Terminaison d'un algorithme

Trois questions à se poser quand on fabrique un algorithme :
– est-ce qu'il donne un résultat ? ⇝ terminaison ;
– est-ce qu'il donne le/un bon résultat ? ⇝ validité ;
– est-ce qu'il donne le résultat en temps raisonnable ? ⇝ complexité.

1 Algorithme d'Euclide (1)

Rappeler l'algorithme d'Euclide (on suppose disposer des opérations arithmétiques et de la partie entière). Justifier qu'il se termine et donne le bon résultat.

Une résolution

On cherche le pgcd (a, b) de deux entiers a et b supposés positifs.

Sage

```
fonction pgcd(a,b entiers)
def pgcd(a,b):
    x = a
    y = b
    while y!=0:
        c = y
        q = floor(x/y)    // floor est la partie entière
        y = x-y*q
        x = c
    return x
```

Deux propriétés sont utiles :
– quels que soient les entiers a, b, q, $\text{pgcd}(a, b) = \text{pgcd}(b, a - bq)$;
– quel que soit a, $\text{pgcd}(a, 0) = a$.

La boucle a pour effet de remplacer le couple de variables (x, y) par $(y, x - yq)$. La première propriété montre que si, à l'entrée dans la boucle `while`, on a : $\text{pgcd}(x, y) = \text{pgcd}(a, b)$, alors cela reste vrai en sortie de boucle. L'égalité étant vraie à la première entrée dans la boucle, elle est vraie en sortie –pour autant qu'on en sorte.

L'assertion « le pgcd des variables x et y est la réponse souhaitée » reste vraie à chaque passage dans la boucle : c'est ce qu'on appelle un *invariant de boucle*. L'usage de telles propriétés permet de certifier la validité d'un algorithme.

Quant à la boucle, on finit par en sortir car les valeurs de la variable y, qui est remplacée par le reste d'une division par y à chaque passage dans la boucle, forment une suite strictement décroissante.

2 Fractions égyptiennes

On fixe un rationnel a/b, où a et b sont des entiers strictement positifs et $a < b$. On souhaite l'écrire comme somme d'inverses d'entiers tous distincts :

$$\frac{a}{b} = \frac{1}{n_1} + \frac{1}{n_2} + \cdots + \frac{1}{n_r}, \quad n_1 > n_2 > \cdots > n_r > 0.$$

La méthode proposée procède de la façon suivante : choisir n_1 minimal parmi les entiers n tels que $a/b > 1/n$; remplacer a/b par $a_1/b_1 = a/b - 1/n_1$; recommencer avec a_1/b_1 autant que nécessaire.

1. Traduire la méthode en pseudo-code, puis l'implémenter.
2. Justifier que l'algorithme qu'il se termine et donne une suite finie $(n_1, \dots, n_r)$ satisfaisant aux contraintes imposées.
3. On rappelle que la série harmonique, $\sum 1/n$, diverge. Étendre l'algorithme précédent à toutes les fractions a/b, sans hypothèse sur l'ordre de a et b.

Une résolution

1. On peut procéder ainsi (syntaxe SAGE) :

Sage
```
def frac(a,b,L):
    if a==0:
        return L
    else:
        x = a
        y = b
        while x != 0:
            n = ceil(y/x)
            L.append(n)
            z = y
            y = n*z
            x = n*x-z
        return L
```

ou, de façon récursive :

Sage
```
def decomp(a,b,L):
    if a==0:
        return L
    else:
        n = ceil(b/a)
        L.append(n)          // on ajoute n à la fin de la liste L
        return decomp(a*n-b,n*b,L)
```

L'image de a/b par la fonction « plafond » `ceil`, notée $\lceil a/b \rceil$, est le plus petit entier supérieur ou égal à a/b. C'est donc $-\lfloor -a/b \rfloor$, où $\lfloor \cdot \rfloor$ désigne la partie entière ; c'est l'unique entier n tel que

$$n - 1 < \frac{b}{a} \leq n, \quad \text{ou encore } \frac{1}{n} \leq \frac{a}{b} < \frac{1}{n-1}.$$

C'est bien le plus petit entier tel que $1/n$ est inférieur ou égal à a/b.

On appelle cette fonction par : `decomp(a,b,[])`. Le troisième argument est une liste qui s'allonge à chaque passage dans la boucle.

2. L'assertion suivante est un invariant de boucle :

$$\frac{a}{b} = \frac{x}{y} + \sum_{\ell \in L} \frac{1}{\ell} \quad \text{et} \quad x \geq 0.$$

En effet, au premier passage, la liste L est vide donc la somme de droite est nulle, et $a/b = x/y$. En fin de boucle, on a remplacé x par $x' = nx - y$ et y par $y' = ny$ et on a ajouté n à la liste L'. Par suite :

$$\frac{x'}{y'} = \frac{nx - y}{ny} = \frac{x}{y} - \frac{1}{n} = \frac{a}{b} - \sum_{\ell \in L} - \frac{1}{n},$$

comme on le souhaitait. Par ailleurs, par définition de la partie entière supérieure `ceil`, on a :

$$n - 1 < \frac{y}{x} \leq n - 1, \quad \text{donc : } 0 \leq nx - y < x.$$

Ceci finit de prouver qu'on a bien un invariant de boucle et entraîne que la boucle se termine.

