Master Sciences, Technologies, Santé
Mention Mathématiques - spécialité Enseignement des mathématiques

Algorithmique et graphes, thèmes du second degré

Algorithmique de base pour le lycée

Éric SOPENA, eric.sopena@labri.fr

Plan de la présentation

- **Introduction**
- **Les notions du programme**
- **Exemples de mise en œuvre à l'aide d'outils logiciels (AlgoBox, Python)**

Première partie

Introduction

Qu'est-ce qu'un algorithme ?

Un algorithme décrit un ***enchaînement*** d'opérations permettant, en un temps ***fini***, de ***résoudre*** toutes les instances d'un problème donné.

```
     3 8 4            6 1 9   <- données
  x    2 3         x    3 2   <-
  1 1 5 2            1 2 3 8
  7 6 8            1 8 5 7
  8 8 3 2          1 9 8 0 8  <- résultat
```

Un algorithme produit ainsi un **résultat** *dépendant des* **données** *(instance) du problème.*

Un premier exemple

Résolution d'une équation du 2nd degré de la forme

$$ax^2 + bx + c = 0$$

(données du problème : a, b et c)

1. je calcule le discriminant : $\Delta = b^2 - 4ac$
2. trois cas à considérer :
   a. $\Delta < 0$: pas de solution
   b. $\Delta = 0$: une solution double, $x = -b / 2a$
   c. $\Delta > 0$: deux solutions,
      $x_1 = (-b + \text{racine}(\Delta)) / 2a$
      $x_2 = (-b - \text{racine}(\Delta)) / 2a$

Un deuxième exemple

Algorithme d'Euclide : déterminer le PGCD de deux entiers positifs a et b (données du problème)

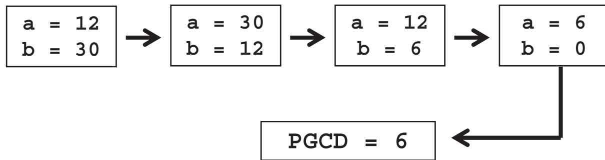
Basé sur les propriétés suivantes :

- PGCD(a,0) = a
- si b ≠ 0, PGCD(a,b) = PGCD(b, a mod b)

1. si b = 0, le PGCD vaut a
2. dans le cas contraire, remplacer a par b, et b par a mod b, puis recommencer en 1.

Cette séquence se répète tant que l'égalité b = 0 *ne se produit pas…*

Un deuxième exemple



1. si `b = 0`, le `PGCD` vaut `a`
2. dans le cas contraire, remplacer `a` par `b`, et `b` par `a mod b`, puis recommencer en 1.

Cette séquence se répète tant que l'égalité `b = 0` *ne se produit pas…*

Caractéristiques d'un algorithme

L'expression des algorithmes nécessite l'utilisation d'un ***langage*** (convention d'écriture) rigoureux, structuré, compréhensible et non ambigu.

Il n'existe pas de langage normalisé à ce niveau, mais les conventions adoptées sont suffisamment similaires pour être comprises de tous…

On s'attend à ce qu'un algorithme soit ***correct*** (le résultat obtenu est correct) et se termine ***en un temps fini***.

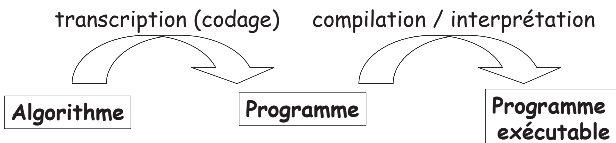
*Ces deux propriétés, **correction** et **terminaison**, sont souvent « prouvables » mathématiquement…*

… sans négliger le **temps de calcul** nécessaire…

*On parle de **complexité** de l'algorithme…*

Algorithmique et programmation

Un algorithme peut être ***transcrit*** dans un langage de programmation, afin d'être ***exécuté*** par une machine.



Tout algorithme doit être ***universel***, c'est-à-dire être indépendant du langage de programmation dans lequel il sera transcrit.

(cf. l'algorithme d'Euclide, IIIe s. av. J.C. !…)

Remarque fondamentale…

On ne peut concevoir un algorithme pour résoudre un problème donné que si l'on connait soi-même une méthode permettant de le résoudre !...

Ainsi, au lycée, l'enseignement de l'algorithmique se doit **d'accompagner** *l'enseignement des mathématiques…*

Deuxième partie

Les notions du programme

Éléments de base de l'algorithmique

- La notion de variable, l'opération d'affectation
  manipulation des données (calculs)
- Les opérations d'entrée-sortie
  récupération des données, affichage des résultats
- Les structures conditionnelles
  traitements alternatifs selon une condition
- Les structures répétitives
  répétition d'une séquence d'opérations

La notion de variable

Un algorithme a besoin de manipuler des ***« objets »*** (données du problème, données intermédiaires de calcul, résultats) ayant une certaine ***valeur***. Ces objets sont appelés des ***variables***.

