# SAVOIRS

# Thème 1 - Les outils de l'informatique

## [S1.1] Les bases de l'algorithmique

Un algorithme peut se définir comme une suite d'instructions qui doivent, lorsqu'elles sont appliquées dans l'ordre et correctement, conduire à un résultat voulu.

Une comparaison courante consiste à faire le parallèle entre algorithme et recette de cuisine : si l'on suit toutes les étapes pas à pas, on doit bien obtenir le même résultat que sur la photo du livre de cuisine.

Cette analogie est limitée car elle omet une particularité essentielle des algorithmes : la prise de décision. En effet, les branchements conditionnels ou les boucles conditionnelles vont permettre d'adapter les instructions à effectuer aux conditions rencontrées.

Les algorithmes sont aujourd'hui omniprésents dans la vie courante, avec le développement très rapide des outils numériques : la recherche du meilleur itinéraire sur un logiciel de guidage, les propositions ciblées d'articles sur les réseaux sociaux, la reconnaissance de la voix dans les assistants personnels des téléphones…

L'exemple le plus connu d'algorithme en mathématiques est sans doute l'algorithme d'Euclide qui permet de décrire en quelques phrases une méthode simple pour calculer le plus grand diviseur commun de deux nombres entiers quelconques. Comme son nom l'indique, cette méthode était connue dès l'Antiquité.

Deux questions théoriques intéressent les informaticiens qui étudient les algorithmes.

- Un algorithme donné termine-t-il et est-il correct ? C'est-à-dire va-t-il toujours renvoyer le résultat attendu sans rencontrer de bug ou bloquer sur un cas particulier inattendu ?
- Est-il efficace ? Savoir qu'un algorithme va retourner le résultat recherché, mais seulement après plusieurs milliards d'années de calcul n'est pas forcément intéressant…

Ces études ne sont pas au programme des classes de BCPST, mais peuvent être effleurées empiriquement.

### ◇ Les fonctions

Une fonction, en informatique, peut être vue comme un conteneur pour un algorithme : on fournit à la fonction les données nécessaires au calcul et celle-ci nous renvoie le ou les résultats.

L'avantage des fonctions est de pouvoir n'écrire qu'une seule fois un algorithme et de le réutiliser ensuite facilement pour construire des procédures plus complexes. Par exemple, on peut imaginer définir une fonction qui calcule le discriminant d'une équation du second degré avant de l'utiliser dans un algorithme qui résout cette équation.

La première étape dans l'écriture d'une fonction est de bien réfléchir aux entrées et aux sorties de celle-ci.

◇ **Les branchements conditionnels**
Il est important dans la construction d'un algorithme d'avoir la possibilité d'écrire des alternatives suivant les cas rencontrés.

Par exemple, dans la résolution d'une équation polynomiale du second degré, le signe du discriminant va déterminer la suite du raisonnement.

En pseudo-code, on écrira :

Sans alternative

**si** *condition* **alors**
└ commandes

Avec alternative

**si** *condition* **alors**
| commandes
**sinon**
└ commandes

◇ **Les boucles**
Les algorithmes réclament souvent de faire des calculs répétitifs, c'est-à-dire d'enchaîner un grand nombre de fois les mêmes commandes.

Par exemple, pour faire le produit de deux matrices carrées d'ordre $n$, il va falloir calculer $n^2$ sommes comportant chacune $n$ produits de coefficients.

Les boucles vont permettre d'écrire facilement de telles répétitions. Il existe deux types de boucles et il y a une question à se poser pour choisir quel type utiliser : connaît-on à l'avance le nombre de répétitions ?

- Si c'est le cas, on choisira une boucle inconditionnelle, dite boucle « Pour » :

  Sans précision, on comprend que la variable de boucle $k$ va prendre toutes les valeurs entières comprises entre $a$ et $b$, en commençant par $a$, puis $a+1$, etc.

  Suivant les langages, il est possible de modifier le pas de la boucle ; par exemple, à chaque itération, la valeur de $k$ augmente de deux.
- Sinon, on choisira une boucle conditionnelle, dite boucle « Tant que » dont les répétitions seront soumises à la vérification d'une condition.

  Pour une telle boucle, il est indispensable de s'assurer que les variables qui apparaissent dans la condition vont évoluer de telle sorte que cette condition ne sera plus vérifiée à partir d'un certain moment et, ainsi, éviter de tomber dans une boucle infinie.

