Informatique II (J1CP3020)

Philippe Duchon

U. Bordeaux 1

2012-13

Programme

- Programmes et fonctions **récursifs**

Programme

- Programmes et fonctions **récursifs**
- Des récurrences pour les complexités d'algorithmes

Programme

- Programmes et fonctions **récursifs**
- Des récurrences pour les complexités d'algorithmes
- Des algorithmes un peu plus performants pour le tri

Un exemple bateau

- Fonction factorielle : $n! = 1.2.\ldots.n$

Un exemple bateau

- Fonction factorielle : $n! = 1.2.\ldots.n$
- Définition classique : la factorielle d'un entier positif ou nul n, c'est (par définition) le produit de tous les entiers compris (au sens large) entre 1 et n.

Un exemple bateau

- Fonction factorielle : $n! = 1.2.\ldots.n$
- Définition classique : la factorielle d'un entier positif ou nul n, c'est (par définition) le produit de tous les entiers compris (au sens large) entre 1 et n.
- Définition par récurrence : la factorielle d'un entier positif ou nul n, notée $n!$, c'est
  - 1, si $n = 0$
  - $(n-1)!.n$, si $n > 0$

Un exemple bateau

- Fonction factorielle : $n! = 1.2.\ldots.n$
- Définition classique : la factorielle d'un entier positif ou nul n, c'est (par définition) le produit de tous les entiers compris (au sens large) entre 1 et n.
- Définition par récurrence : la factorielle d'un entier positif ou nul n, notée $n!$, c'est
  - 1, si $n = 0$
  - $(n-1)!.n$, si $n > 0$

La fonction (ou suite) est définie par récurrence, un terme en fonction des précédents.

La factorielle en Python


```
def facto(n):
    res = 1
    for i in range(n):   // 0,1,...,n-1
        res = res * (i+1) // i+1, pas i
    return(res)
```

Fonctions récursives

- En informatique, **une fonction est récursive si un appel à la fonction peut, soit directement soit indirectement, entraîner un ou plusieurs autres appels à la fonction elle-même.**

Fonctions récursives

- En informatique, **une fonction est récursive si un appel à la fonction peut, soit directement soit indirectement, entraîner un ou plusieurs autres appels à la fonction elle-même.**
- La plupart des langages de programmation autorisent l'écriture de fonctions récursives

Fonctions récursives

- En informatique, **une fonction est récursive si un appel à la fonction peut, soit directement soit indirectement, entraîner un ou plusieurs autres appels à la fonction elle-même.**
- La plupart des langages de programmation autorisent l'écriture de fonctions récursives
- Exemple :

```
def facto(n):
    if (n==0):
        return(1)
    return(n*facto(n-1))
```

Fonctions mutuellement récursives


```
def f(n):
    if (n<=1):
        return(n)
    return(g(n)+g(n-1))

def g(n):
    return(f(n-1)*f(n-2))
```

Avantages. . .

- Du code qui “colle” parfaitement à des définitions récursives

Avantages. . .

- Du code qui “colle” parfaitement à des définitions récursives
- Souvent, on évite l’écriture de boucles : moins de problèmes d’initialisation de variables, ou d’indices de boucles faux

Avantages. . .

- Du code qui “colle” parfaitement à des définitions récursives
- Souvent, on évite l’écriture de boucles : moins de problèmes d’initialisation de variables, ou d’indices de boucles faux
- Cela correspond à l’état d’esprit de l’informaticien, qui aime les définitions “inductives”

Dangers...

- Il est facile d'écrire des programmes **qui ne se terminent jamais**

Dangers...

- Il est facile d'écrire des programmes **qui ne se terminent jamais**
- Si on ne fait pas attention, on peut écrire des programmes **très peu efficaces** (qui s'exécutent très lentement)

Le problème de la terminaison

- Une fonction récursive **doit toujours** avoir **au moins un cas “terminal”** (qui n’entraîne pas d’appel récursif)

Le problème de la terminaison

- Une fonction récursive **doit toujours** avoir **au moins un cas "terminal"** (qui n'entraîne pas d'appel récursif)
- Le langage **ne résoud pas d'équations :** si $f(x)$ est défini en fonction de $f(y)$, alors $f(y)$ ne doit pas être défini en fonction de $f(x)$

Le problème de la terminaison

- Une fonction récursive **doit toujours** avoir **au moins un cas "terminal"** (qui n'entraîne pas d'appel récursif)
- Le langage **ne résoud pas d'équations :** si $f(x)$ est défini en fonction de $f(y)$, alors $f(y)$ ne doit pas être défini en fonction de $f(x)$
- Souvent (mais pas toujours), la fonction est appelée récursivement avec un paramètre (entier, positif ou nul) **strictement inférieur**, et le cas terminal correspond à un paramètre 0 ou 1.

Complexité d'une fonction récursive

- Pour évaluer la complexité d'une fonction récursive, il faut tenir compte de la complexité des appels récursifs

Complexité d'une fonction récursive

- Pour évaluer la complexité d'une fonction récursive, il faut tenir compte de la complexité des appels récursifs
- Typiquement, cela se traduit par une **définition par récurrence** de la complexité

Complexité d'une fonction récursive

- Pour évaluer la complexité d'une fonction récursive, il faut tenir compte de la complexité des appels récursifs
- Typiquement, cela se traduit par une **définition par récurrence** de la complexité
- On écrit la récurrence, et on la résoud soit exactement, soit **de manière approchée**

Puissance

On peut définir de deux manières, mathématiquement, la fonction puissance $p(x, n) = x^n$ ($x \in \mathbb{R}$, $n \in \mathbb{N}$) :

$$
\begin{aligned}
p(x,n) &= 1 \text{ si } n = 0 \\
p(x,n) &= x.p(x, n-1) \text{ si } n \geq 1 \\
q(x,n) &= 1 \text{ si } n = 0 \\
q(x,n) &= (q(x, n/2))^2 \text{ si } n > 0,\ n \text{ pair} \\
q(x,n) &= x.(q(x, (n-1)/2))^2 \text{ si } n \text{ est impair}
\end{aligned}
$$

Exercice. . .

- **Prouver** que pour tout $n \geq 0$ et tout $x \in \mathbb{R}$, $p(x, n) = q(x, n) = x^n$
- Écrire **deux fonctions** `Python`, traductions récursives (les plus simples possibles) des deux définitions.
- Notons $C_p(n)$ et $C_q(n)$, respectivement, les nombres de multiplications entraînées par l'appel $p(x, n)$ (resp. $q(x, n)$). **Donner une définition par récurrence de $C_p(n)$ et de $C_q(n)$.**
- **Résoudre** la récurrence pour $C_p(n)$.