

Chapitre 8 – La récursivité

Concept fondamental en mathématiques et en informatique :
manière particulière d'appréhender certains problèmes.

Généralités

- ❑ Un module (procédure ou fonction) récursif est un module qui s'appelle lui-même : il est *défini en référence à lui-même*.
- ❑ Un module ne peut pas être défini uniquement en référence à lui-même sans qu'on introduise de définition circulaire.
→ *condition de terminaison*
- ❑ Pour une récursivité qui boucle indéfiniment, l'arrêt se fait quand toute la mémoire disponible est utilisée.
- ❑ Le calcul effectif d'un module récursif s'effectue à partir du moment où on a atteint le point d'appui.
- ❑ La suite des appels d'un module récursif est une pile :
 - à chaque appel, le contexte d'exécution (les paramètres effectifs, les variables locales et l'adresse de retour) est empilé sur la pile (zone de la mémoire),
 - après l'exécution du module, on retourne à l'adresse au sommet de la pile et on désempile.

Récurtivité fondée sur une relation de récurrence

Pour l'évaluation d'une suite récurrente, on a le choix entre un algorithme itératif et un algorithme récursif

Exemple 1 : la fonction factorielle.

$$u_0 = 1$$

$$u_n = n * u_{n-1} \text{ si } n > 0$$

```
Fonction fac (n : entier) : entier
Var i, f : entier
Début
    f ← 1
    Pour i ← 2 à n Inc +1 Faire
        f ← i * f
    FinPour
    retourne(f)
Fin
```

```
Fonction fac (n : entier) : entier
Var f : entier
Début
    Si n = 0
        Alors f ← 1
        Sinon f ← n * fac(n-1){@2}
    FinSi
    retourne(f)
Fin
```

Pile pour l'appel de fac(3)

écrire(fac(3)){@1}

~~@2; n=0~~

~~@2; n=1~~

~~@2; n=2~~

@1; n=3

$$f = 3 * (2 * (1 * (1))) = 6$$

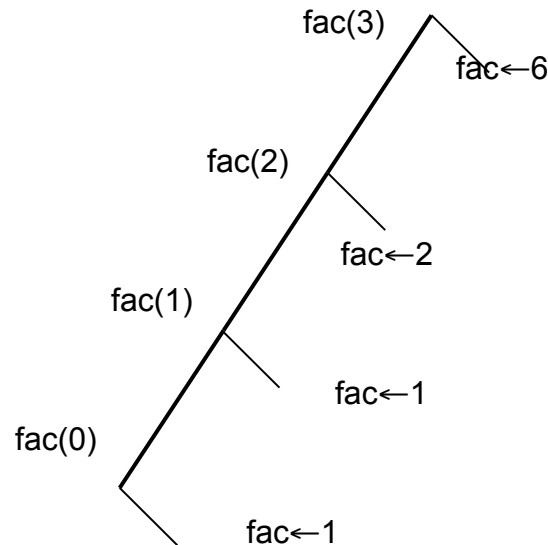
Arborescence des appels récursifs

On peut visualiser les différents appels réalisés par un graphe *nommé arborescence des appels récursifs*

- le premier appel est extérieur et correspond au seul sommet du graphe qui n'a pas de prédécesseur (racine),
- chaque sommet du graphe correspond à un appel de l'algorithme et est identifié par le nom du module exécuté avec le contexte d'exécution,
- il existe un arc entre deux sommets X et Y ssi l'exécution qui correspond au sommet X engendre un nouvel appel correspondant au sommet Y.

Arborescence des appels récursifs

- On peut enrichir les graphes des appels en indiquant les actions réalisées sur chaque version exécutée.



- L'intérêt de la récursivité se justifiera surtout par une nouvelle manière d'aborder les problèmes.

Suite de Fibonacci

Exemple 2 : la suite de Fibonacci

$$u_0 = u_1 = 1$$

$$u_n = u_{n-1} + u_{n-2} \text{ si } 1 < n$$

```
Fonction fibo (n : entier) : entier
Const max = 10
Type tab = tableau[1..max] d'entier
Var i : entier
      t : tab
Début
t[0] ← 1
t[1] ← 1
Pour i ← 2 à n Inc +1 Faire
      t[i] ← t[i-1] + t[i-2]
FinPour
retourne(t[n])
Fin
```

```
Fonction fibo (n : entier) : entier
Var f : entier
Début
Si n ≤ 1
      Alors f ← 1
      Sinon f ← fibo(n-1){@2} + fibo(n-2){@3}
FinSi
retourne(f)
Fin
```

Appel de fibonacci(4)

écrire(fibo(4)){@1}

~~@3 ; n=0~~

~~@2 ; n=1~~

~~@3 ; n=2~~

~~@3 ; n=1~~

~~@3 ; n=0~~

~~@2 ; n=1~~

~~@2 ; n=2~~

~~@2 ; n=3~~

~~@1 ; n=4~~

Fibo = 1 + 1 + 1 + 1 + 1 = 5

Sommes de 1 : peu performant car il y a répétition du calcul, la méthode est non optimale.

