Terminaison et Correction

Philippe Lac
(philippe.lac@ac-clermont.fr)
Malika More
(malika.more@u-clermont1.fr)

IREM Clermont-Ferrand

Stage Algorithmique

Année 2010-2011

Trois questions

Face à un algorithme

- Est-ce-qu'il donne un résultat ?
  ou bien est-ce-qu'il ne s'arrête jamais ?

  Terminaison
- Est-ce-qu'il donne le résultat attendu ?
  ou bien est-ce-qu'il calcule n'importe quoi ?

  Correction
- Est-ce-qu'il donne le résultat en un temps raisonnable ?
  ou bien est-ce-qu'il faut attendre plusieurs siècles ?

  (Complexité)

Parfois c'est simple

On est sûr qu'un algorithme se termine :

- Lorsque le nombre d'instructions à effectuer est connu à l'avance

On est sûr qu'un algorithme est correct :

- Lorsque l'algorithme reproduit directement la spécification

L'âge de Toutou

Fonction `AgeHumain(x)`

début
  si $x \geq 19$ **alors**
    Donner à y la valeur $10 \times x - 90$
  sinon
    si $x \geq 17$ **alors**
      Donner à y la valeur $6 \times x - 14$
    sinon
      si $x \geq 2$ **alors**
        Donner à y la valeur $4 \times x + 20$
      sinon
        Donner à y la valeur $8 \times x + 12$
      fin
    fin
  fin
  retourner : y
fin

Un être humain et un chien ne vieillissent pas de la même manière.
Pour déterminer l'équivalent humain de l'âge d'un chien de moins de 15 kg, on utilise le tableau ci-dessous, où x représente l'âge réel de l'animal (en années), et y l'équivalent humain en terme de vieillissement.

x	y
moins de 2	$8x + 12$
de 2 à 17	$4x + 20$
de 17 à 19	$6x - 20$
19 et plus	$10x - 90$

L'âge de Toutou

Fonction `AgeHumain(x)`

début
- **si** $x \geq 19$ **alors**
  - Donner à y la valeur $10 \times x - 90$
- **sinon**
  - **si** $x \geq 17$ **alors**
    - Donner à y la valeur $6 \times x - 14$
  - **sinon**
    - **si** $x \geq 2$ **alors**
      - Donner à y la valeur $4 \times x + 20$
    - **sinon**
      - Donner à y la valeur $8 \times x + 12$
    - **fin**
  - **fin**
- **fin**
- **retourner** : y

fin

Terminaison

- C'est un « algorithme-calcul »
- Il se termine simplement quand toutes les instructions sont effectuées

Correction

- L'algorithme correspond à la formule
- La formule est très simple

Somme des n premiers entiers

Fonction `Somme(`n`)`

début
  Donner à S la valeur 0
  pour *i de 1 à n* **faire**
    Donner à S la valeur $S + i$
  fin
  retourner : S
fin

Somme des n premiers entiers

Fonction `Somme`(n)

début
  Donner à S la valeur 0
  pour *i de 1 à n* **faire**
    Donner à S la valeur $S + i$
  fin
  retourner : S
fin

Terminaison

- Il y a une boucle, mais le nombre de passages dans la boucle est connu *a priori*
- Donc l'algorithme se termine toujours

Somme des n premiers entiers

Fonction `Somme(`n`)`

début
  Donner à S la valeur 0
  pour *i de 1 à n* **faire**
    Donner à S la valeur $S + i$
  fin
  retourner : S
fin

Correction

- On vérifie que l'initialisation de S est correcte
- On vérifie que l'itération est correcte
- On vérifie que le nombre d'itérations est correct
- Donc l'algoritmhe est correct

Souvent c'est moins simple

Terminaison :

- On ne connaît pas *a priori* le nombre d'instructions effectuées

Correction :

- La spécification est compliquée
- La spécification ne dit pas comment obtenir le résultat

Dans les exemples qui suivent, on va se concentrer sur le cas d'une boucle `tant que`

