Cours 7 – Récursivité

I Introduction

Exemple : Fonction factorielle

Commençons par écrire une version itérative de la fonction factorielle, *i.e.* une fonction définie uniquement à l'aide de boucles for ou while, qui ne s'appelle pas elle-même.

```
def factorielle(n):
    P=1
    for k in range(1,n+1):
        P=P*k
    return P
```

Maintenant, en remarquant que $n! = n*(n-1)!$, écrivons une version récursive de la fonction factorielle, *i.e.* une fonction faisant appel à elle-même

```
def factorielle_rec(n):
    #cas de base : 0!=1
    if n==0:
        return 1

    #"hérédité"
    return n*factorielle_rec(n-1).
```

I.1 Conception d'une fonction récursive

Exemple : Algorithme naïf d'exponentiation

On cherche à calculer x^n à l'aide d'un algorithme récursif simple, ne comportant que des multiplications.

Le cas de base est : $\boxed{x^0 = 1}$

Pour obtenir x^n à partir de x^{n-1}, il suffit de faire : $\boxed{x^n = x * (x^{n-1})}$

On résume dans la fonction suivante :

```
def puissance(x,n):
    if n==0:
        return 1
    return x*puissance(x,n-1)
```

II Exemples de fonctions récursives

Exemple 1 : Suite de Fibonacci

Ecrire une fonction récursive `fibo(n)` qui calcule le n-ième terme de la suite de Fibonacci définie par $\begin{cases} F_0 = F_1 = 1 \\ \forall n \in \mathbb{N}, F_{n+2} = F_n + F_{n+1} \end{cases}$

```
def fibo(n):
    if n==0 or n==1:
        return 1
    return fibo(n-2)+fibo(n-1)
```

Exemple 2 : Exponentiations rapides

Ecrire deux fonctions récursives d'exponentation rapide (permettant de calculer x^n sans utiliser $**$)

La première version doit reposer sur les identités suivantes : $\begin{cases} x^{2k} = x^k \times x^k \\ x^{2k+1} = x \times x^k \times x^k \end{cases}$

```
def expo_rapide_1(x,n):
    #x^0=1
    if n==0:
        return 1
    #Cas n pair, k=n/2
    if n%2==0:
        y=expo_rapide_1(x,n/2)
        return y*y
    #cas n impair, k=(n-1)/2
    else:
        y=expo_rapide_1(x,(n-1)/2)
        return x*y*y
```

La seconde version doit reposer sur les identités suivantes : $\begin{cases} x^{2k} = (x \times x)^k \\ x^{2k+1} = x \times (x \times x)^k \end{cases}$

```
def expo_rapide_2(x,n):
    #x^0=1
    if n==0:
        return 1
    #Cas n pair, k=n/2
    if n%2==0:
        return expo_rapide_2(x*x,n/2)
    #cas n impair, k=(n-1)/2
    else:
        return x*expo_rapide_2(x*x,(n-1)/2)
```

III Complexité

III.1 Complexité et suites récurrentes

Que peut-on dire de la complexité asymptotique d'un algorithme dont la complexité $C(n)$ suit une relation de récurrence du type :

1. $C(n) = C(n-1) + R$: Complexité linéaire
2. $C(n) = qC(n-1)$: Complexité exponentielle
3. $C(n) = aC(n-1) + b$: Complexité exponentielle
4. $C(n) = aC(n-1) + bC(n-2)$: Complexité exponentielle
5. $C(n) = C(n/2) + k$: Complexité logarithmique

III.2 Analyse de complexité de fonctions récursives

III.2.1 Un premier exemple

On considère la suite récurrente définie par $\begin{cases} u_0 = 2 \\ \forall n \in \mathbb{N}^*, u_n = \frac{1}{2}(u_{n-1} + \frac{3}{u_{n-1}}) \end{cases}$ qui converge vers $\sqrt{3}$ et permet donc d'en obtenir une approximation numérique.

Ecrire une fonction récursive permettant d'obtenir u_n puis estimer sa complexité en fonction de n.

```
def u(n):
    if n==0:
        return 2
    y=u(n-1)
    return (y+3/y)/2
```

On a $C(n) = C(n-1) + k$. La complexité est linéaire

III.2.2 Itératif ou récursif ? Exemple 1 : Factorielle

On effectue n multiplications dans les deux fonctions. La complexité calculatoire est identique. (En revanche, la complexité spatiale est linéaire pour la version récursive, et constante pour la version itérative)

III.2.3 Itératif ou récursif ? Exemple 2 : Suite de Fibonacci

La complexité pour la fonction récursive est exponentielle.

On peut obtenir une complexité linéaire avec une version itérative. Par exemple :

```
def fibo_iter(n):
    #Fn et Fn+1
    fn=1
    fn1=1
    for k in range(1,n+1):
        x=fn+fn1 #Fn+2
        fn=fn1
        fn1=x
    return fn
```