

*Initiation à l'algorithmique.
Classe de 2^{nde}*

Définition d'«algorithme» dans l'Encyclopaedia Universalis :

" Un algorithme est une suite finie de règles à appliquer dans un ordre déterminé à un nombre fini de données pour arriver, en un nombre fini d'étapes, à un certain résultat, et cela indépendamment des données. "

Le mot algorithme provient du nom d'un célèbre mathématicien arabe de la première moitié du IX^{ème} siècle: Muhammad ibn Musa al Khwarizmi.

La traduction dans un langage compréhensible par l'ordinateur d'un algorithme est un *programme*.

Un algorithme est indépendant des ordinateurs qui l'exécutent et des langages dans lesquels il est traduit.

3 types de structures sont nécessaires (et a priori suffisantes) pour réaliser tout algorithme :

-La structure séquentielle : les instructions sont réalisées les unes à la suite des autres dans l'ordre d'écriture.

-La structure alternative : les instructions ne sont réalisées que si une condition particulière est remplie.

-La structure répétitive : des instructions identiques sont répétées un certain nombre de fois.

Exemple d'algorithme séquentiel qui calcule les nombres a et b dont on ne connaît que la somme et le produit.

Afficher « somme = »
Demander la somme S
Afficher « produit = »
Demander le produit P
Attribuer à D la valeur $S^2 - 4P$
$\frac{S - \sqrt{D}}{2}$
Attribuer à A la valeur
$\frac{S + \sqrt{D}}{2}$
Attribuer à B la valeur
2
Afficher « a= »
Afficher A
Afficher « b= »
Afficher B

*Traduction de cet algorithme en langage Casio
Graph (toutes séries)*

Dans le menu PRGM, choisir NEW et donner un nom au programme, « SOMPROD », puis EXE.

==SOMPROD==
"SOMME=" ↓
?→S ↓
"PRODUIT=" ↓
?→P ↓
S ² - 4P→D ↓
(S - √ D) ÷ 2→A ↓
(S + √ D) ÷ 2→B ↓
"A=" ↓
A▲
"B=" ↓
B

Traduction de cet algorithme en langage TI 83 Plus
Appuyer sur PRGM, et choisir NEW, puis Create New (touche 1). Taper le nom « SOMPROD », puis ENTER. Le signe « : » est appelé l'invite de commande.

: PROMPT S,P
: S ² - 4P→D
: (S - √ D)/2→A
: (S + √ D)/2→B
: Disp "A=",A
: Disp "B=",B

Il n'y a plus qu'à essayer. Par exemple en cherchant deux nombres dont la somme vaut 11 et le produit 24.
Recommencer avec une somme égale à 5 et un produit égal à 6, puis le contraire.
Recommencer avec une somme égale à 5 et un produit égal à 3, puis le contraire.

Le programme présente au moins un défaut :

dans le cas où les valeurs fournies sont fausses, il est possible que le problème n'admette pas de solution. Cela se traduit alors par $D < 0$, et une erreur de la calculatrice au moment de calculer \sqrt{D} .

Pour remédier à ce problème, on peut utiliser une **structure alternative** (structure de choix).

Elle se présente sous la forme :

Si condition

Alors

Instruction {Si vraie}

Sinon

Instruction {Si fausse}

Fin Si

Remarques : l'indentation est souvent utilisée pour clarifier l'algorithme. De même, les accolades signalent souvent un commentaire (facultatif) destiné à faciliter la compréhension de l'algorithme.

En l'occurrence, voilà ce que l'on peut ajouter

```
Afficher « somme = »
Demander la somme S
Afficher « produit = »
Demander le produit P
Attribuer à D la valeur  $S^2 - 4P$ 
Si ( $D < 0$ )
Alors
    Afficher « Valeurs impossibles »
Sinon
    Attribuer à A la valeur  $\frac{S - \sqrt{D}}{2}$ 
    Attribuer à B la valeur  $\frac{S + \sqrt{D}}{2}$ 
    Afficher « a = »
    Afficher A
    Afficher « b = »
    Afficher B
Fin Si
```

