Análise Algoritmo Fibonacci

Roberto Alves Neto

Avaliação preguiçosa

Em python 2 era usado eager evaluation:

```
>>> a = range(10)
>>> print a
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> a[3]
3
>>>
```

Já em python 3, lazy evaluation:

```
>>> a = range(10)
>>> print(a)
range(0, 10)
>>> a[3]
3
>>>
```

Tabela Auxiliar

```
f(4) = f(4-1) + f(4-2)
     = f(3) + f(2)
     = f(3-1) + (3-2) + f(2-1) + f(2-2)
     = f(2) + f(1) + f(1) + f(0)
```

Quando ele calcular o primeiro **f(2)** o valor é salvo em um mapa, e na segunda chamada de **f(2)** ele já sabe o valor, sem precisar calcular.

Recursão em cauda para loop

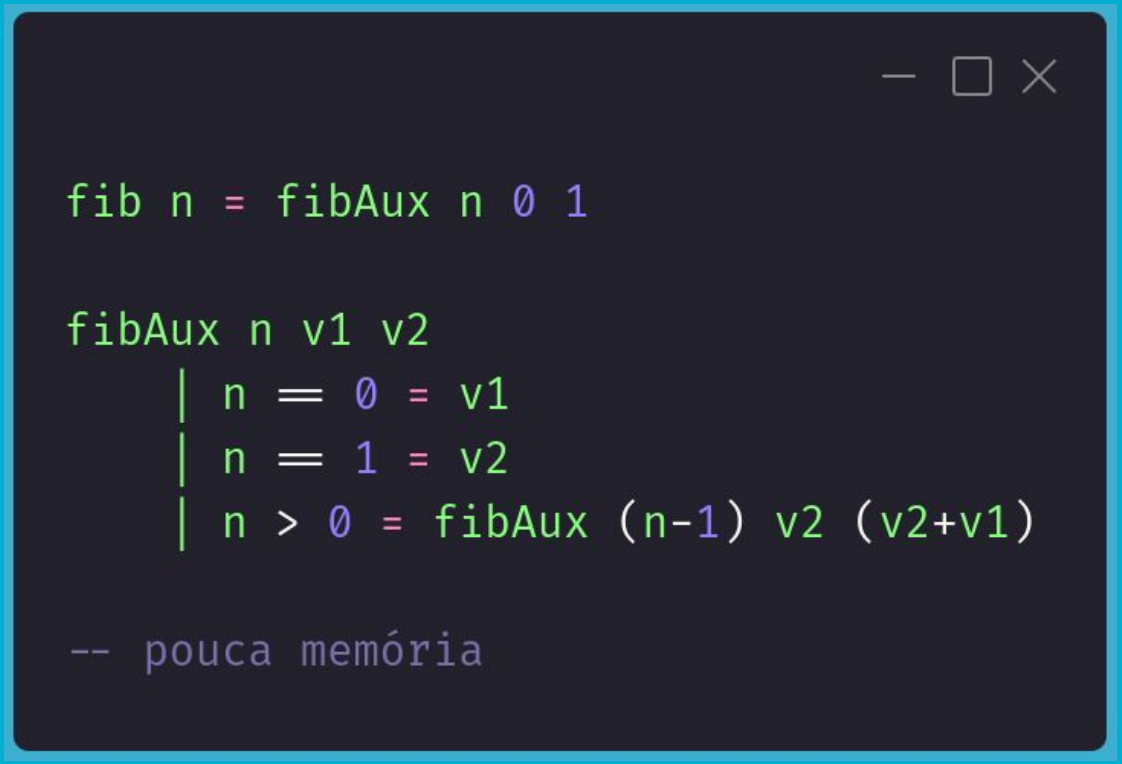
Exemplo em python:

```python
n = 10      # fib de 10
n1, n2 = 0, 1
count = 0

while count < n:
    aux = n1 + n2
    n1 = n2
    n2 = aux
    count += 1

print(n1)   # 55
```

Código Fibonacci

```
fib (0) = 0
fib (1) = 1
fib (n) = fib (n-1) + fib (n-2)
-- O(1.6^n)
-- muita memória
```

Código Fibonacci em cauda



```haskell
fib n = fibAux n 0 1

fibAux n v1 v2
    | n == 0 = v1
    | n == 1 = v2
    | n > 0 = fibAux (n-1) v2 (v2+v1)

-- pouca memória
```

Código Fibonacci da net

```haskell
fib = 0 : 1 : zipWith (+) fib (tail fib)

-- O(n²)
-- pouca memória
```

Fib de 10

FIB NORMAL:

0:00.00 elapsed

3408kb memory

FIB CAUDA:

0:00.00 elapsed

3576kb memory

FIB LIST:

0:00.00 elapsed

3728kb memory

Fib de 20

FIB NORMAL:

0:00.01 elapsed

4492kb memory

FIB CAUDA:

0:00.00 elapsed

3688kb memory

FIB LIST:

0:00.00 elapsed

3580kb memory

Fib de 30

FIB NORMAL:

0:00.28 elapsed

4616kb memory

FIB CAUDA:

0:00.00 elapsed

3408kb memory

FIB LIST:

0:00.00 elapsed

3692kb memory

Fib de 100

FIB NORMAL:

^CCommand terminated by signal 2

2:06:02 elapsed

6336kb memory

FIB CAUDA:

0:00.00 elapsed

3676kb memory

FIB LIST:

0:00.00 elapsed

3676kb memory

Fib de 1000

FIB NORMAL:

NT - Nem Tentei

FIB CAUDA:

0:00.00 elapsed

4276kb memory

FIB LIST:

0:00.00 elapsed

3968kb memory

Compilação com melhoria de eficiência:

		30	35	40
Fib Normal:	time	0:00.28	0:02.91	0:32.68
	memory	4616kb	4900kb	6620kb
Fib Norm -O :	time	0:00.09	0:00.82	0:08.87
	memory	4628kb	4532kb	4724kb
Melhoria em porcentagem:		68%	72%	73%

Compilação com melhoria de eficiência:

		10.000	100.000	1.000.000
Fib Cauda:	time	0:00.01	0:00.28	0:39.81
	memory	6236kb	61680kb	173476kb
Fib Cauda -O :	time	0:00.00	0:00.27	0:38.54
	memory	6124kb	60140kb	157704kb
Melhoria em porcentagem:		0%	3.6%	3.2%

Material consultado

https://www.ic.unicamp.br/~oliveira/doc/mc102_2s2004/Aula19.pdf

http://www.facom.ufu.br/~madriana/PF/recursao.pdf

https://qastack.com.br/programming/13042353/does-haskell-have-tail-recursive-optimization

https://stackoverflow.com/questions/1105765/generating-fibonacci-numbers-in-haskell

https://pt.stackoverflow.com/questions/4282/qual-a-diferen%c3%a7a-entre-recurs%c3%a3o-e-recurs%c3%a3o-de-cauda