LA NOTION D'ALGORITHME

« L' informatique n'est pas plus la science des ordinateurs que l'astronomie n'est celle des télescopes »
(E.Dijkstra, Turing award 1972)

On peut donc parler de science informatique à part entière. Si l'informatique a d'abord été au service des autres disciplines, elle a aussi permis l'émergence de nouveaux concepts comme celui de bases de données, de réseaux, d'intelligence artificielle, etc ... L'informatique c'est aussi une nouvelle façon de penser, d'apprendre ...

Citer des domaines d'utilisation de l'ordinateur

Quelques rubriques sont en place, à illustrer ou à compléter ...

- Mettre en forme : traitement de texte, tableurs, présentation, ...
- Exécuter des calculs : calculs numériques, calculs formels, ...
- Simuler : jeux vidéo, simulateurs (avions, navires, ...)
- Traiter les données : archiver, stocker, gérer l'accès à des bases de données
- Connecter, contrôler à distance : envoyer/recevoir des messages, piloter un drone, un satellite, un robot, une imprimante, gérer un réseau, ...
- Gestion : gestion des paies, comptabilité, comptes bancaires ...
- Images : analyse et retouche d'images, ...
- d'autres idées ? ...

Le caractère polyvalent de l'ordinateur

L'ordinateur n'est pas comme une machine à café ou tout autre automate prêt à fonctionner à sa sortie d'usine et qui va toujours effectuer la même tâche. C'est plus qu'une calculatrice qui elle est spécialisée.
Au contraire l'ordinateur ne sait qu'exécuter qu'un petit nombre de tâches élémentaires (additionner deux registres, faire des opérations logiques de base ...) mais il le fait très vite (plusieurs milliards d'opérations par secondes) et il le fait bien.
En organisant ces tâches élémentaires (travail du système d'exploitation et du programmeur), on peut lui demander de réaliser des tâches plus complexes. On conserve la main sur la façon d'organiser ces tâches. De là provient la polyvalence de l'ordinateur.
En ajoutant des procédés pour communiquer avec l'extérieur, l'ordinateur permet d'échanger des informations d'où le nom "d'informatique".

Comment s'y prendre ?

Comment passer d'un problème concret à une tâche réalisable par un ordinateur ?
Le problème concret est en général formulé dans le langage commun avec tout ce que cela comporte d'ambiguïté, de sous-entendus, de références culturelles. Tel quel, cela n'est pas transmissible ni intelligible pour l'ordinateur.

Il faut tout d'abord se donner une méthode pour résoudre le problème concret.

Définition

Un algorithme est une suite d'instructions à exécuter dans un ordre bien précis dans le but de résoudre un problème donné, de rendre un service spécifié.

En général, les instructions à suivre font référence à des données fournies au départ (ou en cours de traitement). L'algorithme doit pouvoir s'adapter à ce jeu de données.
Il y a donc un jeu de données au départ et un service rendu à l'arrivée, l'algorithme décrivant donc comment passer de l'un à l'autre.

Un jeu de données particulier pour les données attachées au problème à résoudre s'appelle une instance du problème.

Un algorithme est de nature déterministe : avec les mêmes entrées, il produira les mêmes résultats.

Un programme est la traduction dans un langage d'un ou plusieurs algorithmes.

Un programme court réalisant une tâche simple est appelé un script.
En général, un programme retourne quelque chose : il s'agit du résultat correspondant à la résolution du problème posé.
On parle d'implémentation d'un algorithme lorsqu'on le traduit dans un langage compréhensible par l'ordinateur.

Dans les exemples suivants, identifier ce que peut-être un jeu de données et ce qui est retourné.

- Se rendre au lycée Paul Valéry
- Résolution d'un trinôme du second degré à coefficients réels.
- Recette de cuisine.
- Déterminer si un point M est à l'intérieur du triangle ABC ou non.
- Organiser « au mieux » le parcours d'un voyageur de commerce devant se rendre dans plusieurs villes.
- ...

Ingrédients nécessaires

Il s'agit d'identifier les concepts universels qui sont nécessaires pour couvrir une large gamme de méthodes qui apparaissent dans la description des algorithmes.

On va examiner un exemple concret. **Supposons donc qu'on veuille cuisiner un quatre-quart.**
La recette à suivre est la suivante :

> mélanger 4 oeufs frais entiers, rajouter le même poids que les oeufs pour le sucre, le beurre et la farine
> rajouter une pincée de sel, malaxer. Faire cuire 50 minutes environ.

