Analyse d'Algorithmes et Génération Aléatoire

TD 1 : Génération d'Entiers

Table des matières

1 Nombres aléatoires

Exercice** 1 **: Le centre du carré

John von Neumann a proposé, en 1946, la méthode itérative suivante afin de générer des entiers décimaux à 10 chiffres. A partir d'un nombre décimal N à 10 chiffres, on génère son successeur en calculant N^2 et en ne retenant que les 10 chiffres du milieu. Si le carré ne contient pas 20 chiffres, on le précède de suffisamment de 0 afin d'obtenir un entier à 20 chiffres.
Ainsi le successeur de 1234567890 est le nombre 1578750190.

Question 1.1 Quel est le successeur de 1010101010 ? (calcul à la main possible !)

Question 1.2 Quelle est la suite d'entiers générée avec la graine 100000 ? (calcul à la main possible !)

Question 1.3 Coder l'algorithme de von Neumann (suivant le langage de programmation choisi, il faudra être plus ou moins astucieux).

Question 1.4 Générer les 10 premiers entiers générés pas l'algorithme de von Neumann avec la graine 1234567890.

Exercice** 2 **: Restriction aux nombres à 2 chiffres

On reprend l'idée de l'exercice précédent, mais en prenant en considération des nombres à 2 chiffres plutôt que 10 chiffres.

Question 2.1 Coder l'algorithme.

Question 2.2 En prenant chacune des 100 graines distinctes possibles (00, 01, 02, ...) déterminer le nombre de graines qui génèrent une suite contenant l'entier 00 ? Quelle est la conséquence de l'apparition de 00 ?

Question 2.3 Quelles sont les graines générant le plus d'entiers distincts ? Et quel est ce nombre d'entiers distincts ?

Exercice** 3 **: Analyse d'un générateur

L'idée consiste à analyser un générateur d'entiers (du type de l'Exercice 1.1) en traçant une graphique tel que pour tout $x \in [0, 999]$ on associe $f(x)$ qui est la période engendrée par x pour ce générateur.

Question 3.1 Coder cet analyseur de générateur, prenant en entrée une chaine de caractères (qui correspond au nom du programme), et construisant le graphique des périodes de chaque entier de 0 à 999.

Question 3.2 La période d'un générateur n'est pas la seule caractéristique importante. Donner une autre caractéristique et concevoir un programme permettant de l'analyser.

Exercice* 4 *: Algorithme K

Knuth a défini un générateur aléatoire d'entiers. Son pseudo-code est donné ci-dessous. On part d'un entier X sur 10 chiffres décimaux, et l'algorithme produit le nombre à 10 chiffres suivant pour constituer une suite d'entiers aléatoires.

Les étapes de l'algorithmes sont a effectuées séquentiellement, sauf s'il est dit explicitement de sauter à une étape (différente de la suivante cf. 2.).

1. Soit $Y \longleftarrow \lfloor X/10^9 \rfloor$. Donc Y est le digit le plus significatif de X. Les étapes de 2. à 13. seront répétées exactement (Y+1) fois.
2. Soit $Z \longleftarrow \lfloor X/10^8 \rfloor \mod 10$. Donc Z est le second digit le plus significatif de X. Aller directement à l'étape (3+Z).
3. Si $X < 5 \cdot 10^9$, alors remplacer par : $X \longleftarrow X + 5 \cdot 10^9$.
4. Remplacer par : $X \longleftarrow \lfloor X^2/10^5 \rfloor \mod 10^{10}$.
5. Remplacer par : $X \longleftarrow (1001001001 \cdot X) \mod 10^{10}$.
6. Si $X < 10^8$, alors remplacer par : $X \longleftarrow X + 9814055677$. Sinon remplacer par : $X \longleftarrow 10^{10} - X$.
7. Remplacer par : $X \longleftarrow 10^5 \cdot (X \mod 10^5) + \lfloor X/10^5 \rfloor$ (c'est-à-dire, inverser les 5 digits de poids fort avec les 5 de poids faible).
8. Remplacer par : $X \longleftarrow (1001001001 \cdot X) \mod 10^{10}$.
9. Décrémenter chaque digit strictement positif de 1.
10. Si $X < 10^5$, alors remplacer par : $X \longleftarrow X^2 + 99999$. Sinon remplacer par : $X \longleftarrow X - 99999$.
11. Tant que $X < 10^9$, remplacer par : $X \longleftarrow 10 \cdot X$.
12. Remplacer par : $X \longleftarrow \lfloor X \cdot (X-1)/10^5 \rfloor \mod 10^{10}$.
13. Si $Y > 0$, alors remplacer $Y \longleftarrow Y - 1$ et retourner à l'étape 2. Sinon l'algorithme se termine et la valeur de X est renvoyée.