Calcul du pgcd de deux entiers

Fonction `Euclide(`*a*,*b*`)`

début
- Donner à x la valeur a
- Donner à y la valeur b
- **répéter tant que** $y \neq 0$
  - Donner à *temp* la valeur y
  - Donner à y la valeur $x \mod y$
  - Donner à x la valeur *temp*
- **fin**
- **retourner** : x

fin

Calcul du pgcd de deux entiers

Fonction `Euclide(`*a*,*b*`)`

début
  Donner à *x* la valeur *a*
  Donner à *y* la valeur *b*
  répéter tant que $y \neq 0$
    Donner à *temp* la valeur *y*
    Donner à *y* la valeur $x \mod y$
    Donner à *x* la valeur *temp*
  fin
  retourner : *x*
fin

L'algorithme se termine si la condition d'arrêt de la boucle se réalise

- C'est-à-dire si *y* s'annule
- Dans la boucle, *y* est remplacé par un reste $\mod y$
- À chaque passage, *y* décroît strictement, tout en restant ≥ 0
- Par conséquent, *y* finit par atteindre 0
- Et on sort de la boucle
- Conclusion : l'algorithme se termine

Calcul du pgcd de deux entiers

Fonction `Euclide(a,b)`

début
- Donner à x la valeur a
- Donner à y la valeur b
- **répéter tant que** $y \neq 0$
  - Donner à *temp* la valeur y
  - Donner à y la valeur $x \mod y$
  - Donner à x la valeur *temp*
- **fin**
- **retourner** : x

fin

L'algorithme se termine si la condition d'arrêt de la boucle se réalise

- C'est-à-dire si y s'annule
- Dans la boucle, y est remplacé par un reste $\mod y$
- À chaque passage, y décroît strictement, tout en restant ≥ 0
- Par conséquent, y finit par atteindre 0
- Et on sort de la boucle
- Conclusion : l'algorithme se termine

Calcul du pgcd de deux entiers

Fonction `Euclide(a,b)`

début
- Donner à x la valeur a
- Donner à y la valeur b
- **répéter tant que** $y \neq 0$
  - Donner à *temp* la valeur y
  - Donner à y la valeur $x \mod y$
  - Donner à x la valeur *temp*
- **fin**
- **retourner** : x

fin

L'algorithme est supposé calculer $\text{pgcd}(a, b)$

- Au départ, on a $x = a$ et $y = b$
- Dans la boucle, on remplace (x, y) par $(y, x \mod y)$
- On sait que $\text{pgcd}(x, y) = \text{pgcd}(y, x \mod y)$
- Donc à chaque étape, $\text{pgcd}(x, y) = \text{pgcd}(a, b)$
- En sortie de boucle, on a $y = 0$ et on renvoie x
- On sait que $\text{pgcd}(x, 0) = x$
- Conclusion : l'algorithme calcule bien $\text{pgcd}(a, b)$

Calcul du pgcd de deux entiers

Fonction `Euclide(a,b)`

début
- Donner à x la valeur a
- Donner à y la valeur b
- **répéter tant que** $y \neq 0$
  - Donner à *temp* la valeur y
  - Donner à y la valeur $x \mod y$
  - Donner à x la valeur *temp*
- **fin**
- **retourner** : x

fin

L'algorithme est supposé calculer $\mathrm{pgcd}(a, b)$

- Au départ, on a $x = a$ et $y = b$
- Dans la boucle, on remplace (x, y) par $(y, x \mod y)$
- On sait que $\mathrm{pgcd}(x, y) = \mathrm{pgcd}(y, x \mod y)$
- Donc à chaque étape, $\mathrm{pgcd}(x, y) = \mathrm{pgcd}(a, b)$
- En sortie de boucle, on a $y = 0$ et on renvoie x
- On sait que $\mathrm{pgcd}(x, 0) = x$
- Conclusion : l'algorithme calcule bien $\mathrm{pgcd}(a, b)$

Outil pour la terminaison

Définition

On appelle convergent une quantité qui prend ses valeurs dans un ensemble bien fondé et qui diminue strictement à chaque passage dans une boucle.

