Feuille d'exercices n°11 - Algorithmes probabilistes

Notions abordées
- test d'égalité probabiliste (pour des polynômes)
- test de primalité probabiliste
- échantillonage

Exercice 1 : Échantillonage

Notations. Cet exercice manipule des tableaux, aussi lorsque A est un tableau de taille q, on pourra utiliser le raccourci syntaxique $x \in A$ pour désigner : "$\exists i \in [\![0, q-1]\!], A[i] = x$".
On se pose le problème suivant :

ÉCHANTILLONAGE :		
	Entrée :	Un tableau T de n éléments et un entier $k \in [\![0, n]\!]$
	Sortie :	Un tableau contenant k éléments de T

Levons les ambiguïtés : le tableau résultat R doit s'envoyer injectivement dans le tableau initial. Aussi, il existe une injection $\varphi : [\![0, k-1]\!] \to [\![0, n-1]\!]$ telle que $\forall i \in [\![0, k-1]\!], R[i] = T[\varphi(i)]$. On remarque que φ n'est pas supposée croissante, il ne s'agit pas d'une extractrice puisque l'ordre dans R n'importe pas.
On suppose dans la suite, pour simplifier les raisonnements, que le tableau T contient des éléments deux à deux distincts. On souhaite finalement assurer que les éléments soient choisis de manière équiprobable, *i.e.* $\forall p \in [\![0, n-1]\!], \mathbb{P}\big(T[p] \in R\big) = \frac{k}{n}$. On se propose ici d'étudier l'algorithme 1, où $\mathcal{U}$ désigne le choix uniforme.

Algorithme 1 : Algorithme probabiliste d'échantillonage

```
Entrée : Un tableau T de taille n, un entier k ⩽ n
Sortie : Un tableau de k éléments de T
Res ← T[0..k − 1] ;
I ← k ;
tant que I < n faire
    j ← U([[0, I]]) ;
    if j < k then
        Res[j] ← T[I] ;
    I ← I + 1 ;
retourner Res ;
```

Q. 1 Calculer, pour chaque indice $i \in [\![0, n[\![$, la probabilité que $T[i]$ soit retenu dans Res. On pourra distinguer deux cas selon que $i < k$ ou non. En déduire que l'algorithme proposé convient.

Exercice 2 : Vérification d'égalité polynomiale

On souhaite dans cet exercice représenter en machine les polynômes non nuls partiellement factorisés. On ne s'intéresse dans cet exercice qu'à des polynômes non nuls. Un polynôme **simple** (c'est-à-dire réduit à un facteur) peut être représenté en machine par un tableau donnant les coefficients devant chaque monôme, celui de plus haut degré en tête, et le terme constant à la fin. Un polynôme quelconque (c'est-à-dire un produit de polynômes simples) est alors représenté par une liste de tableaux. Ainsi l'ordre dans les tableaux de coefficients est significatif contrairement à l'ordre dans la liste de tableaux (puisque le produit de polynômes est commutatif).
Par exemple le polynôme $P = X^4+2X^3-13X^2-14X+24$ admet les deux factorisations suivantes.

- $(X^2-X-6)(X^2+3X-4)$
- $(X^2+X-2)(X^2+X-12)$

Il admet donc, entre autres, les deux représentations suivantes en machine.

- `[[|1; -1; -6|]; [|1; 3; -4|]]`
- `[[|1; 1; -2|]; [|1; 1; -12|]]`

Avec une telle représentation il est aisé de calculer le produit de deux polynômes : il suffit de concaténer les deux listes. En revanche cette représentation n'étant pas unique, le test d'égalité entre deux polynômes n'est pas trivial.

Q. 1 Rappeler comment, étant donnés les tableaux représentant deux polynômes simples, on calcule le tableau représentant leur produit. Donner la complexité de cet algorithme en fonction des degrés des polynômes.

Q. 2 Proposer deux méthodes en temps linéaire pour évaluer un polynôme simple.

Q. 3 En déduire la complexité de l'évaluation d'un polynôme nom simple.

Q. 4 Si P un polynôme de degré n est représenté par T une liste de K polynômes simples de degrés $(n_i)_{i\in[\![1,K]\!]}$, combien d'opérations sont nécessaires pour développer (par produits successifs de gauche à droite) ce polynôme et le resprésenter alors par un polynôme simple ?
En déduire quelle serait la complexité pire cas du test d'égalité (déterministe) qui consisterait à développer le produit des facteurs de chaque polynôme puis à tester l'égalité des coefficients. Donner le comportment asymptotique de cette complexité en fonction de la taille de l'entrée au moyen d'un Ω.

