info602 : Mathématiques pour l'informatique
TD 2 : générateurs aléatoires

Pierre Hyvernat
Laboratoire de mathématiques de l'université Savoie Mont Blanc
bâtiment Chablais, bureau 17, poste : 94 22
email : Pierre.Hyvernat@univ-smb.fr
www : http://www.lama.univ-smb.fr/~hyvernat/

Partie 1 : Générateurs pseudo-aléatoires simples

Exercice 1 : congruences linéaires

Une méthode simple pour générer des nombres pseudo-aléatoire entre 0 et m est d'utiliser la formule

$$x_{n+1} = ax_n + c \pmod m$$

avec un choix approprié de a, c, m et x_0.

Question 1. Expliquez pourquoi un tels générateur est toujours périodique et donnez une borne précise sur sa période.

Question 2. Calculez les nombres générés ainsi que la période si on choisit $a = 6$, $c = 1$, $m = 24$ et $x_0 = 4$.

Question 3. Les vieilles implémentations de la fonction `rand` dans la GLibc utilisaient $m = 2^{31}$. Pourquoi ?

Quelles précautions faut il prendre pour gérer correctement les dépassements de capacité ?

Question 4. Pourquoi ne faut il pas utiliser un générateur congruentiel linéaire avec $m = 2$ pour générer des *bits* aléatoires ?

Question 5. La page de manuel de la fonction `rand` sur mon système contient la note suivante :

> `However, on older rand() implementations, and on current implementations on different systems, the lower-order bits are much less random than the higher-order bits.`

Regardez comment le bit de poids faible de x évolue lorsqu'on utilise une puissance de 2 pour m, et expliquez cette remarque.

Exercice 2 : registres à décalage et rétroaction

Question 1. Les générateurs de type *"Linear feedback shift register"* ("Registre à décalage à rétroaction linéaire") permettent de générer des suites de bits ressemblant à des suites aléatoires. Les plus simples sont de type *Fibonacci* où chaque nouveau bit est obtenu en faisant le XOR de quelques bits précédents. Par exemple :

$$b_n = b_{n-3} \oplus b_{n-4}$$

Calculez les bits générés si on commence avec un état qui ne contient que des 1.

Idem, mais si on commence avec un état qui ne contient que des 0.

Question 2. Un tels générateur est forcément périodique. Pourquoi ?

Quelle est la période maximale d'un tels générateur qui utilise les valeurs des n bits précédents ?

Question 3. On considère maintenant le générateur

$$b_n = b_{n-1} \oplus b_{n-4} \oplus b_{n-6} \oplus b_{n-30}$$

Dans cet exemple, on peut stocker les 30 bits précédents dans un nombre entier (32 bits). Le bit généré est calculé, puis on fait un décalage à gauche et on ajoute le bit généré dans le bit de poids faible (ou alors on fait un décalage vers la droite, et on ajoute le bit généré dans le 30ème bit).

En supposant que `state` contient un entier correspondant aux 30 bits précédents, trouvez une suite d'instructions C la plus courte possible qui calcule le bit généré ainsi que le nouvel état.

Question 4. Écrivez un programme en C pour calculer la période de l'exemple donné ci dessus si on commence avec `state = 0x3fffffff`.

(Remarque : le générateur boucle sur l'état initial...)

Partie 2 : quelques programmes aléatoires

Question 1. On souhaite tirer un nombre parmi $\{0, 1, 2, 3, 4\}$ avec la méthode suivante : *on tire un dé standard à 6 faces, et on prend le modulo* 5 *du résultat.*

- Expliquez pourquoi le résultat n'est pas équitable en donnant la probabilité d'obtenir chacun des nombres 0, 1, 2, 3 et 4.
- Le problème est il toujours présent si on utilise le modulo 3 pour obtenir un nombre parmi $\{0, 1, 2\}$?

Question 2. On utilise maintenant la fonction `rand()` (au lieu d'un lancé de dé à 6 faces) suivie d'un modulo 5. (La fonction `rand` renvoie un nombre pseudo aléatoire entre 0 et $2^{31} - 1 = 2147483647$, qui est un nombre premier.)

- Le résultat est il équitable, *théoriquement parlant* ?
- Le résultat est il équitable, *pratiquement parlant* ?

N'oubliez pas de justifier !

Question 3. Donnez une manière théoriquement équitable pour tirer un nombre uniformément entre $0 \leq r < n$ à partir d'un générateur qui renvoie des nombres $0 \leq r \leq$ `RAND_MAX` (avec $n \leq$ `RAND_MAX`).

