

Cours d'algorithmique Récursivité - EISTI - ING 1

Ecole Internationale des Sciences du Traitement de l'Information

Généralités

- ▶ Idée très simple : une fonction peut s'appeler elle-même exactement comme un appel d'une autre fonction : dès l'appel récursif achevé l'exécution du programme continue (et sans effet sur les variables et paramètres de l'appel initial)
- ▶ Exemples favoris : factoriel, Fibonacci
- ▶ Valable pour fonctions et procédures
- ▶ Récursion mutuelle : deux ou plusieurs fonctions s'appellent entre elle (donc appel avant déclaration pour quelques unes)

Mauvais exemples classiques

- ▶ la fonction récursive est une méthode bizarre de calculer factoriel (une boucle simple suffit)
- ▶ la fonction récursive est une très mauvaise méthode de calculer fibonacci (le temps de calcul est exponentiel !)
- ▶ même si une méthode récursive est plus élégante et claire, on doit aussi considérer l'efficacité

Exemple de factoriel

```
fonction factrec(E n: entier): entier  
  si n=1 ou n=0 alors retourner 1  
  sinon retourner n*factrec(n-1)  
  fsi  
finfonction
```

```
fonction factit(E n: entier): entier  
variables  
  res: entier  
  i: entier  
  res ← 1  
  pour i ← 2 a n pas 1  
    res ← res*i  
  fpour  
  retourner res  
finfonction
```

Exemple de fibonnaci

```
fonction fibo(E n:entier):entier
  si n<2 alors retourner 1
  sinon retourner fibo(n-1)+fibo(n-2)
  fsi
finfonction
```

Complexité

On constate ici que cette version réursive est très coûteuse.

Par exemple l'appel $fibonacci(4)$ va calculer 2 fois $fibonacci(2)$:

- ▶ $fibonacci(4) = fibonacci(3) + fibonacci(2) =$
 $fibonacci(2) + fibonacci(1) + fibonacci(2) = \dots$
- ▶ Complexité exponentielle !

Meilleurs exemples

- ▶ x^n par calcul de carrés : on saute de x^i à x^{2i} en une seule multiplication ; plus difficile à gérer avec une boucle
- ▶ Les tours de Hanoï : déplacer une tour de n disques de taille différente d'une colonne à une autre en utilisant une seule colonne auxiliaire selon la règle qu'on ne peut déplacer qu'un disque à la fois et chaque colonne a toujours ses disques en ordre décroissante de taille
- ▶ Affichage en binaire (une boucle simple affiche les chiffres dans l'ordre inversé)
- ▶ Boucles imbriquées de profondeur inconnue
- ▶ Algorithmes "diviser pour régner"

Exemple du calcul de x^n

```
fonction powerrec1(E x:entier ,E n:
entier):entier
si n=0 alors
retourner 1
sinon retourner x*powerrec1(x,n-1)
fsi
finfonction
```

Récurréce : $x^n = x * x^{n-1}$
Complexité : $O(n)$
Exemple : powerrec1(5,4)=
5*powerrec1(5,3)=
5*5*powerrec1(5,2)=
5*5*5*powerrec1(5,1)=
5*5*5*5*powerrec1(5,0)=
5*5*5*5*1 : 5 multiplications

```
fonction powerrec2(E x:entier ,E n:
entier):entier
si n=0 alors
retourner 1
sinon si n mod 2 =0 alors
retourner sqr(powerrec2(x,n/2))
sinon
retourner x*sqr(powerrec2(x,n/2))
fsi
finfonction
```

Récurréce : $x^{2i} = (x^i)^2$ et $x^{2i+1} = x * (x^{2i})$
Complexité : $O(\log_2(n))$
Exemple : powerrec2(5,4)=
sqr(powerrec2(5,2))=
sqr(sqr(powerrec2(5,1)))=
sqr(sqr(5*powerrec2(5,0)))
sqr(sqr(5*1)) : 3 multiplications

Paramètres et variables locales

- ▶ Les paramètres et les variables déclarées localement dans une fonction (si elle est récurtivité ou non)
 - ▶ sont créées au moment de l'appel/activation de la fonction
 - ▶ continuent à exister jusqu'à la sortie finale de l'appel
 - ▶ sont inaccessibles et immuables pendant d'autres appels imbriqués (sauf utilisation de pointeurs...)
- ▶ Donc, quand une fonction s'appelle récurtivité, à un moment de l'exécution du programme, il existe un nombre indéfini d'occurrences de ses paramètres et variables, mais, en général une seule occurrence accessible.
- ▶ Possibilité d'échec parce que l'espace mémoire ne suffit pas pour toutes ces variables.

