

Introduction aux algorithmes

1. Un monde informatisé

Vous êtes nés dans un monde dans lequel les ordinateurs occupent une place prépondérante : ordinateurs de travail, internet, ordinateurs de la voiture, smartphones, etc. Ils servent à des tâches diverses : traitement de texte, comptabilité, prévisions météo, registres de l'administration, statistiques, etc.

L'accélération de la puissance de calcul des ordinateurs et la puissance du matériel informatique sont captées par les lois de Moore. Selon ces lois, l'évolution est exponentielle. Ainsi, une des deux lois de Moore stipule que le nombre de transistors des microprocesseurs sur une puce de silicium double tous les deux ans. La loi de Moore ne s'appuie pas sur une compréhension physique de l'évolution de la fabrication des microprocesseurs ; elle transcrit un état de fait et spéculé sur l'avenir, avec un certain succès (voir Figure 1). Elle a été formulée en 1975 !

Le cours d'informatique de 1^{ère} n'est pas un cours de bureautique. Il porte pour l'essentiel sur l'informatique théorique qui est une discipline académique enseignée à l'université. L'informatique théorique s'intéresse entre autres à la théorie du calcul et des algorithmes. Le célèbre informaticien E. Dijkstra (1930-2002) disait :

« Computer science is no more about computers than astronomy is about telescopes ».

2. Résolution algorithmique d'un problème

L'informatique théorique s'intéresse à la résolution algorithmique de problèmes. Un problème algorithmique est un problème qu'on cherche à résoudre au moyen d'une procédure non ambiguë qu'une machine (un ordinateur) pourrait accomplir.

De nombreuses activités humaines impliquent une procédure algorithmique :

- trouver le chemin le plus court entre deux endroits,
- optimiser le rangement de cartons dans un dépôt ou un camion,
- factoriser un nombre,
- organiser les données dans un fichier,
- élaborer la grille horaire des trains dans une gare...
-

De façon générale, la résolution algorithmique d'un problème comprend deux ingrédients :

- les données en entrée et en sortie (sous une représentation donnée),
- un algorithme, qui résout le problème en utilisant la représentation.

Voyons ces deux composantes dans ce problème classique : le passage de la rivière.

3. Le problème du passage de la rivière

Dans le problème du passage de la rivière, un homme habite d'un côté d'une rivière et souhaite passer de l'autre côté un chou, une chèvre et un loup. Sa barque est juste assez grande pour le transporter lui et un autre élément (le chou ou la chèvre ou le loup). De plus, l'homme ne peut pas laisser la chèvre seule avec le chou, sans quoi ce dernier sera mangé par la chèvre ; de même il ne peut pas laisser le loup et la chèvre seuls, au risque que cette dernière se fasse dévorer par le loup. Le problème s'énonce simplement :

Comment l'homme peut-il faire passer la rivière au loup, à la chèvre et au chou sans qu'aucun ne se fasse manger ?

Une approche simple – dite force brute – consiste à essayer toutes les possibilités (vous le ferez dans ce cas en exercice). Mais quelque soit l'algorithme, il faut au préalable spécifier la représentation du problème. Seules sont conservées certaines caractéristiques du problème. Par exemple, il est clair que la largeur de la rivière, l'âge de l'homme, la date de l'évènement, le poids de la chèvre, le sexe du loup ou le nombre de feuilles de chou ne sont pas des caractéristiques importantes du problème. En fait, on doit uniquement distinguer les quatre passagers de la barque et de quel côté de la berge chacun d'eux se trouve.

Pour cela, on définit un quadruplet, c'est-à-dire une suite ordonnée d'éléments, suivant un ordre que l'on se fixe, par exemple : l'homme, le chou, la chèvre, le loup. On nomme les deux berges, par exemple G pour gauche et D pour droite. Ce quadruplet ainsi défini est la représentation du problème.

Supposons que tout le monde est à gauche au départ, la position initiale est donnée par le quadruplet [G, G, G, G]. La position finale souhaitée est [D,D,D,D].

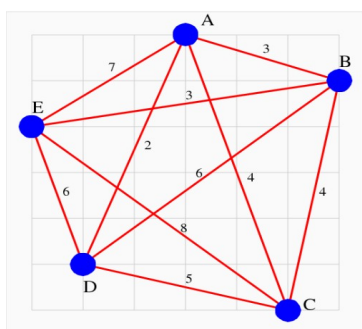


Figure 2: Illustration pour le problème du passage de la rivière. source : <https://mark-borg.github.io/blog/2016/river-crossing-puzzles/>

4. Limites à la résolution algorithmique d'un problème

Une fois que l'on dispose d'un algorithme, une question importante surgit: Est-ce qu'une solution au problème peut-être trouvée en un temps de calcul raisonnable ? Si tel n'est pas le cas, l'algorithme n'aura pas d'utilité pratique. Il se peut que deux algorithmes soient capables de résoudre un problème mais que l'un soit plus efficace que l'autre car plus rapide ou nécessitant moins de mémoire, par exemple, pour arriver au résultat.

