Induction et récursion

Induction et **récursion** sont deux concepts intimement liés, deux facettes d'une même idée. Dans l'usage courant, les deux mots sont employés de façon parfois interchangeable, et en tout cas avec une frontière assez floue ; c'est pour cela qu'on a pris le parti de traiter les deux concepts dans un même page.

Formellement, l'induction est une technique de preuve mathématique employée pour démontrer des théorèmes sur les nombres naturels ou sur d'autres structures infinies munies d'un ordre bien fondé (souvent, des structures définies **récursivement**). De façon encore plus formelle, l'induction est donnée comme un axiome d'un système de preuve, c'est le cas, par exemple, du système axiomatique de Peano.

D'un autre côté, la récursion est une technique de définition et construction d'objets mathématiques (fonctions, ensembles, etc.).

Ce que les deux concepts ont en commun, c'est l'idée d'étudier les propriétés d'un objet (par exemple, une propriété d'un entier n) en les réduisant aux propriétés d'objets *plus petits* (par exemple, les propriétés de $n-1$).

Induction

Dans sa forme la plus simple, le **principe d'induction** est une *technique de preuve* qui permet de prouver qu'un prédicat est vrai pour tous les entiers. Une *preuve par induction* (on dit aussi *preuve par récurrence*) procède en deux étapes :

- On démontre que le prédicat est vrai pour un nombre fini de cas *initiaux* ;
- On démontre que si le prédicat est vrai pour un cas quelconque, alors il est vrai aussi pour le cas *suivant*.

On peut faire remonter son origine aux mathématiques classiques, par exemple l'argument d'Euclide prouvant qu'il existe une infinité de nombre premiers est une forme implicite d'induction. Pascal a été l'un des premiers mathématiciens à en donner une définition explicite, mais une formalisation rigoureuse n'est apparue qu'au 19e siècle dans les travaux de Peano, Boole, etc.

Preuves par induction

Soit $P(n)$ un prédicat portant sur une variable libre n représentant un entier. Le but d'une *preuve par induction* est de montrer le prédicat

$$\forall n.\, P(n)$$

Il existe plusieurs formes équivalentes du *principe d'induction*. La plus commune, aussi appelée *induction faible* ou *simple*, procède en deux étapes :

- Un **cas de base** (aussi appelé **initialisation**), où on démontre le prédicat $P(0)$;
- Un **cas récursif** (ou **hérédité**, ou **pas inductif**), où on démontre le prédicat $P(n) \Rightarrow P(n+1)$.

De ces deux *lemmes*, le principe d'induction déduit $\forall n.\, P(n)$. En effet on sait par la *base* que $P(0)$, puis en utilisant l'hérédité on déduit $P(1)$ de $P(0)$, puis $P(2)$ de $P(1)$, et ainsi de suite.

L'*induction forte* est une autre forme d'induction couramment utilisée. Dans une preuve par induction forte il suffit de démontrer que

- $\forall k < n.\, P(k)$ implique $P(n)$

pour déduire la vérité du prédicat $\forall n.\, P(n)$. Cette forme d'induction est équivalente à l'induction faible, en effet prouver $P(n)$ par induction forte revient à prouver $\forall k < n.\, P(k)$ par induction faible.

Note : Le cas de base de l'induction forte est implicitement contenu dans sa définition. En effet lorsque $n = 0$, prouver $(\forall k < 0.\, P(k)) \Rightarrow P(0)$ revient à prouver $P(0)$, car la quantification sur la gauche est vide (et donc toujours vraie).

Exemples

On donne ici deux exemples classiques de preuves par induction.

Sommes partielles de la série arithmétique

Soit n un entier positif, on a l'égalité

$$\sum_{i=0}^{n} i = \frac{n(n+1)}{2}.$$

Preuve. Le cas de base est immédiat, en effet on obtient 0 à droite et à gauche de l'égal. Pour prouver l'hérédité, on suppose que

$$\sum_{i=0}^{n} i = \frac{n(n+1)}{2}$$

et on essaye de montrer que

$$\sum_{i=0}^{n+1} i = \frac{(n+1)(n+2)}{2}.$$

Or

$$\sum_{i=0}^{n+1} i = (n+1) + \sum_{i=0}^{n} i,$$

et par *hypothèse de récurrence* on peut remplacer la somme sur la droite par

$$(n+1) + \frac{n(n+1)}{2} = \frac{2(n+1)+n(n+1)}{2} = \frac{(n+1)(n+2)}{2}.$$

Sommes partielles de la série géométrique

Soit $a > 0$ un nombre réel et n un entier positif, on a l'égalité

$$\sum_{i=0}^{n} a^i = \frac{a^{n+1}-1}{a-1}.$$

Preuve. On vérifie aisément que pour $n = 0$ les deux côtés de l'égalité valent 1. On procède maintenant par récurrence.

$$\sum_{i=0}^{n+1} a^i = a^{n+1} + \sum_{i=0}^{n} a^i = a^{n+1} + \frac{a^{n+1}-1}{a-1},$$

où la dernière égalité découle de l'*hypothèse de récurrence*. Ceci est égal à

$$\frac{(a-1)a^{n+1} + a^{n+1} - 1}{a-1} = \frac{a^{n+2}-1}{a-1},$$

ce qui conclut la preuve.

