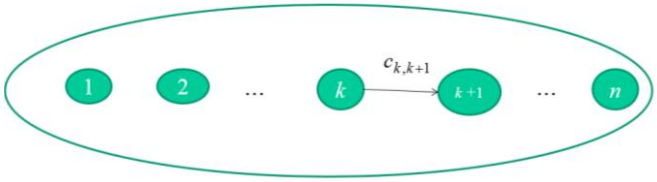
→ Program Dinamis : lebih dari satu rangkaian keputusan yang dipertimbangkan.

Pada program dinamis, rangkaian keputusan yang optimal dibuat dengan menggunakan prinsip optimalitas. Prinsip optimalitas yaitu jika solusi total merupakan solusi optimal, maka bagian solusi sampai tahap ke-k juga optimal.

Prinsip optimalitas berarti bahwa jika kita bekerja dari tahap k ke tahap $k + 1$, kita dapat menggunakan hasil optimal dari tahap k tanpa harus kembali ke tahap awal.

Ongkos pada tahap k +1 = (ongkos yang dihasilkan pada tahap k) + (ongkos dari tahap k ke tahap $k + 1$ atau $c_{k, k+1}$).

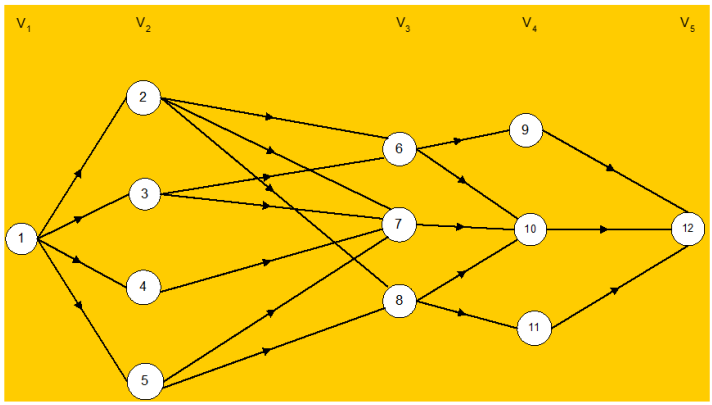
Dengan prinsip optimalitas, dapat dijamin bahwa pengambilan keputusan pada suatu tahap adalah keputusan yang benar dan optimal untuk tahap-tahap selanjutnya. Oleh karena itu, kita tidak perlu kembali ke tahap awal karena hasil optimal dari tahap sebelumnya dapat digunakan kembali tanpa perlu komputasi ulang.



Gambar 2.1 : Ilustrasi prinsip optimalitas
Sumber :
http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2014-2015/Pencocokan%20String%20(2015).ppt

Karakteristik persoalan program dinamis adalah sebagai berikut :

1. Persoalan dapat dibagi menjadi beberapa tahap (*stage*), yang dalam hal ini setiap tahap hanya diambil satu keputusan.
2. Masing- masing tahap terdiri dari sejumlah status (*state*) yang berhubungan dengan tahap tersebut. Secara umum, status yang dimaksud adalah berbagai macam kemungkinan masukan yang ada pada tahap tersebut. Masing-masing tahap ini dapat digambarkan dengan menggunakan graf multitahap (*multistage graph*). Tiap simpul di dalam graf tersebut menyatakan status, sedangkan V_1, V_2, … menyatakan tahap.



Gambar 2.2 : Graf multitahap (*multistage graph*)
Sumber :
http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2014-2015/Pencocokan%20String%20(2015).ppt

3. Hasil dari keputusan yang diambil pada setiap tahap ditransformasikan dari status yang bersangkutan ke status berikutnya pada tahap berikutnya.
4. Ongkos (*cost*) pada suatu tahap meningkat secara teratur (*steadily*) dengan bertambahnya jumlah tahapan.
5. Ongkos pada suatu tahap bergantung pada ongkos pada tahap yang sudah berjalan sebelumnya dan ongkos pada tahap tersebut.
6. Keputusan terbaik pada suatu tahap bersifat independen terhadap keputusan yang dilakukan pada tahap sebelumnya.
7. Adanya hubungan rekursif yang mengidentifikasikan keputusan terbaik untuk setiap status pada tahap k sehingga dapat diambil keputusan terbaik untuk setiap status pada tahap k + 1.
8. Prinsip optimalitas berlaku pada persoalan tersebut.

