Présentation : projet primalité

Sonny Klotz - Idir Hamad - Younes Benyamna - Malek Zemni

UVSQ

30/05/2018

Projet de M1 informatique UVSQ sur les tests de primalité :

- Existence de plusieurs tests de primalité.
- Utilisés pour générer des nombres premiers.
- Performances différentes.

Projet de M1 informatique UVSQ sur les tests de primalité :

- Existence de plusieurs tests de primalité.
- Utilisés pour générer des nombres premiers.
- Performances différentes.

Objectif : implémentation, description (rapport) et mesures de performances pour construire un générateur optimal.

Plan

Principales fonctionnalités de l'application développée :

- Implémentation de tests de primalité :
  1. Test Naif
  2. Test de Wilson
  3. Test de Fermat
  4. Test de Miller-Rabin
  5. Test de Solovay-Strassen
  6. Test AKS
- Mesures de performances.
- Génération de nombres premiers.

Organigramme de l'application et données échangées :

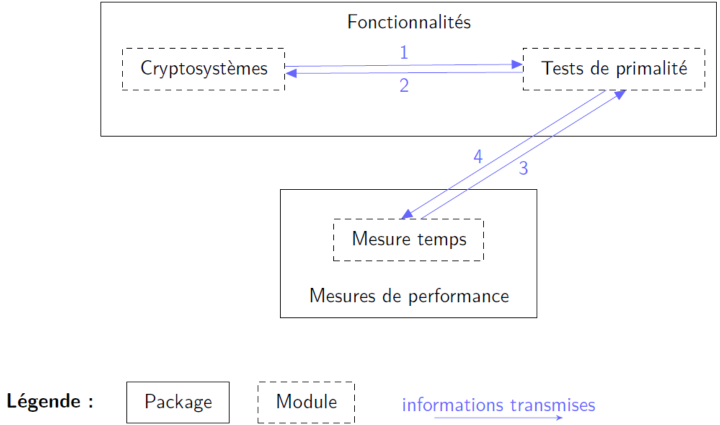

FIGURE 1 – Organigramme des différents modules de l'application

Fonctionnement de l'application :

- Mesures de performance de chaque test.
- Exploitation des résultats de mesures pour construire le générateur optimal.
- Utilisation du générateur par l'utilisateur.

Fonctionnement de l'application :

- Mesures de performance de chaque test.
- Exploitation des résultats de mesures pour construire le générateur optimal.
- Utilisation du générateur par l'utilisateur.

Technologies utilisées :

- Langage C + GMP
- LaTeX
- Gnuplot

Plan

Tests de primalité importants dans la **cryptographie à clé publique**.
Utilisés pour générer de grands nombres premiers.

- **RSA** : phase de génération des clés.
- **ElGamal** : phase d'échange de clés.

Cas de RSA :

- Générer 2 grands nombres premiers distincts p et q.
- Module RSA : $n = p * q$. La taille en bits de p et q est la moitié de celle de n.
- Calcul de $\phi(n) = (p-1) * (q-1)$.
- Clé publique : e premier avec $\phi(n)$.
- Clé privée : $d = e^{-1} \pmod{\phi}(n)$.

Importance des nombres premiers : la **factorisation** de n permet de retrouver la clé privée d.

Plan

Processus de génération des nombres premiers :

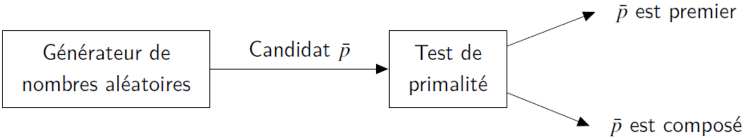

FIGURE 2 – Processus de génération des nombres premiers

Importance du générateur :

- Utilisé par les cryptosystèmes pour générer des nombres premiers.
- Importance pour la sécurité : générateur aléatoire imprévisible.

Génération des nombres premiers

Importance du générateur :

- Utilisé par les cryptosystèmes pour générer des nombres premiers.
- Importance pour la sécurité : générateur aléatoire imprévisible.

Importance du test de primalité :

- Assure que le nombre aléatoire est premier.
- Performances du générateur dépend du choix du test de primalité.

Plan

Teste si un nombre est **premier** ou **composé**.

Existence de plusieurs tests, avec deux type différents :

- **Tests déterministes :** renvoient toujours le bon résultat.
- **Tests probabilistes :** renvoient un résultat avec une certaine probabilité.