Généralisation à d'autres ensembles que les entiers naturels

L'application du principe d'induction n'est pas restreinte aux entiers, en effet il peut être appliqué à tout ordre bien fondé. La propriété fondamentale des ordres bien fondés, à savoir qu'il n'existe pas de *chaîne descendante infinie*, garantit que toute preuve par induction se termine après un nombre fini d'étapes. Ce type d'induction est parfois appelé *induction structurelle*, il est spécialement utile pour raisonner sur des structures de données tels les *arbres*, les *listes*, etc.

Formellement, soit A un ensemble muni d'une relation d'ordre bien fondée $\prec$, et soit $P(a)$ prédicat portant sur une variable libre a représentant un élément de a. Si pour tout $a \in A$ on peut prouver que

$$(\forall b \prec a.\, P(b)) \Rightarrow P(a),$$

alors on peut conclure par induction que $P(a)$ est vrai pour tout $a \in A$.

Preuve. La preuve découle de l'hypothèse de *bonne fondation*. De

$$(\forall b \prec a.\, P(b)) \Rightarrow P(a),$$

on déduit que

$$\neg P(a) \Rightarrow (\exists b \prec a.\, \neg P(b)).$$

Supposons que $P(a)$ soit faux, à cause du prédicat précédent il existe a' tel que $a' \prec a$ et $P(a')$ est faux. Pour la même raison, on peut trouver $a'' \prec a'$ tel que $P(a'')$ est faux, et ainsi de suite. On construit donc une chaîne infinie

$$a \succ a' \succ a'' \succ \cdots,$$

ce qui contredit l'hypothèse de bonne fondation de A.

Axiome d'induction

voir aussi Entiers de Peano.

Le principe d'induction peut être prouvé sous une hypothèse de bonne fondation, comme nous l'avons fait dans la section précédente. En ce qui concerne les entiers, on préfère souvent d'énonce le principe d'induction comme un axiome, c'est à dire comme une vérité évidente qui ne nécessite pas de démonstration.

C'est le cas, notamment, dans le système axiomatique de Peano, où le principe d'induction constitué l'un des neuf axiomes définissant les entiers.

Le principe d'induction peut être exprimé par un *schema d'axiomes du premier ordre*, i.e. par une infinité de prédicats de la forme

$$\big(P(0) \wedge \forall n.\,(P(n) \to P(n+1))\big) \Rightarrow \forall n.\,P(n),$$

avec un axiome pour chaque prédicat P avec une variable libre.

Il peut aussi être exprimé par un *prédicat du second ordre*, i.e. par un prédicat où on s'autorise à quantifier les prédicats

$$\forall P.\,\Big(\big(P(0) \wedge \forall n.\,(P(n) \to P(n+1))\big) \Rightarrow \forall n.\,P(n)\Big).$$

La différence entre les deux formulations est trop subtile pour être abordée dans ce cours, mais il est bon de savoir qu'elle a des conséquences importantes.

Les axiomes d'induction sont équivalents à la bonne fondation, c'est à dire que sur tout ensemble sur lequel on impose un axiome d'induction on peut construire un ordre bien fondé (et inversement, comme cela a été montré dans la section précédente).

Récursion

La récursion est une technique de définition d'objets mathématiques. Comme pour l'induction, on *définit récursivement* (ou *inductivement*) une famille d'objets en deux étapes:

- On donne un nombre fini de définitions explicites d'objets de *base* ;
- On définit tous les autres objets en fonction d'objets *plus petits* définis précédemment.

Par exemple, la définition des entiers selon les axiomes de Peano peut être vue comme une construction récursive de l'ensemble $\mathbb{N}$. D'autres objets qu'on définit récursivement sont les suites (dites *récurrentes*), les fonctions, et, en informatique, les arbres, les files, les piles, etc.

Exemples

Les constructions récursives que l'on rencontre le plus souvent définissent des fonctions sur les entiers ou sur d'autres structures dotées d'un ordre bien fondé.

