Introduction à la cryptographie
Devoir Maison - Test de Miller-Rabin

BENKARRAD Alaa Eddine.
CHERIFI Abdelaziz.

Réponse 1.

1. Le langage de programmation que nous avons choisi est : Langage JAVA
2. La bibliothèque permettant de générer des nombres entiers de grande taille que nous allons utiliser est la bibliothèque BigInteger[1] de JAVA.
3. Les opérations implémentées dans cette bibliothèque sont les opérations habituelles (addition, multiplication, division, etc...), en plus de ça, cette bibliothèque offre des opérations de calcul en arithmétique modulaire, calcul du PGCD, génération de nombre premier, test pour savoir si un entier est premier, etc....

Réponse 2.

Les nombres aléatoires sont des nombres qui se produisent dans une séquence telle que deux conditions sont remplies[3] :

1. Les valeurs sont uniformément réparties sur un intervalle ou un ensemble défini.
2. Il est impossible de prédire les valeurs futures en fonction des valeurs passées ou présentes.

Autrement dit, une des conditions les plus importantes d'un nombre aléatoire est d'être indépendant, car cela permet d'établir aucune corrélation entre des nombres successifs. Il faut veiller à ce que la fréquence de ces nombres aléatoires soit approximativement la même. Par conséquent, en théorie, il n'est pas facile de générer un nombre aléatoire long[4].

Pour cela, nous utilisons, dans notre implémentation, la bibliothèque SecureRandom pour générer les nombres aléatoires. Cette classe fournit un générateur de nombres aléatoires cryptographiquement puissant, comme décrit dans la RFC 1750[2].

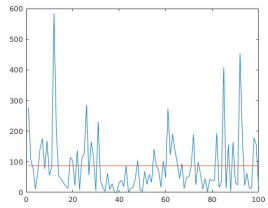
Réponse 6.

En executant notre implémentation avec les 3 nombres (n1, n2 et n3 écrits en hexadécimal) et cpt = 20, nous avons obtenue les resultats suivants :

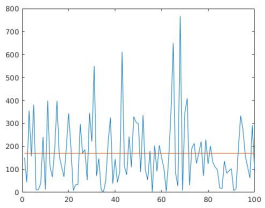
— **n1** est pseudo-premier.
— **n2** est composé.
— **n3** est pseudo-premier.

Réponse 8.

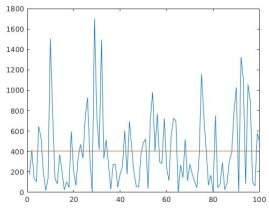
La figure ci-apres represente des courbes des differents tests de la fonction Eval(). Ces resultat ont été pris en executons la fonction Eval() 100 fois avec les valeurs de b suivantes : 128, 256, 512, 1024, 2048 et 4096.



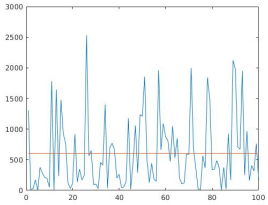
(a) 128 bits



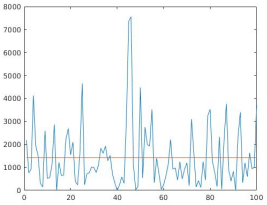
(b) 256 bits



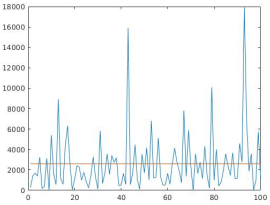
(c) 512 bits



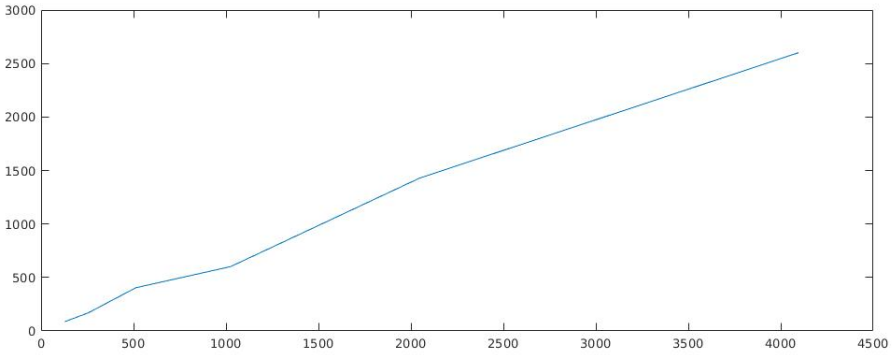
(d) 1024 bits



(e) 2048 bits



(f) 4096 bits



(g) courbe des moyennes

FIGURE 1: tests de fonction eval.

Sur les figures (a-f), l'axe des abscisses représente les differentes executions tandis que celui des ordonnées représente la complexité (). La courbe des executions est en bleue et la courbe en rouge represente la moyenne.

Réponse 9.

La figure 1 (g) ci-dessus représente la courbe de complexité (en ordonnée) selon la taille de nombre en bits (en abscisse).

On peut y remarquer que cette courbe représente une fonction croissante, ce qui implique que la complexité est directement proportionnelle avec la taille de nombre en bits.

En d'autre termes, on remarque que la complexité augmente presque linéairement, ce qui nous amène à constater que ce générateur de nombres aléatoires (basés sur le test de Miller Rabin) est en temps polynomiale.

Réponse 10.

Le test qui permet de garantir si un nombre premier est le test d'AKS (aussi connu comme le test de primalité Agrawal-Kayal-Saxena) ce test est un test déterministe et généraliste, il permet de déterminer de façon certaine si un nombre est premier contrairement aux tests probabilistes.

Sa complexité temporelle originale est en $\mathcal{O}((log\ n)^{10.5})$ en mars 2003, d'autres mathématiciens ont essayés d'améliorer sa complexité en $\mathcal{O}((log\ n)^4)$ en revanche, l'algorithme en cette complexité n'était pas déterministe [5].

Références

[1] BigInteger (JavaDoc), https ://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html.

[2] SecureRandom (JavaDoc),

[3] TechTarget, https ://whatis.techtarget.com/definition/random-numbers

[4] Techopedia, https ://www.techopedia.com/definition/31706/random-number

[5] L'algorithmeAKS, http ://www.trigofacile.com/maths/curiosite/primarite/aks/pdf/algorithme-aks.pdf