Remarques

- Un ensemble *bien fondé* est un ensemble totalement ordonné dans lequel il n'existe pas de suite infinie strictement décroissante.
- En particulier, $\mathbb{N}$, ou $\mathbb{N}^k$ munis de l'*ordre lexicographique*, sont des ensembles bien fondés.
- On a $(a, b) < (c, d)$ pour l'ordre lexicographique lorsque $a < c$ ou $a = c$ et $b < d$.

Outil pour la terminaison

Définition

On appelle convergent une quantité qui prend ses valeurs dans un ensemble bien fondé et qui diminue strictement à chaque passage dans une boucle.

Propriété

L'existence d'un convergent pour une boucle garantit que l'algorithme finit par en sortir.

Terminaison de l'algorithme d'Euclide

Fonction `Euclide(`*a*,*b*`)`

début
- Donner à *x* la valeur *a*
- Donner à *y* la valeur *b*
- **répéter tant que** $y \neq 0$
  - Donner à *temp* la valeur *y*
  - Donner à *y* la valeur $x \mod y$
  - Donner à *x* la valeur *temp*
- **fin**
- **retourner** : *x*

fin

On a vu que *y* est un convergent

Outil pour la correction

Définition

On appelle invariant de boucle une propriété qui, si elle est vraie avant l'entrée dans une boucle, reste vraie après chaque passage dans cette boucle, et donc est vraie aussi à la sortie de cette boucle.

Remarque

Analogie évidente avec une preuve par récurrence :

- Entrée de boucle $\longrightarrow$ initialisation
- Passage dans la boucle $\longrightarrow$ hérédité
- Sortie de boucle $\longrightarrow$ conclusion

Outil pour la correction

Définition

On appelle invariant de boucle une propriété qui, si elle est vraie avant l'entrée dans une boucle, reste vraie après chaque passage dans cette boucle, et donc est vraie aussi à la sortie de cette boucle.

Propriété

La mise en évidence d'un invariant de boucle adapté permet de prouver la correction d'un algorithme.

Correction de l'algorithme d'Euclide

Fonction `Euclide(`*a*,*b*`)`

début
- Donner à x la valeur a
- Donner à y la valeur b
- **répéter tant que** $y \neq 0$
  - Donner à *temp* la valeur y
  - Donner à y la valeur $x \mod y$
  - Donner à x la valeur *temp*
- **fin**
- **retourner** : x

fin

On a vu que
$pgcd(x, y) = pgcd(a, b)$ est un invariant de boucle

Algorithme « aléatoire » d'après Dowek

Fonction `Aléatoire`(n, p)

début
- Donner à x la valeur n
- Donner à y la valeur p
- **répéter tant que** $x \neq 0$ **ou** $y \neq 0$
  - **si** $y > 0$ **alors**
    - Donner à y la valeur $y - 1$
  - **sinon**
    - **si** $x > 0$ **alors**
      - Donner à x la valeur $x - 1$
      - Donner à y une valeur au hasard
    - **fin**
  - **fin**
- **fin**
- **retourner** : (x, y)

fin

Algorithme « aléatoire » d'après Dowek

Fonction `Aléatoire`(n, p)

début
  Donner à x la valeur n
  Donner à y la valeur p
  répéter tant que $x \neq 0$ **ou** $y \neq 0$
    si $y > 0$ **alors**
      Donner à y la valeur $y - 1$
    sinon
      si $x > 0$ **alors**
        Donner à x la valeur $x - 1$
        Donner à y une valeur au hasard
      fin
    fin
  fin
  retourner : (x, y)
fin

Cet algorithme se termine-t-il toujours ?

