Fibonacci- numbers

Tom Smedsaas

Complexity when computing Fibonacci numbers recursively.

Introduction of memoization to make recursive programs much more efficient.

Recursive Fibonacci function

The Fibonacci numbers can be recursively defined:

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

and recursively computed by the function:

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

Demo 1

Complexity for the `fib`-function

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

Let $t(n)$ be the number of addition made by the call `fib(n)`.

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

Can be given an upper estimate by:

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

Can be given a lower estimate by:

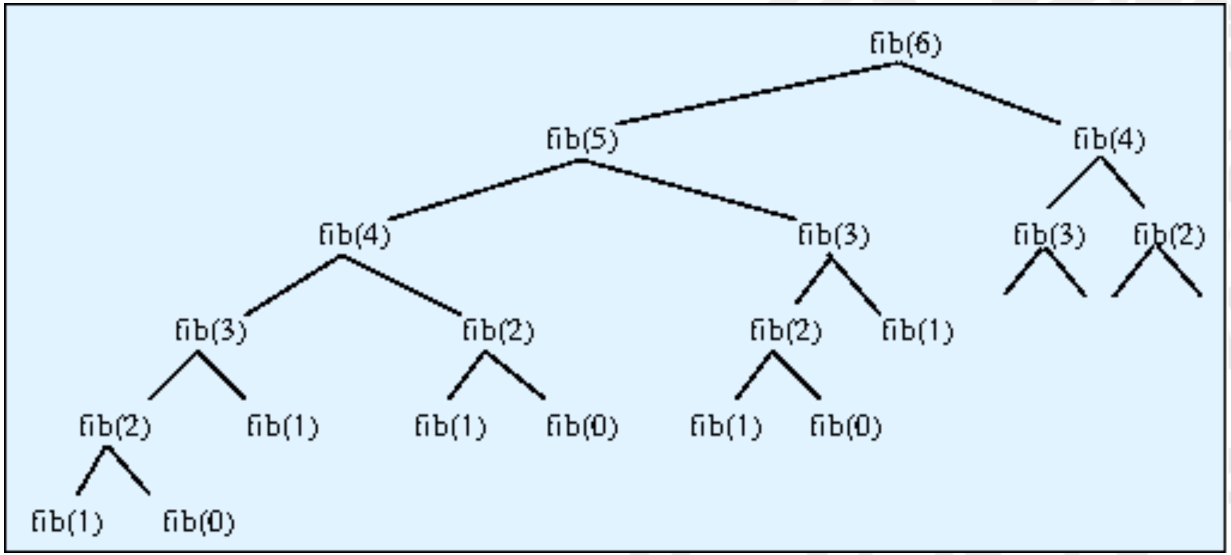
$$t(n) = t(n-1) + t(n-2) > 2t(n-2) + 1 \; \approx \; \sqrt{2}^n$$

It can be shown that $t(n) \approx c \cdot 1.618^n$ Hopeless for large n !

Why does it become so inefficient?

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

What is happening at the call `fib(6)` ?



Memoization

Save calculated values and use these first.

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

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

Now we can calculate very large Fibonacci numbers!

Demo 2

Can we hide the memory variable?

The variable `memory` is now global – not so nice!
It is only used in the function 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]
```

In this way?

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

Demo 3

A working way to hide memory

Let fib be a "wrapper function" to the recursive function and place memory in it:

```
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)
```

Demo 4

Thus

- Variables created in a function are local to that function.
- Function definitions can contain function definitions. Such functions are local to the enclosing function.
- In a local function, the variables in the enclosing function are available.
- Variables declared at the top level (outside all functions) are global Avoid using global variables!

Optional exercise: Rewrite the function exchange with memoization.

A Python specialty

```python
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)
```

Demo 5

UPPSALA
UNIVERSITET

The end

.