Chaque variable possède :

- un nom (***identificateur***), permettant de la désigner sans ambiguïté dans l'algorithme,
- une ***valeur*** (on parle de contenu de la variable),
- un ***type*** (entier, caractère, …), déterminant l'ensemble des valeurs que peut prendre la variable et l'ensemble des opérations qui lui sont applicables.

La notion de variable

Représentation graphique usuelle :

Y
| 384 |

Une variable ***entière*** de nom ***Y*** ayant pour valeur ***384***…

Une variable ne peut avoir qu'une seule valeur.

Ainsi, si l'on modifie la valeur d'une variable, son ancienne valeur est perdue (« écrasée » par la nouvelle).

La notion de variable

Toutes les variables manipulées par un algorithme doivent être ***déclarées***, ce qui permet de préciser leur ***nom*** (identificateur) et leur ***type***.

```
Algorithme Exemple1
# cet algorithme illustre la déclaration de variables
variables
    Y : entier naturel
    réponse : caractère
début
    …
fin
```


L'opération d'affectation

L'opération d'affectation permet de ***donner une valeur*** à une variable (l'ancienne valeur, si elle existait, est remplacée par la nouvelle valeur).

Format général (syntaxe) :

```
<nom_variable>  ←  <expression>
```

- l'expression est évaluée, puis sa valeur est rangée dans la variable
- la variable et l'expression doivent avoir des types compatibles

`A ← 14`	la variable `A` reçoit la valeur `14`
`B ← A+5`	la variable `B` reçoit la valeur `19`

Les opérations d'entrée-sortie

Les opérations d'entrée-sortie permettent de communiquer avec l'utilisateur : ***récupérer la valeur*** de variables (e. g. les données du problème) ou ***afficher la valeur*** d'expressions (e. g. les résultats de l'algorithme).

Format général (syntaxe) :

```
Entrer ( <liste_de_variables> )
Afficher ( <liste_d_expressions> )
```

```
Entrer (A, B)
Afficher ( "La somme vaut", A+B )
```


Les structures conditionnelles

La structure ***Si-Alors-Sinon*** permet de réaliser telle ou telle séquence d'opérations ***selon la valeur d'une condition*** (expression logique, dont l'évaluation donne la valeur VRAI ou FAUX).

Format général (syntaxe) :

```
Si (<condition>)
Alors …
Sinon …
Fin_Si
```

ou

```
Si (<condition>)
Alors …
Fin_Si
```

Les structures conditionnelles

Exemple.

Calcul de la valeur absolue d'un entier relatif

```
Algorithme ValeurAbsolue
# cet algorithme calcule la valeur absolue d'un
# entier relatif
variables
    X : entier relatif
    absX : entier naturel
début
    Entrer (X)
    Si ( X ≥ 0 )
    Alors  absX ← X
    Sinon  absX ← -X
    Fin_Si
    Afficher (absX)
fin
```


Les structures répétitives (boucle Pour)

La boucle ***Pour*** permet de ***répéter*** une séquence d'opérations en faisant ***varier automatiquement*** une variable (appelée ***variable de boucle***). Cette variable de boucle prendra successivement toutes les valeurs d'un ***intervalle*** fixé.

Format général (syntaxe) :

```
Pour <variable> de <expr1> à <expr2>
Faire    …
Fin_Pour
```

La variable de boucle ne doit jamais être modifiée dans le corps de boucle !... (universalité)

Les structures répétitives (boucle Pour)

Exemple.

Affichage de la « table de 8 »

```
Algorithme TableDe8
# cet algorithme affiche la table de
# multiplication par 8
variables
    I : entier naturel
début
    Pour I de 1 à 10
    Faire
        Afficher ( I, " x ", 8, " = ", 8*I )
    Fin_Pour
fin
```

Affichage obtenu
1 x 8 = 8
2 x 8 = 16
3 x 8 = 24
4 x 8 = 32
5 x 8 = 40
6 x 8 = 48
7 x 8 = 56
8 x 8 = 64
9 x 8 = 72
10 x 8 = 80

Les structures répétitives (boucle Tantque)

La boucle ***Tantque*** permet de ***répéter*** une séquence d'opérations tant qu'une condition est vérifiée (c'est-à-dire tant que son évaluation retourne la valeur VRAI).

Format général (syntaxe) :

```
Tantque (<condition>)
Faire    …
Fin_Tantque
```

Attention : mal maîtrisée, cette structure peut conduire à un algorithme qui « boucle » indéfiniment (si la condition ne retourne jamais la valeur FAUX).

Les structures répétitives (boucle Tantque)

Exemple.

Calcul du premier multiple de 11 supérieur à 134

```
Algorithme PremierMultipleDe11SupérieurA134
# cet algorithme calcule le premier multiple de 11
# supérieur à 134
variables
    multiple : entier naturel
début
    multiple = 0
    Tantque ( multiple ≤ 134 )
    Faire  multiple ← multiple + 11
    Fin_Tantque
    Afficher ( multiple )
fin
```


« Faire tourner » un algorithme à la main

Intérêt.