Un robot saurait-il appliquer la recette telle quelle ? Pourquoi ?

> Il n'y a pas assez de détails donnés : faut-il préchauffer le four ? ramollir le beurre ? la pâte doit-elle être homogène ?
> Il y a des références culturelles implicites : le four doit être allumé, le mélange doit être mis dans un moule; oeufs entiers, cela ne comprend pas la coquille ! Le beurre se trouve au frigo, la farine dans le placard ...
> Il faut tout préciser, envisager tous les cas de figure : dans quel ordre on rajoute le sucre, le beurre et la farine; malaxer jusqu'à ce que la pâte soit homogène en précisant comment tester si c'est bien le cas; où se trouvent les ingrédients, que faire s'il n'y en a pas assez, etc ... Il faut dire ce que signifient "une pincée ", "environ 50 minutes" ...
> La recette doit pouvoir s'adapter si les quantités changent, par exemple si on veut faire un gâteau pour 10 personnes au lieu de 5 personnes. On peut préciser des paramètres optionnels : rajouter ou non de la vanille, un zeste de citron ...
> On peut faire appel à des savoir-faire décrits ailleurs sans avoir à les décrire à nouveau : faire fondre le beurre au bain-marie, séparer les blancs des jaunes d'oeufs, ... Il faut juste connaître ce qui est fourni en entrée et le résultat final du procédé.

De cet exemple, on peut dégager plusieurs ingrédients à mettre en oeuvre pour décrire un algorithme :

- **<u>Traitement séquentiel</u> des opérations : l'algorithme spécifie l'ordre des instructions.**
- **Pour anticiper tous les cas de figure possibles, on doit inclure des tests du type « <u>si ... alors ... sinon</u> ... »**
- **Pour préciser la durée d'exécution ou le nombre de répétitions de certaines instructions, on doit inclure des commande du type « <u>tant que</u> » ou bien « répéter ».**
- **Certains blocs d'instructions peuvent être regroupés car ils pourront être réutilisés tels quels**
  Ces blocs s'appellent des <u>fonctions</u> .
  Les fonctions sont à la base du traitement <u>modulaire</u> des algorithmes : décomposer un problème compliqué en problèmes élémentaires.
- **L'algorithme doit tenir compte des données qui sont susceptibles de prendre différentes valeurs dans un domaine spécifié. Ces données sont des <u>variables</u>.**

Ces fonctionnalités sont disponibles sur un ordinateur grâce à l'architecture du processeur. Voir le document sur l'architecture des ordinateurs pour plus de détails.

Qualités d'un algorithme

Un algorithme doit aussi avoir des qualités auxquelles il faut veiller dès la conception.

- **Correction**
  Il faut qu'il accomplisse sa tâche !
  Un algorithme est correct si pour chaque instance autorisée, l'algorithme se termine en produisant la sortie attendue.

- **Efficacité**
  Un algorithme peut-être correct mais inefficace : soit il prend trop de temps, soit il monopolise trop de ressources. L'efficacité se mesure par la complexité de l'algorithme.
  La concision d'un algorithme n'est pas forcément un gage d'efficacité
  Pour étudier la complexité, on utilise les outils de l'analyse asymptotique.
- **Terminaison**
  Un algorithme doit se terminer ! Cela peut ne pas être le cas si une séquence d'instructions se répète indéfiniment. On parle alors de bug.

Il y a souvent des bugs résiduels d'où la phase de mise au point des programmes
On peut tester la correction et la terminaison d'un algorithme à l'aide d'une batterie de tests, en prenant un échantillon significatif de paramètres du programme.
On peut aussi chercher à prouver la correction et la terminaison (à l'aide d'invariants de boucle, du formalisme de la logique ...).
Malgré toutes ces précautions, souvent des bugs susbsistent (voir ci-dessous).

