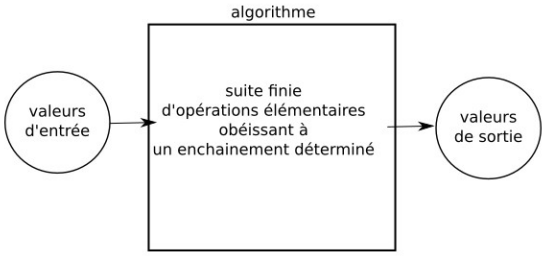
Algorithmes

Le terme algorithme vient du nom du mathématicien perse du 9e siècle Muhammad Ibn Mūsā al-Khuwārizmī. Cette notion est donc très antérieure à la création du premier ordinateur. **Mais qu'est-ce qu'un algorithme ? Et comment comparer l'efficacité de différents algorithmes ?**

I. Aglorithme

I.1. Définition(s)



Mais, qu'est-ce qu'une "opération élémentaire" ?
Ici aussi, définir "opération élémentaire" n'est pas chose aisée ! On peut tout de même dire qu'une "opération élémentaire" est une action simple, qui doit être facilement compréhensible pour la personne chargée d'effectuer cette action.

On peut donc rencontrer des algorithmes dans la vie quotidienne :

- Une recette de cuisine peut être vue comme un algorithme avec **des entrées** (les ingrédients, le matériel utilisé), **des instructions élémentaires simples** (frire, flamber, rissoler, braiser, blanchir, etc.) à faire dans un ordre précis, **un résultat** (le plat préparé).
- Le tissage, surtout tel qu'il a été automatisé par le métier Jacquard est une activité que l'on peut dire algorithmique.
- Un casse-tête, comme le cube Rubik, peut être résolu de façon systématique par un algorithme qui mécanise sa résolution.

En informatique, les algorithmes décrivent les opérations élémentaires (affectation d'une donnée de type simple, calcul arithmétique ou logique, comparaison…) **en "langage naturel"** (le français pour nous). Le passage à un langage de programmation (Python, Java, C++, …) s'appelle l'implémentation. **L'algorithmique est l'étude des algorithmes.**

I.2. Terminaison

Dès que l'on utilise une boucle non bornée ("tant que"), il y a un risque de boucle infinie. Pour éviter cela, il faut s'assurer qu'il y a bien un variant de boucle (une variable dont la valeur change lors de chaque passage dans la boucle) **et que la condition d'arrêt de la boucle sera validée.**

Exemple 1 La boucle se termine car l'écart entre i et 10 diminue de 1 à chaque passage.	Exemple 2 Boucle infinie car pas de variant (étourderie : j à la place de i)	Exemple 3 Boucle infinie car condition d'arrêt tj fausse (i diminue de 3 en 3 sans être un multiple de 3. Sa valeur deviendra négative sans passer par 0)
Affecter 0 à la variable i Tant que i < 10 *… Instructions ne modifiant pas i ...* i = i +1	Affecter 0 à la variable i Tant que i < 10 *… Instructions ne modifiant pas i ...* j = j +1	Affecter 10 à la variable i Tant que i != 0 *… Instructions ne modifiant pas i ...* i = i - 3

I.3. Correction

Pour s'assurer qu'un algorithme est correct **(correction totale)**, il faut démontrer deux choses: que l'algorithme se termine **(terminaison)** et que le résultat de l'algorithme est effectivement de la forme énoncée par la spécification **(correction partielle)**.

II. Complexité d'un algorithme

Le calcul de la complexité d'un algorithme permet de mesurer sa performance. Il existe deux types de complexité : complexité spatiale (pour quantifier l'utilisation de la mémoire) et la complexité temporelle (pour quantifier la vitesse d'exécution). **Nous n'étudierons ici que la complexité temporelle.**

L'objectif d'un calcul de complexité temporelle est de pouvoir comparer l'efficacité de différents algorithmes pour un même problème afin de déterminer l'algorithme optimal.
Ce calcul doit être indépendant du choix d'un langage de programmation (et éventuellement du compilateur associé à ce langage) ou d'un ordinateur (notamment son processeur).

II.1. Décompte des opérations élémentaires

La complexité en temps d'un algorithme sera exprimée par une fonction, notée T comme Time. **Réaliser un calcul de complexité en temps revient à compter le nombre d'opérations élémentaires : affectation d'une donnée de type simple, calcul arithmétique ou logique, comparaison…**

Quelques règles utilisées pour le décompte

• Séquences d'instructions $T(\{ I_1 , I_2 \}) = T(I_1) + T(I_2)$
Exemple :T(« Affecter 3 à la variable i. Ajouter 1 à i ») = 2

• Tests $T(\text{si C alors } I_1 \text{ sinon } I_2) \leq T(C) + \max(T(I_1),T(I_2))$
Exemple : T(« Si i<4 alors afficher True sinon afficher False ») = 2

• Boucles $T(\text{pour i de a à b faire L}) = \sum_{i=a}^{b} T(L)$ Exemple : T(« afficher une liste de longueur n ») = n

D'après la 4eme règle, plus les données seront volumineuses, plus il faudra d'opérations élémentaires pour les traiter. T est donc une fonction de n, la taille des données d'entrée.

II.2. Cas extrêmes

On peut distinguer deux cas extrêmes :

- la complexité dans le meilleur des cas (c'est la situation la plus favorable, par exemple : recherche d'un élément situé à la première position d'une liste)
- la complexité dans le pire des cas (c'est la situation la plus défavorable, par exemple : recherche d'un élément dans une liste alors qu'il n'y figure pas)

Le plus souvent, on calculera la complexité dans le pire des cas, car elle est la plus pertinente. Il vaut mieux en effet toujours envisager le pire.

II.3. Ordre de grandeur

Pour comparer des algorithmes, un ordre de grandeur asymptotique, noté O (« grand O »), de leur complexité temporelle peut suffire. Une fonction $T(n)$ est en $O(f(n))$ (« *en grand O de f(n)*«) si :

$$\exists n_0 \in \mathbb{N} \wedge c \in \mathbb{R}+, \forall n \in \mathbb{R}+ \quad n \geq n_0 \Rightarrow |T(n)| \leq c|f(n)|$$

Autrement dit : $T(n)$ est en $O(f(n))$ s'il existe un seuil n_0 à partir duquel la fonction T est toujours dominée par la fonction f, à une constante multiplicative fixée c près.

Quelques règles sur les ordres de grandeur asymptotique

• Si f(n) est un polynôme de degré d, alors f(n) est $O(n^d)$. Exemple : $3n^2 + 7$ est en $O(n^2)$
• Donner la classe la plus basse possible. Exemple : On dit que « 2n est O(n) » même si on a aussi « 2n est $O(n^2)$ »
• Donner l'expression la plus simple. Exemple : On dit que « 3n + 5 est O(n) » même si on a aussi « 3n + 5 est O(3n) »

Huit fonctions sont souvent utilisées en analyse algorithmique

- Constante ≈ 1
- Factorielle ≈ n!
- Cubique ≈ n^3
- N-log-N ≈ n log(n)
- Quadratique ≈ n^2
- Linéaire ≈ n

- Exponentielle ≈ 2^n
- Logarithmique ≈ log(n)