DEUX ALGORITHMES DU PGCD, PLUS UN

Henri LOMBARDI
Irem de Besançon

Résumé : *Cet article sert à illustrer le point de vue de l'auteur selon lequel la plupart (sinon toutes) des démonstrations en mathématiques relèvent en dernière analyse d'une démarche algorithmique. Nous proposons une analyse du théorème du pgcd, tel qu'il résulte de l'algorithme d'Euclide d'une part, et tel qu'il résulte de la démonstration abstraite le plus couramment présentée aujourd'hui d'autre part. Nous argumentons pour mettre en évidence que la preuve abstraite est le déguisement d'une preuve qui est également de nature algorithmique. Cependant l'algorithme sous-jacent à la démonstration abstraite n'est pas le même que celui d'Euclide. Aussi est-il intéressant de les comparer. Nous terminons en indiquant dans une annexe comment se situer par rapport à une autre manière populaire de calculer le pgcd de deux entiers, basée sur la décomposition en facteurs premiers.*

L'anthyphérèse

Dans « Les Eléments », ce que nous appelons l'algorithme d'Euclide est avant tout une méthode géométrique pour trouver une plus grande commune mesure, si elle existe, à deux grandeurs de même nature.

Considérons par exemple deux segments de droites, AB et CD qui admettent pour commune mesure un segment EF, ce qui signifie que l'unité EF est contenue un nombre entier de fois dans AB et CD. Supposant AB > CD, on commence par retrancher autant de fois qu'il est possible le segment CD du segment AB. S'il ne reste rien, c'est que CD était une commune mesure, et c'est manifestement la plus grande possible. S'il reste quelque chose, que nous notons GH, alors toute commune mesure à AB et CD est aussi une commune mesure à CD et GH. On peut donc continuer le processus. Comme GH est strictement plus petit que CD, le nombre de fois que EF est contenu dans GH est inférieur au nombre de fois qu'il est contenu dans CD. Ainsi le processus s'arrête après un nombre fini d'étapes et fournit une commune mesure, nécessairement multiple entier de EF.

Voici donc un résultat qui n'était pas a priori évident : la commune mesure trouvée est multiple de toute autre commune mesure.

* Cet article est inspiré d'un atelier proposé aux journées de l'APMEP de la Rochelle. Le texte original proposé pour cet atelier a été publié dans le Bulletin APMEP. Concernant l'orthographe de Bezout (ou Bézout), il s'agit d'un problème fort controversé, je laisse l'éditeur choisir celle qu'il préfère.
Henri Lombardi

Si maintenant on raisonne avec les nombres entiers a et b qui mesurent AB et CD par rapport à l'unité EF , le processus devient un calcul avec des entiers positifs qui fournit un commun diviseur g de a et b, et tout autre diviseur commun de a et b divise g, donc est plus petit que g.

Ceci démontre que le plus grand commun diviseur de a et b est multiple de tout autre diviseur commun.

Par exemple avec $a = 268$ et $b = 108$, on trouve que les diviseurs communs à a et b sont successivement les mêmes que les diviseurs communs aux couples successifs que voici

$$\begin{bmatrix} 268 \\ 108 \end{bmatrix} \to \begin{bmatrix} 160 \\ 108 \end{bmatrix} \to \begin{bmatrix} 52 \\ 108 \end{bmatrix} \to \begin{bmatrix} 52 \\ 56 \end{bmatrix} \to$$

$$\begin{bmatrix} 52 \\ 4 \end{bmatrix} \to \begin{bmatrix} 48 \\ 4 \end{bmatrix} \to \begin{bmatrix} 44 \\ 4 \end{bmatrix} \to \ldots \to \begin{bmatrix} 4 \\ 4 \end{bmatrix} .$$

Et donc les diviseurs communs à 268 et 108 sont les diviseurs de 4, qui est le plus grand d'entre eux. La maîtrise d'un algorithme de division permet naturellement d'accélérer significativement le calcul par rapport à l'algorithme qui n'utilise que des soustractions :

$268 = 2 \times 108 + 52$ i.e. $52 = 268 - 2 \times 108$
$108 = 2 \times 52 + 4$ i.e. $4 = 108 - 2 \times 52$
$52 = 13 \times 4$ (reste 0)

