Analyse d'Algorithmes et Génération Aléatoire (AAGA)

Organisation

Équipe pédagogique : Pour les semaines 1 à 7 :

- Antoine Genitrini
- Mehdi Naima
- Amaury Curiel

Pour les semaines 8 à 14 filières ING et R&D :

- Alix Munier Kordon
- Claire Hanen

Pour les semaines 8 à 14 filières ALT :

- Bình Minh Bùi Xuân

Emploi du temps des 7 premières semaines d'enseignement :

- Vendredi de 13h45 à 15h45 : cours en salle 54.55.205
- Vendredi de 16h à 18h : TD/TME en salle 14.15.507 et 24.25.303

Attention, l'emploi du temps change à partir de la semaine 8

(échange avec UE TAS, donc ça passe le jeudi matin)

Évaluation (ING + R&D) *à vérifier*

- Examen réparti 1 : 40% de la note finale.
- Devoir de programmation : 10% de la note finale.
- Examen réparti 2 : 50% de la note finale.

Évaluation (ALT)

- Examen réparti 1 : 40% de la note finale.
- La deuxième partie sera décidée par l'enseignant responsable Bình Minh Bùi Xuân.

Contexte des enseignements

- Génération aléatoire d'entiers
- Génération aléatoire de structures combinatoires de base
- Génération aléatoire de structures de graphes
- Introduction aux classes de complexité

  Examen réparti 1
- Analyse d'algorithmes

  Examen réparti 2

Contents

Bibliographie :

- D. Knuth : The art of computer programming
- S. S. Skiena : The algorithm desing manual
- D. L. Kreher and D. R. Stinson : Combinatorial algorithms
- W. H. Press et al. : Numerical recipes: the art of scientific computing

Pour aller plus loin : thèse de Florette Martinez

1- Génération d'entiers : Introduction

"*Le vrai hasard étant hasardeux, contentons-nous* d'un *pseudo-hasard et adaptons-le à nos besoins*" (Jean-Paul Delahaye dans Pour la Science 1998).

Depuis des années, on tente d'enlever tout hasard au sein d'un système informatique :

- Aujourd'hui un calcul composé de milliards d'opérations peut être lancé plusieurs fois et donnera à chaque fois le même résultat.
- Algorithme de correction d'erreurs (Transmission de données et stockage de données).

Pourtant maintenant que l'on gère l'élimination du hasard, on a besoin de le faire réapparaître. Trois grands domaines :

1. La simulation (par exemple pour modéliser un feu de forêt, l'évolution de l'infection d'une maladie)
2. L'algorithmique (Algorithmes probabilistes : par exemple l'Algorithme de Kager pour MIN-CUT. (*Calculer la probabilité d'erreur ?*)
3. La cryptographie (par exemple : Masque jetable et Chiffrement RSA)

Pour ces trois types d'application, ce n'est pas le même hasard qui est nécessité ! Dans la plupart des langages de programmation, il y a un générateur de hasard appelé "pseudo random generator".

Attention : Python depuis la version 2.3 utilise Mersenne Twister comme générateur d'aléa dans se biblothèque standard. Voilà une alerte présentée dans la documentation :

> "The pseudo-random generators of this module should not be used for security purposes."

Vous, en tant qu'experts en informatique, vous n'avez pas besoin de connaître toutes les raisons qui peuvent poser problème. Mais vous devez réagir correctement face à ce type de message.

LIVE DEMO !!!! : *(Exemple valable au moins depuis Java 6)*

- Avec deux entiers consécutifs générés, je peux deviner tous les suivants en moins de 10 lignes de code python.
- Pire : avec deux entiers consécutifs, je retrouve tous les précédents !

Trois degrés de hasard

- **Hasard faible :** Satisfaction des tests statistiques. Peut être produit par des algorithmes rapides. Utile en simulation.
- **Hasard moyen :** Imprévisibilité pour un observateur ne disposant que de moyens de calcul réalistes. Peut être produit (vraisemblablement) par algorithmes moyennement rapides. Utile en cryptographie.
- **Hasard fort :** Imprévisibilité totale, incompressibilité, contenu maximum en information. Ne peut jamais être produit par algorithme, mais vraisemblablement par des moyens physiques. Utile en cryptographie et en théorie de l'information.