Boucle inconditionnelle

**pour** *k allant de a à b* **faire**
└ commandes

Boucle conditionnelle

**tant que** *condition* **faire**
└ commandes

◇ **La récursivité**
Les fonctions récursives sont une manière élégante de traduire des définitions mathématiques récurrentes.

L'idée est d'écrire une fonction qui dans quelques cas simples va renvoyer directement le résultat et qui sinon va s'appeler elle-même.

L'exemple le plus classique est sans doute celui de la factorielle, qui repose sur la formule bien connue :

$$0! = 1! = 1 \qquad \text{et} \qquad \forall n \in \mathbb{N}^*,\ n! = n \times (n-1)!.$$

Boucle

**Données :** $n$ entier naturel
**début**
  $p \leftarrow 1$
  **pour** $i$ *allant de* 1 *à* $n$ **faire**
    $p \leftarrow i * p$
  **retourner** $p$

Fonction récursive

**Données :** $n$ entier naturel
**début**
  **si** $n \leqslant 1$ **alors**
    **retourner** 1
  **sinon**
    **retourner** $n *$ `Fact`$(n-1)$

## [S1.2] Les structures de données

Un point important avant de se lancer dans l'écriture d'un projet informatique est de réfléchir à la manière d'organiser les données que l'on va manipuler dans les algorithmes.

Un bon choix pour la structure de données va faciliter les opérations qui vont suivre et améliorer l'efficacité du code car les méthodes associées seront bien adaptées au problème concret que l'on cherche à modéliser.

On présente ici quelques structures de données classiques rencontrées en BCPST.

### ◇ Les chaînes de caractères

En informatique, un caractère est un symbole quelconque : une lettre, un chiffre, un élément de ponctuation, un espace… Les chaînes de caractères sont donc utiles pour manipuler les données textuelles.

Pour une chaîne de caractères fixée, on connaît sa longueur et on peut accéder à un caractère d'indice donné. Il est aussi possible d'extraire des sous-chaînes directement.

> ✓ On adoptera la convention « la numérotation des indices des caractères pour une chaîne de caractères commence à 0 ». Ainsi, les caractères d'une chaîne de longueur $\ell$ sont numérotés de 0 à $\ell - 1$.

### ◇ Les listes

Les listes permettent de garder en mémoire un grand nombre de données (souvent numériques) ; par exemple, on pourra dans une liste calculer et stocker les $n$ premiers termes d'une suite.

Ici encore, on connaît la longueur d'une liste donnée et on peut accéder à un élément précis. Pour une liste, il va être possible de modifier cet élément.

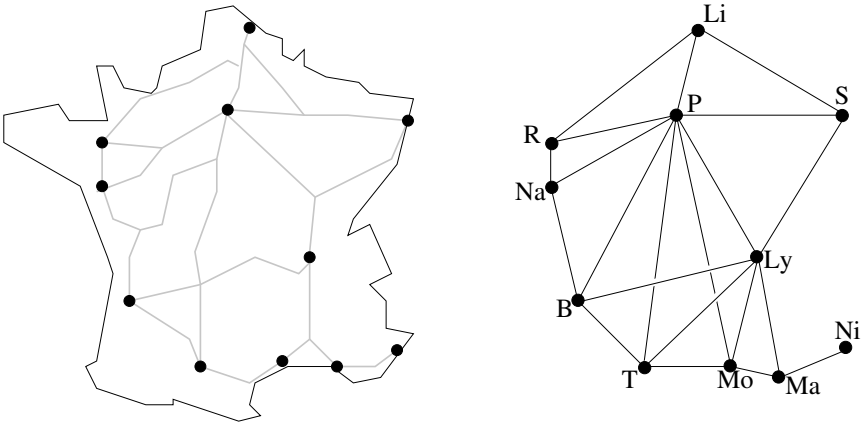
> ✓ Pour les listes aussi, on adopte la convention « la numérotation des éléments commence à 0 ».

Une liste de listes va permettre de représenter facilement une matrice.

◇ **Les graphes**

Les graphes permettent de mettre en évidence des liens entre les objets.

