La récursivité

Driss MATROUF
Maître de conférences HDR

Recursivité

- Une fonction est dite récursive si elle comporte dans son corps un (ou plusieurs) appel(s) à elle-même.

- Le raisonnement récursif :

- On veut résoudre un problème de taille N : P(N)
- Si Résoudre(P(N-1)) $\Rightarrow$résoudre(P(N)) et Si $P(n_0)$ est facile alors P(N) est résolu

Evidence : $P(n_0) \Rightarrow P(n_0+1) \Rightarrow P(n_0 + 2) \ldots . \Rightarrow P(N)$

Exemple

Problème :

je souhaite calculer Somme (des n premiers entiers)

Somme (des n premiers entiers) = Somme (des n-1 premiers entiers) + n

```
int somme(int n)
{
	if(n==1) return (1) ;
	return(somme(n-1)+n)) ;
}
```

Appel:somme(4)

somme(4)=somme(3)+4

somme(3)=somme(2)+3

somme(2)=somme(1)+2

somme(1)=1

Recursivité

```
int somme(int n)
{
        if(n==1) return (1) ;
        return(somme(n-1)+n)) ;
}
```

+ Dans une fonction récursive il faut absolument qu'il le traitement du cas trivial

Ici : n=1

+ Dans cette fonction on passe d'un appel avec n vers un appel avec n-1,

n → n-1 → n-2 . . ....2→1

1 → 2 → . . ... → n-1 → n

Ce qui conduira forcement vers le cas trivial

Un autre exemple

Afficher un tableau d'entier de N éléments :

```
void afficher(int T[], int d, int f)
{
	if(d<=f)
	{
	cout<<T[d]<<"," ;
	afficher(T,d+1,f);
	}
	// else c'est le cas trivial, on fait rien
}
```

T={1,3,6,8}

afficher(T,0,3) → 1, afficher(T,1,3) → 1,3, afficher(T,2,3) → 1,3,6, afficher(T,3,3) → 1,3,6,8

afficher(T,4,3) → 1,3,6,8,

Afficher à l’envers

```
void afficher(int T[], int d, int f)
{
	if(d<=f)
	{
		afficher(T,d+1,f);
		cout<<T[d]<<"," ;
	}
	// else c’est le cas trivial, on fait rien
}
```

T={1,3,6,8}

afficher(T,0,3) → afficher(T,1,3) ,1 → afficher(T,2,3) 3,1, → afficher(T,3,3) 6,3,1, → afficher(T,4,3) 8,6,3,1,

Calcule de la puissance

a^n ???

Naïf

$a^n = a^{n-1} \times a$

```
int puissance(int a, int n)
{
     if(n==0) return(1) ;
     else return (puissance(a,n-1)xa) ;
}
```

$a^6 \rightarrow a^5 \times a \rightarrow a^4 \times a \times a \rightarrow a^3 \times a \times a \times a \rightarrow a^2 \times a \times a \times a \times a \rightarrow a \times a \times a \times a \times a \times a \rightarrow 1 \times a \times a \times a \times a \times a \times a$

Intelligent

Si n est pair $a^n = (a^{n/2})$

Sinon $a^n = (a^{n/2}) \times a$

```
int puissance(int a, int n)
{
     if(n==0) return 1;
     int r=puissance(a,n/2) ;
     if((n % 2)==0) return(r*r) ;
     else return(r*r*a) ;
}
```

$a^6 \rightarrow a^3 \times a3 \rightarrow (a \times a \times a)$

$a^{12} \rightarrow a^6 \times a^6 \rightarrow a^3 \times a^3 \rightarrow a \times a \times a$

Afficher les chiffres composant un entier

N=5678 doit afficher 5-6-7-8

```
void afficher(int n)
{
    if(n==0) return ;
    else
    {
        cout<<n %10<<"-" ; //L1
        afficher(n/10) ; // L2
    }
}
```

Afficher(5678) → 5-afficher(678)
→ 5-6-afficher(78) → 5-6-7-afficher(8)
→ 5-6-7-8-afficher(0) → 5-6-7-8-

ET si on voulait afficher
8-7-6-5

Il suffit d'inverser L1 et L2 :

```
void afficher(int n)
{
    if(n==0) return ;
    else
    {
    afficher(n/10) ; // L2
    cout<<n %10<<"-" ; //L1
    }
}
```

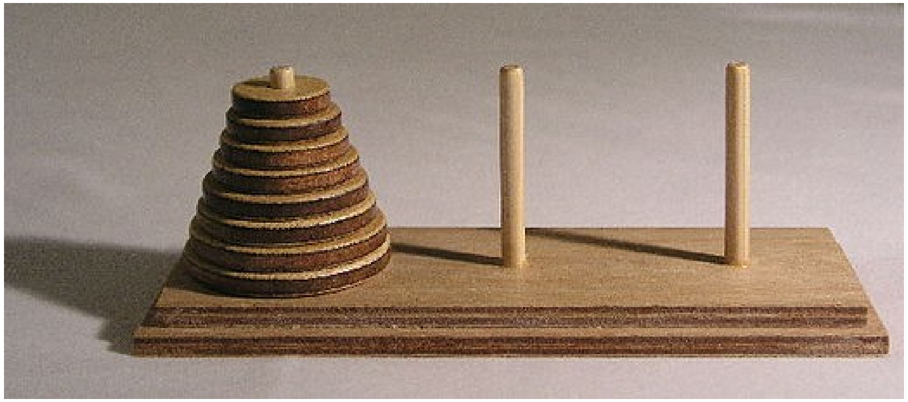
afficher(5678) → afficher(567)8-
→afficher(56)7-8- → afficher(5)6-7-8-
→ 5-6-7-8-

Les tours de Hanoï

Les tours de Hanoï (originellement, la tour d'Hanoïa) sont un jeu de réflexion imaginé par le mathématicien français Édouard Lucas, et consistant à déplacer des disques de diamètres différents d'une tour de « départ » à une tour d'« arrivée » en passant par une tour « intermédiaire », et ceci en un minimum de coups, tout en respectant les règles suivantes :

+ on ne peut déplacer plus d'un disque à la fois ;

+ on ne peut placer un disque que sur un autre disque plus grand que lui ou sur un emplacement vide.



Les tours de Hanoï

+ On déplace n-1 disques de T1 vers T2 (en utilisant T3)
+ On déplace le disque n de T1 vers T3
+ On déplace les n-1 disques de T2 vers T3 (en utilisant T1)

```
Algorithme Hanoi(Tige &T1, Tige &T2, tige & T3,n)
    si(n==0) rien à faire
    sinon
      Hanoi(T1,T3,T2,n-1)
      deplacer un disque de T1 vers T3
      Hanoi(T2,T1,T3,n-1)
```

En résumé

+ Une fonction récursive doit comporter

- Un cas d’arrêt dans lequel aucun autre appel n’est effctué

- Un cas général dans lequel un ou plusieurs autres appels sont effectués

+ La chaîne d’appel doit conduire au critère d’arrêt

+ L’écriture sous forme récursive est toujours plus simple que l’écriture sous forme itérative