Algorithmique :

Introduction

Table des matières

1.Introduction....3
a)Informatique, Information....3
b)Connaissances....3
c)Codage....4
2.Algorithme....5
a)Les origines....5
b)En pratique....5
c)Notion de traitement....6
d)Quelques exemples....7
e)Et l'ordinateur dans tout cela ?....8
3.Les qualités essentielles d'un bon algorithme....9
a)Entrées....9
b)Sorties....9
c)Finitude....9
d)Définition....10
e)Efficacité....10
4.En résumé....11
Annexes....12
5.Index des illustrations....12

1. Introduction

a) Informatique, Information

Voici la définition qu'en donne le Larousse :

informatique : Science du traitement automatique et rationnel de l'information en tant que support des connaissances et des communications.

information : Élément de connaissance susceptible d'être codé pour être conservé, traité ou communiqué.

Il ressort de ces deux définitions trois concepts importants :

- la notion de connaissance,
- la notion de codage,
- et la notion de traitement.

Nous allons détailler dans la suite de cette introduction ces trois notions.

b) Connaissances

Pour commencer, qu'est-ce que la connaissance ?

De quelle manière appréhendons-nous le monde qui nous entoure ?

En y réfléchissant, on peut distinguer trois types de connaissances :

La connaissance déclarative : elle concerne le "quoi"

Elle donne des définitions et des propriétés caractéristiques de la notion étudiée.

Par exemple, on peut définir la racine carrée d'un réel positif ou nul x comme étant l'unique réel y positif ou nul dont le carré est x .

Cette définition permet de tester si un réel est bien la racine carrée d'un autre, mais elle ne donne pas de méthode pour déterminer la racine carrée d'un réel.

La connaissance impérative : elle concerne le "comment"

Elle donne des procédés permettant de construire les objets étudiés.

Par exemple, vous avez peut-être rencontré au lycée la méthode de Héron, permettant d'obtenir des valeurs approchées de plus en plus précises de la racine carrée d'un réel x :

1. Partir d'une approximation x de $\sqrt{a}$.
2. Remplacer x par la moyenne de x et de $\frac{a}{x}$.
3. Recommencer tant qu'on n'a pas atteint la précision voulue.

Les mathématiques montrent qu'un tel procédé converge, ce qui répond bien à nos préoccupations.

La connaissance conditionnelle : elle s'occupe du "quand"

Elle donne les conditions dans lesquelles une connaissance doit être utilisée. Par exemple, on peut donner des contextes dans lesquels il peut être intéressant d'utiliser la racine carrée d'un réel, quand on connaît des propriétés de son carré.

c) Codage

Pour manipuler une connaissance, nous devons la coder, de manière, comme indiqué par la définition du Larousse, à pouvoir la stocker, la traiter ou la transmettre. Pour cela, nous utilisons des symboles : lettres, chiffres, signaux (lumineux, sonores, électromagnétiques...), pictogrammes, etc.

Voici quelques exemples :

Information	Codage
Les voitures peuvent passer	feu vert
Un appel arrive	sonnerie
Danger de radioactivité	pictogramme 
l'année 2011	chiffre romain : MMXI
SOS	codage morse : … --- ...

Le langage : le plus souvent, ce n'est pas un symbole qui est utilisé pour coder une information, **mais un ensemble de symboles**, appartenant à un certain **alphabet**, assemblés selon un certain nombre de « **règles »**.

L'association d'un alphabet et de ces règles s'appelle un langage.

La syntaxe : **c'est l'ensemble des règles d'assemblage des symboles**.

Elle indique quelles sont les "phrases" effectivement constructibles dans le langage donné, en général en fournissant une grammaire.

Par exemple, "1 2 = + 4" est une phrase non valide du langage des expressions arithmétiques.

Il est en général relativement simple de tester si une phrase est syntaxiquement correcte, et il existe de nombreux outils, en particulier dans le monde de l'informatique, permettant d'accomplir cette tâche de façon automatique.

La sémantique : elle exprime **la signification associée aux groupes de symboles**.

C'est elle qui établit la correspondance entre le codage et les éléments de connaissance. Cependant, une phrase peut très bien être syntaxiquement correcte mais ne pas exprimer une connaissance valide, comme la phrase "1 + 2 = 4".