Terdapat dua pendekatan yang digunakan dalam PD (program dinamis), yaitu :

1. PD maju (*forward* atau *top-down*)
2. PD mundur (*backward* atau *bottom-up*).

Misalkan x_1, x_2, …, x_n menyatakan peubah (*variable*) keputusan yang harus dibuat masing-masing untuk tahap 1, 2, …, *n*. Maka, kedua pendekatan adalah sebagai berikut :

1. Program dinamis maju. Program dinamis bergerak mulai dari tahap 1, terus maju ke tahap 2, 3, dan seterusnya sampai tahap *n*. Runtunan peubah keputusan adalah x_1, x_2, …, x_n.
2. Program dinamis mundur. Program dinamis bergerak mulai dari tahap *n*, terus mundur ke tahap $n-1$, $n-2$, dan seterusnya sampai tahap 1. Runtunan peubah keputusan adalah x_n, x_{n-1}, …, x_1.

Prinsip optimalitas pada PD maju :
Ongkos pada tahap $k+1$ = (ongkos yang dihasilkan pada tahap k) + (ongkos dari tahap k ke tahap $k+1$).
$k = 1, 2, \ldots, n-1$.

Prinsip optimalitas pada PD mundur:
Ongkos pada tahap k = (ongkos yang dihasilkan pada tahap $k+1$) + (ongkos dari tahap $k+1$ ke tahap k).
$k = n, n-1, \ldots, 1$.

Secara singkat, langkah-langkah pengembangan algoritma program dinamis adalah sebagai berikut :

1. Karakteristikkan struktur solusi optimal.
2. Definisikan secara rekursif nilai solusi optimal.
3. Hitung nilai solusi optimal secara maju atau mundur.
4. Konstruksi solusi optimal.

III. Dasar Teori Barisan Fibonacci

Barisan bilangan adalah urutan bilangan yang memiliki aturan atau pola tertentu. Elemen-elemen dari suatu barisan bilangan disebut dengan suku. Ada beberapa barisan bilangan, seperti barisan aritmetika, barisan geometri, barisan persegi, dan barisan Fibonacci.

Barisan bilangan Fibonacci ditemukan oleh Leonardo Pisano, dikenal juga dengan Fibonacci. Ia adalah seorang matematikawan terbesar pada abad pertengahan yang lahir di Pisa, Italia pada tahun 1170. Meskipun Leonardo lahir di Pisa, tetapi ia lebih banyak menyerap ilmu pengetahuan dari orang-orang Timur, karena ia ikut ayahnya yang bekerja di Aljazair.

Gambar 3.1 : Leonardo Pisano alias Fibonacci
Sumber : http://f.tqn.com/y/math/1/S/L/F/fibonacci.jpg

Barisan ini diperoleh Fibonacci dari pengamatannya terhadap peternakan kelinci. Pada abad ke-13, Leonardo Pisano menuliskan suatu masalah di bukunya Liber Abaci (Book of the Abacus or Book of Calculating). Inilah masalah yang terdapat pada buku tersebut : "Berapa banyak pasangan kelinci yang beranak-pinak selama satu tahun jika diawali dari sepasang kelinci (jantan dan betina) dan kelinci tersebut tumbuh jadi dewasa dan bisa kawin setelah mereka berumur satu bulan sehingga setiap bulan kedua, masing-masing kelinci betina selalu melahirkan sepasang kelinci baru?".

Dari hasil pengamatannya, Fibonacci memeroleh jawaban sebagai berikut : Total banyaknya pasangan kelinci pada setiap awal bulan berturut-turut adalah:

1, 1, 2, 3, 5, 8,13,21,34,55,89,144,233, 377,dst .....

Barisan inilah yang disebut dengan barisan Fibonacci. Suku pertama dan kedua bernilai satu, sedangkan suku ketiga dan seterusnya merupakan hasil penjumlahan dari dua suku sebelumnya.

