25 mai 2022
Master 1 M.F., Orsay

Corrigé du partiel de M.A.O. Calcul Formel

Le but de ce problème est de présenter l'algorithme ϱ de Pollard pour factoriser un entier (partie 3) et pour résoudre le problème du logarithme discret (partie 4).

Partie 1 : Suites récurrentes d'ordre 1

1. On a $u_{n+T} = u_n$ pour $n = i_0$ par définition de i_0 et j_0, puisque $i_0 + T = j_0$. En outre si c'est vrai pour un entier $n \geq i_0$, alors c'est vrai pour $n+1$ puisque $u_{n+1+T} = f(u_{n+T}) = f(u_n) = u_{n+1}$.
2. Tout d'abord, si n est multiple de T et $n \geq i_0$ alors $2n = n + kT$ pour un certain entier k, d'où $x_n = u_{2n} = u_n$ d'après la question précédente. Pour la réciproque on constate que la suite $(u_{i_0+\ell})_{\ell \geq 0}$ est périodique de période T, et que T est sa plus petite période puisque les $T-1$ premiers termes de cette suite sont deux à deux distincts (par définition de j_0). En outre les termes de cette suite sont tous distincts de $u_0, \ldots, u_{i_0-1}$ (qui sont eux-mêmes deux à deux distincts). Donc si on a $u_n = u_{2n}$ alors $n \geq i_0$, et $2n - n = n$ est un multiple de T.
3. Dans tout ensemble formé par T entiers consécutifs il y a un multiple de T. C'est le cas dans l'ensemble $[\![i_0, j_0 - 1]\!]$ puisque $j_0 = i_0 + T$; il suffit donc d'appliquer la question précédente.
4. L'algorithme 1 nécessite de calculer u_j pour tout $j \in [\![1, j_0]\!]$; chacun de ces calculs, à partir de la valeur précédente u_{j-1}, revient à évaluer f une fois. Au total, le coût en temps est donc de $j_0 \leq S$ évaluations de f. En termes de mémoire, il faut stocker les valeurs précédentes pour pouvoir les comparer à la nouvelle valeur obtenue, ce qui nécessite de stocker S éléments de E (dans le cas le pire où $j_0 = S$).

   Quant à l'algorithme 2, il nécessite de calculer le couple (u_j, x_j) pour tout $j \in [\![1, j_0 - 1]\!]$ au pire (d'après la question 3). Chaque couple s'obtient à partir du précédent en évaluant f trois fois, puisque $u_j = f(u_{j-1})$ et $x_j = f(f(x_{j-1}))$. Au total on a donc besoin de $3(j_0 - 1) \leq 3S$ évaluations de f. En termes de mémoire, il suffit de stocker à chaque itération le couple (u_j, x_j) précédent : le coût en mémoire est en $O(1)$ éléments de E.

   Ces deux algorithmes ont donc le même coût en temps, à savoir $O(S)$ évaluations de f : la valeur de la constante implicite dans le symbole O n'est pas importante. En revanche ils n'ont pas du tout le même coût en espace : l'algorithme 2 est nettement préférable. *Il s'agit de l'algorithme de détection de cycle de Floyd, appelé aussi algorithme du lièvre et de la tortue car la suite (x_n) va "deux fois plus vite" que la suite (u_n). Pour l'application qui est faite à la partie 3, l'entier n pourra valoir quelques dizaines de milliards (par exemple) et l'économie de mémoire est alors déterminante.*
5. $(\star)$

Partie 2 : Suites aléatoires

6. Le nombre de $(n+1)$-uplets de E^{n+1} formés d'éléments deux à deux distincts est $S(S-1)\dots(S-n)$: on a S choix pour x_0, puis (une fois que x_0 est choisi) on a $S-1$ choix pour x_1, puis $S-2$ pour x_2, et ainsi de suite jusqu'à $S-n$ pour x_n.
7. Notons n la partie entière de $1+4\sqrt{S}$. La probabilité que $X_0, \dots, X_n$ prennent des valeurs deux à deux distinctes est $\frac{S(S-1)\dots(S-n)}{S^{n+1}} = \prod_{i=1}^{n}(1-\frac{i}{S})$, d'après la question précédente. On a donc $P(A) = 1 - \prod_{i=1}^{n}(1-\frac{i}{S})$.
8. Comme $e^x \geq 1+x$ pour tout réel x, on a

$$1-P(A) = \prod_{i=1}^{n}\Big(1-\frac{i}{S}\Big) \leq \prod_{i=1}^{n}\exp(-i/S) = \exp\Big(-\frac{1}{S}\sum_{i=1}^{n} i\Big) = \exp\Big(-\frac{n(n+1)}{2S}\Big).$$

