UPPSALA UNIVERSITET

Fibonacci-tal

Tom Smedsaas

Komplexitet vid rekursiv beräkning av Fibonacci-tal.

Vi ska också introducera tekniken “memoization” för att effektivisera vissa rekursiva program.

UPPSALA UNIVERSITET

Rekursiv Fibonacci-funktion

Fibonacci-talen kan definieras rekursivt:

$$F_n = \begin{cases} 0 & \text{om } n = 0 \\ 1 & \text{om n} = 1 \\ F_{n-1} + F_{n-2} & \text{om } n > 1 \end{cases}$$

och beräknas med en rekursiv funktion:

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

UPPSALA UNIVERSITET

Demo 1

Komplexitet för `fib`-funktionen

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

Låt $t(n)$ vara antalet additioner som görs av anropet `fib(n)`.

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

Kan uppskattas uppåt:

$$t(n) = t(n-1) + t(n-2) + 1 < 2t(n-1) + 1 = 2^n - 1$$

Kan uppskattas nedåt:

$$t(n) = t(n-1) + t(n-2) > 2t(n-2) + 1 \approx \sqrt{2}^n$$

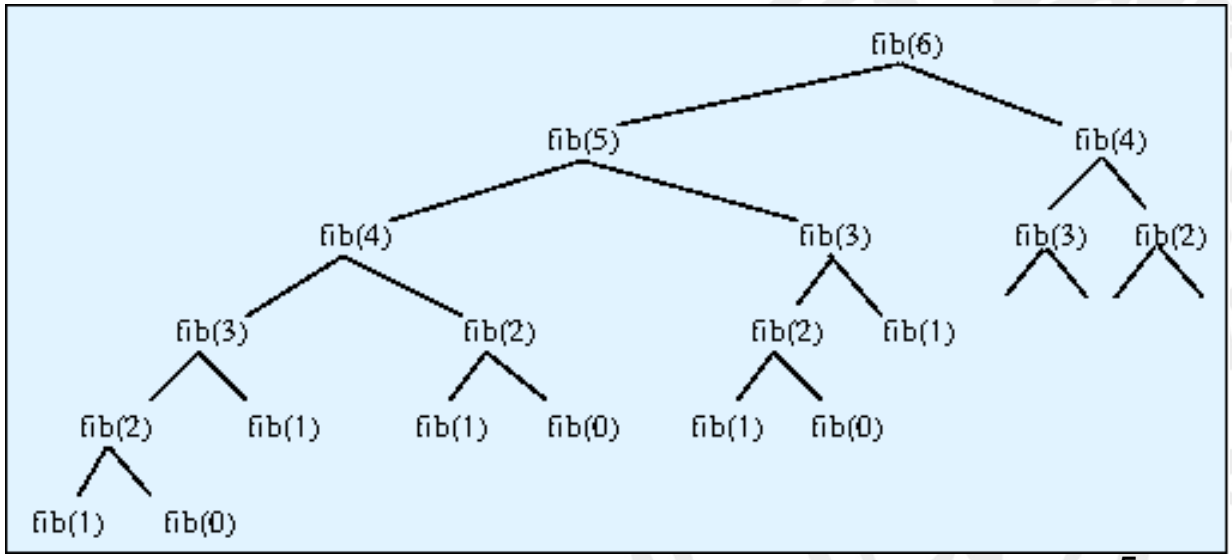
Man kan visa att $t(n) \approx c \cdot 1.618^n$ Hopplöst för stora n !

UPPSALA UNIVERSITET

Varför blir det så ineffektivt?

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

Vad händer vid anropet `fib(6)` ?



UPPSALA UNIVERSITET

”Memoization”

Spara beräknade värden och använd dessa i första hand.

```
memory = {0:0, 1:1}

def fib(n):
    if n not in memory:
        memory[n] = fib(n-1) + fib(n-2)
    return memory[n]
```

Kan nu beräkna väldigt stora Fibonacci-tal!

UPPSALA UNIVERSITET

Demo 2

Kan man dölja variabeln memory?

Variabeln memory är nu global – inte så vackert!
Den används ju bara i funktionen fib.

```
memory = {0:0, 1:1}

def fib(n):
    if n not in memory:
        memory[n] = fib(n-1) + fib(n-2)
    return memory[n]
```

Så här?

```
def fib(n):
    memory = {0:0, 1:1}
    if n not in memory:
        memory[n] = fib(n-1) + fib(n-2)
    return memory[n]
```

UPPSALA UNIVERSITET

Demo 3

UPPSALA UNIVERSITET

Bättre försök att gömma `memory`

Låt fib vara en ”omslagsfunktion” till den rekursiva funktionen och placera `memory` i den:

```
def fib(n):

    memory = {0:0, 1:1}

    def _fib(n):
        if n not in memory:
            memory[n] = _fib(n-1) + _fib(n-2)
        return memory[n]

    return _fib(n)
```

UPPSALA UNIVERSITET

Demo 4

UPPSALA UNIVERSITET

Alltså

- Variabler skapade i en funktion är lokala i den funktionen.
- Funktionsdefinitioner kan innehålla funktionsdefinitioner. Sådana funktioner är lokala till den omslutande funktionen.
- I en lokal funktion är variablerna i den omslutande funktionen tillgängliga.
- Variabler deklarerade på högsta nivån (utanför alla funktioner) är global Undvik att använda globala variabler!

Frivillig övning: Skriv om funktionen `exchange` med memoization.

UPPSALA UNIVERSITET

Pythonspecialitet

```
from functools import lru_cache

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

UPPSALA UNIVERSITET

Demo 5

UPPSALA UNIVERSITET

The end

.