Chapitre VI

Récursivité 2 : quelques approfondissements

Figure VI.1 – Un exemple de peinture "récursive"

Sommaire

1 Récursivité terminale 2

1.1 Définition 2

1.2 Structure générale d'un algorithme récursif terminal 2

1.3 Premier exemple : factorielle récursive terminale 3

1.4 Quelques exemples 4

a - Suite de Fibonacci récursive terminale 4

b - PGCD récursif terminal 4

c - Un tri récursif terminal 5

1.5 En résumé 5

2 Dérécursivation 6

2.1 Principe et intérêt 6

2.2 Exemples de mise en oeuvre de la dérécursivation 6

a - Factorielle 6

b - Suite de Fibonacci 6

c - PGCD 7

1 Récursivité terminale

1.1 Définition

Définition - (1.1) - 1:

Une fonction récursive est dite **terminale** *si elle renvoie directement la valeur obtenue par son appel récursif. Ainsi, dans une fonction récursive terminale, la valeur renvoyée par l'appel récursif est directement la valeur obtenue par celui-ci sans autre calcul, et donc* **sans remontée de calcul dans la pile d'exécution**.

Avantages :

- Absence de toute remontée dans une pile d'exécution $\Longrightarrow$ moindre complexité spatiale, en particulier si le nombre de récursions est important.
- Certains langages compilateurs détectent les structures récursives terminales et optimisent l'occupation mémoire $\Longrightarrow$ aucun "surcoût machine" par rapport à la version itérative (ce peut être le cas de Python qui permet aussi de réaliser des programmes compilés).
- L'écriture d'une structure récursive terminale est souvent plus claire et très facile à interpréter car proche d'une structure itérative.
- Les fonctions récursives terminales sont très facilement transformables en leurs homologues itératives : **c'est la dérécursivation.** **(cf 2)**

1.2 Structure générale d'un algorithme récursif terminal

Les structures de fonctions suivantes ne sont pas récursives terminales :

Listing VI.1 –

```
Définition f(p)
    si condition(P)     #Condition sur les paramètres P pour entrer dans la récursion
    alors
        Traitement_1(P) #Traitement 1 des paramètres d'appel
        f(g(P))         #Appel à la fonction récursive f avec les paramètres g(P) modifiés par la fonction g
        Traitement_2(P) #Traitement 2 des paramètres d'appel initial (avant l'appel récursif juste au dessus)
    sinon
        B(P)            #Traitement du cas de base
```

car l'appel récursif n'est pas le dernier traitement du cas récursif engagé. Il faudra en effet une remontée complète de la pile pour évaluer `Traitement_2(P)`.

Listing VI.2 –

```
Définition f(p)
    si condition(P)     #Condition sur les paramètres P pour entrer dans la récursion
    alors
        Traitement_1(P) #Traitement 1 des paramètres d'appel
        P*f(g(P))       #Appel à la fonction récursive f avec les paramètres g(P) modifiés par la fonction g et multiplication par paramètre

    sinon
        B(P)            #Traitement du cas de base
```

car l'appel récursif est impliqué dans un calcul avec les paramètres (multiplication). Il faudra en effet une remontée complète de la pile pour évaluer la valeur invoquée initialement.

La structure d'une fonction récursive terminale est la suivante :

Listing VI.3 –

```
Définition f(p)
    si condition(P)   #Condition sur les paramètres P pour entrer dans la récursion
    alors
        Traitement(P)   #Traitement des paramètres d'appel
        f(g(P))         #Appel à la fonction récursive f avec les paramètres g(P)
    modifiés par la fonction g
    sinon
        B(P)            #Traitement du cas de base
```

Dans cette fonction, l'appel récursif n'est pas inclu dans un traitement, mais est la dernière étape du cas récursif et renvoie directement la valeur invoquée sans calcul. La fonction `f(p)` est donc récursive terminale.

1.3 Premier exemple : factorielle récursive terminale

Rappelons la version récursive classique de la factorielle :

Listing VI.4 – Fonction factorielle en récursif

```python
def fact(n):
    if n==1: #cas de base!!!
        return 1
    else:
        return n*(fact(n-1)) # Appel récursif non terminal
```

Dans cette fonction, l'appel récursif comporte un traitement, la multiplication par n, qui nécessitera un stockage temporaire dans la pile d'exécution.

<u>IDÉE :</u> on souhaite bâtir une **version récursive terminale de la factorielle**.

La solution consiste à stocker dans une variable supplémentaire (qui viendra donc enrichir la liste des paramètres de l'appel, et donc n n'est plus seul!!) le résultat du calcul $\times n$. On bâtit pour cela la fonction `facRTerm(m,n)` suivante :

$$facRTerm(m,n) = \begin{cases} m & \text{si } n = 1 \\ facRTerm(m \times n, n-1) & \text{sinon} \end{cases}$$

Décomposons l'appel $facRterm(m,n)$:

$$\begin{aligned} facRTerm(m,n) &= facRterm(m \times n, n-1) \\ &= facRterm(m \times n \times (n-1), n-2) \\ &\dots \\ &= facRterm(m \times \underbrace{n \times (n-1) \dots \times 2}_{=n!}, 1) \\ &= m \times n! \end{aligned}$$

Par conséquent, on calcule la factorielle de n en appelant `facRTerm(1,n)` $= n!$.

EXERCICE N°1: STRUCTURE DES APPELS

❶ *Ecrire les appels récursifs successifs lorsque l'on calcule par exemple* `facRterm(1,4)`.