D'autre part, il est souvent très compliqué de vérifier si une phrase signifie effectivement quelque chose. Ce sera le travail du locuteur que de vérifier que ses phrases ont un sens.

Il faut bien comprendre qu'une même connaissance peut être codée de différentes façons, selon le langage qu'on emploie. Par exemple, l'entier 71 peut être codé :

- 71 (codage décimal),
- « soixante-et-onze » (codage en langue française),
- « seventy-one » (codage en anglais)
- 1000111 (codage en binaire)

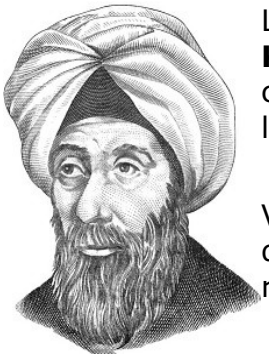
Réciproquement, le groupe de symboles 71 peut représenter:

- une quantité (les dossiers de 71 candidats ont été retenus par une première sélection),
- une étiquette attribuée à une classe d'objet (la ligne de bus 71),
- le code téléphonique du département de Haute-Loire,
- la notation décimale du code ASCII du caractère G,
- la notation octale du code ASCII du caractère 9,
- la notation hexadécimale du code ASCII du caractère q
- ...

Le « contexte » permet en général à un humain de coder ou décoder sans ambiguïté un message, mais c'est un domaine dans lequel les ordinateurs ont encore de gros progrès à faire !

2. Algorithme

a) Les origines



Le mot "algorithme" est une latinisation de la ville d'origine d' **Al-Khwarismi**, mathématicien musulman perse, dont l'ouvrage le plus célèbre, « *Kitab aljabr w'al muqabala* », a permis l'introduction de l'algèbre en Europe.

Voici la définition du Petit Robert : « suite finie séquentielle de règles que l'on applique à un nombre fini de données, permettant de résoudre une classe de problèmes semblables. »

b) En pratique

Un théorème mathématique affirmant que tout nombre positif a une racine carrée est fascinant du point de vue théorique, mais ne nous intéresse que très peu si l'on a effectivement besoin de la valeur de cette racine carrée.

Dans ce cas, on a besoin d'une méthode permettant de calculer cette racine, ou bien,

si cela est impossible, d'en obtenir une valeur approchée aussi précise que possible.

L'algorithmique est la science qui étudie les problèmes du point de vue impératif, concevant des méthodes pour les résoudre, construisant leurs solutions, et étudiant les qualités et les défauts de ces méthodes. C'est elle qui construit les traitements des informations connues afin d'obtenir d'autres informations.

c) Notion de traitement

Cette notion de traitement s'articule autour de trois concepts :

- Description : la méthode de passage des données aux résultats est décrite par un texte.
- Exécution : une réalisation effective du traitement est mise en œuvre sur des données spécifiques.
- Agent exécutant : c'est l'entité effectuant une exécution, cette entité est donc capable de mettre en œuvre la méthode.

Dans le contexte de l'informatique, la description sera souvent exprimée à l'aide d'un algorithme, l'exécutant sera un processeur et une exécution sera appelée un processus.

Mais il y a bien d'autres contextes dans lesquels on fait du traitement de l'information. Par exemple, dans une cuisine, une description sera nommée « recette », l'exécutant sera le cuisinier, et une exécution de la recette permet de passer des ingrédients au plat.

Il ne faut pas confondre la description, l'exécution et l'agent : la recette, en général écrite sur une feuille de papier, n'a pas vraiment de valeur nutritionnelle, et à moins qu'on ait des mœurs spéciales, l'agent ne doit pas être consommé. Seule une exécution particulière de la recette par le cuisinier va fournir un résultat comestible.

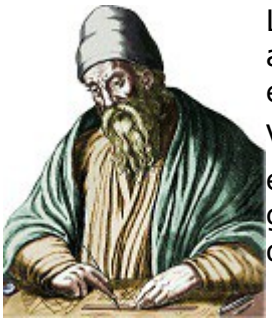
On confond souvent description et exécution. La méthode permettant de multiplier entre eux deux entiers ne doit pas être confondue avec l'exécution du produit 45×10.