Fonctions récursives sur les entiers

Une fonction de $\mathbb{N}$ dans $\mathbb{N}$ est définie récursivement de la façon suivante:

- par un *cas de base* (ou *cas particulier*) qui donne une valeur pour la fonction à un entier fixé (en général 0 ou 1): par exemple $f(0) = 1$;
- par un *cas inductif* (ou *cas général*) qui définit la valeur à un entier n en fonction de sa valeur à un ou plusieurs entiers plus petit: par exemple $f(n) = f(n-1) + n$.

Factorielle

La fonction factorielle $n!$ peut être définie récursivement de la façon suivante:

- $0! = 1$,
- $n! = (n-1)! \cdot n$ pour $n > 0$.

Suite de Fibonacci

La suite de Fibonacci est un exemple de définition récursive basée non seulement sur la valeur de la fonction à $n-1$, mais aussi sur sa valeur à $n-2$. Par conséquent, deux cas de base sont nécessaires afin d'éviter une définition mal formée.

- $F(0) = 0$,
- $F(1) = 1$,
- $F(n) = F(n-1) + F(n-2)$ pour $n > 1$.

La suite de Fibonacci apparaît régulièrement en mathématiques, allez voir sa page Wikipedia est un bon point de départ pour se faire une culture là dessus. Régulièrement, on trouve même des biologistes, des historiens et toute autre sorte de savants qui prétendent, à raison ou à tort, avoir rencontré la suite de Fibonacci dans leurs études. Cette page Wikipedia vous donnera quelques idées.

Nombres de Catalan

Les nombres de Catalan sont un autre exemple de définition récursive basée non seulement sur la valeur de la fonction à $n-1$. Ils sont définis ainsi:

- $C(0) = 1$,
- $C(n+1) = \sum_{i=0}^{n} C(i) \cdot C(n-i)$ pour $n > 0$.

Les nombres de Catalan apparaissent aussi très fréquemment, dès que l'on travaille avec des arbres binaires. On peut montrer l'égalité suivante

$$C(n+1) = \frac{2(2n+1)}{n+2} C(n) \qquad \text{pour tout } n > 0.$$

Exponentielle

La fonction exponentielle est définie de la façon suivante:

- $\exp(0) = 1$,
- $\exp(n) = e \cdot \exp(n-1)$ pour $n > 0$.

Cependant, une autre définition tout à fait légitime est:

- $\exp(0) = 1$,
- $\exp(n) = \exp(n/2)^2$ si $n > 0$ est pair,
- $\exp(n) = e \cdot \exp(\lfloor n/2 \rfloor)^2$ si n est impair.

Cette deuxième définition a un intérêt algorithmique, puisque elle est à la base de l'algorithme d'exponentiation binaire.

Indicatrice d'Euler

L'**indicatrice d'Euler**, généralement notée $\varphi(n)$, compte combien d'entiers plus petits que n n'ont pas de facteur commun avec n. Formellement elle est définie par

$\varphi(n) =$ nombre d'entiers $m \leq n$ tels que $\operatorname{pgcd}(n, m) = 1$.

Si p^e est une puissance d'un nombre premier p, il est facile de se convaincre que

$$\varphi(p^e) = p^{e-1}(p-1).$$

Plus généralement, si m et n sont deux entiers *premiers entre eux* (i.e., tels que $\operatorname{pgcd}(m, n) = 1$), on peut démontrer que

$$\varphi(mn) = \varphi(m)\varphi(n).$$

Ces deux dernières égalités peuvent être prises comme définition récursive de φ à la place de la première définition.

Autres fonctions récursives

On peut définir des fonctions ou des propriétés récursives sur autre chose que les entiers. Les chaînes de caractères, par exemple, s'y prêtent très bien.

Palindrômes

Une chaine de caractères est palindrome si elle se lit de la même façon de gauche à droite et de droite à gauche. Par exemple, *abba* et *radar* sont palindromes. On peut définir la palindromie de la façon suivante:

- La chaîne vide est palindrome,
- toute chaîne d'un seul caractère est palindrome,
- si S est un palindrome, alors xSx est palindrome pour tout caractère x.

Expressions bien parenthésées

On s'intéresse aux chaînes de caractères dans lesquelles les parenthèses ouvrantes et fermantes sont bien distribuées. Par exemple, $(ab(cd))$ est bien parenthésée, mais $(ab))(cd$ ne l'est pas. En ignorant les caractères qui ne sont pas de parenthèses (et qui ne jouent aucun rôle), on se ramène à étudier les chaînes constituées exclusivement de parenthèses. Une expression bien parenthésée est définie alors de la manière suivante:

- La chaîne vide est bien parenthésée,
- si A et B sont bien parenthésées, alors AB est bien parenthésée,
- si A est bien parenthésée, alors (A) est bien parenthésée.