- comprendre ce que fait un algorithme
- vérifier qu'un algorithme conçu est correct, et éventuellement le corriger…

Exemple.

Déterminer si un entier N est parfait
(égal à la somme de ses diviseurs stricts)

Ainsi…

- 6 est parfait : 1 + 2 + 3 = 6
- 14 n'est pas parfait : 1 + 2 + 7 = 10 ≠ 14
- 28 est parfait : 1 + 2 + 4 + 7 + 14 = 28

« Faire tourner » un algorithme à la main

```
Algorithme EntierParfait
# cet algorithme détermine si un entier N est parfait
variables
    N, diviseur, somme : entiers naturels
début
    Entrer (N)
    somme ← 0
    Pour diviseur de 1 à (N div 2)
    Faire
        Si (N mod diviseur = 0)
        Alors Somme ← Somme + diviseur
        Fin_Si
    Fin_Pour
    Si (N = Somme)
    Alors  Afficher (N, " est parfait ")
    Sinon  Afficher (N, " n'est pas parfait ")
    Fin_Si
fin
```


« Faire tourner » un algorithme à la main

```
    somme ← 0
    Pour diviseur de 1 à (N div 2)
    Faire
        Si (N mod diviseur = 0)
        Alors Somme ← Somme + diviseur
        Fin_Si
    Fin_Pour
```

opération	N	diviseur	somme
Entrer (N)	8	?	?
somme ← 0			0
(Pour) diviseur ← 1		1	
N mod diviseur = 0 ? oui			
Somme ← Somme + diviseur			1

« Faire tourner » un algorithme à la main

```
    somme ← 0
    Pour diviseur de 1 à (N div 2)
    Faire
        Si (N mod diviseur = 0)
        Alors Somme ← Somme + diviseur
        Fin_Si
    Fin_Pour
```

opération	N	diviseur	somme
	8	1	1
(Pour) diviseur ← 2		2	
N mod diviseur = 0 ? oui			
Somme ← Somme + diviseur			3
(Pour) diviseur ← 3		3	

« Faire tourner » un algorithme à la main

```
    somme ← 0
    Pour diviseur de 1 à (N div 2)
    Faire
        Si (N mod diviseur = 0)
        Alors Somme ← Somme + diviseur
        Fin_Si
    Fin_Pour
```

opération	N	diviseur	somme
	8	3	3
N mod diviseur = 0 ? non			
(Pour) diviseur ← 4		4	
N mod diviseur = 0 ? oui			
Somme ← Somme + diviseur			7

« Faire tourner » un algorithme à la main

```
    Si (N = Somme)
    Alors  Afficher (N, " est parfait ")
    Sinon  Afficher (N, " n'est pas parfait ")
    Fin_Si
```

opération	N	diviseur	somme
	8	4	7
(Pour) fin de boucle			
N = Somme ? non			
Affichage : non parfait !			

Troisième partie

Exemple de mise en œuvre à l'aide d'outils logiciels (AlgoBox, Python)

Mise en œuvre d'algorithmes : AlgoBox

AlgoBox est un logiciel libre, multi-plateforme et gratuit, d'aide à l'élaboration d'algorithmes dans l'esprit du programme de mathématiques du lycée. Il permet de créer un algorithme, de le faire tourner (exécuter), ou de le faire tourner pas à pas pour détecter d'éventuelles erreurs…

Il propose :

- une interface intuitive
- une saisie guidée (syntaxe « contrôlée »)
- des possibilités graphiques

Démonstration…

Pour télécharger AlgoBox :

http://www.xm1math.net/algobox/

Mise en œuvre d'algorithmes : AlgoBox

AlgoBox offre par ailleurs deux primitives graphiques :

- TRACERPOINT : on indique les coordonnées du point
- TRACERSEGMENT : on indique les coordonnées du point de départ et celles du point d'arrivée

Démonstration…

Programmation des algorithmes : Python

Python est un langage de programmation libre et gratuit, souvent utilisé comme langage d'apprentissage de la programmation.

Seule une petite partie des possibilités de ce langage est pertinente dans le cadre de l'enseignement de l'algorithmique au lycée.

On trouve en ligne de nombreux ouvrages, tutoriels, guides, …

Pour télécharger Python :

http://www.python.org/

Programmation des algorithmes : Python

Exemple de programme python.

Déterminer si un entier N est parfait

```
N = int (input("donnez un entier : "))
somme = 0
for diviseur in range(1,(N//2)+1):
    if (N % diviseur == 0):
        somme = somme + diviseur
if (N == somme):
    print (N, "est parfait")
else:
    print (N, "n'est pas parfait")
    print ("somme des diviseurs =", somme)
```

Démonstration…

Merci de votre attention...

Muhammad Ibn Musa al-Kwarizmi

timbre commémoratif - 06.09.83

Wikimedia Commons