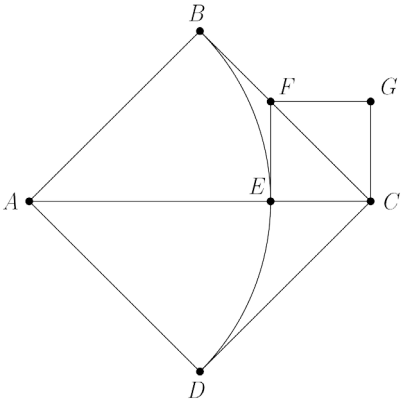
En outre en remontant les égalités ci-dessus on obtient le pgcd 4 comme une combinaison linéaire simple de 268 et 108 :

$$4 = 108 - 2 \times 52 = 108 - 2 \times (268 - 2 \times 108)$$
$$= 5 \times 108 - 2 \times 268$$

Le processus de soustractions alternées (on retranche autant de fois qu'on peut CD de AB, puis autant de fois qu'on peut GH de CD, puis...) s'appelle l'*anthyphérèse* dans les textes anciens.

Plus que pour chercher une commune mesure quand il en existe une, ce procédé était surtout utilisé pour montrer l'impossibilité d'une commune mesure entre deux grandeurs données, lorsqu'on montre que l'anthyphérèse ne peut certainement pas s'arrêter après un nombre fini d'étapes.

Voyons ceci sur l'exemple du côté et de la diagonale du carré :



L'anthyphérèse de la diagonale et du côté du carré

On démarre avec le côté et la diagonale d'un carré ABCD. Le cercle de centre A passant par B coupe la diagonale AC en E, de sorte que AB = AE. On retranche le côté AB de la diagonale AC et l'on obtient EC. On considère alors le carré EFGC. On a FB = FE = EC, la première égalité parce que FB et FE sont les deux tangentes au cercles menées depuis le point F . Ainsi une commune mesure à AB et AC est aussi une commune mesure à AB et EF, donc à BC et BF = EF, donc à FC et EF :

la diagonale et le côté du carré EFGC. Le processus va se répéter à l'identique en remplaçant le carré ABCD par le carré EFGC. Comme le côté EF est moindre que la moitié du côté AB, les côtés des carrés successifs deviendront moindre que tout segment donné par avance (axiome d'Archimède), et ceci montre qu'une commune mesure à AB et AC est impossible.

Le théorème du pgcd

Le « théorème du pgcd » qui nous occupe est le suivant :

Théorème 1 Si a et b sont deux entiers strictement positifs, le plus grand diviseur commun $g > 0$ de a et b est multiple de tout autre diviseur commun.

En fait on a remarqué plus tard que le théorème 1 résulte du théorème suivant, qui concentre toute la difficulté du problème dans un énoncé simple, mais plus fort :

Théorème 2 Si a et b sont deux entiers strictement positifs, il existe un entier $g > 0$ de la forme $ua + vb$ avec $u, v \in \mathbf{Z}$, qui divise a et b.
L'égalité $g = ua + vb$ s'appelle aussi une *relation de Bézout* entre a et b.

Démonstration que le théorème 2 implique le théorème 1. Soit h un diviseur commun de a et b. On obtient $a = ha_1$, $b = hb_1$ donc $g = au + bv = h(a_1u + b_1v)$. Ainsi g est à la fois plus grand que h et multiple de h. Le théorème 1 est bien satisfait.

Le théorème 2 doit donc être compris comme affirmant deux propriétés inattendues du plus grand des diviseurs communs à a et b.

1. La première est que tout diviseur commun à a et b doit diviser g .
2. La seconde est que g peut s'écrire sous la forme $ua + vb$ avec $u, v \in \mathbf{Z}$.

Nous envisageons maintenant deux démonstrations du théorème 2.

Une preuve abstraite classique

Tout d'abord on établit le lemme suivant :

Lemme 3 Toute partie non vide de $\mathbf{N}$ admet un plus petit élément.

Preuve du lemme. Si une partie V de $\mathbf{N}$ contient un élément c alors le plus petit élément de l'ensemble fini $V \cap [\![0...c]\!]$ est aussi le plus petit élément de V .

Preuve abstraite du théorème 2. On considère l'ensemble $V(a, b)$ défini comme suit :

$$V(a, b) = V = (a\mathbf{Z} + b\mathbf{Z}) \cap \mathbf{N}^* = \{ x > 0 \mid \exists u, v \in \mathbf{Z}, x = ua + vb \} .$$

L'ensemble V contient a et b : il est non vide. Soit $g = ua + vb$ le plus petit élément de V. Montrons par l'absurde qu'il divise a et b.

