ALGORITHMES

ISN - Chapitre 1

Table des matières

I A l'origine des algorithmes

I 1 Historique

En quelle année a été inventé le premier algorithme ?

Il y a plus de 2000 ans !

Les algorithmes ne sont pas nés avec l'informatique :
▶ L'algorithme d'Euclide pour calculer le PGCD de 2 entiers est vieux de plus de 2 000 ans !
▶ Des descriptions précises d'algorithmes sont présents dans la Chine ancienne.

L'origine du mot algorithme est lié au nom du savant arabe du IXe siècle : Al-Khwarizmi.
Ce savant arabe a publié plusieurs méthodes de calcul pour le calcul effectif des solutions d'une équation du second degré et grâce à lui les chiffres arabes ont pu se diffuser en Occident.

Ada Lovelace (1815 - 1852) est l'auteur du premier algorithme destiné à être exécuté par une machine, la machine analytique, conçue par Charles Babbage, mais jamais construite de son temps. Un langage de programmation, Ada, est ainsi nommé en son honneur.

I 2 Des définitions d'un algorithme

Une définition simple :
« Un ensemble d'instructions pour résoudre un problème. »

La définition donnée par le B.O. :
Un algorithme se définit comme une méthode opérationnelle permettant de résoudre, en un nombre fini d'étapes clairement spécifiées, toutes les instances d'un problème donnée. Cette méthode peut être exécutée par une machine ou par une personne.

Une définition plus complète à partir de 5 propriétés :
▶ **Finitude** : Un algorithme doit toujours se terminer après un nombre fini d'étapes.
▶ **Précision** : Chaque étape d'un algorithme doit être définie précisément ; les actions à transposer doivent être spécifiées rigoureusement et sans ambiguïté pour chaque cas.
▶ **Entrées** : Quantités, prises dans un ensemble d'objets spécifié, qui sont données à l'algorithme avant qu'il ne commence.
▶ **Sorties** : Quantités qui ont une relation spécifiée avec les entrées.
▶ **Rendement** : Toutes les opérations que l'algorithme doit accomplir doivent être suffisamment élémentaires pour pouvoir être en principe réalisées dans une durée finie par un homme utilisant du papier et un crayon.

I 3 Quelques exemples d'algorithmes dans la vie moderne et dans les maths

▶ L'algorithme d'Euclide.
▶ Le programme de construction d'une figure géométrique.
▶ La transcription des « formules » moléculaires en chimie.
▶ Le code génétique.
▶ L'analyse fonctionnelle en technologie.
▶ Le manuel d'un meuble Ikéa :p

I 4 Ne pas confondre algorithme et programme

Un algorithme est rédigé dans un *pseudo-langage* (en Français).
Il peut ensuite être implémenté dans un langage de programmation donné (Ti-Basic, XCAS, JAVA, Python...).

Un algorithme exprime les **instructions résolvant un problème** donné indépendamment des particularités de tel ou tel langage.

Apprendre l'algorithme, c'est apprendre à manier la structure logique d'un programme informatique.

II Concevoir un algorithme

II 1 Phase de recherche : objectif ? données ? moyens ?

- Commencer par bien cerner le but recherche d'un algorithme. Quelle est sa finalité ? L'objectif voulu ?
- Quelles sont les données dont on dispose pour concevoir l'algorithme ?
- Quels sont les moyens à notre disposition ? Outils mathématiques, outils de tri, d'analyse..

II 2 Les 3 étapes d'un algorithme

II 2 a Entrées

Entrées

Déclaration de données, de variables, ou saisie de ces données.

Il s'agit de repérer les données nécessaires à la résolution d'un problème. Ces données peuvent être numériques, sous forme de textes (*chaines de caractères*), de type logique (*booléen*, deux valeurs possibles, VRAI ou FAUX), ou de type graphique (des points par exemple).

Dans cette phase peut aussi figurer ce qu'on appelle l'entrée des données : la saisie de caractères ou de nombres sur le clavier, ou la lecture de la position du pointeur de la souris, ou encore par la lecture d'un fichier contenant ces nombres ou caractères.

II 2 b Traitement

Traitement

Exécution automatique des instructions permettant de résoudre un problème.

Il s'agit de déterminer toutes les étapes des traitements à faire et donc des *instructions* à donner pour une exécution **automatique**.

Si ces instructions s'exécutent en séquence, c'est-à-dire les unes après les autres, on parle d' **algorithme séquentiel**.
Si les opérations s'exécutent sur plusieurs processeurs en parallèle, on parle d' **algorithme parallèle**.
Si les taches s'exécutent sur un réseau de processeurs, on parle d'algorithme **réparti** ou **distribué**.

