Terminaison d'un algorithme

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

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 $(n_1, \ldots, n_r)$ 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.

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.

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.