Sebenarnya, Fibonacci sendiri tidak banyak menyelidiki lebih lanjut tentang barisan dari masalah yang dikemukakannya itu. Ia juga tidak memberi nama barisannya sebagai Barisan Fibonacci.

Nama itu Barisan Fibonacci baru muncul pada abad ke-19 dan diperkenalkan oleh Lucas, seorang matematikawan Perancis. Lucas mendefinisikan kembali barisan tersebut, dan menamakan barisan tersebut sebagai barisan Fibonacci di mana setiap sukunya diberikan simbol.

Barisan Fibonacci dapat didefinisikan sebagai berikut:

$$F(n) = \begin{cases} 0, & \text{jika } n = 0; \\ 1, & \text{jika } n = 1; \\ F(n-1) + F(n-2) & \text{jika tidak.} \end{cases}$$

Gambar 3.2 : Definisi barisan fibonacci
Sumber :
https://upload.wikimedia.org/math/8/9/8/8985ea0a02eb63c870b242151a900584.png

IV. Implementasi Penyelesaian Masalah Barisan Fibonacci dengan Metode Rekursif

Sebagai langkah pertama, kita perlu mendefinisikan basis dan rekurens untuk barisan Fibonacci. Hal ini serupa dengan definisi barisan Fibonacci yang sudah kita tinjau sebelumnya.

Misalkan barisan Fibonacci dinyatakan dengan fungsi F dan peubah n -> F(n). Maka, basis dan rekurens untuk relasi rekurens Fibonacci adalah sebagai berikut :

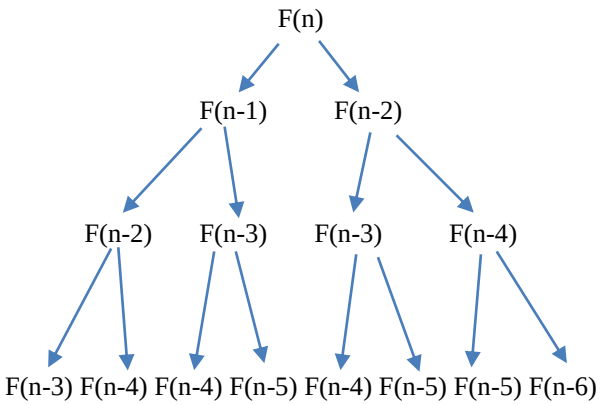
1. Basis : F(n) = 0 untuk n = 0.
2. Basis : F(n) = 1 untuk n = 1.
3. Rekurens : F(n) = F(n-1) + F(n-2) untuk n >1.

Implementasinya dalam bahasa Java adalah sebagai berikut :

```
public long RecursiveFib (long n) {
  if (n<=1) {
    return n;
  }
  else {
    return RecursiveFib(n-1)+RecursiveFib(n-2);
  }
}
```

V. Implementasi Penyelesaian Masalah Barisan Fibonacci dengan Program Dinamis

Perhatikan proses penyelesaian masalah barisan Fibonacci dengan metode rekursif. Metode rekursif tidak menggunakan kembali solusi dari subpersoalan yang sudah diselesaikan. Perhatikan pohon di bawah ini yang menggambarkan proses penentuan bilangan Fibonacci secara rekursif :



Gambar 5.1 : Pohon penyelesaian bilangan Fibonacci dengan metode rekursif
Sumber : Olahan penulis

Dapat kita lihat bahwa terjadi redundansi dalam proses komputasi bilangan Fibonacci dengan metode rekursif. Redundansi yang dimaksud adalah sebagai berikut :

1. Proses komputasi F(n-2) dilakukan sebanyak 2x
2. Proses komputasi F(n-3) dilakukan sebanyak 3x
3. Proses komputasi F(n-4), F(n-5), dan F(n-6) dilakukan sebanyak lebih dari 3x jika proses penyelesaian pada pohon dilanjutkan.

Terdapat banyak komputasi yang sebenarnya tidak perlu dilakukan yang mengakibatkan besarnya kompleksitas waktu dari metode rekursif. Hal inilah yang coba untuk diperbaiki dalam penyelesaian menggunakan program dinamis.

