Feuille d'exercices n°12 - 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 $k \leqslant n$
Sortie : Un tableau de k éléments de T

```
1 Res ← T[0..k − 1] ;
2 I ← k ;
3 tant que I < n faire
4 |   j ← U([[0, I]]) ;
5 |   if j < k then
6 |   |   Res[j] ← T[I] ;
7 |   I ← I + 1 ;
8 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.

Solution

Soit (T,k,n) une entrée de l'algorithme 1.
On introduit $(J_l)_{l\in[\![k,n[\![}$ une suite de variables aléatoires représentant les tirages effectués pour la variable J à la ligne 4 lors de l'appel de l'algorithme 1 sur l'entrée (T,k,n), et R la variable aléatoire décrivant le tableau renvoyé par cet appel. Ainsi les (J_l) sont des variables indépendantes de lois respectives $\mathcal{U}([\![0,l]\!])$. Soit $i\in[\![0,n[\![$.

- Si $i<k$, Res$[i]$ vaut initialement $T[i]$, et par la suite seule des valeurs de la forme $T[i']$ avec $i'>k$ sont ajoutées dans Res, donc $T[i]$ est finalement dans Res ssi la ligne 6 ne concerne jamais la case Res$[i]$, d'où :

$$\begin{aligned}
\mathbb{P}(T[i]\in R) &= \mathbb{P}(\forall l\in[\![k,n[\![, J_l\neq i) && \text{d'après les lignes 4 à 6}\\
&= \prod_{l=k}^{n-1}\mathbb{P}(J_l\neq i) && \text{car les } J_l \text{ sont 2 à 2 indépendantes}\\
&= \prod_{l=k}^{n-1}\frac{\operatorname{card}([\![0,l]\!]\setminus\{i\})}{\operatorname{card}([\![0,l]\!])} && \text{car } J_l\sim\mathcal{U}([\![0,l]\!])\\
&= \prod_{l=k}^{n-1}\frac{l+1-1}{l+1}\\
&= \frac{k}{k+1}\frac{k+1}{k+2}\cdots\frac{n-1}{n}\\
&= \frac{k}{n} && \text{car les facteurs se télescopent}
\end{aligned}$$

- Si $i>k$, $T[i]$ n'est pas initialement dans Res. Ainsi $T[i]$ figure finalement dans Res ssi il est ajouté dans Res au tour où $I=i$ et qu'il n'est pas écrasé par la suite, autrement dit ssi $X_i<k$ et si, dans les tours suivants, la ligne 6 ne concerne jamais la case Res$[X_i]$, d'où :

$$\begin{aligned}
\mathbb{P}(T[i]\in R) &= \mathbb{P}(J_i<k \text{ et } \forall l\in]\!]i,n[\![, J_l\neq J_i)\\
&= \mathbb{P}(\exists p\in[\![0,k[\![,\ J_i=p \text{ et } \forall l\in]\!]i,n[\![, J_l\neq p)\\
&= \sum_{p=0}^{k-1}\mathbb{P}(J_i=p \text{ et } \forall l\in]\!]i,n[\![, J_l\neq p) && \text{car les évènements sont 2 à 2 disjoints}\\
&= \sum_{p=0}^{k-1}\mathbb{P}(J_i=p)\times\prod_{l=i+1}^{n-1}\mathbb{P}(J_l\neq p) && \text{car les } J_l \text{ sont 2 à 2 indép. et indép de } J_i\\
&= \sum_{p=0}^{k-1}\mathbb{P}(J_i=p)\prod_{l=i+1}^{n-1}\frac{\operatorname{card}([\![0,l]\!]\setminus\{i\})}{\operatorname{card}([\![0,l]\!])} && \text{car } J_l\sim\mathcal{U}([\![0,l]\!])\\
&= \sum_{p=0}^{k-1}\mathbb{P}(J_i=p)\times\frac{i+1}{n} && \text{par produit télescopique comme ci-avant}\\
&= \frac{i+1}{n}\sum_{p=0}^{k-1}\frac{1}{\operatorname{card}([\![0,i]\!])} && \text{car } J_i\sim\mathcal{U}([\![0,i]\!]) \text{ et } p\in[\![0,i]\!]\\
&= \frac{i+1}{n}\frac{k}{i+1}\\
&= \frac{k}{n}
\end{aligned}$$

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.

