18. Memoization kết hợp với Decorators

Định nghĩa memorization

Memoization là một kỹ thuật được sử dụng trong tính toán để tăng tốc độ các chương trình. Điều này được thực hiện bằng cách ghi nhớ các kết quả tính toán của đầu vào và xử lý như kết quả của các cuộc gọi hàm. Nếu cùng một đầu vào hoặc gọi hàm với cùng một tham số, kết quả được lưu trữ trước đó có thể được sử dụng lại và giảm được thời gian tính toán. Trong nhiều trường hợp, một mảng đơn giản được sử dụng để lưu trữ kết quả, nhưng có thể sử dụng nhiều cấu trúc khác, chẳng hạn như các mảng kết hợp, được gọi là bảng băm trong Perl hoặc dictionary trong Python.

Memoization có thể được lập trình bởi lập trình viên, nhưng một số ngôn ngữ lập trình như Python cung cấp cơ chế để tự động ghi nhớ các hàm.

Memoization với Function Decorators.

Bạn có thể tham khảo chương về decorator tại đây.

Trong chương trước, tôi đã sử dụng phương pháp dưới đây để tính dãy số Fibonacci.

```
import time
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-1) + fib(n-2)

def measure_cal_fib(x):
    start = time.time()
    result = fib(x)
    done = time.time()
    print result, "runtime = ", done -start

measure_cal_fib(35)

Output:

9227465 runtime =  3.38899993896
```

Dễ nhận thấy cách làm này có độ phức tạp tăng theo hàm mũ (exponential runtime), ví dụ tính fib(n) dựa vào fib(n-1) và fib(n-2) nhưng với cách phía trên ta lại phải tính lại fib(n-2) đã thực sự được tính trong fib(n-1).

Giải pháp? Có thể sử dụng dictionary để lưu lại kết quả tính toán của fib(n-1), fib(n-2) để tính fib(n) một cách nhanh chóng, và ở đây chính là biến “memo”

```
import time
memo = {0:0, 1:1}
def fibm(n):
    if not n in memo:
        memo[n] = fibm(n-1) + fibm(n-2)
    return memo[n]

def measure_cal_fibm(x):
    start = time.time()
    result = fibm(x)
    done = time.time()
    print result, "runtime = ",done - start

measure_cal_fibm(35)


Output:


9227465 runtime = 0.0
```

Cách thứ 2 giúp cho việc tính Fibonacci nhanh hơn cách 1 nhiều lần. Tuy nhiên, bất lợi của phương pháp thứ 2 đó là vẻ đẹp và sự rõ ràng của việc thực thi đệ quy ban đầu bị mất.

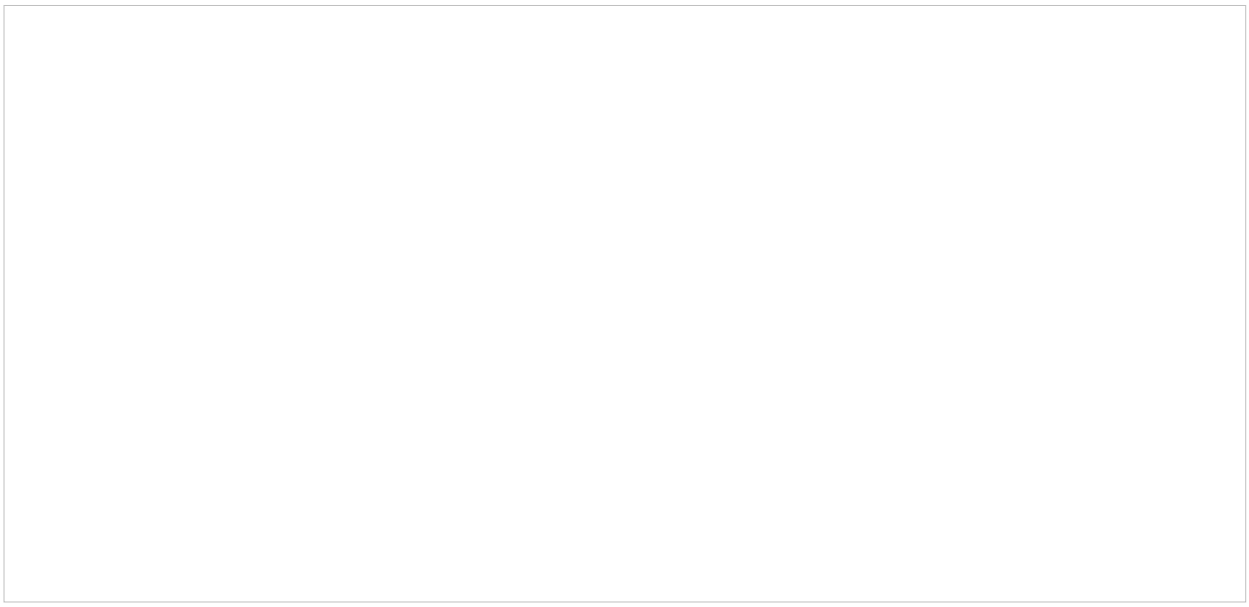
"Vấn đề" đó là chúng ta đã thay đổi mã của hàm đệ quy tính fib.

Đề đảm bảo tính nguyên vẹn của hàm đệ quy ban đầu thì “decorator” là một giải pháp giúp chúng ta giải quyết vấn đề này. Đoạn mã sau không làm thay đổi chức năng hàm fib. Ta dùng cách tiếp cận tương tự với hàm memorize để wrap lại hàm gốc tính fib(n) theo tư duy thông thường.

```
import time
def memoize(f):
    memo = {}
    def helper(x):
        if x not in memo:
```

```
            memo[x] = f(x)
        return memo[x]
    return helper
@memoize
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-1) + fib(n-2)

def measure_cal_fibm(x):
    start = time.time()
    result = fib(x)
    done = time.time()
    print result, "runtime = ",done - start

measure_cal_fibm(35)

Output:

9227465 runtime =  0.0
```

Để mô tả lại quá trình này, tôi dùng hình sau để miêu tả:



Sau khi thực thi fib = memorize(fib), fib sẽ trỏ tới thân của hàm helper(), được trả về từ memorize. Chúng ta cũng có thể nhận thấy rằng mã của hàm gốc fib có thể từ bây giờ chỉ được gọi thông qua hàm "f" của hàm helper. Không có cách nào khác để gọi hàm gốc fib một cách trực tiếp, nghĩa là không có tham chiếu nào vào nó. Tính toán Fibonacci trong lời gọi fib(n-1) và fib(n-2) trong thân hàm fib(n) chính là gọi tới hàm decorated hay chính là đoạn mã của helper thông qua việc ghi nhớ (memorize). Bạn có thể hình dung nó qua hình vẽ sau:

Kết luận

Memoization về cơ bản không phải là một kỹ thuật mới trong lập trình, với mục đích chính là để tối ưu và tăng tốc độ xử lý chương trình thông qua việc ghi nhớ lại kết quả tính toán của những lần thực thi trước đó. Kỹ thuật này đặc biệt hữu ích với phép đệ quy. Với cùng một đầu vào hoặc một lời gọi hàm với các đối số truyền vào có giá trị như nhau, chúng ta không cần phải tính toán lại mà sử dụng ngay kết quả được lưu lại từ lần tính toán trước. Python sử dụng dictionary để lưu trữ các kết quả, kết hợp với công cụ decorator mạnh mẽ để không làm mất đi tính toàn vẹn và rõ ràng của hàm gốc.