CASIO

```
===SOMPROD===
"SOMME=" ↵
?→S ↵
"PRODUIT=" ↵
?→P ↵
 $S^2 - 4P \rightarrow D$  ↵
If  $D < 0$  ↵
Then "IMPOSSIBLE" ↵
Else  $(S - \sqrt{D}) \div 2 \rightarrow A$  ↵
 $(S + \sqrt{D}) \div 2 \rightarrow B$  ↵
"A=" ↵
A▲
"B=" ↵
B ↵
IfEnd
```

TI

```
PROGRAM:SOMPROD
: PROMPT S,P
:  $S^2 - 4P \rightarrow D$ 
: If  $D < 0$ 
: Then
: Disp "IMPOSSIBLE"
: Else
:  $(S - \sqrt{D})/2 \rightarrow A$ 
:  $(S + \sqrt{D})/2 \rightarrow B$ 
: Disp "A=",A
: Disp "B=",B
: End
```

1^{er} PROJET : Calcul du PGCD de deux nombres par l'algorithme d'EUCLIDE.

Dans un premier temps, on utilisera la méthode non optimisée.

$\text{PGCD}(a,b)=\text{PGCD}(a-b,b)$ si $a>b$ ou $\text{PGCD}(a,b-a)$ si $b>a$

```
Afficher « PGCD »
Afficher « A= »
Demander A
Afficher « B= »
Demander B
« Début : »
Si A>B
Alors A - B → A
Sinon B - A → B
FinSi
Si AB ≠ 0 Aller à « Début »
Afficher « Le PGCD est : »
Afficher A+B
```

Attention : cette structure est peu utilisée en algorithmique. Nous l'exploitons ici car le langage de programmation que nous allons utiliser en dispose.

Astuce : puisque $AB=0$, cela signifie que $A=0$ ou $B=0$. La somme des deux est donc égale à celui des deux qui ne vaut pas zéro. Cela évite un test.

Amélioration : une structure répétitive va améliorer la lisibilité de l'algorithme, en évitant le « saut ».

```
Afficher « PGCD »
Afficher « A= »
Demander A
Afficher « B= »
Demander B
Tant que AB ≠ 0
  Si A>B
  Alors
    A - B → A
  Sinon
    B - A → B
  FinSi
Fin TantQue
Afficher « Le PGCD est : »
Afficher A+B
```

La structure « **Tant Que** » répète une série d'instructions tant que la condition initiale est remplie. Elle teste cette condition dès le début de la première répétition, et peut donc, si la condition est fautive dès le début, ne jamais faire appliquer les instructions qui suivent.

Traductions de Tant que

CASIO

```
While condition
Instruction(s)
WhileEnd {Bouton W.End}
```

TI

```
While condition
Instruction(s)
End
```

Essayer le programme avec $A=18$ et $B=12$. Il doit renvoyer 6 comme PGCD.

Essayer le programme avec $A=18$ et $B=3252$. Il doit renvoyer aussi 6 mais il met beaucoup (!) de temps.

On va donc l'optimiser, c'est-à-dire le rendre plus performant.

Il existe plusieurs façons d'optimiser un algorithme.

Soit on utilise une méthode différente pour arriver au même résultat (c'est ce que nous allons faire), soit on utilise des instructions différentes pour traduire la même méthode. L'optimisation peut se faire sur le temps de calcul (c'est en général ce qui est recherché en priorité) ou sur la place que prendra le programme dans la mémoire de l'ordinateur (surtout quand on dispose de très peu de place).

Ici, il suffit d'enlever B autant de fois que l'on peut de A, ou A, autant de fois que l'on peut de B, en une unique opération.

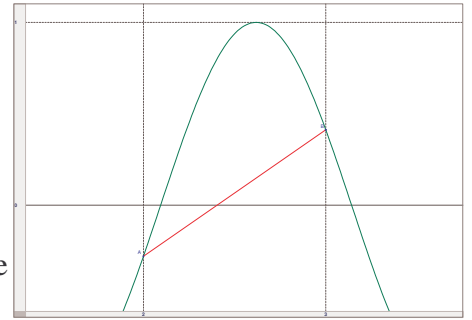
Or, si A est le plus grand, B « rentre » $E(A/B)$ (ou $\text{int}(A/B)$) fois dans A.