Le problème du voyageur de commerce est un exemple type de problème d'optimisation. Le problème consiste à trouver la route la plus courte pour un voyageur de commerce qui doit passer une fois dans chacune des n villes (représentées par des lettres sur la figure ci-dessous). Ce chemin doit commencer et se terminer au même endroit.



Exemple de chemin pour la figure ci-dessous :

$A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \rightarrow A$: distance = $3 + 4 + 5 + 6 + 7 = 25$

Remarque : cette «distance» qu'on souhaite minimiser pourrait aussi bien être un temps de parcours, un prix de péage autoroutier, une dépense énergétique, etc.

L'approche brutale consiste à calculer la longueur de la route pour chacune des combinaisons possibles et de trouver la plus courte parmi toutes. Pour un ensemble de n villes, on peut montrer que le nombre N de chemins à considérer est

$$N = \frac{(n-1)!}{2} \quad \text{où «!» = factorielle (ex : } 5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120 \text{)}.$$

Pour un ensemble de $n = 20$ villes, le nombre de combinaisons est $N = \frac{19!}{2} = 1,2 \cdot 10^{18}$.

En supposant que l'ordinateur à disposition calcule 10^6 routes par seconde, il lui faudra plus de 3210 années pour résoudre ce problème. Pour $n = 50$ villes, le nombre de routes dépasse 10^{63} et cela prendrait plus de temps à résoudre que l'âge de l'Univers...

Le temps de calcul peut être une limitation importante. Dans le jeu d'échecs, par exemple, une approche «force brute» n'est pas réalisable, le nombre de parties différentes possibles étant beaucoup trop grand pour les capacités de calcul des ordinateurs les plus puissants (ce nombre est estimé à 10^{120}). De nombreuses approches algorithmiques ont été proposées, aucune ne permettant d'évaluer toutes les possibilités à partir de n'importe quelle configuration du jeu. Le plus célèbre ordinateur joueur d'échecs est Deep Blue (voir Figure 4), qui

fut le premier à battre un grand maître (Kasparov en 1995). Deep Blue était capable de calculer 200'000'000 de positions par seconde. Aujourd'hui le jeu de go suscite plus d'intérêts chez les programmeurs car il est encore plus difficile à maîtriser pour une machine.

Figure 4 : Garry Kasparov contre Deep Blue de IBM en 1997.

Source : <https://spectrum.ieee.org/how-ibms-deep-blue-beat-world-champion-chess-player-garry-kasparov>



5. Représentations d'un algorithme

a) le *pseudo-code* :

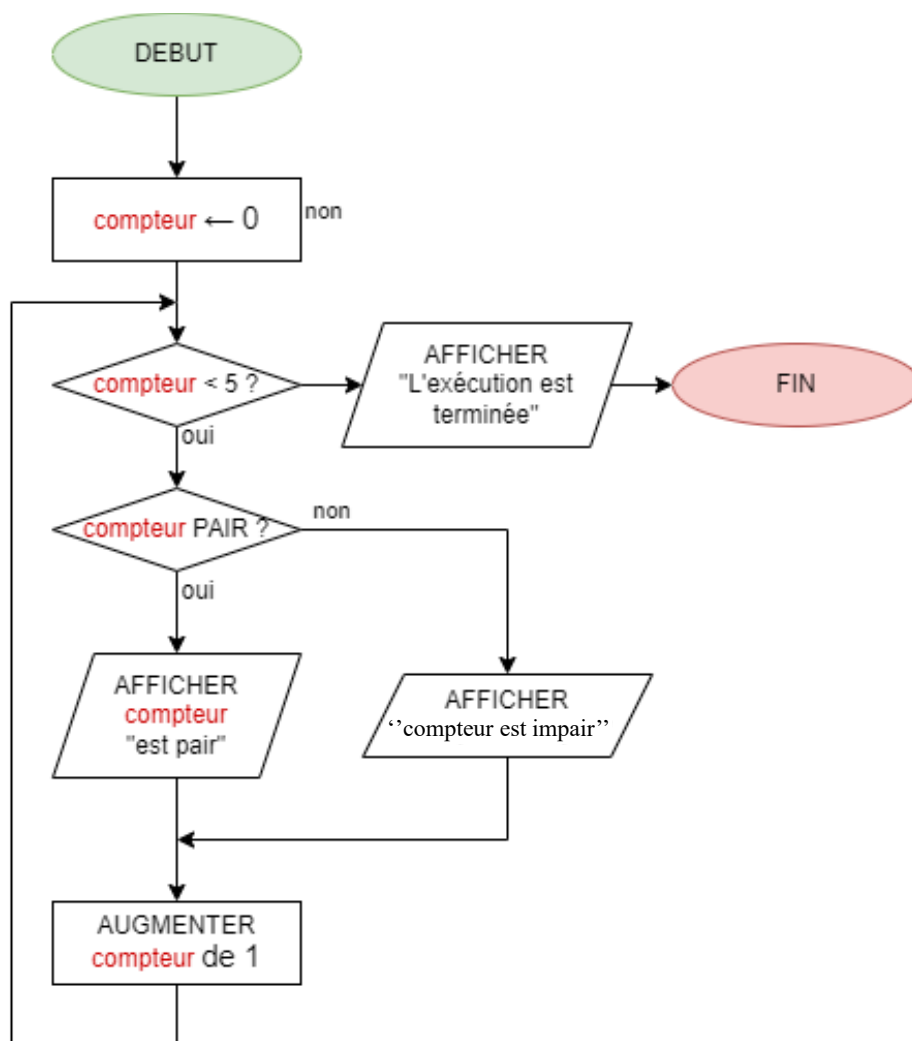
Pour produire un algorithme, on peut écrire une séquence d'instructions en *pseudo-code*. Le *pseudo-code* n'est pas du code informatique dans un langage comme Python mais ça y ressemble. En effet, les langages naturels (français, anglais, etc.) comportent énormément d'ambiguïtés alors que le *pseudo-code* est censé ne pas en contenir. Autrement dit, on pourra facilement transformer le *pseudo-code* en un programme (en Python ou autre) où chaque instruction est comprise par le processeur de manière univoque, ce qui permet de le faire tourner sur un ordinateur.