Si ce n'était pas le cas, on aurait par exemple : g ne divise pas a. Alors en divisant a par g on obtiendrait un reste $r \in \{1, ..., g - 1\}$ avec $a = gq + r$. On aurait alors

$$r = a - gq = a(1 - uq) + (- vq)b$$

et ceci montre que $r \in V$, ce qui contredit le fait que g est le plus petit élément de V .

Une preuve par algorithme

NB : Dans tous les algorithmes encadrés, une ligne qui commence par un # est un commentaire et ne fait pas partie de l'algorithme

proprement dit. Lorsqu'un tel algorithme est implémenté sur machine, on « donne les entrées » (que l'on choisit arbitrairement) à la machine : ce sont les valeurs que l'on attribue aux variables déclarées dans la rubrique **Entrée**. La machine exécute l'algorithme et elle « donne en sortie » les valeurs des variables déclarées dans la rubrique **Sortie**.

Algorithme 1 *Algorithme de calcul du* pgcd*, à la Euclide.*

Entrée : Deux entiers naturels a et b , > 0.
Sortie : Leur pgcd g.
Début
boucle
Tant que b ≠ 0 **faire**
Remplacer a et b respectivement par :
b et le reste de la division de a par b ;
fin tant que ;
fin de boucle
g ← a
Fin.

On considère l'algorithme 1 qui est conforme au calcul que nous avons présenté en introduction, lequel est à peu près celui exposé par Euclide.

Nous affirmons que pour n'importe quelles valeurs entrées pour a et b, l'algorithme fournit un élément g qui satisfait les propriétés requises dans le théorème 2.

Sous une forme un peu plus précise, et sans détruire le contenu des identificateurs a et b donnés en entrée, l'algorithme se réécrit comme dans l'encadré 1 bis. Il nécessite trois identificateurs pour gérer les remplacements successifs des couples considérés. On note que même lorsque a et b sont très grands (une centaine de chiffres par exemple) et si l'algorithme utilise une centaine de divisions, il n'y a besoin que de ces trois identificateurs pour faire le travail. L'algorithme est donc économe du point de vue de l'espace de travail : sur machine, la gomme ne coûte pas cher et le papier est presque inusable, insensible à la gomme.

Algorithme 1 bis *Algorithme de calcul du pgcd, plus précis.*

Entrée : Deux entiers naturels a et b , > 0.
Sortie : Leur pgcd g.
Variables locales : a',b',b" : entiers ≥ 0 ;
Début
initialisation
a' ← a ; b' ← b ;
boucle
Tant que b' > 0 **faire**
b" ← reste de la division de a' par b' ;
a' ← b' ; b' ← b"
fin tant que ;
fin de boucle
g ← a'
Fin.

Preuve de terminaison. Nous vérifions ici que l'algorithme 1 bis s'arrête bien après un nombre fini d'étapes. Cela tient à ce que, à chaque exécution de la boucle, b' est remplacé par un nombre strictement plus petit dans **N**. Il atteint donc nécessairement la valeur 0 (ce qui est le seul moyen de sortir de la boucle) en un nombre fini d'étapes. Il faut également noter que l'instruction « $b"$ ← reste de la division de a' par b' » est toujours exécutée avec un $b' > 0$ et qu'il n'y aura donc pas d'erreur (la division par 0 n'est pas définie) lors de l'exécution du programme.

Preuve de correction. Nous l'avons déjà donnée dans le paragraphe d'introduction, mais

bis repetita placent. Notons a_k et b_k (k = 0, 1, ... ,n) les valeurs successives prises par a' et b' chaque fois qu'on fait le test « $b' = 0$? » en début de boucle. On démarre avec $a_0 = a$ et $b_0 = b$. Si $b_k \neq 0$ la boucle est exécutée et l'on obtient $a_{k+1} = b_k$ et $b_{k+1} = a_k - q_k b_k$, en notant q_k lequotient dans la division.

Alors on a par un calcul immédiat, pour tout $x \in \mathbf{N}$, l'équivalence :

((x divise a_k et b_k) $\Leftrightarrow$ (x divise a_{k+1} et b_{k+1}))

On a donc pour tout $x \in \mathbf{N}$:

(x divise a_0 et b_0) $\Leftrightarrow$ (x divise a_1 et $b1$) $\Leftrightarrow$
$\cdots \Leftrightarrow$ (x divise a_n et b_n)

La dernière valeur du couple (a',b') est (a_n,b_n) avec $a_n > 0$ et $b_n = 0$.