Q. 5 En suivant l'algorithme proposé en classe pour la vérification de produits matriciels, proposer un algorithme probiliste de type Monte-Carlo testant l'égalité de deux polynômes, et qui :

- prend en arguments deux polynômes de degré maximum n, et un paramètre entier k ;
- est de complexité♣ linéaire ;
- lorsque les deux polynômes sont égaux, dit qu'ils le sont ;
- lorsque les deux polynômes sont distincts, dit, à tort, qu'ils sont égaux avec probabilité $\leqslant \frac{1}{k}$.

♣. on compte 1 pour chaque opération arithmétique de base, et 1 pour la génération uniforme d'un nombre aléatoire dans un intervalle $[\![0,m]\!]$ et ce pour tout $m\in\mathbb{N}$

Exercice 3 : Test de primalité probabiliste

On rappelle le petit théorème de Fermat : $\forall n \in \mathbb{N}\setminus\{0,1\}, n$ est premier $\Rightarrow \forall a \in [\![1, n-1]\!], a^{n-1} \equiv 1[n]$
On souhaite s'inspirer de ce résultat pour mettre en place un test de primalité probabiliste. L'idée est de choisir $a \sim \mathcal{U}([\![1, n-1]\!])$, puis de tester si $a^{n-1} \not\equiv 1[n]$. Si tel est le cas n n'est pas premier, sinon on recommence.

1. Résultats mathématiques

On fixe pour les deux premières questions un entier $n \in \mathbb{N}\setminus\{0,1\}$.
On définit alors $G_n = \{x \in [\![1, n-1]\!] \mid \text{PGCD}(x, n) = 1\}$ l'ensemble des entiers de $[\![1, n-1]\!]$ premiers avec n, et on considère l'opération binaire $\cdot$ définie sur G_n par $\forall (x, y) \in G_n^2,\ x \cdot y = xy \mod n$. On pose de plus $E_n = \{a \in [\![1, n[\![\mid a^{n-1} \equiv 1[n]\}$.

Q. 1 Montrer que $(G_n, \cdot)$ est un groupe.

Q. 2 Montrer que $(E_n, \cdot)$ est un sous-groupe de $(G_n, \cdot)$.

Nombres de Carmichael. Un nombre $n \in \mathbb{N}$ est appelé **nombre de Carmichael** si :
- c'est un nombre composé♣ ;
- pour tout entier $a \in [\![2, n-1]\!]$, $\text{PGCD}(a, n) = 1 \Rightarrow a^{n-1} \equiv 1[n]$.

Autrement dit, un nombre n est de Carmichael lorsqu'il n'est pas premier et qu'il n'est pas possible de trouver de trouver un entier a premier avec n qui soit un témoin de la non primalité de n au sens du petit théorème de Fermat♡. Dans toute la suite de l'exercice on note $\overline{\mathcal{C}}$ l'ensemble des entiers naturels qui ne sont pas des nombres de Carmichael.

Q. 3 Soit $n \in \mathbb{N} \setminus \{0,1\} \cap \overline{\mathcal{C}}$. Montrer que si n n'est pas premier, alors $\text{card}(E_n) \leqslant \frac{n-1}{2}$.

2. Algorithme

Q. 4 À partir des remarques précédentes, définir un algorithme probabiliste de type Monte-Carlo, prenant en argument un entier $n \in \mathbb{N} \setminus \{0,1\}$, qui n'est pas un nombre de Carmichael et un paramètre $k \in \mathbb{N}^*$ et testant la primalité de n de sorte que :
- si n est premier alors l'algorithme retourne vrai ;
- si n est un nombre composé, alors l'algorithme retourne vrai avec probabilité $\leqslant \frac{1}{2^k}$.

3. Implémentation en C

Afin de fournir une implémentation convenable de l'algorithme de test de primalité obtenu à la question précédente, il convient de définir une fonction `long expo_mod(long a, long b, long n)` calculant intelligemment $a^b \mod n$. En effet, on ne saurait se contenter d'une implémentation effectuant b multiplications de a puis d'un passage au modulo (car cela risquerait de générer des débordements d'entiers).

Q. 5 Définir en C une fonction **itérative** `long expo_mod(long a, long b, long n)` effectuant de l'ordre de $\log_2(b)$ multiplications d'entiers. Une attention particulière sera accordée au placement des calculs de modulo lors de l'exécution. Par exemple, `expo_mod(435, 591, 5)` retourne `5`.

♣. les nombres composés sont ceux qui ne sont pas premiers
♡. On remarque qu'il n'est pas non plus possible de trouver un tel témoin parmi les non premiers avec n car ceux-ci sont non inversibles dans $\mathbb{Z}/n\mathbb{Z}$

Q. 6 En déduire une fonction `bool primalité_fermat(long n, int k)` implémentant l'algorithme proposé ci-avant.