Algorithmes récursifs

Nour-Eddine Oussous et Éric Wegrzynowski

Licence ST-A, USTL - API2

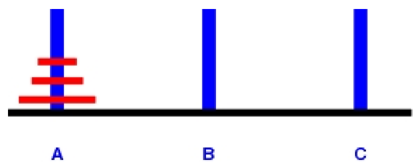
23 septembre 2009

Introduction

- En programmation, de nombreux problèmes résolus par répétition de tâches
- ⇒certains langages (comme PASCAL) munis de structures de contrôles répétitives : boucles `pour` et `tant que`
- Mais certains problèmes se résolvent simplement en résolvant des problèmes identiques

C'est la récursivité

Les tours de Hanoï : le problème



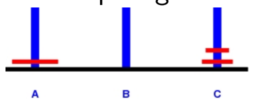
Règles (opérations élémentaires)

1. *déplacer un disque à la fois d'un bâton sur un autre*
2. *ne jamais mettre un disque sur un plus petit*

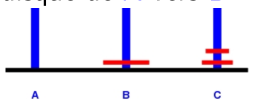
But : transférer la pile de disques de A vers B.

Les tours de Hanoï : une solution ?

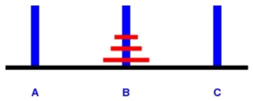
1. Mettre tous les disques sauf le plus grand sur C



2. Déplacer le plus grand disque de A vers B



3. Mettre tous les disques de C sur B



Les tours de Hanoï : une solution

- Les points 1 et 3 de la solution esquissée sont des problèmes de Hanoï avec un disque de moins
- $\Rightarrow$ si on sait résoudre le problème avec $n-1$ disques, alors on sait le résoudre avec n disques
- Or on sait résoudre le problème avec 1 disque
- $\Rightarrow$le problème est résolu pour tout nombre $n \geq 1$ de disques (principe de récurrence)

La solution est récursive

Calcul de dérivées

- Règles de dérivation
  - $(u+v)' = u' + v'$
  - $(u-v)' = u' - v'$
  - $(uv)' = u'v + uv'$
  - $\left(\frac{u}{v}\right)' = \frac{u'v - uv'}{v^2}$
  - ...
- $\Rightarrow$ pour dériver il faut savoir dériver !
- Or on sait dériver les fonctions de base
- $\Rightarrow$on sait dériver toutes les fonctions (dérivables bien entendu !).

Le calcul est récursif

Calcul de factorielle

- Soit à calculer $n! = 1 \cdot 2 \cdot 3 \cdots (n-1) \cdot n$
- On sait que pour $n > 0$, $n! = n \cdot (n-1)!$
- $\Rightarrow$si on sait calculer $(n-1)!$, alors on sait calculer $n!$
- Or on sait calculer $0! = 1$
- $\Rightarrow$on sait calculer $n!$ pour tout $n \geq 0$

Le calcul est récursif

Définition

Définition

Un algorithme de résolution d'un problème P sur une donnée a est dit *récursif* si parmi les opérations utilisées pour le résoudre, on trouve une résolution du même problème P sur une donnée b.

Appel récursif

Dans un algorithme récursif, on nomme *appel récursif* toute étape de l'algorithme résolvant le même problème sur une autre donnée.

Tours de Hanoï : Algorithme

Tours de Hanoï

$H(n, D, A, I)$ = problème de déplacement de n disques depuis la tour D vers la tour A avec la tour intermédiaire I

```
H(n,D,A,I):
  si n=1 alors
     deplacer disque de D vers A
  sinon
     H(n−1,D,I,A);
     deplacer disque de D vers A;
     H(n−1,I,A,D);
  fin si
```


Dérivation : Algorithme

Dérivation

$deriver(f)$ = problème du calcul de la dérivée de f

```
deriver(f):
  si f est une fonction de base alors
     donner la derivee de f
  sinon
     si f est de la forme u+v alors
        deriver(u) + deriver(v)
     si f est de la forme u−v alors
        ...
```


Factorielle : Algorithme

Factorielle

$fact(n)$ = problème du calcul de $n!$

```
fact(n):
  si n=0 alors
     fact(0) = 1
  sinon
     fact(n) = n×fact(n−1)
  fin si
```

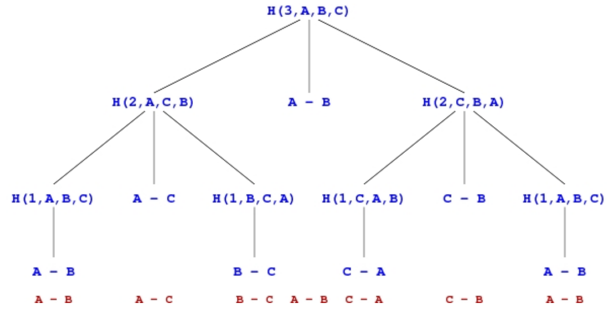
Exécution d'un algorithme récursif

Calcul de 4! :

fact(4) $\Rightarrow$ 4$\cdot$ fact(3) $\Rightarrow$ $4 \cdot 3\cdot$ fact(2) $\Rightarrow$ $4 \cdot 3 \cdot 2\cdot$ fact(1) $\Rightarrow$ $4 \cdot 3 \cdot 2 \cdot 1\cdot$ fact(0) $\Rightarrow$ $4 \cdot 3 \cdot 2 \cdot 1 \cdot 1$ $\Rightarrow$ 24

Exécution d'un algorithme récursif

Exécution de l'algorithme des tours de Hanoï pour déplacer trois disques de la tour A vers la tour B



En rouge, la suite des déplacements effectués au cours de l'exécution de l'algorithme.