Program dinamis yang digunakan adalah program dinamis dengan pendekatan mundur. Kita mulai proses dari F(n) kemudian turun hingga F(0) pada aras (*level*) pohon yang paling bawah.

Pada pohon penyelesaian, kita bisa menganggap setiap aras sebagai tahapan-tahapan program dinamis. Setiap kali menyelesaikan sebuah tahapan, kita akan mencatat solusi dari tahapan tersebut untuk kemudian digunakan pada tahapan selanjutnya.

Pada setiap tahapan, kita sebenarnya menyelesaikan beberapa subpersoalan. Solusi dari subpersoalan akan dicatat sehingga saat kita membutuhkannya, kita tidak mengulang komputasi dari awal, melainkan cukup menggunakan solusi dari subpersoalan yang sudah didapat pada tahap sebelumnya.

Contohnya, pada tahap ke-2, kita akan mendapatkan solusi untuk subpersoalan F(n-2). Maka, pada tahap ke-3 saat kita membutuhkannya, kita cukup menggunakan solusi yang sudah dicatat. Oleh karena itu, pohon tidak akan menguraikan simpul F(n-2) secara rekursif, melainkan F(n-2) pada tahap ke-3 akan menjadi simpul daun dan proses berhenti sampai disitu.

Dari penjelasan di atas, kita dapat melihat bahwa sebenarnya program dinamis tetap memanfaatkan konsep rekursifitas. Hanya saja, solusi subpersoalan yang sudah diselesaikan akan dicatat. Hal ini lazim dikenal dengan istilah *memoization*. Kemudian, solusi subpersoalan ini akan digunakan kembali (*reuse*) pada tahapan-tahapan selanjutnya. Secara singkat, kita dapat menarik kesimpulan bahwa :

Dynamic Programming -> Recursion + Memoization

Implementasinya dalam program Java adalah sebagai berikut :

```
public long DPFib (long n) {
    long f[];
    f = new long [(int)n+1];

    f[0] = 0;
    f[1] = 1;

    for (int i = 2; i <= n; i++)
    {
        f[i] = f[i-1] + f[i-2];
    }

    return f[(int)n];
}
```

VI. Perbandingan Kompleksitas Waktu Antara Metode Rekursif dan Program Dinamis

Kompleksitas waktu metode rekursif adalah sebagai berikut :

$$T(0) = T(1) = 1$$
$$T(n) = T(n-1) + T(n-2)$$

Jika diselesaikan, maka akan diperoleh hasil sebagai berikut :

$$T(n) = a^n$$
$$a = (1 + \sqrt{5})/2$$

Kompleksitas waktunya adalah eksponensial yang termasuk sulit untuk dipecahkan. Dalam notasi Big-O dengan aproksimasi bahwa nilai a mendekati 2 :

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

Pada penyelesaian dengan program dinamis, pada saat *running time*, akan terjadi sebanyak tepat n kali *non-*

memoized calls, yaitu DPFib(1), DPFib(2), DPFib(3), … , DPFib(n). *Non-memoized calls* berarti pemanggilan fungsi rekursif yang tidak menggunakan solusi yang sudah diingat (harus melakukan komputasi).

Untuk setiap *non-memoized calls*, sebenarnya hanya perlu menjumlahkan solusi subpersoalan yang sudah diselesaikan sebelumnya. Misalkan untuk DPFib(5), maka cukup menjumlahkan f(4) + f(3) yang sudah didapat sebelumnya. Oleh karena itu, untuk setiap *non-memoized calls*, kompleksitas waktu untuk *non-recursive work* adalah konstan O(1). Maksud dari *non-recursive work* disini adalah kompleksitas waktu hanya dihitung dari penjumlahan dan mengabaikan waktu pemanggilan fungsi rekursif.

Sehingga, kompleksitas waktu dari program dinamis adalah :

T(n) = jumlah pemanggilan rekursif * kompleksitas waktu penjumlahan untuk setiap pemanggilan
= n * 1
= n

Kompleksitas waktunya adalah linear (polinomial) yang termasuk mudah untuk dipecahkan. Dalam notasi Big-O :

O(n) = n

Maka, dapat ditarik kesimpulan bahwa program dinamis jauh lebih cepat dalam menyelesaikan persoalan bilangan Fibonacci ketimbang metode rekursif berdasarkan kompleksitas waktunya.