La méthode ne donne pas le résultat de cette opération particulière, et l'exécution ne dit pas comment multiplier 45 et 10 !

d) Quelques exemples

On a déjà rencontré quelques exemples d'algorithmes :

- une recette est un algorithme permettant de passer des ingrédients isolés au plat ;
- lorsque votre instituteur (ou institutrice) vous a appris à multiplier ou diviser deux entiers, les méthodes qu'il ou elle vous a données sont des algorithmes.



L'un des premiers algorithmes mathématiques connus est le célèbre algorithme d'Euclide, permettant de calculer le **PGCD** de deux entiers.

Voici comment on peut l'énoncer de façon moderne :

étant donnés deux entiers positifs non nuls m et n, trouvez leur plus grand diviseur commun, c'est-à-dire le plus grand entier positif les divisant tous les deux.

1. Calcul du reste : effectuer la division euclidienne de m par n , soit r le reste de cette division (ainsi, $0 \leqslant r < n$).
2. Est-il nul ?:
    - si $r = 0$, l'algorithme termine, n est le résultat.
    - sinon : faire $m \leftarrow n$, $n \leftarrow r$ et retourner à l'étape 1.

Exercice

Voici un algorithme, appelé méthode de multiplication russe, permettant de calculer le produit de deux entiers a et b :

- si $a = 0$: le résultat est 0 ;
- si $a \neq 0$:
- diviser a par 2 ;
- multiplier b par 2 ;
    - calculer le produit des deux nombres résultant de ces opérations :
- si a est impair ajouter b au résultat.

a) Que doit savoir faire l'agent exécutant pour mettre en œuvre cet algorithme ?

b) Pourquoi est-il bien adapté à un traitement informatique ?

c) Remarquez que pour calculer le produit de a et b, il faut savoir calculer le produit de deux nouveaux entiers (deuxième ligne) ; cette description est-elle valide ? Que doit-on faire pour la rendre valide si ce n'est pas le cas ?

d) Utiliser cet algorithme pour calculer le produit de a = 13 et b = 238

Étape	$a=13$	$b=238$	résultat
1	$\frac{13}{2} \rightarrow quotient\ 6\ reste\ 1$	$b=238$	238
2	$\frac{6}{2} \rightarrow quotient\ 3\ reste\ 0 \rightarrow a=3$	$b=2\times238=476$	476
3	$\frac{3}{2} \rightarrow quotient\ 1\ reste\ 1 \rightarrow a=1$	$b=2\times476=952$	$952+238=1190$
4	$\frac{1}{2} \rightarrow quotient\ 0\ reste\ 1 \rightarrow a=0$	$b=2\times952=1904$	$1904+1190=3094$

e) Et l'ordinateur dans tout cela ?

Un algorithme étant connu, on peut le mettre en œuvre de manière automatique en concevant un mécanisme adapté. Mais allons encore plus loin. Imaginons une machine qui prendrait en entrée un algorithme, et se modifierait de manière à appliquer cet algorithme. Une telle machine pourrait donc réaliser n'importe laquelle des tâches pour lesquelles on possède un algorithme. Une telle machine existe : c'est l'ordinateur.

Ainsi, tout ce que l'on a à faire pour faire faire un nouveau calcul à un ordinateur est de concevoir l'algorithme le réalisant, l'implémenter à l'aide d'un langage de programmation, et à fournir les entrées.

Voici le schéma d'un ordinateur:

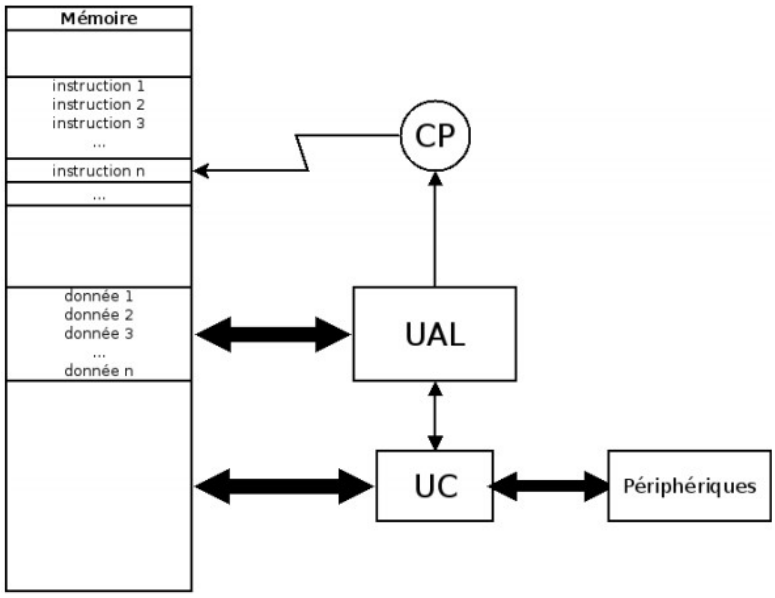


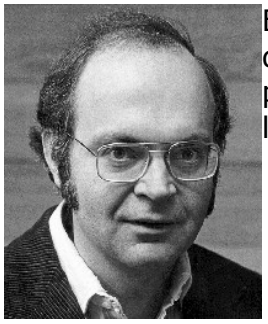
Illustration 1: Schéma d'un ordinateur

L'**UAL**, ou unité arithmétique et logique est le centre de calcul.

Le compteur programme **CP** point sur une adresse de la mémoire, contenant une instruction à faire exécuter par l'unité de contrôle **UC**.

Celle-ci peut charger des données de la mémoire dans l'**UAL**, faire exécuter à celle-ci des opérations, transférer les résultats en mémoire, ou bien les transférer vers les périphériques (écran, carte réseau, clavier, souris, imprimante...).

3. Les qualités essentielles d'un bon algorithme



En plus d'être une suite finie de règles donnant une séquence d'opérations permettant de résoudre un type spécifique de problème, *Donald Knuth* décrit la notion d'algorithme comme ayant les spécificités suivantes:

a) Entrées

Un algorithme peut avoir des entrées, qui sont des données sur lesquelles il va travailler.

Par exemple, les deux entrées de l'algorithme d'Euclide sont les deux entiers m et n .

Ces entrées sont des objets d'un certain ensemble aux propriétés spécifiées, et il ne faut pas s'étonner si on donne à un algorithme des données qui ne respectent pas ces spécificités.

b) Sorties

Un algorithme renvoie une ou plusieurs sorties, qui sont en relation avec les entrées.

Par exemple, la sortie de l'algorithme d'Euclide est l'entier n de l'étape 2, qui est le PGCD des entrées m et n .

Là ou les règles permettant de relier les sorties aux entrées sont les spécifications de l'algorithme.

c) Finitude

Un algorithme doit toujours se terminer après avoir exécuté un nombre fini d'opérations.

Dans l'exemple de l'algorithme d'Euclide, il n'est pas évident de prouver que le procédé s'arrête.

Il faut constater qu'à chaque étape, on remplace n par le reste de la division euclidienne de m par n , qui est strictement inférieur à n .

Ainsi, la suite des valeurs stockées dans la variable n est une suite d'entiers strictement décroissante, qui doit nécessairement prendre la valeur 0 après au plus n passages dans l'étape 1 de l'algorithme.

Il faut bien faire attention au fait qu'il ne suffit pas qu'une méthode s'exprime à l'aide un nombre fini de mots pour mériter le titre d'algorithme : la phrase « avance d'un

pas, tourne à droite, et recommence » est un exemple simple d'une description finie (elle comporte seulement 9 mots !) d'un **processus infini**, qu'on appelle souvent « boucle de la mort » en informatique : le processeur répète indéfiniment la même opération sans rien pouvoir faire d'autre.

Un procédé qui a toutes les qualités d'un algorithme, sauf celle-ci, est appelé **méthode calculatoire**.

Un exemple est le calcul d'un réel défini comme limite d'une suite. Ces méthodes ont bien entendu de l'intérêt, mais le travail permettant d'en tirer des algorithmes nécessite entre autres de régler **le problème de l'arrêt du calcul**.

d) Définition

Chaque étape de l'algorithme doit être précisément et de façon non ambiguë définie.