Appel de fibonacci(4)

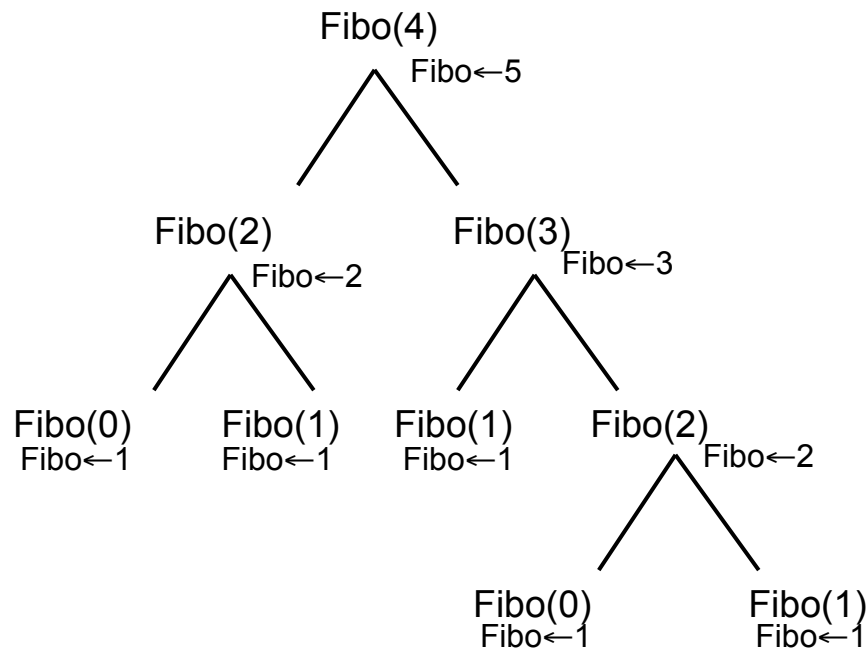


Schéma général d'un algorithme récursif

- ❑ Algorithmes récursifs ne découlant pas d'une suite récurrente.
- ❑ Dans un algorithme récursif, on distingue deux parties : la condition de terminaison (point d'appui) et l(es)'appel(s) récursif(s).

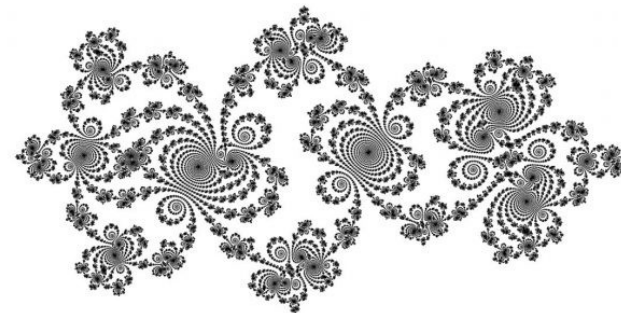
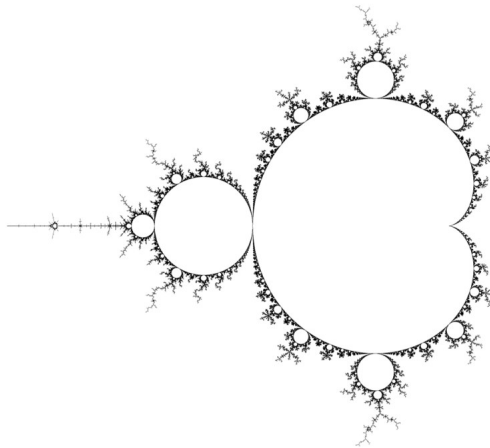
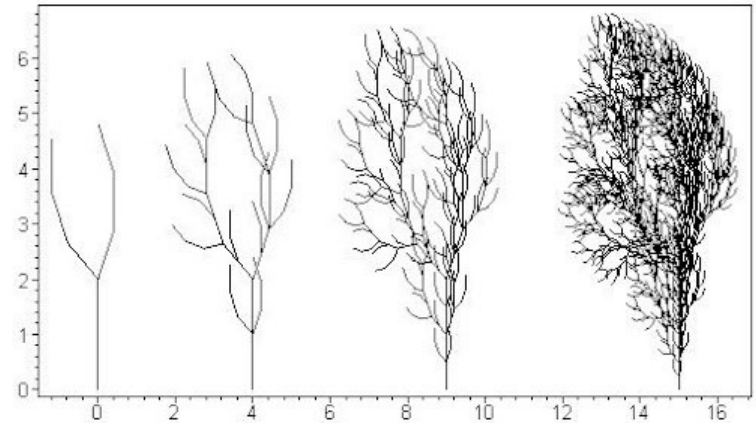
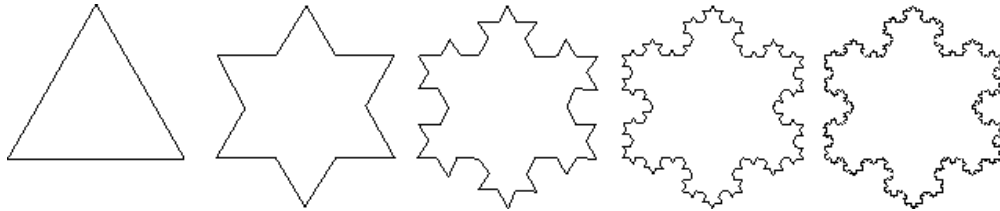
```
Module A( $p_1, \dots, p_n$ )  
Début  
Si cond  
    Alors I1...  
    Sinon I2  
        A( $f(p_1, \dots, p_n)$ )  
        I3  
        A( $g(p_1, \dots, p_n)$ )  
        ...  
    Im  
FinSi  
Fin
```

Définitions

- ❑ Un algorithme est dit *simplement récursif* s'il ne contient qu'un seul appel récursif.
- ❑ Un appel récursif est *terminal* s'il n'est jamais suivi par l'exécution d'instructions dans le module.
- ❑ Une récursivité est *indirecte* ou *croisée* quand un module A sans appel récursif appelle un module B qui appelle A.

Applications : les fractales

<https://youtu.be/zXTpASSd9xE?t=68>



Applications : tours de Hanoi