Utiliser cette propriété pour améliorer l'algorithme, traduire en programme et vérifier avec $A=18$ et $B=3252$.

2^{ème} projet : Recherche de racine d'une fonction par la méthode de la sécante.

Principe mathématique :

On constate que la courbe ci-contre coupe l'axe des abscisses entre les abscisses 2 et 3. Par ailleurs, $f(2)$ et $f(3)$ sont de signes contraires. On convient alors, en première approximation, de remplacer la courbe par le segment $[AB]$, et on estime que l'intersection de $[AB]$ avec l'axe des abscisses (à une abscisse notée c) est « assez proche » de l'intersection de la courbe avec ce même axe.



On constate par ailleurs que $f(c)$ est du même signe que $f(b)$, donc la courbe coupe nécessairement l'axe des abscisses entre les abscisses a et c .

On refait donc le même travail que précédemment, mais en prenant c comme nouvelle valeur de b .

On continue jusqu'à atteindre une précision que l'on a déterminée à l'avance.

Ce programme permet à la graph 25 de combler une de ses lacunes par rapport à ses grandes sœurs.

Soit $M(c,0)$ le point (AB) situé sur (Ox)

\vec{AB} et \vec{MA} sont donc colinéaires, or :

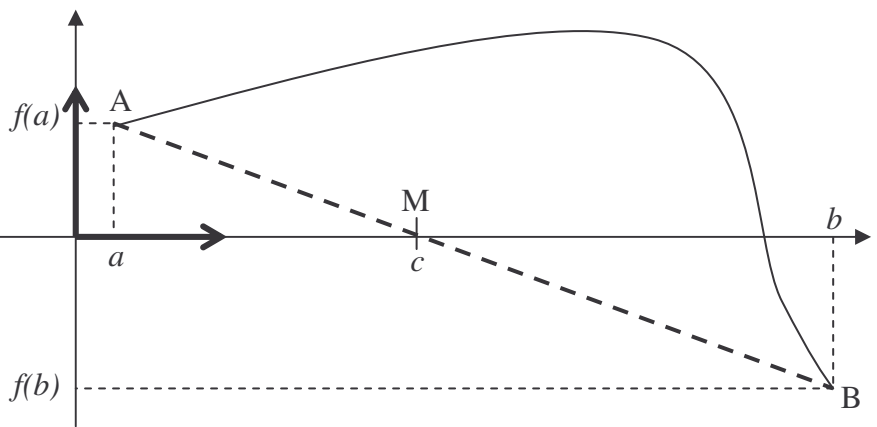
$$\vec{AB} \begin{pmatrix} b-a \\ f(b)-f(a) \end{pmatrix} \text{ et } \vec{MA} \begin{pmatrix} a-c \\ f(a) \end{pmatrix}$$

Le critère analytique de colinéarité donne :

$$(b-a) \times f(a) - (f(b)-f(a))(a-c) = 0$$

ce qui équivaut à :

