Algorithmes Récurrents

1. Introduction à la Récursivité

En algorithmique, un processus est dit **récurrent** (ou récursif) lorsqu'une fonction s'appelle elle-même pour résoudre un problème. Ce concept s'inspire du principe mathématique de récurrence : pour résoudre un problème complexe, on le divise en sous-problèmes identiques mais de taille réduite, jusqu'à atteindre une situation si simple qu'elle se résout directement.

2. Les Deux Piliers d'un Algorithme Récurrent

Pour qu'un algorithme récursif fonctionne sans erreur et ne crée pas de boucle infinie (ce qui provoque un plantage de la mémoire appelé *Stack Overflow*), il doit obligatoirement inclure deux éléments :

- **La condition d'arrêt (le cas de base) :** C'est la condition qui met fin aux appels récursifs. Elle correspond au cas le plus basique du problème, dont la solution est triviale et connue à l'avance.
- **L'appel récursif (l'hérédité) :** C'est le moment où la fonction s'invoque elle-même en modifiant ses paramètres pour se rapprocher inévitablement de la condition d'arrêt.

3. Exemple Pratique 1 : Le Calcul de la Factorielle

La factorielle d'un nombre `n` (notée `n!`) est le produit de tous les entiers positifs inférieurs ou égaux à `n`. Par définition mathématique : `n! = n * (n-1)!`, avec la condition d'arrêt `0! = 1`.

Voici l'algorithme en pseudo-code :

```
Fonction Factorielle(n : Entier) : Entier
Début
    // 1. La condition d'arrêt
    Si n = 0 Alors
        Retourner 1
    // 2. L'appel récursif
    Sinon
        Retourner n * Factorielle(n - 1)
    FinSi
Fin
```

apcpedagogie.com

4. Exemple Pratique 2 : La Suite de Fibonacci

La suite de Fibonacci est un grand classique où chaque terme est la somme des deux termes précédents. Par définition : `F(n) = F(n-1) + F(n-2)`, avec les conditions d'arrêt `F(0) = 0` et `F(1) = 1`.

```
Fonction Fibonacci(n : Entier) : Entier
Début
    // Conditions d'arrêt
    Si n = 0 Alors
        Retourner 0
    Sinon Si n = 1 Alors
        Retourner 1
    // Appels récursifs multiples
    Sinon
        Retourner Fibonacci(n - 1) + Fibonacci(n - 2)
    FinSi
Fin
```

5. Récursivité vs Itération (Boucles classiques)

Le choix entre une approche récursive ou itérative dépend fortement du problème à résoudre :

Critère	Algorithme Récurrent	Algorithme Itératif (Boucles Pour/Tant Que)
Lisibilité du code	Souvent plus court, élégant et très proche de la logique mathématique.	Peut devenir lourd et complexe à lire pour des problèmes de hiérarchie avancés.
Consommation mémoire	Élevée (chaque appel est empilé et mis en attente dans la mémoire).	Faible (utilise des variables locales mises à jour en continu).
Cas d'usage idéal	Parcours d'arbres, de graphes, problèmes "Diviser pour régner" (ex: Tri rapide).	Traitements simples, parcours de tableaux linéaires, calculs basiques continus.

Rédigé et structuré par : **Med Khalil Kribi**

Source : https://apcpedagogie.com/algorithmes-recurrents · téléchargé le 07/11/2026 · © APC Pédagogie · apcpedagogie.com