

# Récurrance et récursivité

## Récurrance

### Les suites

Une suite est une liste **ordonnée** d'éléments appelés "**terme**".

La **longueur** d'une suite correspond au nombre d'éléments qu'elle contient. Il existe quelques noms spécifiques pour certaines suites :

- **vide** désigne une suite composée de 0 éléments et est représentée ainsi :  $\$()\$$
- **couple** désigne une suite composée de 2 éléments (exemple  $\$(1,2)\$$ )
- **finie** désigne une suite composée d'un nombre déterminé d'éléments (exemple *Les nombres entre 1 (inclus) et 3 (inclus)* :  $\$(1,2,3)\$$ )
- **infinie** désigne une suite composée d'un nombre indéterminé d'éléments (exemple "La suite des nombres positifs partant de 0 jusqu'à l'infini, ou la suite de Fibonacci" qui peuvent être représentée par une formule mathématique ou par une suite terminée par ...  $\$(1,2,3,\dots, n)\$$ )

Une suite (qui correspond plus aux collections `List` en Java) est notée avec des parenthèses et ne doit pas être confondue avec un **ensemble** qui est représenté par des accolades (et qui correspond aux collections `Set` en Java)

Une suite peut être transformée en ensemble en retirant tous les doublons qu'elle contient.

Ainsi la suite  $\$(1,2,3,4,3,2)\$$  deviendra l'ensemble  $\$\{1,2,3,4\}\$$ . À savoir que contrairement aux suites, l'ordre ne compte pas dans un ensemble. Ce qu'il peut exister un nombre infini de suites pour un ensemble donné. L'ensemble correspondant à une suite est noté  $\$E(s)\$$

Note: La suite de fibonacci est une suite commençant par `0,1` où chaque élément est la somme des 2 précédents.

## La démonstration par récurrence

Le but de la récurrence est de montrer qu'une propriété  $P(n)$  est vraie pour tout  $n$ .

- Premièrement, l'**initialisation**, qui correspond à la preuve d'une propriété au premier rang demandé. Si on prend l'exemple de domino, l'initialisation est le fait que le premier domino tombe.
- Deuxièmement, l'**hérédité** qui correspond au fait que chaque élément se comportera toujours exactement comme le premier. Autrement dit, que si la propriété est valable pour  $n$  alors, elle sera aussi valable pour  $n+1$ . Ainsi, dans l'exemple du domino, l'hérédité est le fait que chaque domino qui tombe fera tomber son suivant.

TODO démonstration par récurrence héréditaire et emboîtée

L'intérêt de la récurrence en informatique, c'est le fait de pouvoir prouver mathématiquement le fonctionnement d'un programme ainsi diminuant la nécessité de faire un plan de test, car une démonstration vérifie que le programme fournit un résultat correct dans tous les cas de figures.

## La récursivité

La récursivité est une implémentation dans un programme de la notion mathématique de récurrence. Ainsi la récursivité est *la démarche de faire référence à l'objet même de la démarche à un moment du processus* (par exemple une méthode qui s'appelle elle-même sous certaines conditions)

L'approche récursive est une manière de programmer qui s'oppose en quelque sorte à l'approche itérative. L'approche dite "itérative" va utiliser des boucles tandis que l'approche récursive va résoudre un problème en calculant les instances plus petites du même problème.

La récursivité est un moyen simple et élégant de résoudre un problème qui a également l'avantage d'être souvent plus simple et rapide à programmer. En revanche, elle a généralement le désavantage d'être plus gourmande en ressources pour l'ordinateur et ainsi d'être moins efficace.

Voici un exemple de définition d'un problème de manière *explicite*, *itérative* et *récursive* :

Considérons la suite des sommes des  $n$  premiers nombres entiers strictement positifs

- Explicite :  $S_n = \frac{n(n+1)}{2}$
- Itérative :  $S_n = \sum_{i=1}^n i$
- Récursive :  $S_n = S_{n-1} + n$

Ainsi, choisir la récursivité à la place d'une autre approche dépend de son goût personnel pour un style de programmation, de la simplicité de la solution récursive par rapport à une autre, et de l'efficacité recherchée de l'implémentation.

# Créer une méthode récursive

Une méthode récursive va être composée de 2 choses au minimum

1. Une condition d'arrêt, c'est-à-dire un cas de base où on sait avec certitude la réponse.
2. Un appel à la fonction elle-même en décrémentant un de ses paramètres.

Mais on peut éventuellement lui ajouter une condition d'erreur si un paramètre est incorrect

Ainsi voici un calcul de la suite de Fibonacci *récursive*

```
public static int fibonacci(int n) {  
    // 0. condition d'erreur  
    if (n <= 0) throw new IllegalArgumentException("n doit être strictement supérieur à 0");  
  
    // 1. condition d'arrêt de la récursion, on sait que quand n est égal à 2 ou a 1 la réponse sera n-1  
    if (n <= 2) return n-1;  
  
    // 2. appel récursif à la méthode en la décrémentant son paramètre n  
    return fibonacci(n-1) + fibonacci(n-2);  
}
```

## Définition par induction

Il est possible de définir certaines notions par *induction*, c'est-à-dire en procédant en 2 temps :

1. Base initiale ou base de construction : on représente les éléments primitifs de la notion
2. Étape inductive, c'est-à-dire que l'on énonce les règles de la notion à partir des éléments primitifs

Par exemple pour représenter l'ensemble des nombres impairs ( $I$ )

1. Base initiale :  $1 \in I$  (1 est l'élément primitif, car c'est l'instance la plus petite d'un nombre impair)
2. Règle de construction :  $\forall n \in I : (n+2) \in I$  (pour tout nombre impair, ce même nombre + 2 sera aussi impair)

Voici un autre exemple avec un palindrome (séquences de lettres pouvant se lire de la même façon de droite à gauche et de gauche à droite)

1. Base initiale : Toute lettre  $x$  est un palindrome. Une lettre étant l'instance la plus petite d'un palindrome (une lettre est la même se lit de la même manière de droite à gauche et de gauche à droite)
2. Règle de construction : Une lettre  $x$  + un palindrome + lettre  $x$  est aussi un palindrome (par exemple `r`, `d` et `a` sont des palindromes, `ada` est un palindrome et `radar` est également un palindrome)

Cela est un peu équivalent à la manière de construire une méthode récursive en soi. La base initiale étant la condition d'arrêt et la règle de construction étant l'appel récursif.

# Preuve de correction d'un algorithme récursif par induction

La méthodologie est la suivante :

1. Définir quel est le but de l'algorithme et une instance du problème
2. Instances ordonnées par taille (entier  $n$ , taille du tableau, etc)
3. Indiquer la base de l'induction comme vu plus tôt
4. Indiquer la règle de construction comme vu plus tôt également
5. Terminaison : on montre que les appels récursifs se font sur des sous-problèmes

---

Revision #7

Created 24 May 2023 13:09:06 by SnowCode

Updated 24 May 2023 17:32:42 by SnowCode