Règles de conception

Attention

Il existe des algorithmes récursifs qui ne produisent aucun résultat

```
fact(n):
   fact(n) = n×fact(n−1)
```

fact(1) $\Rightarrow$ 1$\cdot$fact(0) $\Rightarrow$ $1 \cdot 0\cdot$fact(-1) $\Rightarrow \dots$

$\Rightarrow$ calcul infini

Règles de conception

Première règle

Tout algorithme récursif doit distinguer plusieurs cas, dont l'un au moins ne doit pas comporter d'appel récursif.

sinon risque de cercles vicieux et de calcul infini

Condition de terminaison, cas de base

Les cas non récursifs d'un algorithme récursif sont appelés *cas de base*.

Les conditions que doivent satisfaire les données dans ces cas de base sont appelées *conditions de terminaison*.

Règles de conception

Attention

Même avec un cas de base un algorithme récursif peut ne produire aucun résultat

```
fact(n):
   si n = 0 alors
      fact(0) = 1
   sinon
      fact(n) = fact(n+1) / (n+1)
   fin si
```

fact(1) $\Rightarrow$ fact(2)/2 $\Rightarrow$ fact(3)/(2 $\cdot$ 3) $\Rightarrow \ldots$

$\Rightarrow$ calcul infini

Règles de conception

Seconde règle

Tout appel récursif doit se faire avec des données plus « proches » de données satisfaisant une condition de terminaison.

Théorème

Il n'existe pas de suite infinie strictement décroissante d'entiers positifs ou nuls.

Ce théorème permet de contrôler l'arrêt d'un calcul suivant un algorithme récursif.

Récursivité simple ou linéaire

Récursivité simple ou linéaire

Un algorithme récursif est *simple* ou *linéaire* si chaque cas qu'il distingue se résout en au plus un appel récursif.

Récursivité simple ou linéaire

L'algorithme de calcul de $n!$ est récursif simple.

```
fact(n):
   si n = 0 alors
      fact(0) = 1
   sinon
      fact(n) = n·fact(n−1)
   fin si
```

Récursivité multiple

Récursivité multiple

Un algorithme récursif est *multiple* si l'un des cas qu'il distingue se résout avec plusieurs appels récursifs.

Dans le cas où il y a deux appels récursifs on parle de récursivité *binaire*.

Récursivité multiple

L'algorithme des tours de Hanoï est récursif binaire

```
H(n,D,A,I):
  si n=1 alors
     deplacer disque de D vers A
  sinon
     H(n−1,D,I,A);
     deplacer disque de D vers A;
     H(n−1,I,A,D);
  fin si
```


Récursivité croisée ou mutuelle

La récursivité peut parfois être cachée.

Récursivité mutuelle

Deux algorithmes sont *mutuellement* récursifs si l'un fait appel à l'autre, et l'autre fait appel à l'un.

Récursivité croisée ou mutuelle

Parité d'un entier

$P(n)$ = prédicat de test de parité de l'entier n.
$I(n)$ = prédicat de test d'« imparité » de l'entier n.

Solution mutuellement récursive

```
P(n):
  si n=0 alors
     P(n) = vrai
  sinon
     P(n) = I(n−1)
  fin si

I(n):
  si n=0 alors
     I(n) = faux
  sinon
     I(n) = P(n−1)
  fin si
```

Récursivité croisée ou mutuelle

Évaluation de P(2) :

P(2) ⇒ I(1) ⇒ P(0) ⇒ vrai

Évaluation de P(3) :

P(3) ⇒ I(2) ⇒ P(1) ⇒ I(0) ⇒ faux

Récursivité en Pascal

- Pascal permet d'exprimer les algoritmes récursifs.
- Les fonctions et les procédures peuvent être récursives.
- Un appel récursif s'écrit simplement en faisant référence au nom de la fonction ou de la procédure.
- Il faut utiliser le mot-clé **forward** pour les fonctions ou procédures mutuellement récursives.

Récursivité en Pascal

Une fonction de calcul de n!

Listing

```
// fact(n) = n!
function fact(n : CARDINAL) : CARDINAL;
begin
  if n=0 then
    fact := 1
  else
    fact := n*fact(n-1);
end {fact};
```


Récursivité en Pascal

Une procédure de résolution des tours de Hanoï :

Listing

```
// hanoi(n,d,a,i) deplace n disques
// de la tour d vers la tour a en utilisant
// la tour i comme tour intermediaire
procedure hanoi(const n : CARDINAL;
                const d,a,i : TOURS);
begin
  if n=1 then
    deplacerDisque(d,a)
  else begin
    hanoi(n-1,d,i,a);
    deplacerDisque(d,a);
    hanoi(n-1,i,a,d);
  end {if};
end {hanoi};
```

Récursivité en Pascal

Les prédicats de test de parité :

Listing

```
//impair(n) = vrai si et seulement si n est impair
function impair(n : CARDINAL) : BOOLEAN; forward;

//pair(n) = vrai si et seulement si n est pair
function pair(n : CARDINAL) : BOOLEAN;
begin
  if n=0 then
    pair := true
  else
    pair := impair(n-1);
end {pair};

function impair(n : CARDINAL) : BOOLEAN;
begin
  if n=0 then
    impair := false
  else
    impair := pair(n-1);
end {impair};
```


Conclusion

- La récursivité est un moyen naturel de résolution de certains problèmes.
- Tout algorithme itératif peut s'exprimer de manière récursive.
- Beaucoup de langages de programmation, dont Pascal, permettent d'exprimer des algorithmes récursifs.