- Si les nombres « aléatoires » sont bornés, c'est facile
- Si on considère des nombres non bornés, c'est plus compliqué, mais on y arrive aussi
- C'est l'occasion de parler du hasard en informatique

Algorithme « aléatoire » d'après Dowek

Fonction `Aléatoire`(n, p)

début
- Donner à x la valeur n
- Donner à y la valeur p
- **répéter tant que** $x \neq 0$ **ou** $y \neq 0$
  - **si** $y > 0$ **alors**
    - Donner à y la valeur $y - 1$
  - **sinon**
    - **si** $x > 0$ **alors**
      - Donner à x la valeur $x - 1$
      - Donner à y une valeur au hasard
    - **fin**
  - **fin**
- **fin**
- **retourner** : (x, y)

fin

Cet algorithme se termine-t-il toujours ?

- Si les nombres « aléatoires » sont bornés, c'est facile
- Si on considère des nombres non bornés, c'est plus compliqué, mais on y arrive aussi
- C'est l'occasion de parler du hasard en informatique

Algorithme « aléatoire » d'après Dowek

Fonction `Aléatoire`(n, p)

début
- Donner à x la valeur n
- Donner à y la valeur p
- **répéter tant que** $x \neq 0$ **ou** $y \neq 0$
  - **si** $y > 0$ **alors**
    - Donner à y la valeur $y - 1$
  - **sinon**
    - **si** $x > 0$ **alors**
      - Donner à x la valeur $x - 1$
      - Donner à y une valeur au hasard
    - **fin**
  - **fin**
- **fin**
- **retourner** : (x, y)

fin

Cet algorithme se termine-t-il toujours ?

- Si les nombres « aléatoires » sont bornés, c'est facile
- Si on considère des nombres non bornés, c'est plus compliqué, mais on y arrive aussi
- C'est l'occasion de parler du hasard en informatique

Algorithme « aléatoire » d'après Dowek

Fonction `Aléatoire`(n, p)

```
début
    Donner à x la valeur n
    Donner à y la valeur p
    répéter tant que x ≠ 0 ou y ≠ 0
        si y > 0 alors
            Donner à y la valeur y − 1
        sinon
            si x > 0 alors
                Donner à x la valeur x − 1
                Donner à y une valeur au hasard
            fin
        fin
    fin
    retourner : (x, y)
fin
```

Cet algorithme se termine-t-il toujours ?

- Si les nombres « aléatoires » sont bornés, c'est facile
- Si on considère des nombres non bornés, c'est plus compliqué, mais on y arrive aussi
- C'est l'occasion de parler du hasard en informatique

Suite de Syracuse

Fonction `Syracuse(`n`)`

début
  Donner à x la valeur n
  répéter tant que $x \neq 1$
    si x *est pair* **alors**
      Donner à x la valeur $x/2$
    sinon
      Donner à x la valeur $3x + 1$
    fin
  fin
  retourner : x
fin

Suite de Syracuse

Fonction `Syracuse(`n`)`

début
  Donner à x la valeur n
  répéter tant que $x \neq 1$
    si x *est pair* **alors**
      Donner à x la valeur $x/2$
    sinon
      Donner à x la valeur $3x + 1$
    fin
  fin
  retourner : x
fin

On ne connaît pas de convergent car personne ne sait si cet algorithme se termine toujours !

Exponentielle rapide

Fonction `ExpoRap(a,n)`

début
- Donner à p la valeur 1
- Donner à b la valeur a
- Donner à m la valeur n
- **répéter tant que** $m > 0$
  - **si** m *impair* **alors**
    - Donner à p la valeur $p \times b$
  - **fin**
  - Donner à b la valeur b^2
  - Donner à m la valeur $\mathrm{quo}(m, 2)$
- **fin**
- **retourner** : p

fin

Exponentielle rapide

Fonction `ExpoRap(a,n)`

début
- Donner à p la valeur 1
- Donner à b la valeur a
- Donner à m la valeur n
- **répéter tant que** $m > 0$
  - **si** m *impair* **alors**
    - Donner à p la valeur $p \times b$
  - **fin**
  - Donner à b la valeur b^2
  - Donner à m la valeur $\mathrm{quo}(m, 2)$
- **fin**
- **retourner** : p

fin

Terminaison :

- On se convainc facilement que m est un convergent
- Conclusion : l'algorithme se termine