Sedikit kekurangan algoritma program dinamis yaitu membutuhkan sedikit tambahan *space memory* untuk menampung larik f dengan kompleksitas ruang O(n). Namun, hal ini dapat ditolerir karena saat ini memori sudah bukan lagi hal yang mahal dibandingkan dengan tuntutan kecepatan penyelesaian / kinerja waktu.

VII. Perbandingan Waktu Eksekusi Antara Metode Rekursif dan Program Dinamis

Kita melihat bahwa berdasarkan kompleksitas waktunya, secara teoritis kinerja program dinamis dalam menyelesaikan masalah bilangan Fibonacci jauh lebih cepat. Akan tetapi, mari kita bandingkan dengan pendekatan langsung yaitu melalui waktu eksekusi dari masing-masing algoritma.

Masukan menyatakan urutan dari bilangan yang ingin dihitung dalam barisan Fibonacci. Pada fungsi DPFib, masukan disimbolkan dengan n. Sedangkan, keluaran menyatakan nilai bilangan Fibonacci dalam barisan pada urutan ke-n.

Misal, n = 2 akan menghasilkan keluaran DPFib(n) = 2, n = 3 akan menghasilkan keluaran DPFib(n) = 3, dan seterusnya. Masukan dan keluaran adalah sebuah variabel bertipe *long* dalam bahasa Java. Oleh karena itu, masukan bernilai antara 0 hingga 92. Jika masukan lebih besar dari 92, akan terjadi *overflow*.

WEMR menyatakan menyatakan waktu eksekusi program yang memanfaatkan metode rekursif untuk masukan yang bersangkutan, sedangkan WEPD menyatakan waktu eksekusi program yang memanfaatkan algoritma program dinamis.

Masukan	Keluaran	WEMR	WEPD
0	0	149 µs	164 µs
10	55	420 µs	197 µs
20	6765	4 ms	236 µs
30	832040	28 ms	278 µs
40	102334155	1 s	364 µs
50	12586269025	105 s	392 µs
60	1548008755920	> 15 menit	451 µs
70	190392490709135	> 2 jam	522 µs
80	23416728348467685	> 20 jam	583 µs
92	7540113804746346429	> 1 minggu	624 µs

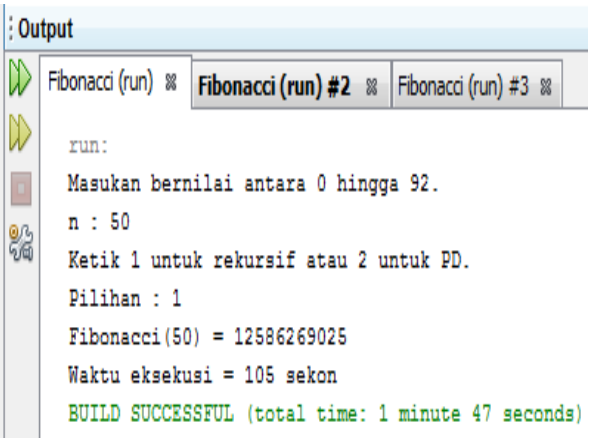
Gambar 7.1 : Tabel perbandingan waktu eksekusi metode rekursif dan program dinamis
Sumber : Olahan penulis

Dapat kita lihat dari tabel bahwa waktu eksekusi program yang memanfaatkan program dinamis berada pada rentang waktu mikro sekon (kurang dari 1 mili sekon). Sedangkan, waktu eksekusi program yang memanfaatkan metode rekursif selalu naik secara drastis seiring bertambahnya nilai masukan.

Hal ini sesuai dengan perbandingan kompleksitas waktu yang sudah dibahas pada bagian sebelumnya. Waktu eksekusi program dinamis naik secara linear (perlahan), sedangkan waktu eksekusi metode rekursif naik secara eksponensial.