Deux types de générateurs d'entiers :

1. True Random Number Generators
2. Pseudo Random Number Generators

True Random Number Generators

- Usage en cryptographie
- Obligation : non prévisible !
- Basé la plupart du temps sur des processus physiques :
  - Un opérateur fait des piles ou face avec une pièce...
  - Calcul de la décroissance de la radioactivité
  - Utilisation de la température du processeur
  - Utilisation de l'horloge
  - Carte Intel accès possible à quelques bits aléatoires (*Digital Random Number Generator*) : 1 entier généré tous les ≈ 500 cycles d'horloge

Il peut y avoir un biais (face 55%, pile 44.9% et tranche 0.1%) : ce n'est pas grave, on peut faire un post-traitement statistique si nécessaire.

Problèmes :

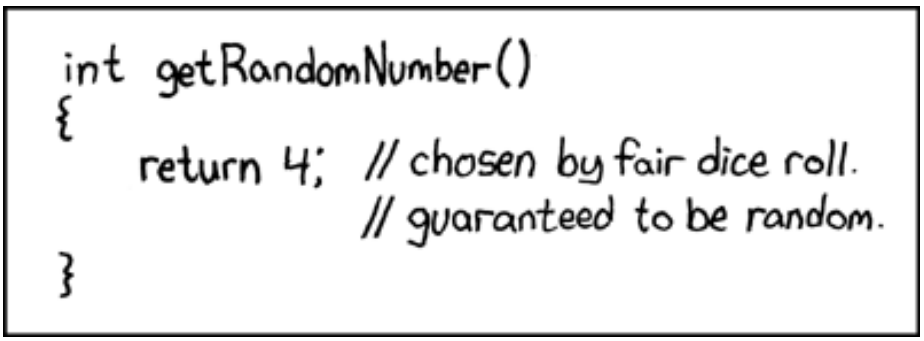
- coût élevé
- vitesse lente de génération
- debug de programmes

- random.org

  Générateur de nombres aléatoires matériel :

- lavarand
- comsire

- ORION



Pseudo Random Number Generators

- Ils sont déterministes : pas réellement d'aléatoire !
- Très rapide
- Portable
- On peut reproduire des séquences d'entiers aléatoires (grâce à une graine)

Problème : trouver un bon Pseudo Random Number Generator !

Les générateurs de nombres pseudo-aléatoires sont, pour la grande majorité, construits avec une idée d'itération :

- Soit E l'ensemble des entiers dans lequel on souhaite générer des nombres.
- On choisit $f : E \to E$.
- On définit "aléatoirement" $X_0 \in E$, nommé la graine.
- Puis on itère : $X_{n+1} = f(X_n)$.

Propriété :

- $\exists \lambda \in \mathbb{N}^*$ appelé période du générateur pour X_0 (avec $\lambda \leq |E|$).
- $\exists n_0 \leq \lambda \mid \forall n \geq n_0, X_{n+\lambda} = X_n$.

Remarques :

- Une fois que n_0 est atteint, on boucle indéfiniment dans la même séquence d'entiers (consécutifs) avec une période λ.
- La période est définie par les entiers i, j les plus proches possible telle que $X_i = X_j$.
- La longueur de la période est cruciale pour qu'un générateur soit de qualité.

Preuve de l'existence de λ ?

Exemple :

$E = [\![1, 8]\!]$, on définit $f(x) = (2x \mod 7) + 1$ et $X_0 = 8$. **Que valent λ et n_0 ?**

Paradoxe des anniversaires :

On veut construire aléatoirement un générateur aléatoire d'entier dans $[\![1, n]\!]$. On choisit une suite d'entiers dans $[\![1, n]\!]$ de façon indépendante et uniforme. **Quelle est la probabilité qu'une suite de j entiers contienne au moins deux entiers identiques ?**

Conclusion ?

2- Exemples de générateurs de nombres pseudo-aléatoires

Il existe une infinité de PRNG !

Question : Proposer quelques PRNG ?

Générateurs congruentiels linéaires

Schéma introduit par Lehmer en 1949. Jusque dans les années 90 c'était la méthode la plus utilisée. Même les générateurs actuels ne sont pas complètement différents et reposent sur des concepts similaires.

Principe :

On a besoin de 4 entiers :

- m : le modulo ($m > 0$)
- a : le multiplicateur ($0 \leq a < m$)
- c : l'incrément ($0 \leq c < m$)
- X_0 : la valeur initiale, appelée graine ($0 \leq X_0 < m$)

On construit une suite de nombre dans $[\![0, m-1]\!]$ de la manière suivante :

$X_{n+1} = (aX_n + c) \mod m.$

$(X_n)_{n \in \mathbb{N}}$ est appelée suite congruentielle (ou modulaire) linéaire.

Exemple :

$X_{n+1} = (2X_n + 3) \mod 10$ en fixant $X_0 = 0$.