Pour cette année d'ISN, seuls les algorithmes séquentiels seront traités.

II 2 c Sortie

Sorties

Affichage ou impression d'un résultat ou de données transformées.

Les résultats obtenus peuvent être affichés sur l'écran, imprimés sur papier, ou bien encore être conservés dans un fichier.

II 3 Les composants d'un algorithme

II 3 a affectations de variables

Les données de l'algorithme peuvent être stockées dans des variables ou « mémoires ».
Ces données sont représentées par un nom (un **identificateur**).
Les identificateurs sont des suites de lettres et chiffres (sans espace !), commençant par une lettre, et qui doivent être choisies judicieusement pour que l'algorithme soit immédiatement lisible et interprétable, pour soi-même et son équipe (et les autres !).

Les données peuvent avoir un type :
- numérique ;
- chaine de caractères ;
- booléen (VRAI ou FAUX) ;
- liste (numériques ou de chaines de caractères).

Exemples d'affectation d'une variable :

A ← 2

La variable nommée A prend pour valeur 2 quel que soit sa valeur précédente.

Compteur ← Compteur + 1

La variable nommée Compteur prend pour valeur la valeur courante de Compteur augmentée de 1.

C ← 2
D ← 3
E ← C + D

A la fin des trois instructions, la variable nommée E a pour valeur 5.

A vous de jouer !
- Proposer un algorithme qui permute les contenus de deux variables numériques A et B.

II 3 b structure alternative : Si

Si, Alors, Sinon

Si *condition* **Alors**
    traitement 1
Sinon
    traitement 2
FinSi

Remarques :
- Il existe également des structures conditionnelles "Si... Alors..." sans l'alternative "Sinon".
- L'évaluation de la condition est un **booléen**, qui a pour valeur VRAI ou FAUX.
- Une structure alternative a toujours un début (ici *Si*) et une fin (ici *FinSi*).

Exemple :
Si $N > 10$ **alors**
    Afficher « Le nombre », N, « est strictement supérieur à 10 »
Sinon
    Afficher « Le nombre », N, « est ..... à 10 »
FinSi

II 3 c structure répétitive : Pour et Tant que

La boucle Pour

Pour $i = 1$ à N **Faire**
    traitement
FinPour

Remarques :
- On effectue les instructions nommées *traitement* N fois.
- Il faut donc connaitre à l'avance le nombre de fois qu'il faudra exécuter l'instruction *traitement* pour utiliser une boucle Pour.

Exemple : que fait cet algorithme ?

Pour $i = 1$ à 10 **Faire**
    Afficher $i * i$
FinPour

Question : peut-on améliorer la programmation ce cet algorithme sur la TI pour un affichage sur une seule ligne ?

La boucle Tant Que

Tant que *condition* **Faire**
    traitement
FinTantQue

Remarques :
- On effectue les instructions nommées *traitement* tant que la condition est vérifiée.
- Il n'est pas nécessaire de connaître le nombre de fois que sera exécuté *traitement* pour utiliser une boucle Tant Que. C'est l'un des principaux avantages de cette boucle par rapport à la boucle Pour.

Exemple : que fait cet algorithme ?

$i \leftarrow 1$
Tant que $i < 100$ **Faire**
    Afficher i
    $i \leftarrow i + 2$
FinTantQue

Question : peut-on ici remplacer la boucle **Tant que** par une boucle **Pour** ?

II 4 Optimisation d'un algorithme

Pour résoudre un problème, il existe souvent un très grand nombre de façons (d'algorithmes !) possibles. Par exemple, pour déterminer le PGCD de deux nombres a et b, il existe l'algorithme d'Euclide (relativement efficace), mais aussi l'algorithme des soustractions (plus long). Mais on pourrait aussi tester tous les entiers naturels inférieurs à a et b et trouver quel est le plus grand de ces entiers divisant à la fois a et b.
On imagine vite que pour des grands nombres, cette méthode n'est que peu judicieuse.

De la même façon, il est toujours intéressant de chercher à optimiser un algorithme :
- Soit au départ, lors de la phase de réflexion, pour éviter de concevoir un algorithme trop compliqué pour la tâche demandée ;
- Soit à la fin, lorsqu'on a obtenu un algorithme qui fonctionne, on peut chercher à l'alléger un peu (nombre de variables utilisées, de boucles, d'instructions), soit à l'optimiser (diminuer le nombre de tests, d'itérations des boucles répétitives etc).