Ce concept nécessite de prendre en compte l'entité qui effectuera les opérations : le processeur.

Ainsi, on ne pourra pas énoncer l'algorithme d'Euclide de la façon dont on l'a énoncé à une personne qui ne sait pas ce qu'est une division euclidienne. De la même façon, on ne décrira pas une recette de cuisine de la même façon à un chef ayant 35 ans d'expérience et à un parfait débutant qui n'a jamais monté des blancs en neige ! Qu'est-ce qu'une « pincée » de sel pour quelqu'un qui n'a jamais fait la cuisine ? Ou une « béarnaise » ?

De la même façon, vous ne pourrez pas concevoir des programmes corrects sans prendre en compte les « connaissances » du langage que vous utiliserez.

e) Efficacité

Chaque algorithme nécessite plusieurs étapes de calcul et l'objectif sera de minimiser les plus coûteuses pour le matériel.

Les algorithmes que nous voulons concevoir ne devront pas seulement effectuer un nombre fini d'opérations pour accomplir leur tâche. Nous aimerions que ce nombre d'opérations soit raisonnablement fini ! Rien ne sert d'avoir un algorithme permettant de trouver la « réponse à la grande question de l'univers » si cet algorithme demande un temps de calcul supérieur à l'âge de l'univers (de l'ordre de 30 milliards d'années d'après les estimations actuelles).

On ne peut malheureusement en général pas connaître le nombre exact d'opérations effectuées par un calcul. Ou plutôt, ce nombre dépend des entrées, et plus précisément de leur taille. Pour des entiers, ce sera leur grandeur, pour un tableau, le nombre de ses cellules...

Par exemple, pour trier un tableau comportant n entrées, les algorithmes peu efficaces ont besoin d'effectuer environ n^2 opérations. Le tri d'un tableau de 10 entrées se fait donc de façon quasi instantanée, par contre, que se passerait-il si l'on souhaitait trier un tableau comportant une soixantaine de millions d'entrées ?

Il faudrait environ 4 millions de milliards d'opérations ; en supposant que l'ordinateur utilisé effectue un milliard de comparaisons par seconde (ce qui est très optimiste !), il faudrait environ 4 millions de secondes pour trier ce tableau, soit 45 jours !

Il n'y a pas que le temps qui soit déterminant dans le fonctionnement d'un algorithme. De la même façon, la mémoire (qu'elle soit vive ou de masse) dont nous disposons est limitée. Dans le cas de l'algorithme d'Euclide, il est assez facile de se convaincre que trois cases mémoire suffisent pour l'ensemble du calcul. En effet, une fois r déterminé dans l'étape 1, on peut se passer de la valeur de m , et donc y ranger l'ancienne valeur de n , puis ranger la valeur de r dans la case n ainsi libérée.

4. En résumé

En pratique, les questions essentielles auxquelles il faut répondre lorsqu'on conçoit un algorithme sont donc :

1. l'algorithme se termine-t-il et ce pour n'importe quelle entrée, et non pas seulement les quelques-unes que l'on a testé ?
2. l'algorithme est-il correct, c'est-à-dire renvoie-t-il le bon résultat dans tous les cas ?
3. l'algorithme est-il performant, ce qui revient à trouver un lien entre le temps et/ou l'espace mémoire nécessaire et la « taille » des entrées ?

Ces questions sont absolument fondamentales de nos jours car la plupart des systèmes numériques sont maintenant « embarqués » dans des outils dont notre tranquillité, notre sécurité, voire notre vie, dépend.

Peut-on par exemple imaginer un système de gestion du freinage d'une voiture qui ne fonctionne pas lorsqu'un capteur est encrassé, ou bien qui réagit dans certaines conditions trois ou quatre secondes trop tard ?

On peut souvent associer les deux premiers points dans la recherche d'une preuve, et souvent, l'analyse de cette preuve permet d'obtenir une estimation du nombre de fois où chaque instruction de l'algorithme est effectuée, permettant de répondre au troisième point.

Annexes

5. Index des illustrations

Index des illustrations

Illustration 1:Schéma d'un ordinateur.....8