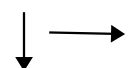
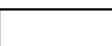
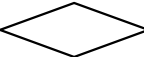
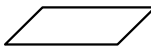
Exemple :

```
VARIABLE: compteur ← 0
TANTQUE compteur < 5 {
    SI compteur EST pair {
        AFFICHER compteur
        AFFICHER «est pair» }
    SINON {
        AFFICHER «compteur est impair» }
    compteur ← compteur + 1 }
AFFICHER «L'exécution du programme est maintenant terminée.»
```

b) l'organigramme :

Une autre manière de représenter un algorithme est en utilisant un *organigramme*. Dans ce cas, on schématise chaque élément de la séquence par des formes qui correspondent à des entrées/sorties, à des calculs, à des conditionnelles :



Symbole	Nom	Rôle
	Flèche	Indique la séquence du programme
	Terminaison	Débute ou termine un programme
	Processus	Exécute une tâche comme un calcul
	Décision	Embranchement conditionnel
	Entrée/Sortie	Demande une entrée ou produit une sortie

Série 3 : Introduction aux algorithmes

1. Donner trois exemples de problèmes pour lesquels on peut développer un algorithme.
2. Pourquoi un algorithme de résolution d'un problème nous aide à écrire un programme pour résoudre ce problème à l'aide d'un ordinateur ?
3. En 1990, le nombre de transistors sur un microprocesseur était 10^6 . D'après la loi de Moore, quel est ce nombre en : a. 1992 / b. 1994 / c. 2000 / d. 2021 ?
4. Dites si chaque problème est *décisionnel* ou *de construction* (ou *d'évaluation*) :
a) Trouver le chemin le plus court entre deux points.
c) trouver si un nombre donné est un nombre premier.
5. Cherchez une solution au problème de la rivière.
Utilisez la force brute (envisager toutes les possibilités raisonnables)
Utilisez l'écriture proposée sous forme de quadruplet.
6. Expliquez à partir d'un exemple ce qu'est l'approche brutale (ou force brute) d'un problème algorithmique.
7. Expliquez à partir d'un exemple en quoi le temps de calcul peut être une limitation importante dans la résolution d'un problème algorithmique.
8.
 - a. Écrivez un pseudo-code pour illustrer une fonctionnalité simple d'un distributeur de billets. On lui donne la somme à distribuer et il donne cette somme au client (si possible en ne donnant pas trop de billets)...
 - b. Construisez l'organigramme de cette même fonctionnalité.
9. Soit l'algorithme suivant écrit en pseudo-code :

```
AFFICHER «Combien de fois allez-vous aux bains des Pâquis par an ?»  
DEMANDER nombre  
SI nombre > 25  
    AFFICHER «Il vaut la peine de prendre un abonnement annuel.»  
SINON  
    AFFICHER «Il n'est pas préférable de prendre l'abonnement annuel.»  
FIN
```

- a. Quel problème cet algorithme est-il censé résoudre ?
- b. Entourez les trois parties suivantes :
 - i. entrée de données (en rouge)
 - ii. traitement de données (en bleu)
 - iii. sortie de résultats (en noir)
- c. Écrivez cet algorithme sous forme d'un organigramme.