Correction des exercices du Chapitre 4 - A. Principes d’algorithmique

IV.2 Prouver la terminaison

- ### Terminaison de PGCD_naif

1ère boucle while :

Variant de boucle	Valeurs	Condition sortie
i décrémenté de 1 à chaque itération	***i = (a, a-1, … 0)***	***i ≤ 0***

Les valeurs du variant de boucle ***i*** forment une suite d’entiers strictement décroissante à partir de ***a*** et par pas de 1. La valeur ***i = 0*** sera forcément atteinte après ***a*** itérations et la boucle se termine.

2ème boucle while :

Variant de boucle	Valeurs	Condition sortie
i décrémenté de 1 à chaque itération	***i = (b, b-1, … 0)***	***i ≤ 0***

Les valeurs du variant de boucle ***i*** forment une suite d’entiers strictement décroissante à partir de ***b*** et par pas de 1. La valeur ***i = 0*** sera forcément atteinte après ***b*** itérations et la boucle se termine.
La 3ème boucle est une boucle for qui se termine forcément.
Conclusion : Toutes les boucles se terminent et l’algorithme aussi.

- ### Terminaison de PGCD_premiers

Fonction facteurs_premiers :

2ème boucle while :

Variant de boucle	Valeurs	Condition sortie
i incrémenté de 1 à chaque itération	$\boldsymbol{i = (2, 3, 4, \dots i_{max})}$ $\boldsymbol{i_{max}}$ = ***temp*** si **i** est premier $\boldsymbol{i_{max}}$ < ***temp*** sinon	***(temp mod i) = 0*** c’est-à-dire ***i*** est un diviseur de ***temp***

Les valeurs du variant de boucle ***i*** forment une suite d’entiers strictement croissante à partir de ***2*** et par pas de 1. Quel que soit la valeur de ***temp***, ***i*** finira forcément par être un diviseur ***temp***, au pire quand ***i = temp***, et la boucle se termine.

1ère boucle while :

Variant de boucle	Valeurs	Condition sortie
temp qui est remplacé par le reste de la division entière par $\boldsymbol{i_{max}}$ à chaque itération	***temp*** vaut ***n*** au départ puis est remplacé par le reste de la division entière de ***n*** par ***i*** avec 2 ≤ ***i*** ≤ ***n***	***premier = Vrai*** c’est-à-dire ***i = temp*** *ou encore **temp** est premier*

Les valeurs du variant de boucle ***temp*** forment une suite d’entiers strictement décroissante à partir de ***n*** et minorée par 2. La boucle se termine dès que temp est premier. Quel que soit la valeur de ***n***, ***temp*** finira forcément par être un nombre premier, au pire quand ***temp = 2***, et la boucle se termine.
Conclusion : Toutes les boucles se terminent et l’algorithme aussi.

Fonction PGCD_premiers

boucle while :

Variant de boucle	Valeurs	Condition sortie
i incrémenté de 1 à chaque itération à moins d'une instruction ***break***	***i = (0, 1, 2, … longueur(facteurs_de_b))***	***i ≥ longueur(facteurs_de_b)***

Les valeurs du variant de boucle ***i*** forment une suite d'entiers strictement croissante à partir de ***0*** et par pas de 1. Car à chaque itération, soit on sort de la boucle par une instruction ***break***, soit on incrémente ***i***. Au pire, i atteint ***longueur(facteurs_de_b)*** et la boucle se termine.
Conclusion : Toutes les boucles se terminent car l'algorithme contient 2 autres boucles ***for*** qui se terminent forcément donc il se termine.

○ Terminaison de PGCD_euclide

boucle while :

Variant de boucle	Valeurs	Condition sortie
reste qui est remplacé par le reste de la division entière de ***u*** par ***v*** à chaque itération	Au départ ***reste = b***, puis ***reste*** devient le reste de la division entière de ***a*** par ***b*** qui est forcément < ***b*** et ≥ ***0.***	***reste = 0***

Les valeurs du variant de boucle ***reste*** forment une suite d'entiers strictement croissante à partir de ***b*** et minorée par 0. ***reste*** atteint donc forcément 0 et la boucle se termine.
Conclusion : L'algorithme ne contient pas d'autres boucles et donc il se termine.

IV.3 Prouver la correction

○ Correction de PGCD_soustraction

L'invariant de boucle est la propriété « ***PGCD(u,v) = PGCD(a,b)*** ».

- Avant la 1ère itération de la boucle ***while***, cette propriété est vraie puisque ***u = a*** et ***v = b***.

- Supposons que cette propriété est vraie avant une itération quelconque de la boucle et notons ***u'*** et ***v'*** les valeurs de ***u*** et ***v*** après cette itération. D'après l'algorithme, on voir que soit ***u' = u - v*** et ***v' = v***, soit ***u' = u*** et ***v' = v – u*** donc ***PGCD(u',v') = PGCD(u-v,v)*** ou ***PGCD(u',v') = PGCD(u,v-u)***. Or un théorème mathématique nous dit que ***PGCD(u-v,v) = PGCD(u,v-u) = PGCD(u,v)***. On en déduit que ***PGCD(u',v') = PGCD(u,v)*** et puisqu'on a fait l'hypothèse que ***PGCD(a,b) = PGCD(u,v)***, on a aussi ***PGCD(u',v') = PGCD(a,b)***. La propriété choisie reste donc vrai après une itération.