❷ *Conclure.*

Pour éviter d'invoquer un appel de fonction contenant le paramètre supplémentaire $m = 1$, il suffit d'imbriquer la fonction `FacRterm` dans une fonction à appeler ne comportant que n comme argument :

Listing VI.5 – Fonction factorielle récursive terminale

```python
def fact(n):
    def facRterm(m,n):
        if n==1:
            return m
        else:
            return facRterm(m*n,n-1)
    return facRterm(1,n)
```

1.4 Quelques exemples

a - Suite de Fibonacci récursive terminale

On rappelle la définition de la suite de Fibonacci de premiers termes $(0, 1)$:

$$u_0 = 0, u_1 = 1, \qquad u_{n+2} = u_{n+1} + u_n$$

et le script de la fonction récursive non terminale correspondante :

Listing VI.6 –

```python
def fiborec(n):
    #algorithme récursif
    if n==1 or n==2 :
        return 1
    return fiborec(n-1)+fiborec(n-2)
```

Exercice n°2: Suite de Fibonacci

❶ *Bâtir une fonction récursive terminale* `fiboRterm(a,b,R,n)` *de calcul de la suite de Fibonacci de premiers termes* (a, b) *et qui renvoie le* $n^{ième}$ *terme de la suite si l'on appelle* `fiboRterm(a,b,1,n)`.

❷ *Détailler les appels successifs à la fonction récursive terminale lors du calcul de* `fiboRterm(a,b,1,n)`, *par exemple en analysant l'appel* `fiboRterm(0,1,1,4)`.

❸ *Comparer le fonctionnement de cette fonction avec celui de la version récursive non terminale. On pourra établir l'arbre des appels de la fonction récursive non terminale afin de conclure, par exemple en analysant là-encore l'appel* `fiborec(4)`

b - PGCD récursif terminal

On rappelle l'algorithme d'Euclide de calcul du PGCD :

- on calcule le reste de $a \mathbin{/\!/} b$ que l'on stocke dans r.
- tant que $a\%b \neq 0$ faire : $\begin{cases} r = a\%b \\ a = b \\ b = r \end{cases}$
- On renvoie b.

Le script récursif terminal est donc :

Listing VI.7 – PGCD récursif terminal

```
def pgcdRterm(a,b):
    if a%b==0:
        return b
    else:
        return pgcdRterm(b,a%b)
```

On constate que l'algorithme d'Euclide est "par essence" **récursif terminal** puisque l'appel récursif, d'une part n'est pas engagé dans un calcul stocké dans la pile d'exécution, mais se veut ici direct, et d'autre part l'appel récusif est la dernière évaluation dans l'algorithme.

c - Un tri récursif terminal

On proposait en exercice n°3 du TD2 l'analyse d'un algorithme de tri par permutation (appelé ici "trinaïf") dans sa version itérative :

Listing VI.8 –

```
def trinaif(L):
    n = len(L)
    for k in range(n-1):
        p, m = max(L, k)
        L[k], L[p] = L[p], L[k]
    return None
```

avec la fonction `max(L,deb)` qui renvoie la position (indice) et la valeur du maximum dans la sous-liste de la liste `L` qui commence à `L[deb]` :

Listing VI.9 –

```
def max(L,deb):
    max=L[deb]
    ind=0
    for p in range(deb,len(L)):
        if L[p]>max:
            max=L[p]
            ind=p
    return p,max
```

Exercice n°3: Version récursive terminale

Proposer une version **récursive terminale** *de cet algorithme de tri.*

1.5 En résumé

Pour élaborer une fonction récursive terminale, il faut :

- au moins un paramètre de la fonction convergent vers le ou les cas de base.
- un paramètre permettant le stockage des opérations normalement en attente dans une pile d'exécution d'une fonction récursive non terminale.
- et évidemment comme toujours la présence d'un ou plusieurs cas de base permettant l'arrêt de la récursivité.

2 Dérécursivation

2.1 Principe et intérêt

- On peut toujours théoriquement transformer une fonction itérative en fonction récursive et vice-versa. Ce n'est parfois pas une bonne idée lorsque la complexité de l'algorithme explose en version récursive[1].

- Toute fonction **récursive non nécessairement terminale** peut être transformée en une fonction itérative, mais parfois au détriment de la complexité.

- Toute fonction récursive terminale est très facilement transformable en son équivalent itératif, et ce en raison du caractère quasi-itératif de la récursivité terminale. Pour ainsi dire, une fonction récursive terminale est simplement une **itération sans commande de boucle**.

DÉFINITION - (2.1) - 2:

Le procédé de transformation d'une fonction récursive en son équivalent itératif porte le nom de dérécursivation.

2.2 Exemples de mise en oeuvre de la dérécursivation

a - Factorielle

Listing VI.10 –

```python
def facRterm(m,n):
        if n=1:
            return m
        else:
            return facRterm(m*n,n-1)
```

Listing VI.11 –

```python
def faciter(n):
    m=n
    while n!=1:
        m=m*(n-1)
        n=n-1
    return m
```

b - Suite de Fibonacci

Listing VI.12 –

```python
def fiboRterm(a,b,R,n):
    if n==0:
        return a
    else:
        return fiboRterm(b,a+b,R,n-R)
```

Listing VI.13 –

```python
def fiboiter(a,b,n):
    u0=a
    u1=b
    p=0
    while p!=n:
        u0,u1=u1,u0+u1
        p=p+1
    return u0
```

1. c'était par exemple le cas du calcul récursif non terminal de la suite de Fibonacci.

c - PGCD

Listing VI.14 –

```python
def pgcdRterm(a,b):
    if a%b==0:
        return b
    else:
        return pgcdRterm(b,a%b)
```

Listing VI.15 –

```python
def pgcd(a,b):
    while a%b!=0:
        a,b=b,a%b
    return b
```