I Algorithme probabiliste

Définition : Algorithme probabiliste

Un algorithme probabiliste est défini comme un algorithme ayant accès à une opération élémentaire `random()` renvoyant un bit uniformément au hasard (0 ou 1 avec probabilité $\frac{1}{2}$).
Plusieurs appels à `random()` donnent des bits mutuellement indépendants.

Exercice 1.
Comment générer un entier dans $[\![0, n-1]\!]$ uniformément au hasard en utilisant `random` ? Avec quelle complexité ?

En pratique, on utilise le plus souvent un générateur pseudo-aléatoire qui renvoie des termes d'une suite u_n où u_0 (graine) est donné par l'utilisateur et $u_{n+1} = f(u_n)$ avec f une fonction bien choisie.
En C :

- `void srand(int)` permet d'initialiser u_0 (souvent avec le nombre de secondes écoulées depuis 1970)
- `int rand(void)` renvoie le prochain terme u_n, où $u_{n+1} = (au_n + b) \mod N$

	OCaml	C
Initialiser avec la graine `s`	`Random.init s`	`srand(s)`
$\mathcal{U}([\![0, n-1]\!])$	`Random.int n`	`rand() % n`
Booléen aléatoire	`Random.bool ()`	`(rand() & 1) == 0`
$\mathcal{U}([0, 1])$	`Random.float 1.`	`((double)rand() / RAND_MAX`

Exercice 2.
En utilisant une fonction `float r()` renvoyant un réel uniformément au hasard dans $[0, 1]$:

1. Écrire une fonction `int bernoulli(float p)` simulant une loi de Bernoulli de paramètre `p`.
2. Écrire une fonction `int geometrique(float p)` simulant une loi géométrique de paramètre `p`.
3. Écrire une fonction `int binomiale(int n, float p)` simulant une loi binomiale de paramètres `n` et `p`.

II Algorithme de Las Vegas

Définition : Algorithme de Las Vegas

Un algorithme probabiliste est de type Las Vegas s'il renvoie toujours un résultat correct mais avec un temps d'exécution variable (pour une même entrée).

Définition : Espérance du temps de calcul

Soit A un algorithme probabiliste.
Sur une entrée x, le nombre $T(x)$ d'opérations élémentaires effectuées par A est une variable aléatoire, à valeurs dans $\mathbb{N} \cup \{+\infty\}$. On dit que l'espérance du temps de calcul de A est en $\mathrm{O}(f(n))$ s'il existe une constante λ telle que $\mathbb{E}(T(x)) \leq \lambda f(|x|)$ pour toute entrée x.

On moyenne donc la complexité sur les différentes exécutions possibles de A pour une entrée x.
Ne pas confondre avec la complexité en moyenne pour un algorithme déterministe qui moyenne la complexité sur toutes les entrées possibles (de taille n).

Exemple : Le tri rapide avec choix aléatoire du pivot est un algorithme de Las Vegas.
Sur un même tableau de taille n, sa complexité peut varier entre $n\log(n)$ et n^2. On peut montrer que son espérance du temps de calcul est en $\mathrm{O}(n\log(n))$.
Sa complexité en moyenne n'est pas définie car ce n'est pas un algorithme déterministe.

Exercice 3.
Soit `void shuffle(int*, int)` une fonction mélangeant un tableau (appliquant une permutation uniformément au hasard sur ses éléments) en complexité linéaire.

1. Écrire une fonction `bool sorted(int*, int)` déterminant si un tableau est trié par ordre croissant.
2. Écrire une fonction `void bozo_sort(int*, int)` triant un tableau à l'aide d'un algorithme de Las Vegas.
3. Soit $T(n)$ le nombre d'itérations de `bozo_sort` pour un tableau de n entiers distincts. Calculer $\mathbb{E}[T(n)]$ et en déduire la complexité moyenne de `bozo_sort`.