Cela signifie que x divise a et b si et seulement si il divise a_n. En conséquence a_n divise a et b. Par ailleurs on a de proche en proche le fait que les entiers a_k et b_k sont tous de la forme $u_k a + v_k b$ avec u_k, $v_k \in \mathbf{Z}$. Ainsi $g = a_n$, qui est la sortie donnée par l'algorithme, vérifie les conclusions du théorème 2.

Comparaison des deux démonstrations

On peut comparer les deux preuves proposées de différents points de vue. Elégance, rigueur, simplicité, facilité de compréhension, conviction de l'interlocuteur, effectivité du résultat annoncé.

Bien que ces critères, hormis le dernier, soient très subjectifs, et que les réponses dépendent beaucoup de notre éducation mathématique ils sont néanmoins très importants. Une grande partie de l'activité de recherche en mathématiques consiste à essayer de simplifier ce qui a déjà été démontré. Souvent la première preuve trouvée pour un résultat important est obscure pour la plupart. La science mathématique ne pourrait pas progresser réellement sans l'activité continue de simplification des preuves.

La première preuve est appelée ici « classique » car c'est celle que l'on trouve désormais le plus souvent exposée [1]. Elle est particulièrement rapide et donne le sentiment d'aller droit au but. On peut la voir comme le résultat d'un effort de simplification de la preuve originelle, qui, dans la mesure où elle peut être pointée historiquement, ressemble beaucoup plus à la preuve par algorithme.

L'argument aussi est dans une certaine mesure plus simple que dans la deuxième preuve. Si l'on n'avait pas déjà la démonstration d'Euclide en tête, il aurait fallu faire preuve d'un peu d'invention pour trouver l'argument décisif : $\forall\ x \in \mathbf{N}$

((x divise a_k et b_k) $\Leftrightarrow$ (x divise a_{k+1} et b_{k+1}))

Par contre la première preuve ne semble pas fournir d'algorithme pour le calcul des entiers relatifs u et v dont on affirme qu'ils existent. Certainement avant Cantor on n'aurait jamais écrit quelque chose qui ressemble à cela.

Il faut une certaine audace pour dire « je les ai trouvés » quand on ne dit pas comment concrètement on peut les avoir. Il faut aussi un certain culot pour considérer l'ensemble infini $V(a, b) \subset N$ comme objet central de la preuve et raisonner avec son plus petit élément.

A priori, si l'on cherche l'algorithme sous-jacent à la preuve abstraite on est tenté de pen-

1. En fait, dans un langage plus abstrait on trouve en général l'énoncé suivant : tout idéal de $\mathbf{Z}$ est principal, et on en déduit ensuite le théorème 2. Nous avons préféré ici donner un résumé de cette preuve classique pour mieux faire ressortir son caractère fulgurant.

ser qu'il va falloir examiner tous les éléments de $V(a, b) \cap [\![0...a]\!]$ et prendre le plus petit d'entre eux. Même si une telle recherche s'avère possible, elle semble très inefficace.

La question suivante se pose alors tout naturellement :

La démonstration classique cache-t-elle un algorithme ?

La réponse est « OUI ! » et nous allons essayer de convaincre le lecteur.

Tout d'abord on peut remarquer que l'algorithme d'Euclide peut être interprété à la lumière de la preuve abstraite de la manière suivante : en considérant les valeurs successives prises par a' on voit que l'algorithme parcourt un chemin dans l'ensemble $V(a, b)$, chemin strictement décroissant, qui s'arrête lorsqu'il ne peut plus descendre. Cependant, le test d'arrêt n'est pas directement lié à la certitude immédiate d'avoir atteint un diviseur commun à a et b. D'ailleurs, comme nous l'avons déjà remarqué la preuve de correction de l'algorithme, quoique assez simple, n'est pas absolument évidente et réclame un peu d'imagination.

Maintenant examinons la preuve classique avec cette idée de parcourir un chemin en descendant strictement dans l'ensemble $V(a, b)$ avec pour objectif de réaliser le théorème 2 : trouver un diviseur commun à a et b dans $V(a, b)$. De ce point de vue, nous voyons que *l'argument « par l'absurde » qui termine la preuve classique, peut, comme très souvent, être vu comme le renversement d'un argument direct.*

L'argument direct est : je vous dis comment descendre dans $V(a, b)$ tant que vous n'avez pas atteint votre objectif (un diviseur commun à a et b dans l'ensemble $V(a, b)$). Son renversement en une preuve par l'absurde donne : plaçons-nous au minimum de $V(a, b)$ et montrons, par l'absurde, que notre objectif est atteint.