Question 4.1 Pour quelle raison ce générateur n'est-il pas en mesure de fournir une infinité d'entiers aléatoires dans le sens où, quelle que soit la graine choisie, la suite générée sera périodique ?

Question 4.2 Encoder cet algorithme dans le langage de votre choix.

Question 4.3 Donner la trace de l'algorithme K en partant de l'entier $X = 6065038420$. On souhaite avoir le numéro de l'instruction exécutée, la valeur de X à la fin de cette instruction et la valeur de Y également. Ainsi, on voudrait compléter le tableau suivant :

Étape	X	Y
au départ	6065038420	
1.	6065038420	6
2.	...	...

Question 4.4 Que conclure de la question précédente ?

Question 4.5 Justifier pour quelle raison l'étape 11. termine toujours.

Question 4.6 En s'appuyant uniquement sur la trace calculer à la Question 3, quel est le premier entier généré par l'algorithme K prenant la graine 3830951656 ? Et le second ? (vérifier à l'aide de votre programme !)

Exercice* 5 *: Détection de période

Nous souhaitons générer une suite d'entiers $X_0, X_1, X_2, X_3, \ldots$ vérifiant $0 \leq X_n < m$, avec m fixé et $n \in \mathbb{N}$. Soit f une fonction vérifiant : $\forall x \in \{0, 1, \ldots, m-1\}$ alors $f(x) \in \{0, 1, \ldots, m-1\}$. On construit une suite basée sur l'itération suivante : $X_{n+1} = f(X_n)$, avec X_0 donné. L'algorithme de von Neumann ou l'algorithme K en sont deux exemples.

Question 5.1 Montrer qu'il existe deux entiers μ et λ tels que les entiers

$$X_0, X_1, \ldots, X_\mu, \ldots, X_{\mu+\lambda-1}$$

sont tous distincts, et $\forall n \geq \mu$, on a $X_{n+\lambda} = X_n$. Déterminer les valeurs minimales et maximales des entiers μ et λ.

Question 5.2 Montrer qu'il existe un entier $n > 0$ tel que $X_n = X_{2n}$; montrer de plus que la plus petite valeur n satisfaisant la condition précédente est telle que $\mu \leq n \leq \mu + \lambda$; enfin, montrer qu'une seule valeur n vérifie les 2 conditions précédentes.

Question 5.3 Donner le pseudo-code d'une fonction prenant la fonction f et la graine X_0 en paramètres et renvoyant les valeurs de μ et λ en seulement $O(\mu + \lambda)$ étapes et avec un espace mémoire borné.

Question 5.4 Coder l'algorithme précédent et le tester sur les algorithmes de von Neumann et K pour des graines de votre choix.

2 Générateurs linéaires congruentiels

Exercice** 6 **: Générateurs génériques

On rappelle qu'un générateur linéaire congruentiel est basé sur l'équation suivante :

$$X_{n+1} = (aX_n + c) \mod m; \qquad X_0 \text{ fixé.}$$

Question 6.1 Encoder un générateur générique prenant en entrée a, c, m et X_n et renvoyant X_{n+1}.

Question 6.2 Tester votre générateur sur un exemple de votre choix en générant les 10 entiers suivant la graine de votre choix.

Question 6.3 Reproduire la Figure 1 du "Pour la Science 1998" : `http://www.lifl.fr/~jdelahay/pls/1998/051.pdf`

Question 6.4 Reprendre la question précédente en utilisant en trois dimensions.

Exercice** 7 **: Générateur de Java

Question 7.1 Le générateur RAND48 est basé sur les paramètres : $a = 25214903917$, $c = 11$ et $m = 2^{48}$. Calculer les 10 premiers termes de RAND48 à partir de la graine 0.

Question 7.2 Le générateur de java est basé sur RAND48. Tous les calculs sont effectués avec une précision de 48 bits, mais le résultat renvoyé par le générateur est un entier signé de 32 bits. Pour passer d'un entier de 48 bits à un entier de 32 bits, les 16 bits de poids faible sont effacés. Coder l'équivalent du générateur de java et vérifier qu'avec la graine (sur 48 bits) 156079716630527 les premiers entiers générés sont :

$$-956251568, -2113186618, 1962154824, 449949881, -1374163520, 392258983, \ldots$$

Exercice** 8 **: Graine du générateur de java

Supposons que vous disposiez de 2 entiers consécutifs, v_1 et v_2 générés par java.

Question 8.1 Donner un pseudo-code permettant de retrouver la valeur V_1 sur 48 bits dont les 32 bits de poids forts correspondent à v_1 et utiliser le générateur de java avec la graine V_1 renvoie v_2.