   Comme $n+1 \geq n \geq 4\sqrt{S}$ on en déduit $1-P(A) \leq e^{-8} < 10^{-3}$ ce qui donne $P(A) > 99,9\%$.
9. $(\star)$
10. En oubliant le 29 février, on peut noter E l'ensemble des jours d'une année, et on a $S = 365$ d'où $4\sqrt{S} \simeq 77,4$. Les dates d'anniversaire des 78 personnes peuvent être considérées comme 78 variables aléatoires $X_0, \dots, X_{77}$. Si on suppose que les dates d'anniversaire de ces personnes sont uniformément réparties dans l'année, et qu'elles sont indépendantes, alors la probabilité qu'elles soient toutes distinctes est inférieure à $1/1000$ d'après la question précédente. *Ce résultat, souvent appelé paradoxe des anniversaires, est généralement cité avec 23 personnes : la probabilité que deux d'entre elles, au moins, aient la même date d'anniversaire est alors supérieure à 50%.*

Partie 3 : Une méthode de factorisation

11. Supposons que $x_n \equiv x'_n$ et $u_n \equiv u'_n$ modulo N. Alors il existe $j \in \mathbb{Z}$ tel que $x_n - u_n = x'_n - u'_n + jN$, et on a alors $\mathrm{pgcd}(x_n - u_n, N) = \mathrm{pgcd}(x'_n - u'_n, N)$. Grâce à cette remarque, il suffit de calculer (au fur et à mesure) les restes de u_n et x_n dans la division euclidienne par N, plutôt que u_n et x_n eux-mêmes. C'est possible car la formule $u_{n+1} = u_n^2 + 1$ montre que le reste de u_{n+1} ne dépend que de celui de u_n. Cela permet de ne manipuler que des entiers compris entre 0 et $N-1$. Si au contraire on manipule les entiers u_n eux-mêmes, les calculs seront beaucoup plus longs car ces entiers deviennent énormes, et on aura même des difficultés à stocker ces entiers en mémoire. En effet on a $u_{n+1} \geq u_n^2$ ce qui donne $u_n \geq 2^{2^n}$ par une récurrence immédiate. Déjà avec $n = 40$ on obtient un entier à $2^{40} > 10^{12}$ bits, et stocker cet unique entier prendrait essentiellement toute la mémoire disponible sur un ordinateur courant.
12. Par l'algorithme d'Euclide, on peut calculer δ_n à partir de $x_n - u_n$ et N en $O(\log(N))$ opérations dans $\mathbb{Z}$.
13. $(\star)$
14. $(\star)$ On trouve 641, comme Euler en 1732 (mais par une méthode différente !) : il s'agit de factoriser le cinquième nombre de Fermat.
15. Pour chaque entier n compris entre 1 et n_0, calculer δ_n à partir de $x_n - u_n$ coûte $O(\log(N))$ opérations dans $\mathbb{Z}$ d'après la question 12. Mais il faut aussi calculer u'_n et x'_n

à partir de u'_{n-1} et x'_{n-1} ; on note ici u'_n (resp. x'_n) le reste dans la division euclidienne de u_n (resp. x_n) par N. En notant f l'application $x \mapsto x^2+1$, et r celle consistant à prendre le reste dans la division par N, on a $u'_n = r(f(u'_{n-1}))$ et $x'_n = r(f(f(x'_{n-1})))$ donc il suffit d'appliquer f trois fois et r deux fois au total. Chaque application de f consiste en une multiplication et une addition. Au total, on a donc $O(1)$ opérations arithmétiques dans $\mathbb{Z}$ à chaque étape, ce qui est négligeable devant les $O(\log(N))$ opérations utilisées pour calculer δ_n. Finalement, l'algorithme utilise donc $O(n_0 \log(N))$ opérations arithmétiques dans $\mathbb{Z}$.

