

Terminaison et validité d'un algorithme récursif

Alix Munier-Kordon et Maryse Pelletier

LIP6
Université P. et M. Curie
Paris

Module 2I003 Algorithmique Élémentaire

Plan du cours

- 1 Définitions, exemple à un seul appel
 - Définition
 - Petit exemple à un seul appel
- 2 La récursivité pas à pas
 - Pile des exécutions
 - Exécution d'une fonction récursive
 - Boucles et fonctions récursives
- 3 Terminaison et validité d'une fonction récursive
- 4 Recherche d'un élément dans un tableau non trié
- 5 Calcul de C_n^p

Définition

Definition

En *mathématiques*, une fonction récursive est une fonction qui est définie à partir d'elle même.

En *informatique*, une fonction est récursive lorsqu'elle peut s'appeler elle-même au cours de son exécution.



Exemple : la fonction factorielle (un seul appel)

$$\forall n \in \mathbb{N}, \text{fact}(n) = \begin{cases} 1 & \text{si } n = 0 \\ n \times \text{fact}(n-1) & \text{sinon} \end{cases}$$

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

Notion de Pile

Une *Pile* P est une structure de données gérée à l'aide des 3 méthodes suivantes :

- 1 $P.append(d)$
Place d en sommet de la pile P .
- 2 $P.pop()$
Dépile le sommet de la pile P et en renvoie la valeur.
- 3 $P.empty()$
Teste si la pile P est vide.

Contexte d'exécution d'une fonction

Pour pouvoir exécuter une fonction (récursive ou non), le code d'exécution doit avoir accès aux valeurs des paramètres de la fonction et des variables locales.

Que devient le contexte d'une fonction qui appelle une nouvelle fonction ?

```
def g(int a) :  
    i=5  
    f(i+2, 8)  
    ...
```

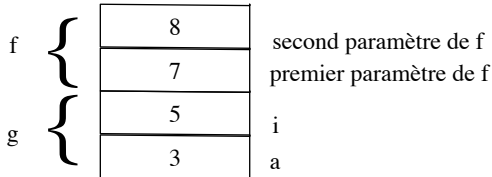
`g(3)`

Pile des exécutions

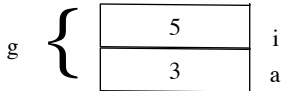
- 1 A chaque appel de fonction, le contexte d'exécution de la fonction appelée est empilé au sommet de la pile des exécutions.
- 2 Quand on termine l'exécution d'une fonction, son contexte d'exécution est dépilé de la pile des exécutions.
- 3 Le sommet de pile contient ainsi le contexte de la fonction en cours d'exécution.

Exemple

A l'appel de $f(i+2, 8)$:



A la fin de $f(i+2, 8)$,
retour à g:



Exemple de la factorielle

```
def fact(n) :  
    print "Appel de fact avec n =", n  
    if n==0 :  
        res=1  
    else :  
        res=n*fact(n-1)  
    print "Resultat pour n =", n, ":", res  
    return res
```

Exemple de la factorielle (suite)

Appel de fact(4)

Appel de fact avec $n = 4$

Appel de fact avec $n = 3$

Appel de fact avec $n = 2$

Appel de fact avec $n = 1$

Appel de fact avec $n = 0$

Resultat pour $n = 0$: 1

Resultat pour $n = 1$: 1

Resultat pour $n = 2$: 2

Resultat pour $n = 3$: 6

Resultat pour $n = 4$: 24

Boucles et fonctions récursives

D'une manière générale, toute fonction contenant un seul appel récursif peut être transformée en une fonction itérative.

```
def factorielleIt(n):  
    res = 1  
    i = 1  
    while i <= n :  
        res = res * i  
        i = i+1  
    return res
```

Terminaison et validité d'une fonction récursive

Terminaison Raisonnement par récurrence.

Validité Raisonnement par récurrence.

On peut souvent grouper les deux raisonnements.

Terminaison et validité de la fonction factorielle

Terminaison Montrer par récurrence sur n que $\forall n \geq 0$, $fact(n)$ se termine.

