

# Algorithmes probabilistes

Quentin Fortier

December 15, 2025

# Algorithme probabiliste

## Définition

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

Comment générer un entier dans  $\llbracket 1, n \rrbracket$  uniformément au hasard en utilisant `random` ? Avec quelle complexité ?

# Algorithme probabiliste

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  $u_{n+1} = (au_n + b) \bmod N$

	OCaml
Initialiser avec la graine s	<code>Random.init s</code>
$\mathcal{U}([0, n - 1])$	<code>Random.int n</code>
Booléen aléatoire	<code>Random.bool ()</code>
$\mathcal{U}([0, 1])$	<code>Random.float 1.</code>

	C
Initialiser avec la graine s	<code>srand(s)</code>
$\mathcal{U}([0, n - 1])$	<code>rand() % n</code>
Booléen aléatoire	<code>(rand() &amp; 1) == 0</code>
$\mathcal{U}([0, 1])$	<code>((double)rand() / RAND_MAX)</code>

# Algorithme probabiliste

## Exercice

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  $p$ .

# Algorithme de Las Vegas

## Définition

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).

# Algorithme de Las Vegas

## Définition

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  $O(f(n))$  s'il existe une constante  $\lambda$  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 fixée  $x$ .

Ne pas confondre avec la complexité en moyenne pour un algorithme déterministe qui moyenne la complexité sur toutes les entrées possibles (de taille  $n$ ).

# Algorithme de Las Vegas

bogo\_sort

**Entrée :** un tableau  $T$  de  $n$  entiers distincts.

**Tant que** non\_est\_trie( $T$ ) :

- └ melanger( $T$ )

**Renvoyer**  $T$

## Exercice

Soit  $T(n)$  le nombre d'itérations de bogo\_sort pour un tableau de  $n$  entiers distincts. Calculer  $\mathbb{E}[T(n)]$ .

# Algorithme de Las Vegas

tri\_rapide

**Entrée :** un tableau T de  $n$  entiers distincts.

**Si**  $n \leq 1$  :

  └ **Renvoyer** T

Choisir un pivot aléatoirement  $p$  dans T.

$T1 \leftarrow$  tableau des éléments de T inférieurs à  $p$ .

$T2 \leftarrow$  tableau des éléments de T supérieurs à  $p$ .

**Renvoyer** concaténer(tri\_rapide( $T1$ ),  $p$ ,  
tri\_rapide( $T2$ ))

# Algorithme de Las Vegas

## Exercice

Soit  $x_1 < x_2 < \dots < x_n$  les éléments de  $T$  triés. On définit  $X_{i,j}$  comme le nombre de fois où  $x_i$  et  $x_j$  sont comparés durant l'exécution de `tri_rapide(T)`.

- ① Montrer que  $X_{i,j} \in \{0, 1\}$ .
- ② Exprimer  $\mathbb{E}[X]$  en fonction des  $X_{i,j}$ , où  $X$  est le nombre total de comparaisons effectuées par `tri_rapide(T)`.
- ③ Montrer que  $\mathbb{E}[X_{i,j}] = \frac{2}{j - i + 1}$ .
- ④ En déduire que l'espérance du temps de calcul de `tri_rapide` est en  $O(n \log n)$ .

# Algorithme de Monte-Carlo

## Définition

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.

# Algorithme de Monte-Carlo

Exemple : Le petit théorème de Fermat affirme que  $p$  est premier si et seulement  $\forall a \in \llbracket 2, n - 1 \rrbracket, a^{p-1} \equiv 1 \pmod{p}$ .

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  $a \in \llbracket 2, p - 1 \rrbracket$  aléatoirement.

**Renvoyer**  $a^{p-1} \equiv 1 \pmod{p}$

# Algorithme de Monte-Carlo

Soit  $A$  un algorithme de Monte-Carlo pour un problème de décision.

## Définition

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  $a^n \equiv a \pmod{n}$  pour tout  $a$  premier avec  $n$ ) peut donner un faux positif.

# Algorithme de Monte-Carlo

## Théorème

Supposons que l'on dispose d'un algorithme Monte-Carlo pour un problème de décision  $\Pi$  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  $\Pi$  sans faux négatif et avec une probabilité de faux positif majorée par  $p^k$ .

# Algorithme de Monte-Carlo

## Exercice

Soit  $\varphi$  une formule de  $k$ -SAT avec  $n$  variables.

### Algorithme 1

**Répéter**  $p$  fois :

$v \leftarrow$  valuation au hasard sur les variables de  $\varphi$

**Si**  $v$  satisfait  $\varphi$  :

**Renvoyer** Vrai

**Renvoyer** Faux

Supposons  $\varphi$  satisfiable.

- ① Soit  $v$  une valuation choisie uniformément au hasard.  
Minorer  $\mathbb{P}(v$  satisfait  $\varphi)$ .
- ② Avec quelle valeur de  $p$  l'algorithme 1 renvoie Vrai avec probabilité au moins  $\frac{1}{2}$  ?

## Algorithme 2

$v \leftarrow$  valuation au hasard sur les variables de  $\varphi$

**Répéter**  $p$  fois :

**Si**  $v$  satisfait  $\varphi$  :

**Renvoyer** Vrai

$C \leftarrow$  clause non satisfaite au hasard de  $\varphi$

$x \leftarrow$  variable au hasard de  $C$

$v(x) \leftarrow \neg v(x)$

**Renvoyer** Faux

On suppose  $\varphi$  satisfiable et on note  $v^*$  une valuation qui satisfait  $\varphi$ . 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  $\varphi$ .

Montrer que  $\mathbb{P}(d(v', v^*) = d(v, v^*) - 1) \geq \frac{1}{k}$ .

## Algorithme 2

$v \leftarrow$  valuation au hasard sur les variables de  $\varphi$

**Répéter**  $p$  fois :

**Si**  $v$  satisfait  $\varphi$  :

**Renvoyer** Vrai

$C \leftarrow$  clause non satisfaite au hasard de  $\varphi$

$x \leftarrow$  variable au hasard de  $C$

$v(x) \leftarrow \neg v(x)$

**Renvoyer** Faux

On suppose  $k = 2$  et on admet que, si  $\varphi$  est satisfiable et  $p = \infty$ , il faut en moyenne  $O(n^2)$  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  $O\left(\frac{1}{n}\right)$  pour déterminer si  $\varphi$  est satisfiable.