Cette preuve indirecte qui commence par un coup de force (plaçons-nous au minimum de $V(a, b)$) et semble se terminer avec un argument par l'absurde peut être facilement remise sous forme d'une preuve directe qui donne le moyen de calculer g. On procède comme suit. On démarre un chemin descendant dans $V(a, b)$ à partir d'un point arbitraire de cet ensemble. Supposons être arrivés au point c. Si c divise a et b on est content. Sinon, si c ne divise pas a on peut le remplacer par le reste de la division de a par c, et de même, si c ne divise pas b on peut le remplacer par le reste de la division de b par c. On continue tant qu'on n'a pas atteint l'objectif.

Notez que l'on réalise ainsi de manière certaine l'objectif « trouver un diviseur commun à a et b dans l'ensemble $V(a, b)$ » mais que l'on ne se soucie pas de savoir si le nombre g obtenu est le minimum de $V(a, b)$. Cet « autre » objectif est atteint, comme dans l'algorithme d'Euclide, sans avoir été recherché en tant que tel.

Exemples

La lectrice est invitée à faire tourner à la main l'algorithme d'Euclide et un algorithme à sa convenance issu des considérations précédentes sur la preuve classique, par exemple avec les nombres 432 et 699, et à comparer leurs performances.

Sans doute en règle générale l'algorithme d'Euclide devrait être plus rapide car les

divisions se font entre deux nombres qui diminuent simultanément, et sont donc de plus en plus rapides à exécuter. Mais dans certains cas un des deux autres algorithmes peut demander beaucoup moins d'étapes.

Voici trois exemples, avec des nombres un peu plus grands, d'exécutions comparées des deux algorithmes

Test 1 Avec les nombres $a = 5492$ et $b = 2341$.

1. Euclide :

$$\begin{bmatrix} 5492 \\ 2341 \end{bmatrix} \to \begin{bmatrix} 2341 \\ 810 \end{bmatrix} \to \begin{bmatrix} 810 \\ 721 \end{bmatrix} \to$$

$$\begin{bmatrix} 721 \\ 89 \end{bmatrix} \to \begin{bmatrix} 89 \\ 9 \end{bmatrix} \to \begin{bmatrix} 9 \\ 8 \end{bmatrix} \to \begin{bmatrix} 8 \\ 1 \end{bmatrix}.$$

2. Inspiré de la démonstration classique, en donnant la priorité à la division de a par c :

$$2341 \xrightarrow{a} 810 \xrightarrow{a} 632 \xrightarrow{a} 436 \xrightarrow{a} 260$$

$$\xrightarrow{a} 32 \xrightarrow{a} 20 \xrightarrow{a} 12 \xrightarrow{a} 8 \xrightarrow{a} 4 \xrightarrow{b} 1$$

3. En donnant la priorité à la division de b par c :

$$2341 \xrightarrow{a} 810 \xrightarrow{b} 721 \xrightarrow{b} 178$$

$$\xrightarrow{b} 27 \xrightarrow{b} 19 \xrightarrow{b} 4 \xrightarrow{b} 1$$

4. En alternant les dividendes :

$$2341 \xrightarrow{a} 810 \xrightarrow{b} 721 \xrightarrow{a} 445 \xrightarrow{b} 116$$

$$\xrightarrow{a} 40 \xrightarrow{b} 21 \xrightarrow{a} 11 \xrightarrow{b} 9 \xrightarrow{a} 2 \xrightarrow{b} 1$$

Test 2 (Fibonacci) Avec les nombres $a = 10946$ et $b = 6765$.

1. Euclide (ici tous les quotients sont égaux à 1) :

$6765 \to 4181 \to 2584 \to 1597 \to 987 \to 610 \to 377 \to 233 \to 144 \to 89 \to 55 \to 34 \to 21 \to 13 \to 8 \to 5 \to 3 \to 2 \to 1$

2. Inspiré de la démonstration classique, en alternant les dividendes :

$$6765 \xrightarrow{a} 4181 \xrightarrow{b} 2584 \xrightarrow{a} 610 \xrightarrow{b} 55 \xrightarrow{a} 1$$

Test 3 Avec les nombres $a = 359226252$ et $b = 95927832$.

1. Euclide :

$95927832 \to 71442756 \to 24485076 \to 22472604 \to 2012472 \to 335412 \to 0$

les quotients successifs sont égaux à 3, 1, 2, 1, 11, 6 et en remontant les égalités correspondantes on obtient $g = 335412 = 47a - 176b$.