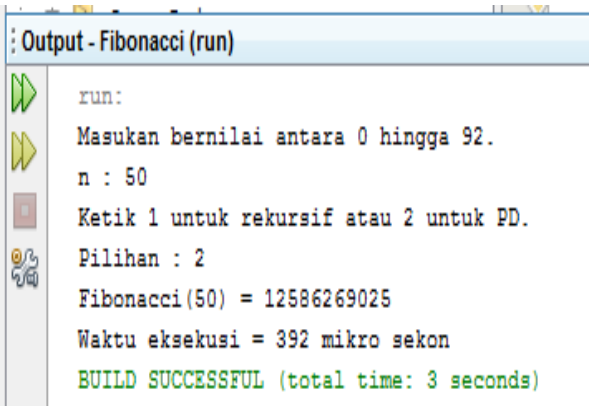
Untuk metode rekursif, waktu eksekusi untuk masukan n = 60 hingga n = 92 tidak saya ketahui secara pasti. Hal ini karena waktu eksekusi sangat lama sehingga saya terpaksa menghentikan program sebelum program tersebut selesai.

Waktu eksekusi untuk n = 60 hingga n = 92 saya perkirakan sesuai dengan kompleksitas waktu metode rekursif, yaitu memiliki perbandingan eksponensial terhadap besar masukan (dengan basis adalah *golden ratio*, yaitu 1,618).



Gambar 7.2 : *Screenshot* hasil eksekusi metode rekursif dengan n = 50

Sumber : Olahan penulis



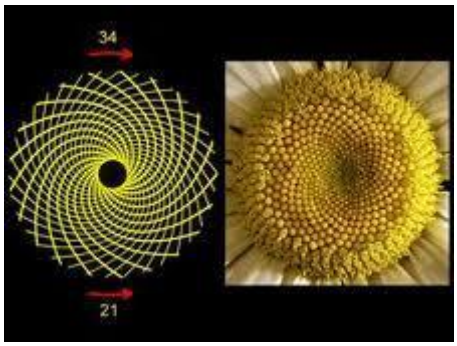
Gambar 7.3 : *Screenshot* hasil eksekusi program dinamis dengan n = 50
Sumber : Olahan penulis

VIII. Aplikasi dan Penerapan Barisan Fibonacci

Seperti sudah dibahas pada pendahuluan, terdapat beberapa kesesuaian barisan Fibonacci dengan alam, yaitu sebagai berikut :

1. Jumlah Daun pada Bunga (petals)
   Mungkin sebagian besar tidak terlalu memperhatikan jumlah daun pada sebuah bunga. Dan bila diamati, ternyata jumlah daun pada bunga itu menganut deret fibonacci. contohnya:
   – jumlah daun bunga 3 : bunga lili, iris
   – jumlah daun bunga 5 : buttercup (sejenis bunga mangkok)
   – jumlah daun bunga 13 : ragwort, corn marigold, cineraria,
   – jumlah daun bunga 21 : aster, black-eyed susan, chicory
   – jumlah daun bunga 34 : plantain, pyrethrum
   – jumlah daun bunga 55,89 : michaelmas daisies, asteraceae *family*

2. Pola bunga juga menunjukkan adanya pola fibonacci ini, misalnya pada bunga matahari.

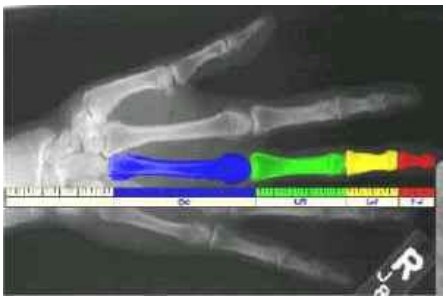


Gambar 8.1 : Pola bunga matahari yang mengikuti pola Fibonacci

Sumber : https://zakiyaasfi.wordpress.com/2012/12/02/uniknya-barisan-fibonacci/

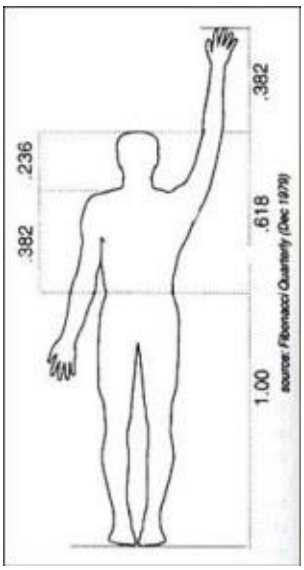
Dari titik tengah menuju ke lingkaran yang lebih luar, polanya mengikuti deret fibonacci.

