Penerapan *Memoization* dalam Permainan Tebak Angka dengan Decorator

Oktavia Nurwinda Puspitasari[1], Mutiara Dian Pitaloka[2], Syadza Puspadari Azhar[3], Kayla Amanda Sukma[4], Muhammad Wafi Raihan[5]

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

Email: oktavia.122450041@student.itera.ac.id mutiara.122450047@student.itera.ac.id syadza.122450072@student.itera.ac.id kayla.122450086@student.itera.ac.id muhammad.122450144@student.itera.ac.id

1. Pendahuluan

Permainan klasik yang populer dikalangan anak-anak dan orang dewasa yaitu permainan tebak angka (*guessing game*), dimana permainan tebak angka ini diimplementasikan dengan berbagai cara, salah satunya dengan menggunakan algoritma pencarian biner. Algoritma yang pencarian biner adalah algoritma yang efisien untuk mencari nilai tertentu dalam daftar yang diurutkan. Algoritma yang bekerja dengan membagi daftar menjadi dua bagian secara berulang dan membuang setengah dari daftar yang tidak mengandung nilai yang dicari.Namun, algoritma pencarian biner dapat menjadi tidak efisien jika daftar angkanya sangat besar. Hal ini karena algoritma ini perlu melakukan banyak operasi perbandingan untuk menemukan nilai yang dicari.

Dalam meningkatkan efisiensi pencarian kata pada permainan tebak angka disini kita akan menggunakan Teknik memorization yang akan sangat berguna untuk menyimpan dan mengakses hasil komputasi yang telah dihitung sebelum nya tanpa menghitung kembali. Sehingga *memoization* akan sangat berguna Ketika terdapat permintaan pencarian yang sama sehingga tidak membuat program banyak bekerja berulang kali karena informasi kata-kata sebelumnya telah ditemukan dan disimpan.

2. Tujuan

Adapun tujuan dari penulisan laporan Penerapan *Memoization* dalam Permainan Tebak Angka dengan Decorator yaitu:

a. Menerapkan *closure* dengan menggunakan fungsi *decorator* dengan metode *memoization* pada permainan tebak angka.
b. Memahami penggunaan fungsi *decorator* dalam meningkatkan efisiensi dan kenyamanan selama program dijalankan.

3. Metode

a. *Time dan Functools*

Pada penerapan permainan tebak angka terdapat dua *library* yang digunakan yaitu *time* dan *functools*. *Library time* adalah *library* pada python yang digunakan untuk melakukan operasi yang berkaitan dengan waktu dan tanggal [1]. Pada kasus ini

library time digunakan untuk menghitung jumlah waktu yang digunakan untuk mengeksekusi saat pengguna melakukan permainan tebak angka.

Selanjutnya *library* yang digunakan adalah *functools*. *Functools* merupakan *library* yang disediakan pada python yang digunakan untuk penggunaan *decorators* yang lebih fleksibel. Dengan library ini penggunaan fungsi *decorators* dapat digunakan dengan lebih luas lagi dan dikembangkan dengan menambahkan fungsionalitas tanpa mengubah fungsi tersebut. Pada kasus ini *functools* yang digunakan adalah "*functools.wrap*" dimana fungsi ini akan memastikan nama fungsi dan atribut tetap konsisten setelah dilakukan fungsi *decorations*.

b. *Decorators*

Dalam bahasa pemrograman python, *decorators* merupakan suatu fungsi yang dapat melengkapi dan mengubah suatu fungsi dengan cara yang bersih. *Decorators* dapat digunakan untuk mengelola suatu kode terutama saat terdapat fitur khusus pada kode kita [2]. Secara ringkas, fungsi *decorators* dapat digunakan untuk mengubah suatu perilaku fungsi tanpa mengubah kode aslinya.

Fungsi ini bekerja dengan membungkus dengan memodifikasi fungsi tersebut. *Decorators* dapat digunakan dengan dengan metode *inner function* dan juga python *closure*. Dengan menggunakan fungsi decorators kita dapat mengelola suatu kode dengan cara yang dinamis tanpa mempengaruhi kode yang tidak berhubungan [3].

c. *Memoization*

Memoization adalah metode yang digunakan untuk mengoptimalkan program dengan menyimpan hasil eksekusi yang akan digunakan pada eksekusi berikutnya. Konsep *memoization* menggunakan implementasi *cache* namun hanya tertuju untuk caching pada *function*. Dengan menggunakan *memoization* kita dapat mengoptimasi proses eksekusi pada *function*s. Akan tetapi terdapat beberapa kondisi dimana *memoization* dapat digunakan dan tidak dapat digunakan. Jika *return function* yang digunakan konsisten dengan argumen maka *function* akan tidak cocok dengan penerapan *memoization* [4]. Pada kasus ini *memoization* digunakan untuk menyimpan hasil tebakan ke dalam *cache*, sehingga pengguna dapat menebak angka lebih dari sekali. *Memoization* dapat mengurangi jumlah rekursif untuk meningkatkan kinerja pas permainan tebak angka.