On peut identifier plusieurs sortes de bugs :

- **erreur sémantique : le programme fournit bien un résultat mais ce n'est pas celui qui était prévu. Par exemple, les instructions ne sont pas forcément décrites dans le bon ordre, les boucles ne sont pas forcément parcourues le bon nombre de fois, les initialisations de variables sont incorrectes ...**

- **erreur de syntaxe : on doit utiliser un langage pour rendre l'algorithme intelligible par l'ordinateur. Un algorithme peut-être correct et son implémentation dans le langage choisi incorrecte.**

- **erreurs à l'exécution : certaines instances du problème peuvent conduire à une instruction impossible à effectuer (comme une division par 0).**

- **Mauvais emploi de bibliothèques : une instruction utilise un autre programme de façon inappropriée. Par exemple, le type ou le nombre des paramètres requis par ce programme auxiliaire ne sont pas respectés. C'est pourquoi un algorithme doit être documenté : nature des paramètres, nature de ce qui est retourné.**

Une erreur de conception ou de programmation peut avoir des conséquences dramatiques.

Juste quelques exemples pour frapper les esprits :

- En 2010, un mini Krach provoqué par des ordres d'achat-vente échappant à tout contrôle (trading haute fréquence) a provoqué une perte de 9% de la capitalisation boursière.
- Installation ratée d'un nouveau logiciel de courtage chez Knight Capital en août 2012 : perte de 461 millions $ et rachat par un concurrent.
- Introduction en bourse de la plate-forme boursière BATS en 2012 : bug dans l'algorithme d'introduction de nouvelles actions. Le cours tombe de 16$ à quelques centimes.
- Bugs informatiques récurrents causant de graves disfonctionnement sur la paye des militaires dans le logiciel Louvois de paiement des soldes.
- Mise au point du F16 américain : comportements aberrants : au sol, le train d'atterrissage accepte de se rentrer, au passage de l'équateur, l'avion pique du nez (erreur dans les formules de coordonnées sphériques, l'altitude devenant négative).
- Sonde Phobos Grunt russe : échec en 2012 par perte de contrôle de la sonde, dysfonctionnement logiciel.
- Crash d'ariane 5 lors de son premier vol : des changements de direction aberrants parce que les entiers étaient codés sur 16 bits et que certaines accélérations prenaient des valeurs qui ne pouvaient pas être codées sur 16 bits.
- Sonde Mars climate orbiter : la sonde s'écrase sur Mars. Problèmes de conversion système métrique/unités anglo-saxonnes, le projet faisant intervenir des équipes internationales utilisant les deux systèmes d'unité.
- Sonde européenne Schaparelli (octobre 2016) s'écrase sur Mars : une sonde mesurant les accélérations a indiqué brièvement une valeur aberrante, le calculateur alors une altitude de -2km et a considéré que la sonde avant atterri. Il a alors commandé l'arrêt des rétro-fusées et l'éjection du bouclier thermique. La sonde est alors tombée en chute libre.
- USS York Town 1997 : un porte-avion américain perd le contrôle de ses moteurs à cause d'une division par 0 sans contrôle d'exception.
- Elections en Allemagne au Schleswig Holstein en 1992 : seuil de 5% pour avoir des élus ; la barre est atteinte par les Verts après arrondi du pourcentage 4.97 %. Après rectification, ils perdent leur siège au profit du SPD qui devient du coup majoritaire.

Résumé

On retiendra comme caractéristiques universelles des algorithmes :

- **5 ingrédients permettant de traiter tous les problèmes :**
  - **traitement séquentiel**
  - **variables**
  - **boucles**
  - **tests conditionnels (conditions de branchements)**
  - **fonctions (ou procédures) pour le traitement modulaire**

- **3 qualités :**
  - **correction**
  - **terminaison**
  - **complexité**

Comment l'ordinateur peut-il exécuter un programme ?

Comment passe-t-on d'un programme décrit en langage courant à des instructions compréhensibles pour l'ordinateur ?
L'ordinateur assure un traitement de l'information. La communication avec l'ordinateur se faisant par échange de données via des entrées-sorties (l'unité centrale échangeant les données avec les supports de stockage, l'écran, les réseaux, ...).
Toutes ces données doivent être codées.
Le codage se fait en binaire. Une unité élémentaire d'information s'appelle un bit (binary digit) : il correspond à 0 ou 1. Une suite brute de bits peut coder n'importe quoi a priori. L'architecture interne de l'ordinateur permet d'organiser et structurer ce flux de nombres binaires (cf cours sur l'architecture des ordinateurs).
Toutes les instructions doivent donc être codées en binaire, l'ordinateur étant capable de réaliser les opérations en binaire.
C'est le langage machine.

oui mais ...

le programmeur veut pouvoir se détacher de l'architecture interne de l'ordinateur pour se concentrer sur le programme lui-même.