Question 8.2 Quelle est la conséquence du pseudo-code précédent ?

Question 8.3 Encoder votre pseudo-code et *tester* sa correction.

Exercice** 9 **: Deviner les entiers précédents générés par java

Le but de cet exercice est de retrouver les entiers précédemment générés par java.

Question 9.1 Poser la multiplication suivante à la main, comme vous l'avez appris à l'école primaire.

$$1101 \times 1110.$$

Question 9.2 Poser la multiplication suivante à la main, sachant que les nombres sont en base 2.

$$1101 \times 1110.$$

La multiplication binaire donne 10110110.

Question 9.3 Supposons que vous n'ayez pas accès à la division, mais que vous souhaitiez trouver x vérifiant (en binaire) $1101 \times x = 10110110$. A l'aide de la question précédente, trouver un algorithme permettant de trouver x.

Question 9.4 A partir de deux entiers v_1 et v_2 générés de façon consécutive par java, donner un algorithme permettant de retrouver la valeur v_0 qui correspond à l'entier qui a été renvoyé par le générateur juste avant v_1.

Question 9.5 Encoder l'algorithme de la question précédente et le tester.

Exercice 10 ***: Inverser un générateur générique***

Question 10.1 Adapter l'algorithme final de l'exercice précédent à un générateur linéaire congruentiel quelconque, sachant qu'on suppose connu a, c, m et les 2 entiers consécutifs v_1 et v_2.
Attention : dans le cas précédent, le multiplicateur a était impair.

Question 10.2 Effectuer des tests sur des générateurs différents.

Exercice 11 ***: Le générateur RANDU***

Dans les années 60, IBM a introduit le générateur congruentiel suivant sur certaines de ses machines : $a = 65539$, $c = 0$ et $m = 2^{31}$.
Ce choix de valeur avait été effectué car $65539 = 2^{16} + 3$.

Question 11.1 Pour quelle raison a avait été choisi égal à 65539 ?

Question 11.2 Calculer $a^2 \mod 2^{31}$.

Question 11.3 En déduire une équation donnant X_{n+2} en fonction de X_n et X_{n+1}.

Question 11.4 Grâce à l'équation précédente, montrer géométriquement le gros inconvénient de ce générateur.

3 Le twister de Mersenne

Exercice 12 ***: Le générateur***

Question 12.1 A l'aide de la description ci-dessous en pseudo-code (de la page wikipedia) coder l'algorithme dans le langage de votre choix.

```
// Create a length 624 array to store the state of the generator
int[0..623] MT
int index = 0

// Initialize the generator from a seed (not the single possible way to initialize the generator !)
function initialize_generator(int seed) {
    index := 0
    MT[0] := seed
    for i from 1 to 623 { // loop over each other element
        MT[i] := lowest 32 bits of(1812433253
                        * (MT[i-1] xor (right shift by 30 bits(MT[i-1]))) + i) // 0x6c078965
    }
}

// Extract a tempered pseudorandom number based on the index-th value,
// calling generate_numbers() every 624 numbers
function extract_number() {
    if index == 0 {
        generate_numbers()
    }

    int y := MT[index]
    y := y xor (right shift by 11 bits(y))
    y := y xor (left shift by 7 bits(y) and (2636928640)) // 0x9d2c5680
    y := y xor (left shift by 15 bits(y) and (4022730752)) // 0xefc60000
    y := y xor (right shift by 18 bits(y))

    index := (index + 1) mod 624
    return y
}

// Generate an array of 624 untempered numbers
function generate_numbers() {
    for i from 0 to 623 {
        int y := (MT[i] and 0x80000000)                       // bit 31 (32nd bit) of MT[i]
                        + (MT[(i+1) mod 624] and 0x7fffffff)   // bits 0-30 (first 31 bits) of MT[...]
        MT[i] := MT[(i + 397) mod 624] xor (right shift by 1 bit(y))
        if (y mod 2) != 0 { // y is odd
            MT[i] := MT[i] xor (2567483615) // 0x9908b0df
        }
    }
}
```

4 Complexité de Kolmogorov

Exercice 13 ***: Compression***

Générer un fichier binaire de bits "pseudo-aléatoires" de 1Mo avec une proportion 1 de $p_1 = 0.0, 0.1, 0.2, ..., 0.9, 1.0$. Puis le compresser (e.g. avec gzip). Tracer la taille divisée par la taille initiale en fonction de p_1. Tracer sur le même graphe la courbe $y = -x\log_2(x) - (1-x)\log_2(1-x)$. Que peut on conclure ?