3. Tubuh Manusia



Gambar 8.2 : Ruas-ruas tangan manusia
Sumber : https://zakiyaasfi.wordpress.com/2012/12/02/uniknya-barisan-fibonacci/

Bila Anda ukur panjang jari Anda, kemudian Anda bandingkan dengan panjang lekuk jari, maka akan ketemu 1.618.



Gambar 8.3 : Ukuran tubuh manusia
Sumber : https://zakiyaasfi.wordpress.com/2012/12/02/uniknya-barisan-fibonacci/

Penjelasannya sebagai berikut :
– Coba bagi tinggi badan Anda dengan jarak pusar ke telapak kaki, maka hasilnya adalah 1.618.
– Bandingkan panjang dari pundak ke ujung jari dengan panjang siku ke ujung jari, maka hasilnya adalah 1.618.
– Bandingkan panjang dari pinggang ke kaki

dengan panjang lutut ke kaki, maka hasilnya adalah 1.618
– Semua perbandingan ukuran tubuh manusia adalah 1.618. Benarkah? Silahkan Anda membuktikannya sendiri.

Pada ilmu ekonomi khususnya bagi seorang investor, barisan Fibonacci digunakan dalam sebuah sistem investasi bernama *Elliot wave theory*. Dengan teori ini, investor dapat memprediksi *levels of support and resistance*, *trend changes*, dan *price targets*. Tujuannya yaitu agar investor dapat memprediksi lebih lanjut harga stok dan memutuskan untuk berinvestasi atau tidak. Selain itu, barisan Fibonacci juga sering digunakan dalam pasar Forex.

Dalam ilmu komputer, beberapa kegunaan barisan Fibonacci adalah sebagai berikut:

1. Digunakan sebagai dasar struktur data Fibonacci *heap*. Struktur data ini digunakan untuk tujuan mempercepat kinerja beberapa algoritma praktikal.
2. Digunakan sebagai dasar algoritma Fibonacci *search technique*. Algoritma ini berguna untuk mencari elemen pada larik terurut. Algoritma yang digunakan untuk *searching* pada larik terurut umumnya adalah *binary search*, namun itu hanya optimal bila waktu pencarian untuk setiap elemen adalah sama. Situasi lain (seperti *memory* yang sudah di-*cached* yang memungkinkan Anda mencari secara lebih cepat elemen selanjutnya yang dekat dari elemen sebelumnya yang sudah dicari) memungkinkan penggunaan Fibonacci *search* secara lebih optimal dibandingkan *binary search*.

IX. Kesimpulan

Masalah bilangan Fibonacci bisa diselesaikan dengan berbagai algoritma, dua diantaranya adalah menggunakan prinsip rekursif dan menggunakan prinsip program dinamis. Melalui hasil eksperimen, masalah bilangan Fibonacci lebif efektif diselesaikan dengan algoritma program dinamis.

Hal ini dilihat dari segi kompleksitas waktu dan waktu eksekusi untuk menyelesaikan masalah bilangan Fibonacci. Kompleksitas waktu metode rekursif adalah $O(2^n)$ atau eksponensial, sedangkan kompleksitas waktu program dinamis adalah O(n) atau linear.

Maka, untuk n bernilai besar, kinerja program dinamis secara signifikan jauh lebih baik dan waktu eksekusi jauh lebih cepat. Hal ini didukung dengan hasil eksperimen.

Namun, sebenarnya masih ada algoritma yang lebih mangkus dari program dinamis untuk menyelesaikan masalah ini, yaitu menggunakan prinsip perpangkatan matriks Fibonacci. Algoritma ini memiliki kompleksitas waktu O(log n) yang sedikit lebih baik dari algoritma program dinamis.

X. Ucapan Terima Kasih

Pertama-tama, saya ingin mengucapkan rasa syukur kepada Allah, Tuhan Yang Maha Esa, karena berkat karunia dan rahmat-Nya lah penulis dapat menyelesaikan makalah ini.