Exercice :

Exprimer X_{n+k} en fonction de X_n, a, c, m et k. On supposera $a \geq 2$.

Propriété :

Si (X_n) est une suite congruentielle linéaire, alors les suites extraites $(X_{n_0+i_0 n})$ avec n_0 et i_0 fixés sont des suites congruentielles linéaires.

Preuve ?

Donner une formule directe pour calculer X_n (en fonction de X_0, a, c, m, et n) ?

- Choix du module m ?
  - Pour générer des bits, ne pas prendre $m = 2$, sinon 010101 dans le meilleur cas
  - Prendre $m =$ grand nombre premier
- Choix du multiplicateur a et de l'incrément c ?
  - On veut la période la plus longue possible, mais pas seulement ($a = c = 1$) !

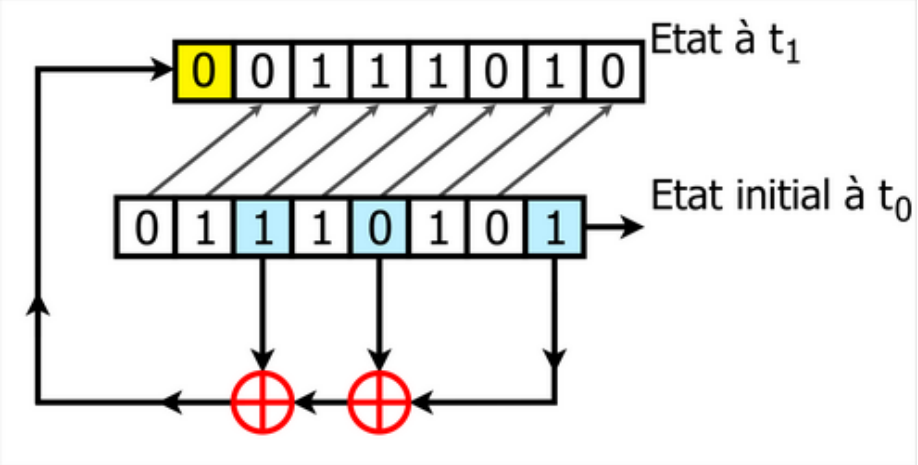
Générateur Blum Blum Shub

Soit n un produit de deux entiers premiers, chacun de la forme $4m + 3$. On prend comme graine un entier x sans facteur commun avec n. On définit ensuite $x_{i+1} = x_i^2 \mod n$. La parité de x_i donne la suite pseudo-aléatoire de bits proposée par BBS.

Si la graine est choisie aléatoirement, et si "trouver les racines carrées modulo n est un problème difficile", alors aucun algorithme rapide ne fait mieux que le hasard pour prédire le m-ième digit à partir des précédents.

Générateur lent, mais très sûr (d'un point de vue cryptographie). Peu utilisé en simulations.

Registre à décalage



Le Mersenne Twister

- C'est un générateur de nombre pseudo-aléatoire particulièrement réputé pour sa qualité.
- Développé par M. Matsumoto et T. Nishimura en 1997.
- Basé sur un type particulier de registre à décalage à rétroaction et tient son nom d'un nombre premier de Mersenne.
- C'est le générateur par défaut de Python, PHP, Maple, Matlab, R, GNU Multiple Precision Arithmetic Library...

Avantages :

- Sa période : $2^{19937} - 1$ (c'est un nombre premier de Mersenne)
- Passe la grande majorité des tests statistiques (notamment Die Hard tests)

Inconvénients :

Espace d'état trop grand : une période de l'ordre de 2^{512} devrait être suffisante (aussi bon et plus rapide) pour n'importe quelle application.

Remarques :

- La période est aussi grande car l'algorithme est basé sur un ensemble de 624 entiers (32 bits) indépendants.

- Il faut pouvoir fournir une graine (relativement aléatoire) de 624 entiers : en général, on fait appel à un autre générateur aléatoire pour construire la graine.
- Il existe une version simplifiée datant de 2006 avec une période de $2^{607} - 1$.

3- Suite aléatoire : qu'est-ce que c'est ?

Paradoxalement, la théorie des probabilité n'est pas la mieux adaptée pour définir ce qu'est une suite aléatoire. Considérons les deux suites, de 20 bits, suivantes :

- 01010101010101010101
- 11010001101001011001

Le bon sens dirait que la deuxième est plus aléatoire que la première et pourtant on peut prouver que chacune a la même probabilité d'apparaître $1/2^{20}$ (si on prend une pièce parfaite et qu'on fait pile ou face).