Teste si un nombre est **premier** ou **composé**.

Existence de plusieurs tests, avec deux type différents :

- **Tests déterministes :** renvoient toujours le bon résultat.
- **Tests probabilistes :** renvoient un résultat avec une certaine probabilité.

Algorithme	Année	Type
Naïf (Crible d'Eratosthène)	-240	Déterministe
Fermat	1640	Probabiliste
Wilson	1770	Déterministe
Miller-Rabin	1976	Probabiliste
Solovay-Strassen	1977	Probabiliste
AKS	2002	Déterministe

Plan

Méthode la plus intuitive de tester si un nombre est premier ou composé, à laquelle on a effectué plusieurs optimisations :

Algorithme 1 : Test naïf

Données : un entier n
pour *tout nombre premier* $p \leqslant \sqrt{n}$ **faire**
  si p *divise* n **alors**
    retourner composé;
retourner premier;

Complexité : $O(\sqrt{n})$ opérations.

Utilisation du **crible d'Eratosthène** (III^e siècle av. J.-C.) pour pré-calculer et stoker dans une table tous les nombres premiers $\leqslant \sqrt{n}$:

Algorithme 2 : Crible d'Eratosthène

Données : un entier N qui correspond à $\sqrt{n}$

Créer une liste L de couples *(entier, primalité)*, pour les entiers allant de 2 jusqu'à N, avec une primalité initialisée à "premier" : ***L = {(2, premier), (3, premier), ..., (N, premier)}*** ;

plusGrandPremier $= N$;

pour *tout nombre p marqué "premier" de la liste L (de manière croissante)* **faire**

  si $p^2 >$ ***plusGrandPremier*** **alors**

    retourner L;

  i $= 2$;

  tant que $p * i < N$ **faire**

    Marquer "composé" l'entier à la position $p * i$;

    Mettre à jour ***plusGrandPremier*** ;

    $i{+}{+}$;

Plan

Test basé sur une propriété simple :

Théorème (Théorème de Wilson)

Un entier $n > 1$ est un nombre premier si et seulement si

$$(n-1)! + 1 \equiv 0 \pmod{n}$$

Algorithme 3 : Test de Wilson

Données : un entier n

si $(n-1)! + 1 \equiv 0 \pmod{n}$ **alors**

  retourner premier;

sinon

  retourner composé;

Complexité : $O(n)$.

Plan

Test de primalité probabiliste basé sur le petit théorème de Fermat :

Théorème (Petit théorème de Fermat (énoncé 1))

Si p est un nombre premier, alors pour tout nombre entier a premier avec p

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

Énoncé la première fois en 1640 par *Pierre de Fermat*.

- Choix de $1 < a < n$.
- Calcul de $\mathbf{a^{n-1} \pmod{n}}$.
- k répétitions.

Algorithme 4 : Test de Fermat

Données : un entier n et le nombre de répétitions k
pour $i = 1$ *jusqu'à* k **faire**
    Choisir aléatoirement a tel que $1 < a < n$;
    si $a^{n-1} \not\equiv 1 \pmod{n}$ **alors**
        retourner composé;
retourner probablement premier;

Présence de nombres **pseudo-premiers** et nombres de **Carmichael**.

Définition (Nombre pseudo-premier)

Un nombre pseudo-premier est un nombre premier probable (un entier naturel qui partage une propriété commune à tous les nombres premiers) qui n'est en fait pas premier. Un nombre pseudo-premier provenant du théorème de Fermat est appelé nombre pseudo-premier de Fermat.

Définition (Nombre de Carmichael)

Un entier positif composé n est appelé nombre de Carmichael si pour tout entier a premier avec n,

$$a^{n-1} \equiv 1 \pmod{n}$$

Complexité : dépend de

- la multiplication modulaire : $C_{mult}(n)$
- l'exponentiation modulaire ***Square And Multiply*** : $log_2(n) \cdot C_{mult}(n)$

$\Longrightarrow$ Complexité test de Fermat : $O(k \cdot log_2(n) \cdot C_{mult}(n))$.

Plan

Probabiliste, initialement publié en 1976, basé sur un raffinement du petit théorème de Fermat.

- Écriture $\mathbf{n-1=2^s t}$ pour $n>0$ et t impair
- Exploitation dans la propriété du petit théorème de Fermat : $a^{n-1} \equiv 1 \pmod{n}$
- Établissement de la propriété de primalité, si :

$$a^{2^j t} \equiv 1 \pmod{n} \quad \text{pour un } j \in \{0, 1, ..., s-1\},$$

  alors n est composé.