Penulis juga ingin berterima kasih kepada Bapak Dr. Ir. Rinaldi Munir, M.T. dan Ibu Dr. Nur Ulfa Maulidevi, ST, M.Sc. selaku pengampu mata kuliah Strategi Algoritma. Sehingga, penulis mampu untuk membuat makalah terkait program dinamis ini.

Selain itu, saya ingin mengucapkan terima kasih kepada orang tua yang telah memberikan dorongan untuk terus berkarya dan rekan-rekan yang telah membantu penyusunan makalah ini.

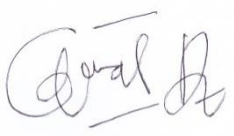
Referensi

[1] Munir, Rinaldi. 2009. ”Diktat Kuliah IF 2251 Strategi Algoritmik”. Bandung:Program Studi Teknik Informatika STEI ITB.
[2] http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2014-2015/Program%20Dinamis%20(2015).ppt
Diakses pada 4 Mei 2015 pukul 13.30
[3] https://www.quora.com/What-are-the-real-life-applications-of-Fibonacci-series
Diakses pada 4 Mei 2015 pukul 16.00
[4] https://www.stocktrader.com/2009/05/26/fibonacci-numbers-investors-sequence-elliot-wave-theory/
Diakses pada 4 Mei 2015 pukul 17.00
[5] https://zakiyaasfi.wordpress.com/2012/12/02/uniknya-barisan-fibonacci/
Diakses pada 4 Mei 2015 pukul 19.30

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, 9 Mei 2016



Geraldi Dzakwan 13514065

LAMPIRAN

Berikut saya lampirkan kode program yang saya buat untuk pengujian dan perbandingan metode rekursif dan program dinamis dalam menyelesaikan masalah bilangan Fibonacci.

```
package fibonacci;

//Author
//Nama  : Geraldi Dzakwan
//NIM   : 13514065

import java.util.*;

public class Fibonacci {

    //Fungsi untuk menghitung bilangan Fibonacci
    secara rekursif
    public long RecursiveFib (long n) {
        if (n<=1) {
            return n;
        } else {
            return RecursiveFib(n-1) +
            RecursiveFib(n-2);
        }
    }

    //Fungsi untuk menghitung bilangan
    Fibonacci dengan dynamic programming
    public long DPFib (long n) {
        long f[];
        f = new long [(int)n+2];

        f[0] = 0;
        f[1] = 1;

        for (int i = 2; i <= n; i++)
        {
             f[i] = f[i-1] + f[i-2];
        }

        return f[(int)n];
    }

    //Driver program
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);

        int n;
        //Validasi masukan, masukan lebih besar
        dari 92 akan menyebabkan overflow
        do {
            System.out.println("Masukan bernilai
            antara 0 hingga 92.");
            System.out.print("n : ");
            n = sc.nextInt();
        } while ((n<0) || (n>92));

        int p;
        do {
            System.out.println("Ketik 1 untuk
            rekursif atau 2 untuk PD.");
            System.out.print("Pilihan : ");
            p = sc.nextInt();
        } while ((p!=1) && (p!=2));

        Fibonacci F = new Fibonacci();
        long StartTime = System.nanoTime();

        if (p==1) {

            System.out.println("Fibonacci(" + n +
            ") = " + F.RecursiveFib(n));
        } else if (p==2) {
            System.out.println("Fibonacci(" + n +
            ") = " + F.DPFib(n));
        }

        long StopTime = System.nanoTime();
        long ElapsedTime = StopTime - StartTime;

        if (ElapsedTime > 1000000000) {

            System.out.println("Waktu eksekusi =
            " + ElapsedTime/1000000000+ "
            sekon");
        } else if (ElapsedTime > 1000000) {

            System.out.println("Waktu eksekusi =
            " + ElapsedTime/1000000+ " mili
            sekon");
        } else if (ElapsedTime > 1000) {

            System.out.println("Waktu eksekusi
            = " + ElapsedTime/1000+ " mikro
            sekon");
        } else {

            System.out.println("Waktu eksekusi
            = " + ElapsedTime+ " nano sekon");
        }
    }
}
```