16. On considère l'image de u_n dans $\mathbb{Z}/p\mathbb{Z}$. Cela définit une suite d'éléments de $\mathbb{Z}/p\mathbb{Z}$ à la laquelle on peut appliquer la question 3 de la partie 1 : il existe un entier $n < p$ tel que u_n et x_n aient la même image modulo p. Comme p divise aussi N, on a $p|\delta_n$ donc $\delta_n > 1$. On en déduit que n_0 existe et, par minimalité, que $n_0 \leq n < p$.
17. Supposons que la suite des réductions modulo p des u_n *ressemble* à une suite aléatoire. Alors la question 8 de la partie 2, appliquée à cette suite, montre qu'avec une probabilité supérieure à $99,9\%$ il existe $i, j \in \mathbb{N}$ tels que $0 \leq i < j \leq 1 + 4\sqrt{p}$ et $u_i \equiv u_j \bmod p$. En reprenant les notations de la partie 1 (toujours dans $E = \mathbb{Z}/p\mathbb{Z}$), on a alors $j_0 \leq 1+4\sqrt{p}$ donc la question 3 donne $n_0 \leq 4\sqrt{p}$ par minimalité de n_0.
18. D'après les questions 15 et 16, le coût de l'algorithme est en $O(p \log(N))$ opérations dans $\mathbb{Z}$. Si on suppose que le comportement de la suite modulo p est aléatoire, le coût est beaucoup plus faible : $O(\sqrt{p} \log(N))$ opérations avec une probabilité supérieure à $99,9\%$.
19. ($\star$) L'algorithme ϱ de Pollard date de 1975. Une variante a été proposée par Brent en 1980, puis Brent et Pollard ont réussi (dans un article paru en 1981) à factoriser F_8 en deux heures sur un ordinateur de l'époque : c'est l'application la plus connue de cet algorithme.
20. ($\star$)
21. Cet algorithme est très efficace quand N possède un "petit" facteur premier. Dans le cas de RSA, il n'est donc pas particulièrement utile puisqu'on choisit N comme étant le produit de deux très grands nombres premiers : si p et q ont essentiellement la même taille, celle de $\sqrt{N}$, alors l'agorithme présenté ici coûte $O(N^{1/4} \log N)$ opérations ce qui est meilleur que l'algorithme naïf (consistant à tester tous les diviseurs possibles jusqu'à $\sqrt{N}$) mais reste très lent.

    L'algorithme échoue lorsque N divise $x_{n_0} - u_{n_0}$. C'est bien sûr le cas lorsque N est premier : on ne peut pas espérer détecter de diviseur non trivial de N dans ce cas. C'est pourquoi il est raisonnable de faire d'abord un test de primalité pour vérifier que N est composé ; il existe des tests de primalité très efficaces même si N est très grand. Supposons maintenant que N est composé. Il peut arriver que l'algorithme échoue quand même, mais cela semble très peu probable. En effet, par exemple si $N = pq$ avec p et q premiers distincts, il semble raisonnable de penser que les suites obtenues à partir de (u_n) par réduction modulo p et q seront indépendantes (en un sens volontairement imprécis), et que les entiers n_0 associés n'auront aucune raison d'être les mêmes. En cas d'échec on peut essayer à nouveau en modifiant la valeur de u_0. D'ailleurs on peut aussi modifier la fonction $f(x) = x^2 + 1$ utilisée pour définir la suite (u_n) par récurrence. En effet cette fonction n'a rien de particulier, sauf qu'elle s'évalue rapidement.

Partie 4 : Calculs dans un groupe

22. Il suffit de calculer ω^{n/p_i} pour tout i compris entre 1 et t. Ces éléments de G sont tous distincts du neutre si, et seulement si, ω est un générateur de G. En utilisant l'exponentiation rapide, le coût est en $O(t \log n)$ multiplications dans G.