Solution

On récupère les degrés n_1 et n_2, on crée un tableau résultat de la bonne taille (*i.e.* $n1+n_2$), on remplit ce tableau (un peu toutes les cases en parallèle), en faisant un double boucle sur les coeffs des deux tableaux (pour chaque (i,j) on ajoute a_i+b_j au coefficient de degré $i+j$ du produit).
Complexité en $O(n_1n_2)$ (si on remplit case après case on risque d'avoir du $O((n_1+n_2)n_1n_2)$.

Q. 2 Proposer deux méthodes en temps linéaire pour évaluer un polynôme simple. Donner leur complexité en fonction du degré.

Solution

En lisant lit les polynômes en commençant par les coefficients de plus haut degré on peut utiliser la méthode de Horner.
En lisant dans l'autre sens il faut penser à une variable qui retient x^n pour ne payer qu'une multiplication par x à chaque tour (+ une autre multiplication pour le coefficient et une addition pour accumuler le résultat). Si on l'oublie, on risque fort de proposer un code de complexité quadratique.

Q. 3 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 sucessifs de gauche à droite) ce polynôme et le resprésenter alors par un polynôme simple ?
En déduire quel 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.

Solution

Si on calcule le produit de K polynômes simples de degrés $(n_i)_{i\in[\![1,K]\!]}$, donc de tailles $(t_i)_{i\in[\![1,K]\!]}$ avec $t_i = n_i + 1$sans utiliser la commutativité ni l'associativité du produit, on aurait $(t_1 + 1) \times ((t_2) + (t_1 + t_2 \times t_3) + \dots$.
Montrons que la complexité pire cas est en $\Omega(n^2)$. Pour cela il suffit de considérer la famlle de pire cas (un pire cas pour chaque $n \in \mathbb{N}$) où le polynome P est factorisé sous la forme $P_1 \times P_2$ avec P_1 et P_2 de degrés $n_1 = \lfloor \frac{n}{2} \rfloor$ et $n_2 = \lceil \frac{n}{2} \rceil$. En effet les multiplier coûte alors $(n_1 + 1) \times (n_2 + 1) \in \Omega(n_1^2) = \Omega(n^2)$.

Q. 4 Donner un algorithme **déterministe** testant l'égalité de deux polynômes en $\mathcal{O}(n^2)$, et ce sans développer les polynômes.

Solution

On teste l'égalité sur les entiers de $[\![0, n]\!]$, c'est suffisant par un argument de dimension. En effet les deux polynomes vivent dans $\mathbb{R}_n[X]$ qui est de dimension $n + 1$, donc leur différence aussi, et donc celle-ci est nulle si elle s'annule en $n + 1$ points.

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 $\leqslant \frac{1}{k}$.

Solution

On choisit un entier au hasard dans $[\![1, kn]\!]$, et on teste si les évaluations des deux polynomes coïncident en ce point.
Si les polynomes sont égaux, l'algorithme affirmera que tel est le cas.
Si les polynômes sont distincts, l'algorithme affirmera à tort qu'ils sont égaux avec probabilité inférieure à $\frac{1}{k}$. En effet, considérons deux polynômes P_1 et P_2 de degré inférieur à n et notons C l'ensemble des points sur lesquels ils coïncident.

$$C \stackrel{\text{déf}}{=} \{x \in \mathbb{R} \mid P_1(x) = P_2(x)\}$$

Si $P_1 \neq P_2$ on sait qu'il ne peut y avoir strictement plus de n points où ils coïncident, soit $\mathrm{card}(C) \leqslant n$. Donc si $X \sim \mathcal{U}([\![1, kn]\!])$,

$$\mathbb{P}(P_1(X) = P_2(X)) = \mathbb{P}(X \in C) = \frac{\mathrm{card}(C \cap [\![1, kn]\!]}{\mathrm{card}([\![1, kn]\!])} \leqslant \frac{n}{kn} = \frac{1}{k}$$

Ainsi l'algorithme renvoie V à tort avec probabilité inférieure à $\frac{1}{k}$.