4. Pembahasan

Pada modul ini, kami menggunakan fungsi *decorator* dengan teknik *memoization*. Program akan dirancang sedemikian rupa sehingga angka yang ditebak tidak sama dengan apa yang ditebak sebelumnya. Pengguna diminta untuk menebak angka dari 1-100, program akan terus berjalan hingga pengguna berhasil menebak angka yang disimpan oleh program. Ketika angka telah ditebak, program akan menunjukkan waktu yang dibutuhkan dalam menjalankan fungsi ini.

```
import functools
import time
import random
```

Program dimulai dengan mengimport modul-modul yang dibutuhkan, yaitu *functools*, *time*, dan *random*. Modul *functools* berisi berbagai *helper functions* yang dapat digunakan dengan fungsi-fungsi pada *Python*. Modul *time* pada program ini digunakan untuk menghitung waktu yang dibutuhkan untuk program berjalan. Modul *random* digunakan untuk menentukan angka random dari 1-100 yang akan disimpan pada program dan menjadi angka yang harus ditebak oleh pengguna.

```
def memoize(func):
    cache = {}
    @functools.wraps(func)
    def wrapper(*args):
        if args not in cache:
            cache[args] = func(*args)
        return cache[args]
    return wrapper
```

Kode di atas mendefinisikan sebuah fungsi *memoize* sebagai fungsi *decorator* utama untuk menyimpan hasil dari pemanggilan fungsi sehingga ketika fungsi dipanggil lagi dengan argumen yang sama, hasilnya akan diambil dari *cache* tanpa harus melakukan perhitungan ulang. Fungsi ini menggunakan modul '`functools.wraps`' yang digunakan untuk memastikan bahwa fungsi *wrapper* yang dihasilkan memiliki metadata yang sama dengan fungsi aslinya. Di dalam fungsi *memoize* juga terdapat fungsi *wrapper* yang akan menggantikan fungsi asli ketika fungsi tersebut dipanggil. Fungsi ini memiliki argumen '`*args`' yang memungkinkan fungsi *decorator* menerima argumen sebanyak apapun.

```
def display_time(func):
    @functools.wraps(func)
    def wrapper(*args, **kwargs):
        start_time = time.time()
        result = func(*args, **kwargs)
        end_time = time.time()
        print(f"Elapsed time for {func.__name__}: {end_time -
start_time} seconds")
        return result
    return wrapper
```

Fungsi diatas merupakan fungsi yang digunakan untuk menghitung waktu eksekusi dari sebuah fungsi. Fungsi *wrapper* dimulai dengan menghitung waktu eksekusi sejak fungsi dijalankan. Ketika fungsi selesai dijalankan, waktu akan dihitung kembali untuk mendapatkan waktu akhir eksekusi. Durasi eksekusi dihitung dengan mengurangkan waktu akhir dengan waktu awal. Fungsi ini akan mengembalikan hasil dari pengurangan tersebut.

```
def x():
  return random.randint(1, 100)
```

Fungsi x() digunakan untuk membuat angka random yang dimulai dari 1 hingga 100. Angka tersebut akan disimpan dan digunakan sebagai angka yang harus ditebak oleh pengguna ketika program dijalankan. Angka yang dihasilkan akan berbeda tiap kali program dijalankan, sehingga pengguna dapat menebak angka yang berbeda setiap sesi.

```
@memoize
def tebak_angka(target):
    guess = int(input("Tebak angka (1-100): "))
    if guess < target:
        print("Tebakan terlalu rendah!")
        return tebak_angka(target)
    elif guess > target:
        print("Tebakan terlalu tinggi!")
        return tebak_angka(target)
    else:
        print("Selamat! Anda menebak dengan benar.")
```