Exponentielle rapide

Fonction `ExpoRap(a,n)`

début
- Donner à p la valeur 1
- Donner à b la valeur a
- Donner à m la valeur n
- **répéter tant que** $m > 0$
  - **si** m *impair* **alors**
    - Donner à p la valeur $p \times b$
  - **fin**
  - Donner à b la valeur b^2
  - Donner à m la valeur $\mathrm{quo}(m, 2)$
- **fin**
- **retourner** : p

fin

Prenons $a = 5$ et $n = 13$

- Avant la boucle
  - $p = 1$, $b = 5$, $m = 13$
- Boucle : $m = 13$ (impair)
  - $p = 5$, $b = 5^2$, $m = 6$
- Boucle : $m = 6$ (pair)
  - $p = 5$, $b = 5^4$, $m = 3$
- Boucle : $m = 3$ (impair)
  - $p = 5^5$, $b = 5^8$, $m = 1$
- Boucle : $m = 1$ (impair)
  - $p = 5^{13}$, $b = 5^{16}$, $m = 0$
- $m = 0$: Sortie de boucle
- On retourne $p = 5^{13}$

Exponentielle rapide

Fonction `ExpoRap(a,n)`

début
- Donner à p la valeur 1
- Donner à b la valeur a
- Donner à m la valeur n
- **répéter tant que** $m > 0$
  - **si** *m impair* **alors**
    - Donner à p la valeur $p \times b$
  - **fin**
  - Donner à b la valeur b^2
  - Donner à m la valeur $\mathrm{quo}(m, 2)$
- **fin**
- **retourner** : p

fin

Correction

- On vérifie que $p \times b^m = a^n$ est un invariant de boucle :
  - Avant : $p \times b^m = 1 \times b^n = a^n \longrightarrow$ OK
  - Si $m = 2k$: $p' \times b'^{m'} = p \times (b^2)^k = p \times b^{2k} = p \times b^m = a^n \longrightarrow$ OK
  - Si $m = 2k + 1$: $p' \times b'^{m'} = p \times b \times (b^2)^k = p \times b^{2k+1} = p \times b^m = a^n \longrightarrow$ OK
  - Sortie : $m = 0$ donc $p \times b^m = p$
- Conclusion : l'algorithme calcule bien a^n

Algorithme d'Euclide étendu

On calcule simultanément le pgcd de a et de b et un couple de coefficients de Bézout, entiers relatifs (u, v) tels que $a \times u + b \times v = \mathrm{pgcd}(a, b)$

q	r	u	v
	$r_0 = a$	$u_0 = 1$	$v_0 = 0$
	$r_1 = b$	$u_1 = 0$	$v_1 = 1$
$q_2 = \mathrm{quo}(r_0, r_1)$	$r_2 = r_0 - r_1 \cdot q_2$	$u_2 = u_0 - u_1 \cdot q_2$	$v_2 = v_0 - v_1 \cdot q_2$
$\ldots$	$\ldots$	$\ldots$	$\ldots$
$q_{i+1} = \mathrm{quo}(r_{i-1}, r_i)$	$r_{i+1} = r_{i-1} - r_i \cdot q_{i+1}$	$u_{i+1} = u_{i-1} - u_i \cdot q_{i+1}$	$v_{i+1} = v_{i-1} - v_i \cdot q_{i+1}$
$\ldots$	$\ldots$	$\ldots$	$\ldots$
$q_n = \ldots$	$r_n = \ldots$	$u_n = \ldots$	$v_n = \ldots$
$q_{n+1} = \ldots$	$r_{n+1} = 0$		

Exercice

1. Réaliser les calculs sur un exemple et vérifier les résultats.
2. Montrer qu'on a $r_n = \mathrm{pgcd}(a, b)$ et $a \times u_n + b \times v_n = r_n$.
3. Écrire l'algorithme d'Euclide étendu, et le programmer en `Scilab` ou en `Xcas`.
4. Prouver la terminaison et la correction de cet algorithme.

FIN