♣. 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

Solution

Au besoin on pourra rappeler les résultats suivants du cours d'arithmétique.
Lemme d'Euclide : Soit $(a, b, q, r) \in \mathbb{Z}^4$. Si $a = bq + r$ alors PGCD$(a, b) =$ PGCD(b, r).
Lemme de Bézout : Soit $(a, b) \in \mathbb{Z}^2$. Il existe $(u, v) \in \mathbb{Z}^2$ tel que $au+bv = 1$ ssi PGCD$(a, b) = 1$

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.

Solution

- Montrons que $\cdot$ est interne. Soit $(x, y) \in G_n^2$. Les facteurs premiers de x et n sont disjoints, de même pour y et n, les facteurs premiers de xy sont donc disjoints de ceux de n, finalement $\text{PGCD}(xy, n) = 1$ et donc $x \cdot y = xy \mod n$ est aussi premier avec n et $x \cdot y \neq 0$.
- $\cdot$ est associatif par associativité de la multiplication et propriété du modulo, et même commutative.
- Pour tout $x \in G_n$, on a $x \cdot 1 = 1 \cdot x = x$.
- Si $x \in G_n$ alors x est premier avec n, et il existe alors, d'après le théorème de Bézout, deux entiers u et v tels que $xu + vn = 1$ donc $xu = 1 \mod n$ en choisissant donc $y = u \mod n$ on a $x \cdot y = 1$, et $y \in G_n$ car l'égalité $xu + vn = 1$ et la réciproque du théorème de Bézout assure que y est aussi premier avec n.

On remarque que le dernier point montre que les nombres premiers avec n sont inversibles pour $.$ dans $\mathbb{Z}/n\mathbb{Z}$. En particulier un entier a non premier avec n n'est pas inversible dans $\mathbb{Z}/n\mathbb{Z}$, donc a^{n-2} n'est pas sont inverse, soit $a^{n-1} \not\equiv 1[n]$.

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

Solution

- $E_n \subseteq G_n$, en effet, si $a \in [\![1, n]\!], a^{n-1} \equiv 1[n]$ alors, donc $\text{PGCD}(a, n) | a | a^{n-1}$ et $\text{PGCD}(a, n) | n$ donc $\text{PGCD}(a, n) | 1$ donc $\text{PGCD}(a, n) = 1$.
- $1^{n-1} \equiv 1[n]$ donc $1 \in E_n$
- Si $x \in E_n$ et $y \in E_n$ on a $x^{n-1} \equiv 1[n]$ et $y^{n-1} \equiv 1[n]$ donc $x^{n-1}y^{n-1} \equiv 1[n]$ et donc $(xy)^{n-1} \equiv 1[n]$, finalement $(xy \mod n)^{n-1} \equiv 1[n]$ d'où $x \cdot y \in E_n$
- Si $x \in E_n$, on a $x^{n-1} \mod n = 1$ donc $\underbrace{x \cdot x \cdot \ldots x}_{n-1} = 1$, soit y l'inverse de x dans G_n, on a alors $1 = \underbrace{x \cdot x \cdot \ldots x}_{n-1} \underbrace{y \cdot y \cdot \ldots y}_{n-1} = 1 \cdot \underbrace{y \cdot y \cdot \ldots y}_{n-1} = \underbrace{y \cdot y \cdot \ldots y}_{n-1}$ donc $y^{n-1} \equiv 1[n]$.

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.

Solution

D'après la page Wikipédia des nombres de Carmichael :

- le plus petit nombre de Carmichael est 561 ;
- il existe une infinité de nombres de Carmichael ;
- les 10 premier sont 561, 1105, 1729, 2465, 2821, 6601, 8911, 10585, 15841, 29341 ;
- plus d'info sur `https://oeis.org/A002997`.

