Informatique X
Initiation à la programmation récursive

Laurent Kaczmarek

PCSI^2 2013-2014
Lycée Louis Le Grand

Lundi 6 janvier 2014

Les fonctions récursives

Définition et premier exemple

- Une fonction récursive est une fonction qui fait appel à elle-même. Très souvent un algorithme récursif est lié à une relation de récurrence permettant de calculer la valeur d'une fonction pour un argument n à l'aide des valeurs de cette fonction pour des arguments strictement inférieurs à n.
- Exemple : calcul par récurrence de x^n pour $x \in \mathbb{R}$ et $n \in \mathbb{N}$.
- Le programme en `Python` s'écrit:

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

Suite de l'exemple

- La machine applique la règle suivante : `puissance(x,n)=x` $\times$ `puissance(x,n-1)` tant que l'exposant est différent de 0, ce qui introduit des calculs intermédiaires jusqu'à aboutir au cas de base : `puissance(x,0)`$= 1$. Les calculs en suspens sont alors achevés dans l'ordre inverse jusqu'à obtenir le résultat final.
- Comme pour les boucles avec conditions d'arrêt, il faut s'assurer que le cas de base sera atteint en un nombre fini d'étapes sinon l'algorithme "boucle" indéfiniment.
- Pour la fonction `puissance` précédente la terminaison est garantie si n est entier positif ou nul, et il y a bouclage si $n < 0$.

De récursif en itératif

- Dans ce cas particulier, il est facile de transformer l'algorithme récursif en algorithme itératif :

```
>>> def puissanceIter(x,n):
        p=1
        while n!=0:
            p=x*p
            n=n-1
        return(p)
```

- De façon générale, les fonctions vérifiant une relation de récurrence de la forme $f(0) = f_0$ et, pour $n > 0$, $f(n) = g(n, f(n-1))$ se prêtent aussi facilement à un codage itératif que récursif.

Terminaison et correction d'une fonction récursive

Raisonnement par récurrence

- On prouve le plus souvent conjointement la correction et la terminaison d'une fonction récursive au moyen d'un raisonnement par récurrence.
- Pour $n \in \mathbb{N}$, notons $HR(n)$: l'appel `puissance(x,n)` termine et renvoie x^n pour tout $x \in \mathbb{N}$.
- $HR(0)$ est clairement vraie (cf. le programme ci-dessus);
- Soient $n \in \mathbb{N}$ et $x \in \mathbb{N}$. Supposons $HR(n)$ vraie. Considérons `puissance(x,n+1)`. Comme $n+1 \neq 0$, la fonction commence par un appel de `puissance(x,n)`. Par $HR(n)$, cet appel termine et renvoie x^n. On en déduit que `puissance(x,n+1)` termine et renvoie $xx^n = x^{n+1}$. Ainsi $HR(n+1)$ est vraie.

Complexité d'une fonction récursive

On peut chercher à calculer :

- le nombre d'opérations effectuées lors d'un appel, ie la complexité de la fonction;
- le nombre d'appels récursifs (qui sera du même orde de grandeur que la complexité si chaque appel donne lieu à un nombre d'opérations ne dépendant pas de n).

L' exemple d'une suite récurrente

- $u_0 = 1, \quad u_{n+1} = \frac{1}{2}\left(u_n + \frac{2}{u_n}\right).$
-
```
>>> def racineDeDeux(n):
        if n=0:
           return(1.)
        else:
           x=racineDeDeux(n-1)
           return(0.5*(x+2./x))
```

Complexité d'une fonction récursive

L' exemple d'une suite récurrente

- Notons $C(n)$ le nombres d'opérations effectuées lors de l'appel de `racineDeDeux(n)`.
- On a $C(0) = 1$ et $C(n+1) = 4 + C(n)$.
- On en déduit que $C(n) = 4n + 1$ pour tout n.

Attention !

- Que pensez-vous de la fonction suivante ?

```
>>> def bis(n):
        if n=0:
            return(1.)
        else:
            return(0.5*(bis(n-1)+2./bis(n-1)))
```

- La complexité explose : $C(n+1) = 4 + 2C(n)$.

Exercices

Exercice 1 : la factorielle

Programmer de manière récursive le calcul de la fonction factorielle.

Exercice 2 : la suite de Fibonacci

Programmer de manière récursive le calcul de la suite de Fibonacci.

Exercice 3 : la division euclidienne

En utilsant l'algorithme des soustractions successives, écrire une fonction récursive `modulo(a,b)` qui retourne le reste de la division euclidienne de `a` par `b`.

Exercice 4 : le codage binaire

Écrire une fonction récursive qui calcule la décomposition binaire de tout nombre entier n.

Exercices

Exercice 5 : l'algorithme d'Euclide

En utilisant l'algorithme d'Euclide, écrire une fonction récursive `pgcd(a,b)` qui retourne le plus grand commun diviseur de deux entiers naturels `a` et `b`.

Exercice 6 : l'exponentiation rapide

Étant donnés un réel positif a et un entier n, on remarque que

$$a^n = \begin{cases} (a^n)^2 & \text{si } n \text{ est pair} \\ a\left(a^{\frac{n-1}{2}}\right)^2 & \text{si } n \text{ est impair.} \end{cases}$$

En déduire une fonction récursive calculant a^n. Comparer le nombre d'opérations effectuées par cet algorithme par rapport à l'algorithme naïf.