Terminaison et validité d'un algorithme récursif

Alix Munier-Kordon et Maryse Pelletier

LIP6
Sorbonne Université
Paris

Module 2I003 Algorithmique Elémentaire

Plan du cours

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)

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

```
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)
```

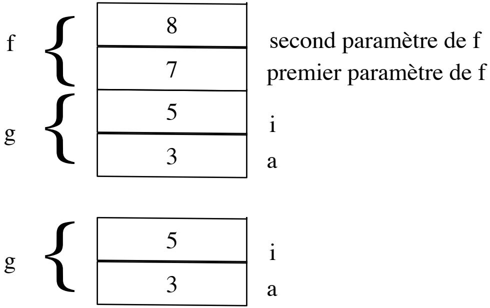
Pile des exécutions

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
```

Terminaison et validité d'une fonction récursive $f(x)$

Soit $\mathcal{P}$ l'ensemble des valeurs du paramètre x de f

Terminaison Démontrer par récurrence que $\forall x \in \mathcal{P}$, $f(x)$ se termine.

Validité Démontrer par récurrence que $\forall x \in \mathcal{P}$, $f(x)$ est valide.

On peut 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 $k \times$`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 $i \in \{0, \cdots, n-1\}$ tel que $tab[i] = elem$ si $elem \in tab[0 \cdots n-1]$,
- -1 sinon

Combinaisons

Definition

Une *combinaison* d'ordre k d'un ensemble E à n éléments est un sous-ensemble de E ayant k éléments.

Remarque : dans une combinaison, il n'y a pas de répétition et l'ordre n'a pas d'importance.

Par exemple, dans l'ensemble $\{1, 2, 3, 4, 5\}$:

- $\{2, 3, 5\}$ est une combinaison d'ordre 3
- $\{5, 2, 3\}$ est la même combinaison
- $\{5, 2, 5\}$ n'est pas une combinaison.

Calcul du nombre de combinaisons

Theorem

Soit $n \geq 0$ et $k \in \{0, \ldots, n\}$.
Le nombre de combinaisons d'ordre k d'un ensemble E à n éléments est noté C_n^k et vérifie $C_n^k = \frac{n!}{k!(n-k)!}$.

Par exemple il y a 10 combinaisons d'ordre 3 dans l'ensemble $\{1, 2, 3, 4, 5\}$, autrement dit $C_5^3 = 10$.

Triangle de Pascal

Theorem (Triangle de Pascal)

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

C_n^k	k=0	k=1	k=2	k=3	k=4	k=5
n=0	1					
n=1	1	1				
n=2	1	2	1			
n=3	1	3	3	1		
n=4	1	4	6	4	1	
n=5	1	5	10	10	5	1

Calcul des C_n^k pour $n \geq 0$ et $k \in \{0, \cdots, n\}$

```
def Comb(n,k):
    if (k==n) or (k==0):
        return 1
    return Comb(n-1,k)+Comb(n-1,k-1)

>>> Comb(5,5)
1
>>> Comb(4,2)
6
>>> Comb(5,3)
10
```

Terminaison et validité de la fonction `Comb`

Terminaison Montrer par récurrence sur n que $\forall n \geq 0$, $\forall k \in \{0, \cdots, n\}$, `Comb(n,k)` se termine.

Validité Montrer par récurrence sur n que $\forall n \geq 0$, $\forall k \in \{0, \cdots, n\}$, `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) or (k==0):
            res=1
    else:
            res = Comb(n-1,k)+Comb(n-1,k-1)
    print('C(' , n, ',' , k , ')=␣' , res)
    return res
```

Exécutions de `Comb(3,2)`

```
>>> Comb(3,2)
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
Appel avec n= 1 et k= 1
C( 1 , 1 )=  1
Appel avec n= 1 et k= 0
C( 1 , 0 )=  1
C( 2 , 1 )=  2
C( 3 , 2 )=  3
3
```

Arbre des exécutions de `Comb(3,2)`