Dengan menggunakan `@memoize`, fungsi *memoize* akan dijalankan pada fungsi setelahnya yaitu fungsi '`tebak_angka`' dimana fungsi *memoize* digunakan untuk mempercepat proses menebak angka dengan menyimpan hasil tebakan sebelumnya. Fungsi '`tebak_angka`' ini akan meminta inputan angka dari 1-100 yang setelah itu akan melalui proses *if else* dengan kondisi apabila tebakan yang dimasukkan pengguna lebih kecil daripada target yang disimpan, maka program akan mengeluarkan pesan "Tebakan terlalu rendah!" dan jika tebakan lebih besar daripada target yang diberikan, maka program akan mengeluarkan pesan "Tebakan terlalu tinggi!". Setelah itu kedua kondisi tersebut akan mengulang fungsi '`tebak_angka`' sehingga terdapat *looping*. *Looping* ini akan berhenti ketika pengguna berhasil menebak angka target yang disimpan oleh program.

```
@display_time
def main():
    print("Selamat datang di permainan tebak angka!")
    target = x()  # Angka yang harus ditebak
```

```
    tebak_angka(target)
```

Pada tahap ini, fungsi ‘`display_time`’ akan dijalankan sehingga ketika program berhenti bekerja, program akan menampilkan banyak waktu yang dibutuhkan untuk program bekerja. Fungsi main() merupakan fungsi utama yang akan menyimpan angka target hasil dari pengambilan angka secara acak dari fungsi x(). Setelah itu, fungsi ini akan menjalankan fungsi ‘`tebak_angka`’ untuk memulai permainan tebak angka.

```
main()
```

Setelah program selesai dibuat, program dapat dijalankan dengan memanggil fungsi main(). Fungsi ini akan menyimpan angka target yang berbeda setiap kali program dijalankan ulang. Adapun contoh hasil dari permainan dapat dilihat pada gambar output berikut.

```
Selamat datang di permainan tebak angka!
Tebak angka (1-100): 50
Tebakan terlalu rendah!
Tebak angka (1-100): 86
Tebakan terlalu rendah!
Tebak angka (1-100): 90
Tebakan terlalu rendah!
Tebak angka (1-100): 95
Tebakan terlalu tinggi!
Tebak angka (1-100): 93
Tebakan terlalu tinggi!
Tebak angka (1-100): 92
Tebakan terlalu tinggi!
Tebak angka (1-100): 91
Selamat! Anda menebak dengan benar.
Elapsed time for main: 27.94391918182373 seconds
```

Gambar 1. Output hasil permainan tebak angka

5. Kesimpulan

Studi ini mengilustrasikan penerapan *memoization* dalam permainan tebak angka menggunakan fungsi *decorator* dalam bahasa pemrograman Python. *Memoization* digunakan untuk meningkatkan efisiensi permainan dengan menyimpan hasil tebakan sebelumnya, menghindari perhitungan ulang yang tidak perlu. Fungsi *decorator* digunakan untuk mengelola perilaku fungsi tanpa mengubah kode aslinya.

Penerapan *memoization* dalam permainan tebak angka memungkinkan pengguna untuk menebak angka lebih cepat dan efisien, terutama saat terdapat pengulangan argumen. Dengan memanfaatkan *memoization*, program dapat menyimpan dan mengambil hasil tebakan sebelumnya dari *cache* saat tebakan yang sama dilakukan lagi.

Selain itu, penggunaan fungsi display_time membantu dalam mengukur waktu eksekusi program secara keseluruhan, memberikan pemahaman yang lebih baik tentang kinerja program dan potensi untuk pengembangan lebih lanjut dalam peningkatan efisiensi kode.

Secara keseluruhan, teknik memoization dan fungsi decorator memberikan kontribusi yang signifikan dalam meningkatkan efisiensi dan kenyamanan dalam pengembangan perangkat lunak.

6. DAFTAR PUSTAKA

[1] N. Afif, "Memoization di Python," 9 October 2020. [Online]. Available: https://naufalafif58.medium.com/memoization-di-python-3e1ef419d371. [Accessed 25 April 2024].

[2] N. A. Prayogo, "E-book Dasar Pemrograman Golang," [Online]. Available: https://dasarpemrogramangolang.novalagung.com/1-berkenalan-dengan-golang.html. [Accessed 25 April 2024].

[3] A. Richard, "Memodifikasi Fungsi dengan Python Decorator (Bagian 1)," 23 January 2021. [Online]. Available: https://agusrichard.medium.com/python-decorators-d5bbde71a730. [Accessed 25 April 2024].

[4] "9 Library Python Terbaik untuk Data Analytics 2023," 25 August 2023. [Online]. Available: https://revou.co/panduan-teknis/library-python. [Accessed 25 April 2024].