Penerapan Decorator Dan Memorization Untuk Meningkatkan Kecepatan Algoritma Fibonacci

Khoirul Anam[1], Rendi Alexander Hutagalung[2], Andre Hadiman Rotua Parhusip[3], Cintya Bella[4], Lisa Diani Amelia[5]

Jurusan Sains Data, Fakultas Sains, Institut Teknologi Sumatera, Lampung Selatan, Indonesia

Email: khoirul.122450039@student.itera.ac.id rendi.122450057@student.itera.ac.id lisa.122450021@student.itera.ac.id cintya.122450066 @student.itera.ac.id andre.122450108@student.itera.ac.id

1. Pendahuluan

Bilangan Fibonacci adalah deret angka di mana setiap angka adalah jumlah dari dua angka sebelumnya. Deret ini dimulai dengan 0 dan 1. Jadi, beberapa angka pertama dalam deret Fibonacci adalah 0, 1, 1, 2, 3, 5, 8, 13, ... dengan setiap bilangan berikutnya yaitu jumlah dari dua buah bilangan sebelumnya, jadi bilangan Fibonacci keempat adalah penjumlahan dari bilangan Fibonacci kedua dan ketiga yaitu satu ditambah dua yaitu tiga. Deret bilangan Fibonacci dinamai oleh penemunya yaitu Leonardo of Pisa, yang dikenal sebagai Fibonacci.

Salah satu faktor penting yang mempengaruhi kebutuhan waktu suatu algoritma adalah keefektifan. Jika algoritma tidak efektif, maka pertambahan kecepatan eksekusi dalam komputer tidak mengubah jumlah waktu yang dibutuhkan algoritma untuk menyelesaikan suatu persoalan secara signifikan. Pertambahan kecepatan eksekusi pada komputer dapat dianalogikan sebagai pertambahan kecepatan dari operasi elementer seperti penjumlahan, perkalian, dan sebagainya. Namun, algoritma itu sendiri dapat dianalogikan sebagai strategi dalam menyelesaikan suatu persoalan. Dengan mempersingkat waktu yang diperlukan untuk melakukan operasi-operasi elementer, algoritma tersebut dapat menyelesaikan suatu persoalan dengan lebih cepat.

Namun, Seberapa lama waktu komputasi yang dibutuhkan untuk menyelesaikan suatu persoalan antara algoritma yang efektif. Maka, kami akan membahas algoritma yang digunakan untuk menemukan bilangan Fibonacci. Untuk menemukan cara yang lebih cepat untuk menemukannya, Anda perlu memahami teori bilangan, khususnya sifat-sifat bilangan Fibonacci. Kami menggunakan *Decoration* dan *Memorization* pada algoritma fibonacci untuk menentukan sendiri seberapa besar pengaruh keefektifan algoritma terhadap waktu yang dibutuhkan untuk menyelesaikan suatu persoalan.

2. Metode

Dalam menjalankan pemrograman penerapan memorization untuk meningkatkan kecepatan algoritma fibonacci. Strategi pendekatan algoritma yang kami gunakan yaitu ;

1. Rekursif

Fungsi rekursif digunakan untuk memanggil dirinya sendiri. Fungsi rekursif kami gunakan untuk perhitungan deret fibonacci.

2. Memorization

Memorization digunalan untuk menghasilkan pemanggilan fungsi pada argumen tertentu dapat disimpan, sehingga jika argumen tersebut dipanggil kembali, hasilnya dapat diambil langsung dari penyimpanan tanpa menghitung ulang. Dalam kode ini, memorisasi dilakukan dengan menggunakan dekorator memoize. Dekorator memoize mendefinisikan proses memorisasi.

3. Decorator

Decorator digunakan untuk mengubah fungsi atau metode tanpa mengubah kode secara langsung. Dalam algoritma ini, decorator memoize digunakan untuk menerapkan memoirsasi pada fungsi Fibonacci. Dengan menerapkan decorator *@memoize* pada fungsi Fibonacci, setiap panggilan fungsi akan diarahkan ke fungsi *wrapper* yang telah diubah untuk menyimpan dan memeriksa hasil memorisasi.