2. Inspiré de la démonstration classique, en alternant les dividendes :

$$95927832 \xrightarrow{a} 71442756 \xrightarrow{b} 24485076$$

$$\xrightarrow{a} 16435188 \xrightarrow{b} 13751892$$

$$\xrightarrow{a} 1677060 \xrightarrow{b} 335412.$$

Algorithme implicite dans la démonstration classique, formes précises

En suivant l'idée informelle de l'algorithme il y a beaucoup de chemins qui s'ouvrent à nous pour descendre dans $V(a, b)$: bien souvent, c ne divisera ni a ni b et l'on aura donc le choix entre les deux divisions. Par exemple on peut choisir l'algorithme 2 dans lequel on essaie à chaque fois la division de a par c et celle de b par c.

Preuve de terminaison. Nous vérifions tout d'abord que l'algorithme s'arrête bien après un nombre fini d'étapes. Cela tient à ce que, à chaque exécution de la boucle, g est remplacé par un nombre strictement plus petit dans $\mathbf{N}^*$. On sortira donc nécessairement de la boucle après un nombre fini d'étapes. Il faut également noter que les instructions

– « $r \leftarrow$ reste de la division de a par g », et

– « $r \leftarrow$ reste de la division de b par g

sont toujours exécutées avec un $g > 0$ et qu'il n'y aura donc pas d'erreur lors de l'exécution du programme.

Preuve de correction. Au moment où l'on sort de la boucle, on a $g > 0$ qui divise a et b. En outre g reste dans l'ensemble $a\mathbf{Z} + b\mathbf{Z}$ chaque fois qu'il est réaffecté (calcul immédiat). L'algorithme réalise bien le théorème 2.

Insistons sur le fait que la preuve de correction est plus simple que celle donnée pour l'algorithme d'Euclide, ce qui reflète la très grande simplicité de la preuve classique (de laquelle on a dérivé l'algorithme).

Enfin notons qu'on peut imaginer de nombreuses variantes, comme par exemple l'algorithme 3, où l'on essaie aléatoirement la division par a ou b.

Récapitulatif : comparaison des deux types d'algorithmes en tant que démonstrations des théorèmes 1 et 2

Les différences entre les deux démonstrations algorithmiques que nous avons présentées nous semblent pouvoir être résumées comme suit :

1. La preuve via l'algorithme d'Euclide démontre à la fois les théorèmes 1 et 2 mais elle récla-

Algorithme 2 *Algorithme de calcul du pgcd, implicite dans la preuve classique.*

```
Entrée : Deux entiers naturels a et b, > 0.
Sortie : Leur pgcd g.
Variables locales : r, r' : entiers ≥ 0 ;
Début
                # initialisation
    g ← inf(a, b) ;
                # boucle
    Répéter
      r ← le reste de la division de a par g ;
      Si r > 0 alors g ← r  fin si ;
      r' ← le reste de la division de b par g ;
      Si r' > 0 alors g ← r'  fin si ;
    jusqu'à ce que r = r' = 0
                # fin de boucle
Fin.
```

Algorithme 3 *Algorithme de calcul du pgcd, implicite dans la preuve classique, variante.*

```
Entrée : Deux entiers naturels a et b, > 0.
Sortie : Leur pgcd g.
Variables locales : r, r', c : entiers ≥ 0 ;
alea : choisi à chaque fois au hasard entre a et b ;
Début
                # initialisation
    g ← alea ;
                # boucle
    Répéter
      c ← alea ; r ← le reste de la division de
                    c par g ;
      Si r > 0 alors g ← r
      sinon c ← a + b – c ; r' ← le reste de la
                              division de c par g ;
        Si r > 0 alors g ← r'  fin si
      fin si ;
    jusqu'à ce que r = r' =0
                # fin de boucle
Fin.
```

me un petit effort d'invention pour convaincre que l'objectif est bien atteint (c'est-àdire qu'on a bien calculé un diviseur commun à a et b).

2. La preuve via l'algorithme qui découle de la preuve classique démontre uniquement le théorème 2 mais ne réclame aucun effort d'imagination, car l'algorithme est guidé par le but à atteindre dans le théorème 2. Par contre il faut une démonstration annexe pour en déduire le théorème 1.

Une bonne rédaction de la preuve classique, débarrassée de ses artifices

Nous proposons maintenant une rerédaction de la démonstration classique, qui évite tout aussi bien les difficultés inhérentes à l'emploi du lemme 3 que celles inhérentes aux preuves par l'absurde. Nous noterons $\mathrm{Rst}(x, y)$ le reste de la division euclidienne de x par y lorsque $x, y \in \mathbf{N}^*$.