On peut aussi parler d'optimisation d'un algorithme lorsqu'on cherche à l'améliorer, en augmentant le nombre de cas qu'il peut traiter (par exemple anticiper les erreurs de l'utilisateur).

Un exemple simple : comment optimiser l'algorithme suivant :

```
1  Demander a, b et c
2  si b^2 - 4*a*c > 0 alors
3  |  Afficher « l'équation a deux solutions réelles distinctes », (-b - sqrt(b^2 - 4*a*c))/(2*a), « et »,
   |    (-b + sqrt(b^2 - 4*a*c))/(2*a)
4  sinon
5  |  si b^2 - 4*a*c = 0 alors
6  |  |  Afficher « l'équation a une unique solution », -b/(2*a)
7  |  sinon
8  |  |  si b^2 - 4*a*c < 0 alors
9  |  |  |  Afficher « l'équation n'a pas de solution réelle »
10 |  |  fin
11 |  fin
12 fin
```

III Quelques exemples d'algorithmes à programmer

III 1 La suite de Syracuse

Présentation

La suite de Syracuse d'un nombre entier naturel N est définie par récurrence, de la manière suivante :
- $u_0 = N$
- Pour tout entier naturel n :
  - ▶ si u_n est pair, alors $u_{n+1} = \dfrac{u_n}{2}$;
  - ▶ si u_n est impair, alors $u_{n+1} = 3u_n + 1$

Travail à effectuer

1. Écrire un algorithme en *pseudo-code* qui en entrée lit l'entier N, puis calcule et affiche les 5 premiers termes de cette suite.
2. Tester l'algorithme avec différentes valeurs de N.
3. Modifier l'algorithme pour afficher les 20 premiers termes de la suite.
4. Programmer ce nouvel algorithme sur la calculatrice.
5. Quelle conjecture peut-on émettre sur la suite de Syracuse ?
6. Modifier l'algorithme pour qu'il s'exécute tant que le terme calculé de la suite n'est pas égal à 1. Quel est le risque de cet algorithme ?
7. Chercher sur internet une démonstration de cette conjecture.

III 2 La dichotomie

Présentation

On cherche un mot dans un dictionnaire de 60 000 mots ordonnés dans l'ordre alphabétique.

Une méthode non optimisée :
On ouvre le dictionnaire à la première page et on compare le mot recherché au premier mot du dictionnaire, puis au deuxième, puis au troisième, etc... jusqu'à trouver le mot recherché, ou arriver au dernier mot de la dernière page.
Évaluation du temps : avec 60 000 mots et une comparaison prenant un quart de seconde, il faudrait, dans le pire des cas, 15 000 secondes, soit plus de 4 heures, pour trouver le mot recherché ou se convaincre qu'il n'appartient pas au dictionnaire.

La méthode par dichotomie :
On ouvre le dictionnaire au milieu et on compare le mot recherché au mot médian. Si le mot recherché est avant le mot médian dans l'ordre alphabétique, on élimine la seconde moitié du dictionnaire, sans même la regarder. S'il est après le mot médian, on élimine la première. En recommençant avec le demi-dictionnaire restant, on élimine ensuite un demi-demi dictionnaire et on continue jusqu'à trouver le mot en question ou obtenir l'ensemble vide, auquel cas le mot recherché n'est pas dans le dictionnaire.
Évaluation du temps : On appelle **logarithme entier** d'un nombre x le nombre de fois qu'il faut le diviser par 2 pour obtenir un nombre inférieur ou égal à 1. On le note $elog(x)$.
$elog(60000) = 16$. Il ne faut donc plus que 16 comparaisons au pire des cas, soit 4 secondes.

Travail à effectuer

1. Soit f la fonction définie sur $\mathbb{R}$ par $f(x) = x^3 - 2x^2 + 5x - 1$.
   On admet que f est strictement croissante sur $\mathbb{R}$ et que l'équation $f(x) = 0$ admet une unique solution α sur l'intervalle $[0\,;1]$.
   Écrire un algorithme en *pseudo-code* qui renvoie un encadrement d'amplitude 10^{-4} près de α.
2. Cet algorithme étant précisément au programme de Mathématiques de TS, programmer cet algorithme sur la calculatrice.

III 3 Addition de 2 nombres exprimés en base 2 (Chapitre INFO 1)

Prochainement, au chapitre INFORMATIONS 1 : représenter l'information

IV Mini projet 01 : Les 3 prisonniers

☞ Télécharger le fichier pdf du mini-projet.