23. On a $g^{b-b'} = \omega^{a'-a}$. Par hypothèse, l'entier $b - b'$ est inversible modulo N. En notant k son inverse, on obtient $g = \omega^{k(a'-a)}$ donc $i = k(a' - a)$ convient. Le coût du calcul de k est $O(\log N)$ opérations arithmétiques dans $\mathbb{Z}$ par l'algorithme d'Euclide étendu, en déterminant une relation de Bezout entre $b - b'$ et N ; le coût du calcul de i est donc aussi en $O(\log N)$ opérations.

24. Si $b = b'$ alors $\omega^{a'-a}$ est l'élément neutre de G ; comme ω est un générateur de G et $a, a' \in [\![0, N-1]\!]$, on en déduit $a = a'$ ce qui est impossible. On a donc $b \neq b'$. Notons $d = \text{pgcd}(b' - b, N)$; on a $1 \leq d \leq N - 1$. En notant k l'inverse de $(b - b')/d$ modulo N/d, la relation $g^{b-b'} = \omega^{a'-a}$ donne $g^d = \omega^{k(a'-a)}$. Par hypothèse, ω est un générateur de G donc il existe i tel que $g = \omega^i$. La relation précédente montre que $di \equiv k(a' - a) \bmod N$ ce qui donne la classe de congruence de i modulo N/d. Il y a d relèvements possibles pour trouver la classe de i modulo N, ce qui revient à déterminer i lui-même. Il suffit de calculer ω^i pour chacun de ces relèvements, et de voir à quel moment on obtient g. Le coût de ce calcul est de $O(\log N)$ opérations arithmétiques dans $\mathbb{Z}$ pour calculer k, puis $O(\log N)$ multiplications dans G pour tester chaque relèvement (par exponentiation rapide). Au total, on utilise donc $O(\log N)$ opérations arithmétiques dans $\mathbb{Z}$ et $O(d \log N)$ multiplications dans G.

25. Notons $u_0 = \omega$, et considérons la suite définie par $u_{n+1} = f(u_n)$. Par une récurrence immédiate, on voit que tous les termes de cette suite sont de la forme $u_n = \omega^{a_n} g^{b_n}$ avec $a_n, b_n \in \mathbb{N}$. La définition de f donne aussi un moyen facile (en $O(1)$ opérations dans $\mathbb{Z}$) de calculer a_n et b_n en fonction de a_{n-1} et b_{n-1}, en supposant qu'on est capable de tester efficacement l'appartenance de u_{n-1} aux ensembles A_0, A_1 et A_2. Comme les ordres de g et ω divisent N, on peut remplacer a_n et b_n par leurs restes dans la division euclidienne par N, ce qui permet de les supposer compris entre 0 et $N - 1$. L'algorithme 2 de la partie 1 fournit un entier n_0 tel que $\omega^{a_{n_0}} g^{b_{n_0}} = \omega^{a_{2n_0}} g^{b_{2n_0}}$, avec $n_0 \leq j_0 - 1 \leq 4\sqrt{N}$ d'après les questions 3 et 8, du moins avec une probabilité supérieure à $99,9\%$ si on suppose que la suite (u_n) se comporte comme une suite aléatoire. L'algorithme échoue si $(a_{n_0}, b_{n_0}) = (a_{2n_0}, b_{2n_0})$, ce qui semble très peu probable (à condition que les parties A_0, A_1 et A_2 soient bien choisies). Dans le cas contraire, la question 24 permet de conclure.

    Le coût en temps du calcul de $(a_n, b_n, a_{2n}, b_{2n}, \omega^{a_n} g^{b_n}, \omega^{a_{2n}} g^{b_{2n}})$ pour les valeurs successives de n (jusqu'à n_0) est en $O(\sqrt{N})$ multiplications dans G, et autant d'opérations arithmétiques dans $\mathbb{Z}$. En notant $d = \text{pgcd}(b_{2n_0} - b_{n_0}, N)$, le coût en temps de l'algorithme de la question 24 est en $O(\log N)$ opérations arithmétiques dans $\mathbb{Z}$ et $O(d \log N)$ multiplications dans G. Au final, si d est négligeable devant $\sqrt{N}/\log N$ (ce qu'on peut supposer, car c'est très probablement vrai), le coût en temps est en $O(\sqrt{N})$ multiplications dans G, et autant d'opérations arithmétiques dans $\mathbb{Z}$.

26. ($\star$)

27. ($\star$)