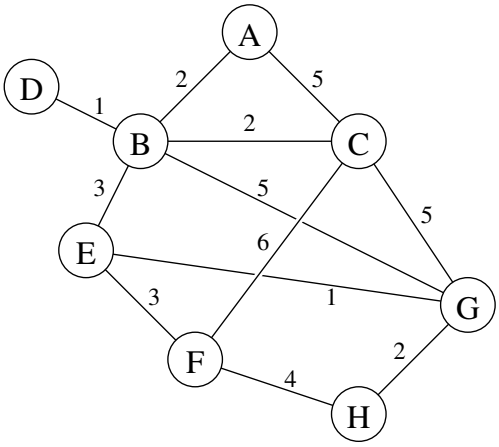
Un exemple très parlant consiste à schématiser une carte routière : les villes de la carte deviennent les *sommets* du graphe, et les routes les *arêtes* du graphe.




Les graphes sont très utilisés en modélisation, souvent en lien avec la notion de réseau (réseaux informatiques et partage des ressources, réseaux sociaux pour « relier » les personnes), mais aussi en probabilités (arbres pondérés) ou encore en biologie (par exemple pour les liens génétiques entre des espèces).

Il y a deux grandes méthodes pour représenter un graphe :

- à l'aide d'une *liste d'adjacence* : on numérote les sommets du graphe et pour chacun d'entre eux, on liste ses voisins (avec la pondération associée) ;
- à l'aide d'une *matrice d'adjacence* : si le graphe possède $n$ sommets, une matrice carrée d'ordre $n$ permet de modéliser le graphe. Plus précisément, si le coefficient d'indice $(i, j)$ est non nul, alors le graphe possède une arête entre les sommets $i$ et $j$ dont la pondération est donnée par la valeur du coefficient.



Par exemple, pour le graphe précédent, on numérote les sommets de 0 à 7 par ordre alphabétique. Le sommet $A$ est relié aux sommets $B$ (avec la pondération 2) et $C$ (avec la pondération 5), donc le premier élément de la liste d'adjacence sera $[(1,2),(2,5)]$ et la première ligne de la matrice d'adjacence sera (0 2 5 0 0 0 0 0).

Ainsi, la liste d'adjacence est donnée par :

```
[[(1,2),(2,5)],
 [(0,2),(2,2),(3,1),(4,3),(6,5)],
 [(0,5),(1,2),(5,6),(6,5)],
 [(1,1)],
 [(1,3),(5,3),(6,1)],
 [(2,6),(4,3),(6,2)],
 [(2,5),(4,1),(7,2)],
 [(5,4),(6,2)]]
```

et la matrice d'adjacence par :

$$\begin{pmatrix} 0 & 2 & 5 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 2 & 1 & 3 & 0 & 5 & 0 \\ 5 & 2 & 0 & 0 & 0 & 6 & 5 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 3 & 1 & 0 \\ 0 & 0 & 6 & 0 & 3 & 0 & 0 & 4 \\ 0 & 5 & 5 & 0 & 1 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 4 & 2 & 0 \end{pmatrix}.$$

✓ La matrice d'adjacence est symétrique. C'est logique car le graphe est *non orienté* : si on peut aller du sommet $C$ vers le sommet $F$, alors on peut aussi aller du sommet $F$ vers le sommet $C$.

On peut ajouter des sens de parcours aux arêtes du graphe (par exemple pour modéliser une rue en sens unique sur une carte routière), la matrice d'adjacence n'est alors plus symétrique.

# Thème 2 - Les outils de l'analyse

Dans ce chapitre, on revient sur le programme d'analyse de première année. L'idée n'est évidemment pas de reprendre l'intégralité de ce programme mais de se focaliser sur les notions et les théorèmes incontournables pour bien aborder les nouveautés de la seconde année (pour avoir tous les détails de ces résultats, le lecteur est invité à reprendre les passages concernés dans le livre de première année).

En premier lieu, on se focalisera sur la notion de limite, outil central de l'analyse, puis, dans la seconde partie, on reprendra les théorèmes essentiels pour les problèmes d'analyse.

Les rappels concernant le calcul d'intégrales seront faits dans le chapitre 3.

## [S2.1] Limites et équivalents

Hormis le résultat sur les suites adjacentes, toutes les propositions énoncées ci-dessous le seront pour des fonctions réelles d'une variable réelle. Pour autant, ils sont facilement adaptables aux suites réelles à l'aide du résultat suivant :

$$\begin{cases} \lim\limits_{x\to a} f(x) = \ell \\ \lim\limits_{n\to+\infty} u_n = a, \end{cases} \quad \Longrightarrow \lim_{n\to+\infty} f(u_n) = \ell.$$

**◇ Utilisation des équivalents et des développements limités pour le calcul des limites**

Face à une forme indeterminée, le calcul de la limite peut parfois sembler compliqué. On rappelle ici deux résultats qui permettent de débloquer la situation.

