Nesta aula...

Sobre a recursividade...

Conteúdo

1 O velho factorial

Definição de factorial

Para $n \geq 0$ define-se o factorial de n como

$$n! = \begin{cases} 1 & \text{se } n = 0 \\ n(n-1)! & \text{se } n \geq 1 \end{cases}$$

1. Mostre que $n!$ fica definido para todo o n inteiro não negativo. Use indução matemática na sua demonstração.

2. Escreva uma função `fact(n)` baseada directamente na definição dada.

Definição indutiva $\Rightarrow$ Programa recursivo

O programa recursivo

Definição:

$$\boxed{n! = \begin{cases} 1 & \text{se } n = 0 \\ n(n-1)! & \text{se } n \geq 1 \end{cases}}$$

$\Rightarrow$ Programa, quase a mesma coisa:

```
def fact(n):
    if n==0:
        return 1
    return n*fact(n-1)
```

Experimentando...

```
>>> fact(1)
1
>>> fact(100)
933262154439441526816992388562667004907159682643816214685929638952175999932299
156089414639761565182862536979208272237582511852109168640000000000000000000000
```

Exercício

O que imprime e o que retorna o programa seguinte com a chamada `fact(5)`?

```
def fact(n):
    a=n
    print a
    if n==0:
         return 1
    t= n*fact(n-1)
    print a
    return t
```

Versão iterativa...

```
def fact(n):
    p=1
    i=1
    while i<=n:
        # ----
        p=p*i
        i=i+1
    return p
```

1. Indique um invariante de ciclo significativo válido na linha `---`.
2. Mostre, usando indução em `n`, que quando o ciclo terminar é `p` $=$ `n`!

2 Combinações

Exercício!

Para $n, m \geq 0$ o número $\binom{m}{n}$ de combinações de m objectos tomados n a n é definido indutivamente por

$$\binom{m}{n} = \begin{cases} 0 & \text{se } n > m \\ 1 & \text{se } n = 0 \\ \binom{m-1}{n} + \binom{m-1}{n-1} & \text{nos outros casos} \end{cases}$$

1. Mostre que $\binom{m}{n}$ fica definido para todo o m e n com $m, n \geq 0$.
2. Escreva uma função `comb(m,n)` baseada directamente na definição dada.
3. Use esta função para imprimir o triângulo de Pascal para $0 \leq m, n \leq 5$.

Mostre que fica definido, caso base

Domínio de definição: $n \geq 0$, $m \geq 0$

```
Caso base $m=0$ ou $n>m$:
                          n
         0   1   2   3   4   5   6   7   8
  ----------------------------------------
   0 |   +   +   +   +   +   +   +   +   +
   1 |   +       +   +   +   +   +   +   +
   2 |   +           +   +   +   +   +   +
   3 |   +               +   +   +   +   +
m  4 |   +                   +   +   +   +
   5 |   +                       +   +   +
   6 |   +                           +   +
   7 |   +                               +
   8 |   +
  ----------------------------------------
```

Mostre que fica definido, passo indutivo

Definido para $m-1 \Rightarrow$ Definido para m Domínio de definição: $n \geq 0$, $m \geq 0$

```
Passo indutivo:
                        n
          0  1  2  3  4  5  6  7  8
     -----------------------------------------
      0 | +  +  +  +  +  +  +  +  +
      1 | +     +  +  +  +  +  +  +
      2 | +        +  +  +  +  +  +
      3 | +           +  +  +  +  +
m-1   4 | +  +  +  +  +  +  +  +  + <-- Hip. indutiva
m     5 | +  o  o  o  o  o  +  +  +
      6 | +                    +  +
      7 | +                       +
      8 | +
     -----------------------------------------
```

Definição indutiva $\Rightarrow$ Programa recursivo

O programa recursivo

Muito fácil, quase a mesma coisa!

```
def binom(m,n):
    if n>m:
        return 0
    if n==0:
        return 1
    return binom(m-1,n)+binom(m-1,n-1)
```

Experimentando...

```
>>> binom(4,2)
6
>>> binom(0,0)
1
>>> binom(4,5)
0
```

Versão iterativa...

Exercício.

1. Escreva uma versão iterativa de `binom` baseada directamente em $\binom{m}{n} = \frac{m!}{n!(m-n)!}$
2. Escreva uma versão iterativa de `binom` baseada directamente em

$$\binom{m}{n} = \frac{m \times (m-1) \times \cdots 2 \times (m-n+1)}{n!}$$

Qual das versões é mais eficiente?

Triângulo de Pascal

```
max=5
linha=0
while linha <= max:
    col=0
    while col <= linha:
        print "%4d" % binom(linha,col),
        col = col+1
    print
    linha = linha+1
```

Resultado

```
   1
   1    1
   1    2    1
   1    3    3    1
   1    4    6    4    1
   1    5   10   10    5    1
```

Dois modos de pensar

Programas recursivos

Modos de pensar/escrever/justificar sobre programas recursivos:

1. Indutivo: baseado no príncípio da indução matemática.
2. Operacional: como funciona de facto o computador?

Dois modos de pensar

Programas recursivos - notas

Notas:

1. Tudo se complica quando as funções não são puras.
2. É em geral fácil passar da definição matemática para o programa recursivo.
3. Muitas vezes as implementações iterativas são mais eficientes que as recursivas.
4. Muitas vezes não há versões iterativas simples.
5. Em certos casos há métodos de converter um programa recursivo num programa iterativo. Estas transformações podem ser efectuadas pelo compilador.