3. Pembahasan

Pada jurnal kali ini kami menggunakan program Memorization dan Decorator untuk algoritma pendekatan yang menghasilkan deret fibonacci sebagai berikut:

```
from functools import wraps

# Decorator untuk memoization
def memoize(func):
    memo = {}
    @wraps(func)
    def wrapper(n):
        if n not in memo:
            memo[n] = func(n)
        return memo[n]
    return wrapper
```

Gambar 1. Penggunan Decorator untuk Memoization

Pada baris pertama dilakukan adalah import wraps dari modul functools. Fungsi wraps yang digunakan untuk mempertahankan informasi metadata dari fungsi asli. Kemudian, mendefinisikan dekorator menggunakan dictionary memo untuk menyimpan hasil pemanggilan fungsi agar tidak terjadi pengulangan dari perhitungan. Selanjutnya, mendekorasi dengan fungsi @wraps(func) untuk memeriksa apakah argumen n sudah ada dalam memo. Setelah itu, membuat

kondisi di mana jika n belum ada dalam memo, maka fungsi func() akan dipanggil dan hasilnya disimpan dalam memo. Pada baris berikutnya, fungsi wrapper mengembalikan hasl dari memo[n].

```
# Fungsi rekursif Fibonacci
@memoize
def fibonacciku(n):
    if n <= 1:
        return n
    else:
        return fibonacciku(n-1) + fibonacciku(n-2)
```

Gambar 2. Program rekursif Fibonacci

Selanjutnya, pada program rekursif fibonacci ini membuat fungsi fibonacciku yang di mana bergunsi untuk menghitung bilangan Fibonacci dan menerima parameter n. Dari pendefinisian fungsi tersebut terdapat suatu kondisi di mana jika n kurang dari atau sama dengan 1, maka fungsi akan langsugn mengembalikan nilai n. Begitu juga sebaliknya, jika n lebih besar dari 1, maka fungsi akan mengembalikan jumlah dari dua pemanggilan rekursif sebelumnya pada fibonacciku(n-1) ditambah fibonacciku(n-2). Pada dekorator @memoize berfungsi untuk menyimpan hasil perhitungan program sebelumnya dalam sebuah dictionary.

```
# Contoh penggunaan
n = 15
print("Nilai Fibonacci ke-", n, "yaitu:", fibonacciku(n))
```

```
Nilai Fibonacci ke- 15 yaitu: 610
```

Gambar 3. Pengaplikasian Dan *Output*

Algoritma terakhir yang dilakukan adalah pengaplikasian setelah dibuat operasi algoritmanya dengan memberikan nilai n sama dengan 15. Kemudian membuat pemanggilan fungsi yang dicetak dengan parameter, yaitu “Nilai Fibonacci ke-” dan fungsi fibonacciku dipanggil dengan argumen n. Hasil pemanggilan tersebut menunjukkan nilai fibonacci sebesar 610. Hal ini menunjukkan bahwa fungsi fibonacciku berhasil melakukan perhitungan bilangan Fibonacci dengan penginputan n sama dengan 15 dan melakukan pengembalian nilai sebesar 610.

4. Kesimpulan

Berdasarkan materi di atas dapat disimpulkan bahwa:

1. Bilangan Fibonacci merupakan deret angka di mana setiap angka adalah jumlah dari dua angka sebelumnya.
2. Materi ini, menggunakan dua teknik yang digunakan dalam algoritma Fibonacci, yaitu rekursif dan memoization. Rekursif digunakan untuk memanggil dirinya sendiri dalam perhitungan deret

Fibonacci, sementara memoization digunakan untuk menyimpan hasil perhitungan sebelumnya dan menggunakannya kembali saat diperlukan. Penggunaan decorator juga dijelaskan sebagai cara untuk mengubah fungsi tanpa mengubah kode secara langsung.

3. Program yang dibahas dimulai dengan penggunaan decorator untuk memoization, di mana kode dijelaskan langkah demi langkah. Selanjutnya, program rekursif Fibonacci diperlihatkan dengan penjelasan tentang cara kerjanya. Terakhir, pengaplikasian dan output dari algoritma setelah diimplementasikan juga dijelaskan.
4. Melalui program tersebut di dapat bahwa penggunaan teknik memorisasi dan decorator dapat meningkatkan kinerja algoritma Fibonacci dengan menyimpan dan mengulang hasil perhitungan sebelumnya. Ini mengarah pada peningkatan efisiensi dan kecepatan dalam menyelesaikan masalah yang melibatkan deret Fibonacci.

5. Referensi

Munir, Rinaldi. 2006. Diktat Kuliah IF2120 Matematika Diskrit,

Edisi Keempat. Informatika Bandung: Bandung

Munir,Rinaldi. “Diktat Kuliah IF3051 Strategi Algoritma”. Institut

Teknologi Bandung. 2009