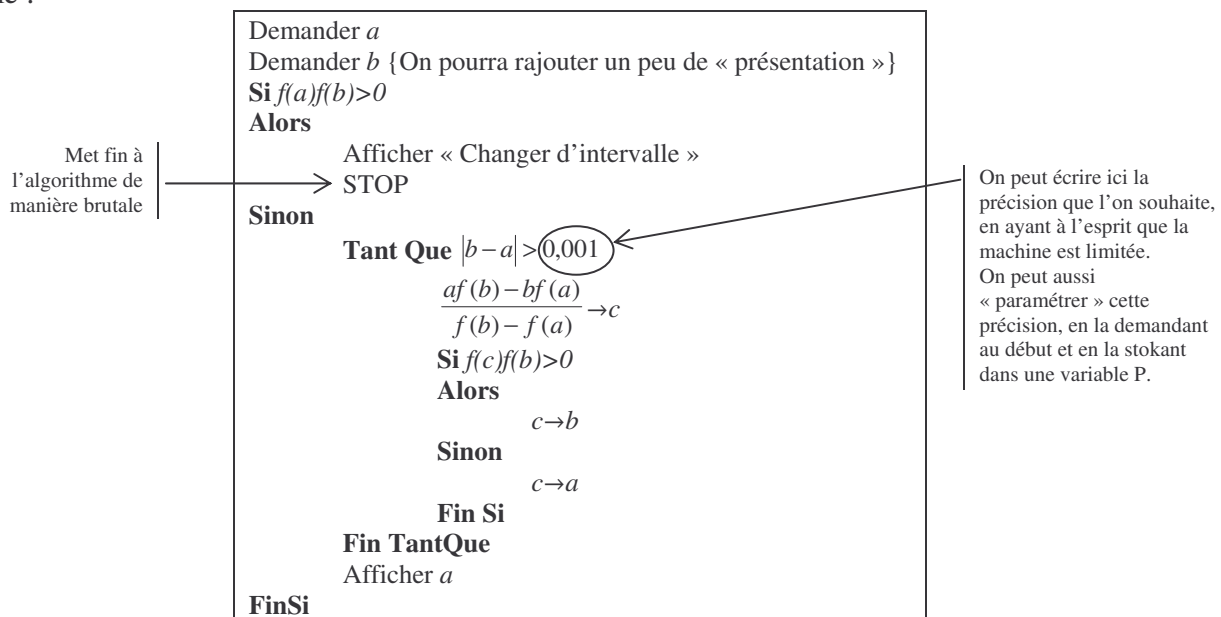
$$c = \frac{a \times f(b) - b \times f(a)}{f(b) - f(a)}$$



car A et B étant de part et d'autre de l'axe des abscisses, leurs ordonnées sont différentes, et on peut donc diviser par $f(b) - f(a)$.

Il reste à tester le signe de $f(c)$ (dans l'exemple ci-dessus, $f(c)$ est positif, pour savoir si c 'est a ou b qui prendra la valeur de c (dans l'exemple ci-dessus, c 'est a)

Algorithme :



Difficultés de mise en œuvre en langage CASIO

(Pas de problème pour TI, il suffit de passer par VARS, puis Y-VARS, puis Function, et Y₁)

Pour calculer Y₁(A), il est nécessaire, avec les CASIO, de :

- Stocker la valeur de A dans la variable X
- Calculer Y₁ (à l'aide de OPTN – VAR – GRPH – Y) et la stocker dans une variable.

Cette manipulation alourdit énormément la traduction de l'algorithme, mais il faut faire avec...

Traductions :

CASIO	TI
"A=" : ?→A ↵ "B=" : ?→B ↵ A→X ↵ Y1→U ↵ B→X ↵ Y1→V ↵ If UV>0 ↵ Then "CHANGER A ET B" ↵ Stop ↵ Else While Abs(B – A)>0.001 ↵ (AV – BU) ÷ (V – U) →C ↵ C→X ↵ Y1→Y ↵ If YV>0 ↵ Then C→B ↵ B→X ↵ Y1→V ↵ Else C→A ↵ A→X ↵ Y1→U ↵ IfEnd ↵ WhileEnd ↵ A	PROMPT A,B If Y1(A)Y1(B)>0 Then Disp "CHANGER A ET B" Stop Else While abs(B – A)>0.001 (AY1(B) – BY1(A))/(Y1(B) – Y1(A)) →C If Y1(C)Y1(B)>0 Then C→B Else C→A End End End Disp A

D'autres idées de programmes, en vrac...

- Ecrire une racine carrée sous la forme la plus simple possible.
- Déterminer les maxima et minima d'une fonction sur un intervalle donné. (utile pour graph 25)
- Déterminer les points d'intersections de deux courbes rentrées en Y₁ et Y₂, sur un intervalle donné. (utile pour graph 25)
- Déterminer une valeur approchée de l'aire comprise entre l'axe des abscisses et la courbe rentrée en Y₁, sur un intervalle donné. (utile pour les futurs élèves de terminale, mais faisable dès maintenant...)