Manipulation de tableaux

- ▶ Toutes les listes multiplicatives avec terminaison n
- ▶ Recherche dichotomique
- ▶ Recherche d'un parcours de l'échiquier par un cavalier

Exemple : recherche dichotomique dans tableau trié

```
fonction rechDic(E tableau t(N):entier ,E taille:entier ,E val:entier):  
    booleen  
    retourner rechDicRec(t,1,taille ,val)  
finfonction  
  
fonction rechDicRec(E tableau t(N):entier ,E d:entier ,E f:entier ,E  
v:entier):booleen  
variables  
m:entier  
si d ≤ f alors  
    m ← f+d/2  
    si t(m) > v alors  
        retourner rechDicRec(t,d,m-1,v)  
    sinon si t(m) < v alors  
        retourner rechDicRec(t,m+1,f,v)  
    sinon  
        retourner vrai  
    fsi  
sinon  
    retourner faux  
    fsi  
finfonction
```

Nous verrons d'autres types de dichotomie moins faciles à écrire sans récursivité lors du prochain cours sur les tris rapides.

Notion de fonctions pures

- ▶ Théorie selon laquelle les fonctions écrites sans affectations et effets de bord sont plus claires et plus facilement écrites sans bugs
- ▶ Plus proche de l'idée mathématique d'une fonction.
- ▶ Les définitions ressemblent à des équations
- ▶ Une question de goût
- ▶ Mais récursion nécessaire pour ce style de programmation

Utilisation obligatoire

- ▶ Quels sont les fonctions pour lesquelles la récursion est essentielle ?
- ▶ Aucune : ce qu'on peut faire avec récursion, on peut toujours faire avec une pile explicite !
- ▶ Jeux de parcourir un arbre sans récursion et sans pile !!

Utilisation conseillée

- ▶ Quand le code est plus simple à écrire et comprendre avec récursion
- ▶ Dans presque tous les cas où l'alternative serait une pile explicite
- ▶ Si je veux parcourir deux arbres en même temps (par exemple je veux les fusionner), je ne peux pas forcément le faire récursivement (mais "coroutines" récursives existent dans certains langages)

Le coût

- ▶ Espace : négligeable :
nécessité de stocker les paramètres et variables locales de chaque appel (activation) ; utilisation invisible d'une pile ; sauf très gros tableaux, cet espace n'est pas important
- ▶ Temps : négligeable :
les instructions supplémentaires exécutées à chaque appel d'une fonction prennent un temps petit par rapport aux calculs de la fonction sauf si ces calculs sont simples
- ▶ La qualité du programme est plus importante et un bon compilateur peut quelquefois générer le même code pour les deux versions (cas de récursion à la fin)

Terminaison

Le risque majeur d'une fonction récurtive est de ne jamais terminer :

- ▶ Toujours commencer par écrire la(les) condition(s) d'arrêt
- ▶ Ecrire ensuite le(s) cas récurtif(s) de telle sorte à ce qu'il(s) converge(nt) vers la(les) conditions(s) d'arrêt

Cette fonction termine-t-elle ?

```
fonction f(E n: entier): entier
si (n=1) alors
  retourner 1
sinon si (n mod 2)=0 alors
  retourner 1+f(n/2)
sinon
  retourner 1+f(3*(n+1)/2)
fsi
finfonction
```

Récursivité

Récursivité terminale

- ▶ Lorsque l'appel récursif est la dernière instruction de la fonction ou de la procédure
- ▶ Permet une traduction immédiate en une version itérative et vice-versa
- ▶ Efficacité équivalente des deux versions (itérative et récursive)

Récursivité non terminale

- ▶ Beaucoup plus délicate à traduire en une version itérative :
 - ▶ Dans le cas d'une récursivité non terminale simple : la rendre terminale en déplaçant les instructions qui suivent l'appel récursif en début de fonction.
 - ▶ Dans le cas d'une récursivité non terminale multiple : pas de solution générique. (ex : tour de Hanoi).