Idée de preuve :

$$a^{2^s t} - 1 = (a^{2^{s-1} t})^2 - 1 = (a^{2^{s-1} t} + 1)(a^{2^{s-1} t} - 1)$$

On reconnait une autre identité remarquable si $s-1>0$ et on réitère.

Algorithme 5 : Test de Miller-Rabin

Données : un entier n et le nombre de répétitions k

Décomposer $n-1=2^s t$, avec $s \in \mathbb{N}^*$ et $t \in \mathbb{N}$ impair ;

pour $i=1$ *jusqu'à* k **faire**

  Choisir aléatoirement a tel que $1 < a < n$;

  $y \leftarrow a^t \pmod{n}$;

  si $y \not\equiv 1 \pmod{n}$ *et* $y \not\equiv -1 \pmod{n}$ **alors**

    pour $j=1$ *jusqu'à* $s-1$ **faire**

      $y \leftarrow y^2 \pmod{n}$;

      si $y \equiv 1 \pmod{n}$ **alors**

        retourner composé;

      si $y \equiv -1 \pmod{n}$ **alors**

        Arrêter la boucle de j et continuer avec le i suivant (sans renvoyer composé);

      retourner composé;

retourner probablement premier;

Répétitions et probabilité d'erreurs : la probabilité que le test renvoie premier à tort après k itérations est de $1/4^k$.

Complexité :

- $log(n)$ multiplications modulaires
- k itérations
- Au total : $O(k \cdot log(n) \cdot C_{mult}(n))$.

Choix du test

La **précision** et la **vitesse d'exécution** font de ce test un des meilleurs choix pour une utilisation en cryptographie.

Exemple : proba d'echec $< 10^{-6} \Rightarrow k = 10$

Plan

Probabiliste, publié en 1977, basé sur le ***critère d'Euler***.

Théorème (Critère d'Euler)

Soient $p > 2$ un nombre premier et a un entier premier avec p

- *Si a est un résidu quadratique modulo p, alors* $a^{\frac{n-1}{2}} \equiv 1 \pmod{p}$.
- *Si a n'est pas un résidu quadratique modulo p, alors* $a^{\frac{n-1}{2}} \equiv -1 \pmod{p}$.

Ceci se résume en utilisant le symbole de Legendre par :

$$a^{\frac{n-1}{2}} \equiv \left(\frac{a}{p}\right) \pmod{p}$$

Algorithme 6 : Test de Solovay-Strassen

Données : un entier n impair et le nombre de répétitions k
pour $i = 1$ *jusqu'à* k **faire**
  Choisir aléatoirement a tel que $2 < a < n$;
  $x \leftarrow \left(\frac{a}{n}\right)$;
  si $x = 0$ *ou* $x \not\equiv a^{\frac{n-1}{2}} \pmod{n}$ **alors**
    retourner composé;
retourner probablement premier;

Complexité : L'évaluation du symbole de Jacobi n'est pas dominant devant l'exponentiation modulaire.

- évaluation du symbole de Jacobi en $log(n)^2$
- $log(n)$ multiplications modulaires
- k itérations
- Au total : $O(k \cdot log(n) \cdot C_{mult}(n))$.

Idée de preuve : Si n premier on a,

$$a^{n-1} = 1 \Leftrightarrow (a^{\frac{n-1}{2}} - 1) \cdot (a^{\frac{n-1}{2}} + 1) = 0$$

- Si $a = x^2$ le premier facteur s'annule (petit théorème de Fermat).
- Sinon le deuxième s'annule.
- On vérifie cela dans notre test à l'aide du symbole de Jacobi.

Plan

Déterministe, publié en 2002, temps polynomial, basé sur une généralisation du petit théorème de Fermat :

Théorème (Petit théorème de Fermat généralisé)

Pour tout entier $n \geqslant 2$ et $a \in \mathbb{Z}$ premiers entre eux,

$$n \text{ est premier} \quad \Leftrightarrow \quad (X+a)^n \equiv X^n + a \pmod{n}$$

Le symbole X représente un symbole formel.

L'évaluation de cette équation est trop longue.