On remplace les codes binaires par des mots clé : par exemple ADD A, B pour l'addition du contenu de deux registres remplacera le code binaire 1011 correspondant à l'addition, et les adresses des deux registres en mémoire.
On parle de langage d'assemblage. C'est un langage de bas niveau spécifique à chaque machine et donc fourni par le constructeur de l'ordinateur.
Pour cette raison, on est toujours tributaire de la machine pour réaliser la programmation de l'ordinateur.

On a donc introduit des langages évolués qui permettent de s'affranchir de la réalité matérielle.
Un langage évolué possède en général

- **des instructions de création et d'affectation de variables.**
- **des instructions permettant d'échanger des informations entre la mémoire et les périphériques.**
- **des structures de contrôle (servent à préciser l'enchaînement des instructions : choix et répétition)**
- **des structures de données (moyen de stocker et d'organiser logiquement des données pour en faciliter l'accès et leur modification). On distingue d'une part le type de données abstrait où on décrit sa structure et les opérations qu'on peut faire et d'autre part, son implémentation dans un langage donné.**
- **possibilité de définir des fonctions et des procédures**
  (regrouper dans un même bloc muni d'un identificateur, tout un bloc d'instructions)
- **la notion d'objet (d'introduction plus récente).**
  (un objet comprend des données sur lesquelles agissent des méthodes)

Des exemples de types de données : tableaux et listes, dictionnaire, piles, files, tas, arbres, date ...
Ce qu'on veut faire des données conditionne le choix de la structure de donnée et l'efficacité d'un programme. Ce choix doit se faire dès la phase de conception de l'algorithme.
En première année, on utilisera seulement la notion de tableau et de liste.

On parle de programmation structurée lorsqu'on dispose des structures de contrôle (branchements et boucles).

On parle de programmation procédurale lorsqu'on dispose des fonctions

On parle de programmation impérative lorsque le programme est un enchaînement structuré d'instructions où on déclare et on modifie des variables, où on appelle des fonctions (programmation modulaire).

Il existe toutefois aussi des langages fonctionnels où l'on évalue seulement des fonctions.

Pour écrire un programme indépendamment du langage dans lequel il sera implémenté, on utilise le langage naturel ou pseudo langage ou pseudo code. La syntaxe d'un tel langage est assez libre et anticipe en général le choix du langage d'implémentation.

Il y a plusieurs sortes de langage

- **langage compilé : (cas de C) le programmateur rentre son code (code source) et le langage le transforme en un exécutable (programme binaire prêt à l'emploi) après compilation (.exe sur windows)**
- **langage interprété : c'est le cas de Python : les instructions (code source) sont interprétées à la volée au fur et à mesure. Il n'y a pas de compilation, ni d'exécutable donc. Le code est directement exécutable. Les erreurs sont détectées lors de l'exécution**

Comparatif

Langage compilé : typage fort des variables, plus rapide car le fichier compilé est prêt à être exécuté.

Langage interprété : typage dynamique (le type des variables est attribué lors de l'affectation, on n'a pas besoin de les déclarer à l'avance), plus lent

PYTHON

- **Logiciel libre. Créé en 1991**

- **Multi-plateforme**

- **Langage interprété**
  - **Typage dynamique**
  - **Ordre des instructions : les objets sont définis avant leur utilisation**

- **Langage de haut niveau**

- **Programmation orientée objet**

- **Langage modulaire : noyau central et modules.**
  Nombreuses bibliothèques, notamment pour le calcul scientifique
  - **matplotlib : pour le tracé de courbes**
  - **Numpy : tableaux de données**
  - **Scipy : calcul scientifique**

- **Version Python 3 minimum pour nous**

- **Syntaxe repose sur l'indentation**

- **Divers IDE (environnement de développement intégré) : permet de**
  - **gérer plusieurs fichiers en même temps**
  - **Aide disponible**
  - **Tester des bouts de code**
  - **« debugger » : exécution du code pas à pas**
  - **Inspection des variables créées.**
  - **IDLE, SPYDER, ...**

- **Des suites complètes intégrant un IDE, le chargement des bibliothèques courantes ...**
  Winpython, Anaconda, ...