Complexité de Kolmogorov

En informatique, la notion de suite aléatoire a convergé vers la notion de suite complexe, ou incompressible.

Définition : *Complexité de Kolmogorov :*

> Soit $a = a_1, a_2, ..., a_n$ une suite binaire. La complexité de Kolmogorov de la suite a est la longueur, en nombre de bits, du plus petit programme permettant à un ordinateur donné de générer a.

Pour nous, un ordinateur correspond à une machine de Turing universelle. La complexité de Kolmogorov est définie à une constante additive près, car elle dépend de l'ordinateur choisi.

Définition : *Indépendance de l'ordinateur :*

> Une suite infinie $a = a_1, a_2, ..., a_n, ...$ est dite aléatoire si $\lim_{n\to\infty} \frac{K(a_1,a_2,...,a_n)}{n} = 1$.

Propriété : *in-calculabilité :*

> La complexité de Kolmogorov n'est pas calculable. En d'autre termes, il n'existe pas de programme informatique qui prenne en entrée a et renvoie $K(a)$.

Propriété :

La plupart des suites de longueur n ont une complexité de Kolmogorov au moins d'ordre n.

Propriété :

Une suite binaire aléatoire $a = a_1, a_2, ..., a_n$ vérifie la loi des grands nombres : $\lim_{n\to\infty} \frac{\sum_{i=1}^{n} a_i}{n} = \frac{1}{2}$.

Remarques :

- Plusieurs autres propriétés statistiques sont valides sur les suites aléatoires.
- La suite des décimales de π non aléatoire, pourtant elle suit les bonnes propriétés de statistiques : $pi = 3.14159265358979323846264338327950\ldots$
- Fabriquer de manière déterministe une suite aléatoire de bits (ou de n'importe quel objet) est contradictoire.

But : construire une suite pseudo-aléatoire, c'est-à-dire une suite qui s'approche d'une suite aléatoire.

Remarque : (Informellement)

On a besoin de s'approcher de l'imprévisible. Une suite a est pseudo-aléatoire si elle est produite par un algorithme et qu'il est algorithmiquement difficile de prévoir avec une probabilité $> \frac{1}{2}$ le bit a_{n+1} en connaissant tous les bits $a_1, a_2, ..., a_n$ précédents (formalisé par Yao en 1982).

Remarque :

On ne sait pas prouver qu'il existe une suite pseudo-aléatoire et le prouver permettrait de conclure $P \neq NP$. Pour le moment, on a une approche plutôt expérimentale pour construire des générateurs. Après construction d'un générateur, on vérifie que les propriétés mathématiques nécessaires (notamment statistiques) sont vérifiées.

Tests statistiques

- Difficile pour un humain de décider manuellement si une suite est aléatoire ou pas (tendance à éviter les choses qui "semblent" non-aléatoires,

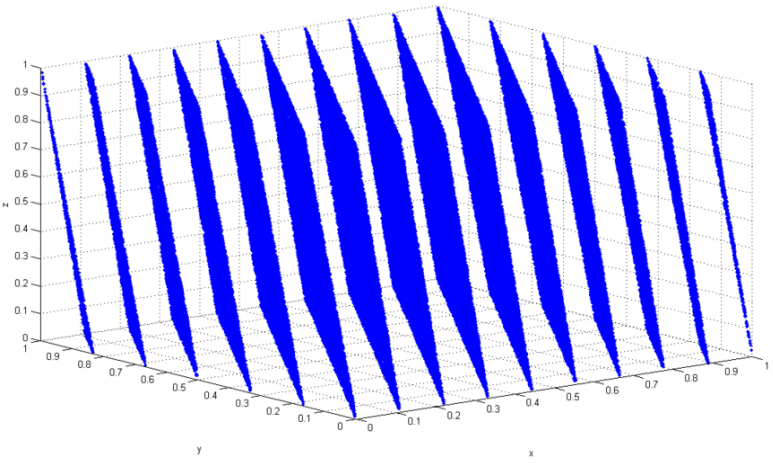
e.g. mêmes nombres consécutifs).

- $\pi = 3.14159265358979323846264338327950$
- Utiliser des tests mécaniques non-biaisés !

Proposer des tests statistiques.

Test spectral

Représentation en 3D de 100 000 valeurs générées par RANDU. Chaque point est détermine par trois tirages pseudo-aléatoires consécutifs. On voit ainsi que les points se trouvent sur un ensemble de 15 plans de



l'espace.

Diehard test and BigCrush test

- Diehard test : https://webhome.phy.duke.edu/~rgb/General/dieharder.php
- BigCrush test