Question 4. On dispose d'un générateur aléatoire `random_bit` de bits indépendants *biaisé* : il produit des `0` avec probabilité 9/10 et des `1` avec probabilité 1/10.

Cherchez une manière de générer des bits aléatoires avec probabilités 1/2 et 1/2.

Exercice 1 : sélection

Question 1. On souhaite afficher exactement 10 éléments aléatoires parmi un tableau de taille 100. Chaque élément doit donc avoir une probabilité 0.1 d'être tiré.

Expliquez pourquoi l'algorithme suivant ne répond pas à la question, et proposez une solution.

```
for (int i=0; i<100; i++) {
    if (rand()%10 == 0) {  // une chance sur 10 de tomber sur 0
        printf("%d\n", T[i]);
    }
}
```

Question 2. Si on note `A(k, n)` la fonction qui doit afficher `k` éléments parmi un tableau de taille `n`, chaque élément doit être pris avec une probabilité `k/n`. Une manière est simplement de regarder si `rand()%n < k`. (Remarque : on ignore le biais de la distribution non-uniforme `random()%n`.)

- Si le premier élément du tableau est affiché, comment doit se comporter la suite de la fonction `A(k, n)` ?
- si le premier élément du tableau n'est pas affiché, comment doit se comporter la suite de la fonction `A(k, n)` ?

Déduisez en une fonction `A(k, n)`, de complexité linéaire, qui affiche `k` éléments parmi `n`. Êtes vous sûr que cette fonction affiche *exactement* `k` éléments ?

Question 3. On peut choisir 1 élément parmi un *flot* de la manière suivante (pseudo Python)

```
s = flot.next()  # premier élément du flot
nb_elem = 1      # nombre d'éléments vus
for x in flot:
    nb_elem += 1
    if randint(0, nb_elem-1) == 0:   # tirage uniforme parmi 0, ..., nb_elem-1
        s = x
```

Il n'est pas nécessaire de connaitre la taille du flot, ni de stocker tous ses éléments !

- Si le flot ne contient qu'un seul élément, il sera contenu dans la sélection `s`.
- Quel sera la valeur de `s`, et avec quelle probabilité, si le flot ne contient que 2 éléments ?
- Et s'il contient 3 éléments, ou n ?

Question 4. Généralisez l'algorithme précédent pour choisir `k` éléments uniformément dans un flot sans connaitre sa taille ni stocker plus que `k` éléments du flots en même temps.

Exercice 2 : mélanger un tableau

On considère l'algorithme suivant pour mélanger un tableau `T` de taille `t` :

```
for (int i=0; i<t; i++) {
    r = random()% t;      // on tire un nombre aléatoire entre 0 et t-1
    tmp = T[i]; T[i] = T[r]; T[r] = tmp;      // on échange T[i] et T[r]
}
```

Question 1. Combien de tirages aléatoires sont effectués ? Combien d'exécutions possibles cela donne t'il ?

Question 2. Il y a $\texttt{t}! = 1 \times 2 \times 3 \times \cdots \times \texttt{t}$ permutations distinctes de `T`. Est t'il possible que chacune d'entre elle soit obtenue le même nombre de fois par l'algorithme précédent ?

Qu'en pensez vous ?

Question 3. La question précédente montre que l'algorithme décrit n'est *théoriquement* pas uniforme, mais il pourrait l'être en pratique.

Ce n'est pas le cas, comme le montre le raisonnement suivant.

- Quel est, après un mélange uniforme de `T = 0, 1, 2, ..., t-1`, la probabilité que la case `T[0]` contienne `t-1` ?
- Quelles sont les 2 seules manières de mettre `t-1` dans la case `T[0]` ?
- Quelle est la probabilité d'obtenir `T[0] = t-1` lorsque la taille du tableau (`t`) est 10, 100, 1000 ? (Utilisez un ordinateur pour faire le calcul !)

Qu'en pensez vous ?

Question 4. On considère maintenant l'algorithme de Fisher-Yates

```
for (int i=t-1; i>0; i--) {
    r = random() % (i+1);      // on tire un nombre aléatoire entre 0 et i
    tmp = T[i]; T[i] = T[r]; T[r] = tmp;        // on échange T[i] et T[r]
}
```

Combien y a t'il d'exécutions possibles ?

Qu'en pensez vous ?