Démonstration élégante et constructive du théorème 2.

Remarque préliminaire qui explique la stratégie utilisée. Si nous trouvons un élément g dans $\mathrm{V}(a, b) = \mathrm{V}$ qui divise a et b, alors tout autre élément de V sera multiple de g. En effet, si $a = ga'$ et $b = gb'$ alors un élément z arbitraire de V s'écrit $z = ua + vb = g(ua' + vb')$. A fortiori, g doit être le plus petit élément de V . On est donc amené à chercher un chemin dans V qui descende tant que l'on n'a pas atteint le but fixé[2] .

Démonstration proprement dite. Définissons une suite g_n par récurrence comme suit[3] .

– Intialisation. On pose $g_0 = \inf(a, b)$. On remarque que $g_0 \in \mathrm{V}$.

– Définition de g_{n+1} à partir de g_n :

1. Si g_n divise a et b, on arrête la suite.
2. Si g_n ne divise pas a, on pose $g_{n+1} = \mathrm{Rst}(a, g_n)$.
   On remarque que $g_{n+1} \in \mathrm{V}$ et $g_{n+1} < g_n$.
3. Si g_n divise a mais ne divise pas b, on pose $g_{n+1} = \mathrm{Rst}(b, g_n)$.
   On remarque que $g_{n+1} \in \mathrm{V}$ et $g_{n+1} < g_n$.

Cette suite est bien définie, et tous ses termes sont dans V . Tant qu'on n'aboutit pas au cas 1, la suite est strictement décroissante, donc on est certain d'aboutir au cas 1.

Naturellement, les preuves de terminaison et de correction de l'algorithme 2 données précédemment font exactement le même travail que cette dernière démonstration, mise là a`titre d'exercice de rédaction, ou si l'on préfère à titre de comparaison avec la démonstration qui commençait cet article, celle inspirée d'Euclide.

Annexe 1 : algorithmes pour le calcul d'une relation de Bézout

Notons qu'il est facile de compléter l'algorithme pour qu'il fournisse des entiers relatifs u et v vérifiant $ua + vb = g$.

Tout d'abord concernant l'algorithme d'Euclide si nous notons a_k et a_{k+1} les valeurs successives prises par a' et b' dans l'algorithme 1, alors on peut calculer de proche en proche des entiers u_j tels que $a_j = u_j a + v_j b$.

L'initialisation est claire avec $a_0 = a$ et $a_1 = b$. Pour calculer u_{k+1} et v_{k+1} nous avons besoin seulement de connaître $u_k, v_k, u_{k-1}, v_{k-1}$

2. Or l'argument par l'absurde de la preuve abstraite classique nous indique ce qu'il faut faire pour descendre plus bas dans V quand le but n'est pas atteint.

3. On donne la priorité à la division de a par c

et le quotient q_k dans la division de a_k par a_{k-1} : en effet si

$$\begin{aligned} a_k &= u_k a + v_k b\ , \\ a_{k-1} &= u_{k-1} a + v_{k-1} b \quad \text{et} \\ a_{k+1} &= a_{k-1} - q_k a_k \end{aligned}$$

on en déduit

$$\begin{aligned} a_{k+1} &= (u_{k-1} a + v_{k-1} b) - q_k (u_k a + v_k b) \\ &= (u_{k-1} - q_k u_k)\, a + (v_{k-1} - q_k v_k)\, b \end{aligned}$$

et donc

$$\begin{aligned} u_{k+1} &= u_{k-1} - q_k u_k \\ v_{k+1} &= v_{k-1} - q_k v_k \end{aligned}$$

On obtient donc l'algorithme suivant :

Algorithme 4 *Euclide étendu : calcul du pgcd et d'une relation de Bézout.*

Entrée : Deux entiers naturels a et b, > 0.
Sortie : Leur pgcd g ainsi que deux entiers relatifs u et v vérifiant ua + vb = g.
Variables locales : a', b', b", u, u', u", v, v', v" : entiers relatifs ; q : entier ≥ 0 ;
Début
initialisation
a'← a ; b'← b ; u← 1 ; v← 0 ; u'← 0 ; v'← 1 ;
boucle
Tant que b' ≠ 0 **faire**
(q, b") ← quotient et reste de la division de a' par b' ;
a' ← b' ; b' ← b" ;
u" ← u – qu' ; u ← u' ; u' ← u" ;
v" ← v – qv' ; v ← v' ; v' ← v" ;
fin tant que ;
fin de boucle
g ← a'
Fin.