Algorithme 7 : Test AKS

Données : un entier $n > 1$

1 - **si** $n = a^b$ *pour des entiers* $a > 1$ *et* $b > 1$ **alors**
  retourner composé;

2 - Déterminer le plus petit entier r tel que $Ord_r(n) > log_2(n)^2$ dans $\mathbb{Z}_r$ (si r n'est pas premier avec n, on passe cet r);

3 - **pour** *tout* $a \leq r$ **faire**
  si $1 < pgcd(a, n) < n$ **alors**
    retourner composé;

4 - **si** $n \leq r$ **alors**
  retourner premier;

5 - **pour** $a = 1$ *jusqu'à* $\lfloor \sqrt{\varphi(r)} log_2(n) \rfloor$ **faire**
  si $(X + a)^n \not\equiv X^n + a$ *dans* $\frac{\mathbb{Z}/n\mathbb{Z}[X]}{(X^r - 1)\mathbb{Z}/n\mathbb{Z}[X]}$ **alors**
    retourner composé;

6 - **retourner** premier;

Principe :

- r choisi à l'étape 2 :
- réduction mod $X^r - 1$ à l'étape 6, on n'a plus l'équivalence dans le théorème.
- Si on prouve l'équation pour plusieurs a alors $n = p^b$ où p est premier.
- L'étape 1 nous permet de conclure si $b = 1$ ou non.

Complexité : Premier test déterministe prouvé en temps polynomial. Estimation la plus utilisée : $O(log(n)^6)$.

Plan

Évolution des tests de primalité :

Évolution des tests de primalité :

- Premiers tests déterministes
  - Test Naïf - Crible d'Eratosthène (III$^{\text{s}}$iècle avant J-C.)
  - Test de Wilson (1770)

Évolution des tests de primalité :

- Premiers tests déterministes
  - Test Naïf - Crible d'Eratosthène (III^{s}iècle avant J-C.)
  - Test de Wilson (1770)

- Tests probabilistes
  - Test de Fermat (1640) longuement utilisé
  - Apparition du test de Solovay-Strassen (1977)
  - Remplacés par le test de Miller-Rabin à partir de 1980 (bonne probabilité)

Évolution des tests de primalité :

- Premiers tests déterministes
  - Test Naïf - Crible d'Eratosthène (III^{s}iècle avant J-C.)
  - Test de Wilson (1770)

- Tests probabilistes
  - Test de Fermat (1640) longuement utilisé
  - Apparition du test de Solovay-Strassen (1977)
  - Remplacés par le test de Miller-Rabin à partir de 1980 (bonne probabilité)

- Test déterministe rapide : AKS (2002), complexité polynomiale, améliorations envisageables

Mesures de performance : temps d'exécution

Algorithme 8 : Mesure temps exécution

Données : tableau tps[6][1025] à remplir, correspondant au temps d'exécution des 6 test pour un nombre de bits entre 0 et 1024
Sorties : tableau tps[6][1025] rempli
pour *chaque test* ($i = 0$ *jusqu'à* 5) **faire**
  pour *pour un nombre de bits allant de 0 à 1024* ($j = 0$ *jusqu'à* 1024) **faire**
    Générer un nombre premier p de j bits;
    tps[i][j] $\leftarrow$ temps pour que le test i vérifie p;

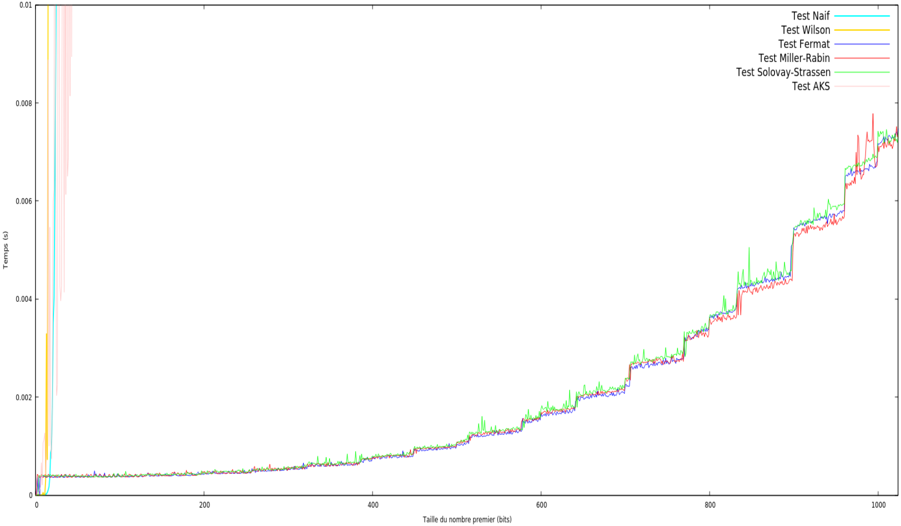

Figure – Temps d'exécution des tests en fonction de la taille en bits du nombre premier