Variante : On pose $a_0 = a$, $b_0 = b$. On définit trois suites d'entiers (éventuellement finies) par récurrence. Soit $k \in \mathbb{N}$, supposons que a_k et b_k ont été définis :
– si $a_k = 0$, le processus s'arrête ;
– si $a_k \neq 0$, on pose :

$$n_{k+1} = \left\lceil \frac{b_k}{a_k} \right\rceil, \quad a_{k+1} = a_k n_k - b_k, \quad b_{k+1} = n_k b_k.$$

En entrant dans la boucle, la variable `a` vaut a_0 et `b` vaut b_0. Si $a_0 \neq 0$, en sortie de boucle, `a` vaut a_1, `b` vaut b_1 et `L` est la liste contenant n_1. Par récurrence sur l'entier k, on démontre qu'au k^{e} passage dans la boucle, `a` vaut a_k à l'entrée et a_{k+1} en sortie, idem pour `b`, et `L`, qui est la liste contenant dans l'ordre $(n_1, \ldots, n_{k-1})$ en entrée, se voit adjoindre n_k à la sortie.
Ainsi, si on montre que $a_{k+1} = 0$ pour un certain entier k, l'algorithme s'arrête au $(k+1)^{\text{e}}$ passage et renvoie $(n_1, \ldots, n_k)$. Mais ceci est presque évident. Tant que $a_k > 0$, on a :

$$\frac{b_k}{a_k} + 1 > n_k, \quad \text{il vient : } a_k > a_k n_k - b_k = a_{k+1}.$$

Comme il n'existe pas de suite strictement décroissante dans $\mathbb{N}$, la suite a_k prend la valeur 0. Ainsi, ce qui prouve la terminaison de l'algorithme, c'est que la variable entière `x` décroît strictement à chaque passage dans la boucle.

3. Soit a/b un rationnel et soit q l'unique entier tel que

$$\sum_{i=1}^{q} \frac{1}{i} \leq \frac{a}{b} < \sum_{i=1}^{q+1} \frac{1}{i}.$$

La divergence de $\sum 1/i$ assure l'existence de q. On pose

$$\frac{a'}{b'} = \frac{a}{b} - \sum_{i=1}^{q} \frac{1}{i}, \quad \text{alors } 0 \leq \frac{a'}{b'} < \frac{1}{q+1} < 1.$$

Si on applique l'algorithme précédent à a'/b', on obtient une suite strictement croissante d'entiers $n_1 < \cdots < n_r$, donc le premier, n_1, satisfait par construction à

$$\frac{1}{n_1} \leq \frac{a'}{b'} < \frac{1}{q+1} < \frac{1}{q}, \quad \text{d'où } q < n_1.$$

Ainsi, la suite $(1, 2, \ldots, q, n_1, \ldots, n_r)$ est strictement croissante et on a :

$$\frac{a}{b} = \sum_{i=1} q\frac{1}{i} + \frac{a'}{b'} = \sum_{i=1} q\frac{1}{i} + \sum_{j=1}^{r} \frac{1}{n_j}.$$

3 Un algorithme « aléatoire » (d'après Dowek)

Hypothèse d'école : on suppose disposer d'une fonction `rnd()` capable de tirer au sort un nombre entier sans limitation de taille (peu importe la loi de probabilité).
On considère la fonction définie `f` de la façon suivante :

Pseudo-code

```
Entrée : deux entiers naturels (n, p)
Définition de la fonction f :
  si p != 0                          //    != signifie "différent de"
     alors renvoyer f(n, p−1)
     sinon si n != 0 alors renvoyer f(n−1, rnd())
                     sinon renvoyer 0
```

1. Pourquoi est-il presque évident que l'algorithme va se terminer si on suppose que la fonction `rnd()` prend ses valeurs dans un intervalle borné $[0, M]$ (où M = le plus grand entier que l'ordinateur peut représenter, par exemple) ? Pourquoi n'est-ce pas une bonne idée d'implémenter l'algorithme, même sous cette hypothèse ?
2. Prouver que cet algorithme se termine pour toute valeur d'entrée.

Une résolution

1. Si on sait que les valeurs de p vont appartenir à l'intervalle $[0, M]$, on peut borner a priori le nombre de passages dans la boucle avant que l'algorithme n'arrive à la valeur $(0, 0)$ en fonction des valeurs à l'entrée (n_0, p_0) par $p_0 + (n_0 - 1)M$.
2. On sent bien que ça va se terminer, mais pour même pour $n = 1$, il n'est pas possible de majorer le nombre d'étapes avant la terminaison : en effet, on va faire un tirage aléatoire dont le résultat n'est pas borné par une fonction a priori des valeur initiales de n et p.