- On en déduit que la propriété reste vraie à la fin de la boucle et que c'est bien un invariant de boucle. Or, à la sortie de la boucle ***u = v*** donc ***PGCD (u, v) = PGCD(u,u)= u***. L'invariant de boucle s'écrit alors ***PGCD(u,v) = u = PGCD(a,b)*** et puisque la fonction retourne ***u***, elle retourne bien ***PGCD(a,b)***. CQFD

IV.4 Calcul de complexité

○ Complexité de PGCD_naif

Dans la première boucle tant que, on fait ***a*** itérations puisque ***i*** varie de ***a*** à ***1*** par pas de 1. De même, on fait ***b*** itérations dans la deuxième boucle tant que. La troisième boucle est une boucle ***for*** qui fait autant d'itérations qu'il y a de diviseurs de ***b***. On ne sait pas exactement combien cela peut faire d'itérations car cela dépend de la valeur de ***b***, mais il y a forcément moins que ***b*** itérations. Au total, on a donc moins que **a + b + b = a + 2b** itérations. Par conséquent, si on multiplie ***a*** et ***b*** par ***n***, le nombre d'itérations est aussi multiplié par ***n*** (***n*a+2*n*b = n*(a+2b)***). Cela montre que la complexité de cet algorithme est linéaire (en O(n)).

○ Complexité de PGCD_soustraction

On ne sait pas combien d'itérations fait la boucle ***tant que*** car ça dépend des valeurs de ***u*** et ***v***. Mais on peut l'évaluer dans le pire des cas, c'est-à-dire si ***v = 1***. Dans ce cas, on voit que ***u*** est décrémenté de ***1*** à chaque itération jusqu'à ce que ***u = 1***. On aura donc ***u*** itérations. Si ***u*** est multiplié par ***n*** alors le nombre d'itérations est aussi multiplié par ***n*** et la complexité est donc bien linéaire.

○ Mesure de la complexité

Pour le PGCD_naif, le pire des cas est lorsque a et b sont premiers entre eux.
Pour le PGCD_soustraction, le pire des cas est lorsque ***b = 1***.

Essai	1	2	3	4	5	6	7	8
N = 10 000 exécutions	a = 385 b = 210	a = 3850 b = 2100	a = 10 205 b = 7 654	a = 385 b = 1	a = 3850 b = 1	a = 385 b = 32	a = 385 b = 211	a = 10 250 b = 7 500
naif	5.3e-01 s	5.7e+00 s	1.9e+01 s	3.7e-01 s	4.0e+00 s	3.9e-01 s	6.0e-01 s	1.8e+01 s
soustraction	5.4e-03 s	7.9e-03 s	2.2e+00 s	2.7e-01 s	3.1e+00 s	4.6e-02 s	1.0e-02 s	6.7e-03 s
Rapport	98	721	9	1,4	1,3	8	60	2700

Pour l'algorithme naif, on a bien une complexité linéaire : quand on passe de essai 1 à essai 2 ou de essai 4 à essai 5, on multiplie ***a*** et ***b*** par 10 et le temps d'exécution est aussi à peu près multiplié par 10.

Pour l'algorithme par soustraction, c'est moins net car en réalité, il est davantage sensible à la différence entre ***a*** et ***b*** qu'aux valeurs de ***a*** et ***b***. Néanmoins, dans le cas ou ***b =1*** (essais 4 et 5), c'est-à-dire dans son pire cas, le temps d'exécution est bien multiplié par 10 quand ***a*** est multiplié par 10.

Par ailleurs, on remarque que l'algorithme naïf est finalement assez peu sensible à son pire cas (essais 3 et 8 ou essais 1 et 7) alors que l'algorithme par soustraction y est effectivement très sensible (essais 1 et 4 ou 2 et 5).

Quand on compare les temps d'exécution de ces deux algorithmes (ligne rapport), on voit que l'algorithme par soustraction est toujours plus efficace que le naïf (tous les rapports sont >1) mais que cela varie énormément (de 1,3 à 2700). Dans son pire cas (***b = 1***), il est quasiment équivalent à l'algorithme naïf (essais 4 et 5). C'est quand ***a*** et ***b*** ont beaucoup de diviseurs (essais 2 et 8) que la différence semble la plus nette.

Enfin, certains résultats sont difficiles à expliquer :

- Pourquoi une telle différence pour les essais 3 et 8 avec l'algorithme soustraction ?
- Pourquoi si peu de différence entre les essais 1 et 7 pour l'algorithme naïf alors que le 7 devrait être un pire cas ?