Validité Montrer par récurrence sur n que $\forall n \geq 0$, $fact(n)$ vaut $n!$.

Terminaison et validité de la fonction factorielle

$\mathcal{P}(k)$, $k \geq 0$: `fact(k)` se termine et retourne $k!$

Par récurrence faible :

Base Pour $n = 0$, `fact(0)` retourne $1 = 0!$.

Donc $\mathcal{P}(0)$ est vérifiée.

Induction Supposons que $\mathcal{P}(k - 1)$ soit vérifiée pour une valeur $k \geq 1$ fixée. `fact(k)` retourne $k \times \text{fact}(k-1)$. Par hypothèse de récurrence, `fact(k-1)` se termine et retourne $(k - 1)!$. Donc, `fact(k)` se termine et retourne $k \times (k - 1)! = k!$. Donc $\mathcal{P}(k)$ est vérifiée.

Conclusion Comme $\mathcal{P}(0)$ est vérifiée, et que, pour tout $k \in \mathbb{N}$, $\mathcal{P}(k) \Rightarrow \mathcal{P}(k + 1)$, on en déduit que $\forall k \in \mathbb{N}$, $\mathcal{P}(k)$ est vérifiée.

Recherche d'un élément dans un tableau non trié

```
def RechercheRec(elem, tab, n) :  
    if n==0 :  
        return -1  
    if tab[n-1]==elem :  
        return n-1  
    return RechercheRec(elem, tab, n-1)
```

```
»> tab=[4,8,3,1,9,8,7]  
»> RechercheRec(8,tab,7)  
5  
»> RechercheRec(5,tab,7)  
-1
```

Terminaison et validité de la fonction RechercheRec

Soit *tab* un tableau d'entiers de taille supérieure ou égale à *n*,
et *elem* un entier.

Terminaison Montrer par récurrence sur *n* que $\forall n \geq 0$,
RechercheRec(*elem*, *tab*, *n*) se termine.

Validité Montrer par récurrence sur *n* que $\forall n \geq 0$,
RechercheRec(*elem*, *tab*, *n*) renvoie

- l'indice maximum $i \in \{0, \dots, n-1\}$ tel que
 $tab[i] = elem$ si $elem \in tab[0 \dots n-1]$,
- -1 sinon

Calcul des C_n^k pour $n \geq 1$ et $k \in \{1, \dots, n\}$

Basée sur le triangle de Pascal :

$$C_n^k = C_{n-1}^k + C_{n-1}^{k-1} \quad \text{pour } 1 \leq k \leq n-1$$

```
def Comb(n, k) :  
    if (k==n) :  
        return 1  
    if (k==1) :  
        return n  
    return Comb(n-1, k) + Comb(n-1, k-1)
```

Terminaison et validité de la fonction Comb

Terminaison Montrer par récurrence sur n que $\forall n \geq 1$,
 $\forall k \in \{1, \dots, n\}$, Comb(n, k) se termine.

Validité Montrer par récurrence sur n que $\forall n \geq 1$,
 $\forall k \in \{1, \dots, n\}$, Comb(n, k) retourne C_n^k .

Exécutions de Comb

```
def Comb(n,k) :  
    print('Appel avec n=', n, 'et k=', k)  
    if (k==n) :  
        res=1  
    elif (k==1) :  
        res=n  
    else:  
        res = Comb(n-1,k)+Comb(n-1,k-1)  
    print 'C(' , n, ', ' , k , ') = ' , res  
    return res
```

Exécutions de Comb

```
»> Comb(4,2)
('Appel avec n=', 4, 'et k=', 2)
('Appel avec n=', 3, 'et k=', 2)
('Appel avec n=', 2, 'et k=', 2)
C( 2 , 2 )= 1
('Appel avec n=', 2, 'et k=', 1)
C( 2 , 1 )= 2
C( 3 , 2 )= 3
('Appel avec n=', 3, 'et k=', 1)
C( 3 , 1 )= 3
C( 4 , 2 )= 6
6
```

Arbre des exécutions de $\text{Comb}(4, 2)$