Et nous pouvons faire le même genre de travail (en fait un peu plus simple) avec l'algorithme 2 :

Algorithme 5 *Algorithme implicite dans la preuve classique, étendu.*

Entrée : Deux entiers naturels a et b, a > b > 0.
Sortie : Leur pgcd g ainsi que deux entiers relatifs u et v vérifiant ua + vb = g.
Variables locales : r, r', q : entiers ≥ 0 ;
Début
initialisation
g ← b ; u ← 0 ; v ← 1 ;
boucle
Répéter
(q, r) ← quotient et reste de la division de a par g ;
Si r > 0 **alors** g ← r ; u ← 1 – qu ; v ←– qv **fin si** ;
(q, r') ← quotient et reste de la division de b par g ;
Si r' > 0 **alors** g ← r' ; u ←– qu ; v ← 1 – qv **fin si** ;
jusqu'à ce que r = r' =0
fin de boucle
Fin.

Annexe 2 : le calcul du pgcd via les décompositions en facteurs premiers

Si l'on prend deux nombres < 121 par exemple 93 et 63, il est facile de trouver leurs décompositions en produit de facteurs premiers : un nombre inférieur à 121 qui n'est pas divisible par 2, 3, 5 et 7 est forcément premier. On trouve donc facilement $93 = 3 \times 31$, $63 = 3^2 \times 7$ et pgcd(93, 63) = 3. Pour chaque facteur premier figurant dans les deux décompositions on prend l'exposant minimum, et cela nous donne le pgcd. En outre cela s'applique aussi bien pour le pgcd de trois entiers par exemple.

Il se cache derrière la simplicité apparente de cet algorithme quelques problèmes redoutables.

Le premier problème est d'ordre théorique : pourquoi diable un nombre entier admet-il une seule décomposition en facteurs premiers ? C'est bien sûr un fait d'expérience pour les petits entiers, mais pour les grands ?

Pour voir que la réponse n'est pas si facile, nous remplaçons l'anneau **Z** par le sous-anneau **A** de **Q**[X] formé par les polynômes P de la forme $a + X^2Q(X)$ ($a \in \mathbf{Q}$, $Q(X) \in \mathbf{Q}[X]$). On peut aussi dire : les polynômes P tels que $P'(0) = 0$. Ou encore $\mathbf{A} = \mathbf{Q}[X^2, X^3]$. Le lecteur se convaincra facilement que $P_2 := X^2$ et $P_3 := X^3$ sont irréductibles dans **A** (le seul facteur unitaire strict de P_2 dans **Q**[X] est X, mais X n'appartient pas à **A**). Maintenant on voit le phénomène étrange suivant : le polynôme $X^6 \in \mathbf{A}$ admet deux décompositions distinctes en produit de facteurs irréductibles dans **A** :

$$X^6 = {P_2}^3 = {P_3}^2 .$$

Quant aux polynômes X^5 et X^6, leurs diviseurs communs (unitaires) sont 1, P_2 et P_3 , et donc il n'y a pas de diviseur commun qui soit multiple de tous les autres diviseurs communs. Donc la propriété inattendue du pgcd de deux entiers donnée par le théorème 1 n'a rien qui lui corresponde dans **A**.

En bref la divisibilité dans des anneaux un peu plus compliqués que Z offre des surprises désagréables.

Alors d'où vient l'unicité de la décomposition en facteurs premiers pour les entiers ? Elle vient de l'algorithme du pgcd, ou plus précisément du théorème 2, duquel on tire facilement que si un nombre premier p divise un produit ab il doit diviser l'un des facteurs. Cela s'appelle le lemme d'Euclide, mais c'est Gauss qui en a apparemment le premier donné une démonstration correcte.

La voici : si p ne divise pas a, puisque p n'a que 1 comme facteur strict, le seul facteur commun à p et a est égal à 1, d'où par la relation de Bézout une égalité $up + va = 1$. Alors si $pq = ab$, cela donne $b = (up + va)b = upb + vab = upb + vpq = p(ub + vq)$: p divise b.

Le deuxième problème est d'ordre pratique : s'il est vrai que pour les petits entiers l'algorithme consistant à les décomposer en facteurs premiers pour trouver leur pgcd est plus rapide que l'algorithme d'Euclide, cela ne va plus du tout pour les grands entiers. Une machine calcule en une fraction de seconde la relation de Bézout entre deux entiers à 1000 chiffres. Elle est en général tout à fait incapable (en l'état actuel de la science) par contre de trouver pour un tel entier les facteurs premiers qui auraient 300 chiffres.