Exercice: On peut démontrer que le nombre d'expressions bien parenthésées contenant exactement n parenthèses ouvrantes est égal au nombre de Catalan $C(n)$. Prouvez cette propriété par récurrence. (**Suggestion:** Commencez par trouver une autre définition récursive des expressions bien parenthésées qui n'utilise qu'un cas de base et un cas inductif).

Définition

De manière intuitive, une fonction (ou une propriété) récursive est une fonction qui est définie en termes d'elle même. Ceci ne suffit pas à définir correctement une fonction, car par exemple l'égalité $f(x) = f(x)$ ne définit aucune fonction en particulier (toute fonction a cette propriété).

Formellement, une fonction récursive peut être définie sur tout ensemble muni d'un ordre bien fondé, ou de façon équivalente en présence d'un axiome d'induction.

Soit A un ensemble muni d'une relation d'ordre bien fondée $\prec$, et soit B un autre ensemble quelconque. Une fonction (récursive) $f : A \to B$ est *bien définie* (ou *cohérente* ou *bien fondée*) si pour tout $a \in A$ la définition de f ne fait intervenir que des valeurs $f(a')$ pour $a' \prec a$.

Nota bene: Cette définition définition n'impose pas que f soit *nécessairement* définie en termes d'elle même pour tout a (considérez, par exemple, la valeur de Fibonacci en 0 et 1). Au contraire, elle **impose** que la fonction **ne soit pas définie en termes d'elle même aux éléments minimaux** de A. En effet, si a est un élément minimal (par exemple $0 \in \mathbb{N}$), il n'y a aucun élément plus petit que a, donc la définition de f ne peut faire recours à aucune autre valeur de f.

Exercice: Pour chacune des fonction récursives définies précédemment, trouvez l'ordre bien fondé qui garantit la cohérence de la définition.

Exercice: Démontrez par Induction que toute fonction bien définie à une valeur unique à chaque élément de A.

Récursion en informatique

La récursivité est tellement omniprésente en mathématiques, que tous les langages de programmation modernes permettent d'écrire des programmes récursifs, c'est à dire des programmes qui s'appellent eux mêmes.

Factorielle

L'exemple suivant, écrit en Python définit une fonction qui calcule la factorielle $n!$ pour tout n positif.

```
def fact(n):
    if n < 0:
        raise Error
    elif n == 0:
        return 1
    else:
        return n * fact(n-1)
```

Exponentiation

En imitant la définition récursive de l'exponentielle, on peut écrire une fonction qui calcule c^n pour tout c et pour tout entier n.

```
def exp(c, n):
    if n < 0:
        return 1 / exp(c, -n)
    elif n == 0:
        return 1
    else:
        return c * exp(c, n-1)
```

Diviser pour régner

Le principe *diviser pour régner* est l'un des outils fondamentaux en algorithmique. L'idée consiste à résoudre un problème en le coupant en deux sous-problèmes de taille à peu près égale, qu'on résout de façon récursive. Ce principe permet souvent d'obtenir des gains d'efficacité considérables.

Exponentiation binaire

En utilisant la seconde définition récursive pour l'exponentielle, on peut ré-écrire la fonction précédente de la manière suivante

```
def bin_exp(c, n):
    if n < 0:
        return 1 / bin_exp(c, -n)
    elif n == 0:
        return 1
    else:
        tmp = bin_exp(c, n // 2)
        tmp *= tmp
        if n % 2 == 1:
            tmp *= c
        return tmp
```

Un appel d'une fonction à elle même est dit un *appel récursif*. Si on compte combien d'appels récursifs ont lieu lors d'un appel à `exp(2,10)` on vérifie que celui-ci appelle `exp(2,9)`, qui appelle `exp(2,8)` et ainsi de suite, pour un total de 10 appels. D'un autre côté un appel à `bin_exp(2,10)` engendre un appel à `bin_exp(2,5)`, puis à `binexp(2,2)`, puis à `bin_exp(2,1)` et enfin à `bin_exp(2,0)`, pour un total de 4 appels. En général, `exp` fera n appels récursifs, alors que `bin_exp` en fera seulement $\lceil \log_2 n \rceil$.

L'algorithme `bin_exp` est appelé *exponentiation binaire*, à cause du lien avec l'écriture de n en base 2.

Exercice: Étudiez le rapport entre l'exponentiation binaire et l'écriture de n en base 2 et donnez-en une version itérative (c'est à dire avec des boucles `for` ou `while` et pas d'appels récursifs).

Pour approfondir

La Tour de Hanoï

Allez voir cet exercice