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

Solution

Si n est composé, puisque $n \in \overline{\mathcal{C}}$ il existe $a \in [\![2, n-1]\!], a \wedge n = 1$ et $a^{n-1} \not\equiv 1[n]$. Un tel a est un alors un élément de $G_n \setminus E_n$, ainsi d'après la question 2 E_n est sous-groupe strict de G_n. D'après le théorème de Lagrange, on en déduit que le cardinal de E_n divise strictement le cardinal G_n. Donc en particulier $|E_n| \leqslant \frac{|G_n|}{2} \leqslant \frac{n-1}{2}$

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 $\leqslant \frac{1}{2^k}$.

Solution

Algorithme 2 : Test de primalité de type Monte-Carlo

Entrée : Un entier $n \in \mathbb{N} \setminus \{0, 1, 2\}$ tel que $n \notin \mathcal{C}$, un paramètre de précision $k \in \mathbb{N}^\star$
Sortie : V si n est premier, F avec probabilité $\geqslant 1 - \frac{1}{2^k}$ sinon

```
Res ← V ;
pour tout ă ∈ [[0, k − 1]] faire
    a ← U([[2, n − 1]]) ;
    si a^{n−1} ≢ 1[n] alors
        Res ← F ;
retourner Res ;
```

♣. 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 $\mathbb{Z}/n\mathbb{Z}$

Solution

D'après le petit théorème de Fermat si n est premier alors Res restera à V à chaque tour, et donc l'algorithme renvoie bien V comme attendu.
Si n est composé et si $X \sim \mathcal{U}([\![2, n-1]\!])$, alors

$$\begin{aligned}
\mathbb{P}(X^{n-1} \equiv 1[n]) &= \mathbb{P}(X \in E_n \setminus \{1\}) && \text{par définition de } E_n \\
&= \frac{\text{card}(E_n \setminus \{1\})}{\text{card}([\![2, n-1]\!])} && \text{car } X \sim \mathcal{U}([\![2, n-1]\!]) \\
&\leqslant \frac{\text{card}(E_n)}{n-2} \\
&\leqslant \frac{n-1}{2} \times \frac{1}{n-2} && \text{d'après la question 3} \\
&\leqslant \frac{1}{2} && \text{car } \frac{n-1}{n-2} \leqslant 1
\end{aligned}$$

Donc si n est composé, et si on note X_k les variables aléatoires représentant les tirages fait pour a au cours de l'appel de l'algorithme sur n, on a

$$\begin{aligned}
\mathbb{P}(\text{l'algo renvoie F sur } n) &= \mathbb{P}(\forall i \in [\![0, k[\![,\ (X_i)^{n-1} \equiv 1[n]) \\
&= \prod_{i=0}^{k-1} \underbrace{\mathbb{P}(X_i)^{n-1} \equiv 1[n])}_{\leqslant \frac{1}{2}} && \text{car les } X_i \text{ sont 2 à 2 indépendantes} \\
&= (\frac{1}{2})^k
\end{aligned}$$

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`.

Solution

```
18 long expo_mod(long a, long b, long n) {
19   long a_it_squared = a % n;
20   long res = 1;
21   while (b != 0) {
22     if (b % 2 == 1) {
23       res = (res * a_it_squared) % n;
24     }
25     a_it_squared = (a_it_squared * a_it_squared) % n;
26     b = b / 2;
27   }
```

```
  return res;
}
```

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

Solution

```
bool primalite_fermat(long n, int k) {
  if (n <= 1) {return false;}
  else if (n == 2 || n == 3) {return true;}
  else {
    for (int i = 0 ; i < k ; i ++) {
      int a = 2 + (rand() % (n-2));    /* a ∈ ⟦2, n − 1⟧*/
      if (expo_mod(a, n-1, n) != 1) { /* a^{n−1} ≢ 1[n] */
        return false;                 /* n est composé */
      }
    }
    return true;                      /* n est probablement premier */
  }
}
```