   Supposons que l'algorithme ne se termine pas pour une certaine valeur (n_0, p_0). L'algorithme produit donc une suite de couples d'entiers $((n_k, p_k))_{k\in\mathbb{N}}$ telle que, pour tout entier k :
   - si $p_k \neq 0$, alors $n_{k+1} = n_k$ et $p_{k+1} = p_k$;
   - si $p_k = 0$, alors $n_k \neq 0$ et $n_{k+1} = n_k - 1$ (et on ne sait rien de p_{k+1}, qui est tiré au hasard).

Soit N la plus petite valeur atteinte par la suite (n_k), et soit $(k_1, k_2, \dots)$ la suite, finie ou infinie, des indices où cette valeur est atteinte. Soit P la plus petite valeur atteinte par la suite $(p_{k_1}, p_{k_2}, \dots)$, et soit k un de ces indices, pour lequel on a $p_k = P$ –et $n_k = N$. Supposons que $p_k \neq 0$, alors $p_{k+1} = p_k - 1$ et $n_{k+1} = n_k = N$, ce qui contredit la minimalité de P. On a donc $P = 0$. Supposons que $N \neq 0$. Alors, comme $p_k = 0$ et $n_k = N > 0$, on a : $n_{k+1} = n_k - 1$, ce qui contredit la minimalité de N. Ainsi, $N = 0$ et $P = 0$, si bien que l'algorithme se termine.

Un peu plus conceptuellement, la variable qui décroît n'est plus un entier, mais le couple (n, p), élément de l'ensemble $\mathbb{N}^2$ ordonné lexicographiquement :

$$(\texttt{lex}) \qquad (n,p) \leq (n',p') \iff [n \leq n' \text{ ou } (n = n' \text{ et } p \leq p')]\,.$$

La terminaison de l'algorithme traduit que cet ensemble totalement ordonné est *bien ordonné*, c'est-à-dire qu'il n'y existe pas de suite strictement décroissante ; de façon équivalente (pourquoi est-ce équivalent ?), toute partie non vide possède un plus petit élément.
Si on dispose de deux ensembles totalement ordonnées (deux ordinaux), leur produit cartésien peut être muni de l'ordre lexicographique, défini par (`lex`). La preuve précédente montre en fait que *le produit de deux ensembles bien ordonnés est bien ordonné.*

4 « Petites suites de Goodstein » (d'après Dowek)

Fixons un entier naturel n, par exemple 5. On va construire une suite d'entiers de la façon suivante. On écrit la suite de chiffres de n en base 2, dans l'exemple, 101_2. On interprète cette suite de chiffre comme un nombre écrit en base 3. Dans l'exemple : $101_3 = 3^2 + 3^0$, c'est dix. On retranche 1. On écrit ce nombre en base 3, on l'interprète en base 4. Dans l'exemple, neuf s'écrit 100_3, ce qui donne $100_4 = 4^2$, c'est seize. On retranche 1. On interprète en base 5. Et ainsi de suite.
L'opération de changement de base remplace une expression $\sum a_k b^k$ par $\sum a_k (b+1)^k$: elle « tire vers le haut » (bottes de sept lieues). L'opération de retrancher 1 est un pas de fourmi « vers le bas ». Qui gagne ?

1. Traduire ceci en pseudo-code et l'implémenter : on suppose savoir faire les conversions d'une base à l'autre ; on affichera les valeurs intermédiaires et le nombre d'itérations nécessaires pour les atteindre. Tester pour des petites valeurs.
2. Prouver que l'algorithme se termine, au sens où la suite stationne à zéro.

Une résolution

1. Voici une façon de coder la fonction à itérer :

Xcas

```
g(x,b):= {
    local y;
    if(x==0) return 0 ;
    y := convert(x,base,b) ;
    y := convert(y,base,b+1) ;
    // afficher(y) ;
    return(y-1) ;
}
```

Quelques essais suggèrent qu'il faut itérer jusqu'à atteindre la valeur 0. On peut faire ainsi :

Xcas

```
itere(x):= {
    local b,m ;
    b := 2 ;
    m := 0 ;
    tantque(x!=0) {
        return("valeur de x et numéro étape : ",x,b-2);
        x := g(x,b) ;
        if(m<x) {m := x ;} ;
        b := b+1 ;
    }
    afficher("max atteint ,nombre d'étapes")
    return(m,b-2)
}
```

2. Soit n un entier et $(a_r, \dots, a_0)$ la suite de ses chiffres en base 2. Le changement de base ne modifie en rien la suite des chiffres ; l'opération de retrancher 1 garde sa longueur constante ou la diminue de 1. Quitte à ajouter des zéros à gauche de sorte à garder la longueur constante égale à $r+1$, on voit ainsi que la suite des chiffres est *une variable strictement décroissante à valeurs dans* $\mathbb{N}^{r+1}$ (orienté lexicographiquement). À moins que la suite soit $(0, \dots, 0)$, cet invariant diminue strictement à chaque étape (penser à l'algorithme de la soustraction...). D'après le raisonnement du paragraphe précédent, l'ensemble $\mathbb{N}^{r+1}$ est bien ordonné (récurrence sur r). Par suite, la suite stationne, et ça ne peut être qu'à la valeur 0.