III Algorithme de Monte-Carlo

Définition : Algorithme de Monte-Carlo

Un algorithme probabiliste est de type Monte-Carlo s'il peut renvoyer un résultat incorrect mais avec toujours le même temps d'exécution pour une même entrée.

Ainsi, l'aléatoire est sur le temps d'exécution pour un algorithme de Las Vegas et sur la valeur de retour pour un algorithme de Monte-Carlo.

Exemple : Le petit théorème de Fermat affirme que p est premier si et seulement $\forall a \in [\![2, n-1]\!], a^{p-1} \equiv 1 \mod p$.
Si p est premier alors l'algorithme de Monte-Carlo suivant renverra toujours Vrai. Si p n'est pas premier, il peut renvoyer Vrai ou Faux.

Test de primalité de Fermat

Entrée : $p \in \mathbb{N}^*$.

Choisir $a \in [\![2, p-1]\!]$ aléatoirement.
Renvoyer $a^{p-1} \equiv 1 \mod p$

Définition : Faux positif, faux négatif

Soit A un algorithme de Monte Carlo pour un problème de décision.
Un faux positif (resp. faux négatif) est une exécution de A qui renvoie Vrai (resp. Faux) pour une instance négative (resp. positive).

Exemple : Le test de primalité de Fermat n'a pas de faux négatif. Un nombre de Carmichael (entier n non premier tel que $a^n \equiv a \mod n$ pour tout a premier avec n) peut donner un faux positif.

Théorème

Supposons que l'on dispose d'un algorithme Monte Carlo pour un problème de décision Π dont la probabilité de faux négatif est nulle et la probabilité de faux positif est majorée par p.
Alors pour tout $k \in \mathbb{N}^*$, on peut obtenir un algorithme de type Monte Carlo résolvant Π sans faux négatif et avec une probabilité de faux positif majorée par p^k.

Preuve :

Exercice 4.
Soit φ une formule de k-SAT avec n variables. On considère l'algorithme suivant :

Algorithme 1

Répéter p fois **:**
    $v \longleftarrow$ valuation uniformément au hasard sur les variables de φ
    Si v satisfait φ **:**
        Renvoyer Vrai
Renvoyer Faux

Supposons φ satisfiable.

1. Soit v une valuation choisie uniformément au hasard. Minorer $\mathbb{P}(v \text{ satisfait } \varphi)$.

2. Avec quelle valeur de p l'algorithme 1 renvoie Vrai avec probabilité au moins $\dfrac{1}{2}$?

On considère un autre algorithme :

Algorithme 2

$v \longleftarrow$ valuation uniformément au hasard sur les variables de φ
Répéter p fois **:**
  Si v satisfait φ **:**
    Renvoyer Vrai
  $C \longleftarrow$ clause non satisfaite uniformément au hasard de φ
  $x \longleftarrow$ variable uniformément au hasard de C
  $v(x) \longleftarrow \neg v(x)$
Renvoyer Faux

On suppose φ satisfiable et on note v^* une valuation qui satisfait φ.
Si v est une valuation, on note $d(v, v^*) = |\{x \mid v(x) \neq v^*(x)\}|$ le nombre de variables pour lesquelles v et v^* diffèrent.

3. Soient v une valuation obtenue à une étape de l'algorithme 2 et v' la valuation obtenue à l'étape suivante. On suppose que v ne satisfait pas φ.
   Montrer que $\mathbb{P}\big(d(v', v^*) = d(v, v^*) - 1\big) \geqslant \dfrac{1}{k}$.

On suppose $k = 2$ et on admet que, si φ est satisfiable et $p = \infty$, il faut en moyenne $\mathrm{O}(n^2)$ itérations pour que l'algorithme 2 renvoie Vrai.

4. En déduire un algorithme de Monte-Carlo en complexité polynomiale et avec une probabilité d'erreur $\mathrm{O}\big(\frac{1}{n}\big)$ pour déterminer si φ est satisfiable.