- En premier lieu, on peut obtenir un équivalent à partir d'un développement limité.

  En effet, si $f$ admet un developpement limité au voisinage de $a$ dont la partie régulière est non nulle, alors $f$ est équivalente au voisinage de $a$ au premier terme de ce développement limité (c'est-à-dire au terme prépondérant).

- Ensuite, on peut trouver une limite à partir d'un équivalent.

  En effet, si $f \underset{a}{\sim} g$ et si $f$ admet une limite $\ell$ (finie ou infinie) en $a$, alors $g$ admet aussi cette limite $\ell$ en $a$.

**◇ Relation d'ordre et limites**

Les inégalités, omniprésentes en analyse, donnent aussi des renseignements importants pour l'existence ou la valeur des limites.

- *Passage à la limite dans une inégalité*
  Si l'on a $f \leqslant g$ sur un voisinage de $a$ et si $f$ et $g$ admettent une limite finie en $a$, alors :

$$\lim_{x\to a} f(x) \leqslant \lim_{x\to a} g(x).$$

✓ Attention, ce résultat ne donne pas l'existence des limites ; il faut savoir *a priori* qu'elles existent.

✓ Attention également au fait que, même si l'on dispose initialement d'inégalités strictes, le passage à la limite ne les conserve pas et les transforme en inégalités larges.

- *Limite par encadrement (ou théorème des gendarmes)*
  Si l'on a $f \leqslant g \leqslant h$ sur un voisinage de $a$ et si $f$ et $h$ admettent une même limite *finie* $\ell$ en $a$, alors $g$ admet aussi une limite finie en $a$ qui vaut $\ell$.

  ✓ Ce résultat donne, lui, l'existence et la valeur de la limite de la fonction $g$.

- *Limite par comparaison*
  Si $f \leqslant g$ sur un voisinage de $a$ et si $\lim\limits_{x\to a} f(x) = +\infty$, alors $\lim\limits_{x\to a} g(x) = +\infty$.
  Si $f \leqslant g$ sur un voisinage de $a$ et si $\lim\limits_{x\to a} g(x) = -\infty$, alors $\lim\limits_{x\to a} f(x) = -\infty$.

  ✓ Ces résultats donnent l'existence et la valeur de la limite.

◇ **Sens de variation et limites**

- *Théorème de la limite monotone*
  Si $f$ est une fonction croissante sur un intervalle $I = ]a, b[$ de $\mathbb{R}$, alors les limites de $f$ en $a$ et $b$ existent et :
  - si $f$ est majorée par un réel $M$ sur $I$, alors la limite de $f$ en $b$ est finie, sinon $\lim\limits_{x\to b} f(x) = +\infty$ ;
  - si $f$ est minorée par un réel $m$ sur $I$, alors la limite de $f$ en $a$ est finie, sinon $\lim\limits_{x\to a} f(x) = -\infty$.

  ✓ Bien évidemment, ce théorème s'adapte facilement au cas d'une fonction décroissante.

  ✓ Autrement dit, une fonction monotone sur un intervalle ouvert admet toujours, en tout point de cet intervalle, une limite finie à droite et à gauche. Aux bornes de cet intervalle, elle admet là aussi des limites qui sont finies ou infinies.

  ✓ L'analogue pour une suite est : une suite croissante est soit majorée, auquel cas elle converge, soit non majorée, auquel cas elle diverge vers $+\infty$ ; une suite décroissante est soit minorée, auquel cas elle converge, soit non minorée, auquel cas elle diverge vers $-\infty$.

  ✓ Ce résultat permet de prouver le critère de convergence des séries ou des intégrales (la positivité de la fonction ou de la suite assurant la croissance de la primitive ou de la somme partielle).

- Deux suites $(u_n)_n$ et $(v_n)_n$ sont dites adjacentes lorsque :
  - l'une est croissante et l'autre décroissante ;
  - la différence $(v_n - u_n)_n$ converge vers 0.

  Dans ces conditions, les deux suites convergent vers une même limite.