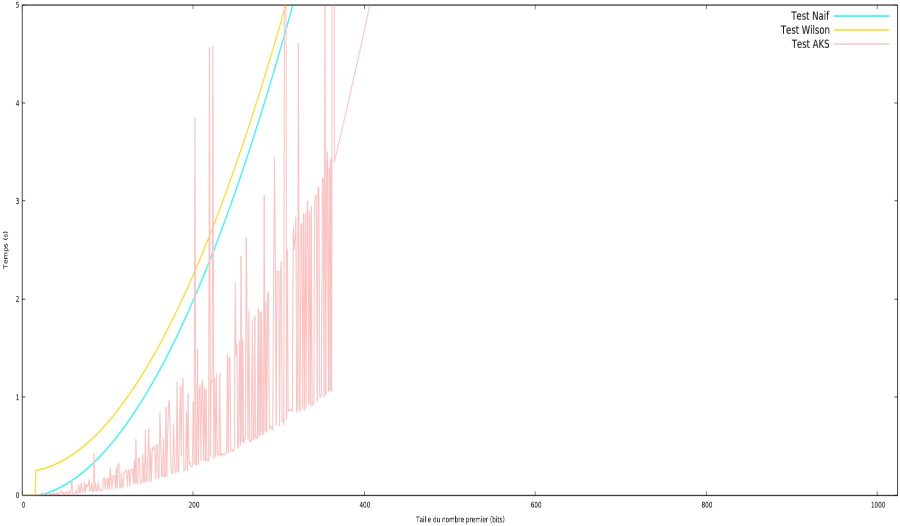

Figure – Temps d'exécution des tests déterministes

Comparatifs - Performances

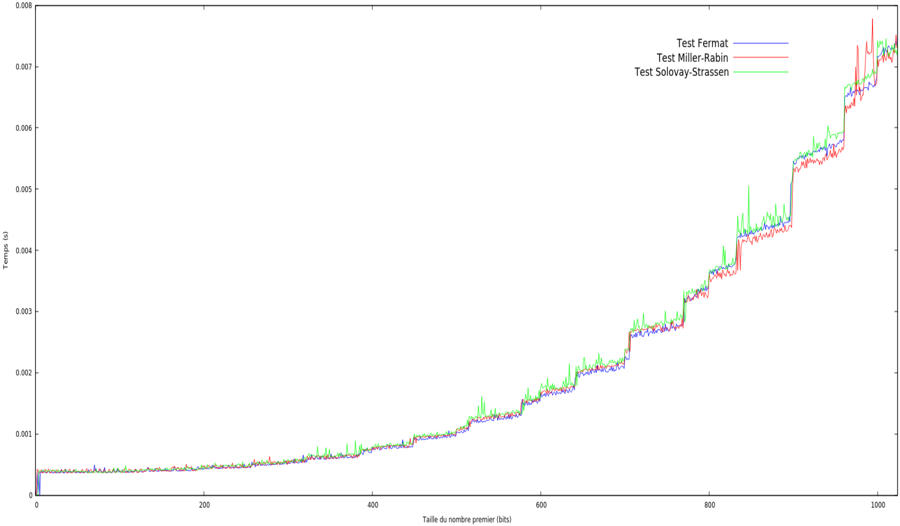

Figure – Temps d'exécution des tests probabilistes

Générateur optimal

Algorithme 9 : RPNG Optimal

Données : la taille t en bits de l'entier premier à générer et le tableau tps[6][1025] des résultats de mesure du temps d'exécution des 6 test pour un nombre de bits entre 0 et 1024

Sorties : un nombre premier de t bits

Trouver le test de primalité d'indice i (i entre 0 et 5) telle que tps[i][t] est minimale (c'est-à-dire le test le plus rapide pour vérifier t bits);

Générer un nombre premier de t bits avec le test de primalité d'indice i;

Plan

Conclusion

- Difficultés : complexités, preuves et AKS.
- Améliorations : répétitions tests probabilistes
- Perspectives : test de primalité optimal