DUMITRAȘCU CĂTĂLIN-NICOLAE
Facultatea de Automatică si Calculatoare
Calculatoare si Tehnologia Informației
Grupa 323, Seria CD

IDENTIFICAREA NUMERELOR PRIME

TEMA – ANALIZA ALGORITMILOR

Dată predare: 14 DEC 2017

Cuprins

1. Introducere 3
2. Algoritmi 4
  2.1. Solovay-Strassen 4
  2.2. Frobenius 5
  2.3. Fermat.....6
3. Comparatie 7
4. Generarea de numere prime 9
5. Sistem de criptare RSA 10
5. Concluzie 11
6. REFERINTE 12

1. Introducere[1][2]

Un test pentru identificare numerelor prime este un algoritm care determina dacă un număr dat ca input este prim sau nu. Printre alte câmpuri ale matematici, aceste teste se folosesc in principal in criptografie. Spre deosebire de factorizare, care este o problemă dificilă de calculat, identificarea numerelor prime este, prin comparație, mai ușoară, timpul de rulare fiind polinomial in concordanță cu inputul.

Aceste teste sunt important atât in teorie cat si in practică, e.g. pentru generarea de chei RSA si pentru verificarea dacă un parametru dintr-un sistem cu public-key (cum ar fi sistemul folosit de bănci pentru a gestiona conturile clienților) care ar trebui să fie prim este într-adevăr prim. Identificarea numerelor prime ar trebui să fie rapidă si ușoară, iar cel mai bun motiv ar fi ca mărimea cheilor din criptosisteme public-key, cum ar fi RSA, se va mări considerabil in următoarele decade, dacă se vor împlini așteptările legate de calculatoarele cuantice. Astfel, pentru a păstra sistemele in siguranță, testele nu ar trebui să ia mult timp sau spațiu.

Pentru aceasta temă, am ales să analizez algoritmi Solovay-Strassen, Frobenius si Fermat. Voi analiza complexitatea lor in diferite cazuri, le voi implementa într-un limbaj de programare si le voi compara unul împotriva celuilalt, cu avantaje si dezavantaje.

Disclaimer: In cercetarea pe care am făcut-o pentru aceasta tema, am întâmpinat câteva noțiuni pe care, in acest moment, nu prea le înțeleg. De cele mai multe probleme am dat la Frobenius, si am decis să prezint sumar caracteristicile acestuia, înlocuind comparația propriu-zisa cu algoritmul lui Fermat. Mulțumesc de înțelegere si sper ca aceasta să nu constituie o problema la notare.

2. Algoritmi[3][4][5][6]

2.1. Solovay-Strassen

Dezvoltat de către Robert M. Solovay si Volker Strassen, acesta este un test probabilistic care determina dacă un număr este *compus* sau *probabil prim*. Deși a fost in mare parte surclasat de testul Bailliei-PSW si de testul Miller-Rabin, acesta are o importanta istorica in arătarea fezabilității criptosistemului RSA.

Euler a demonstrat ca pentru orice număr prim *p* si orice număr întreg *a*:

$$a^{(p-1)/2} = \left(\frac{a}{p}\right) (mod\ p).$$

Fiind dat orice număr impar *n*, putem vedea dacă egalitatea

$$a^{(n-1)/2} = \left(\frac{a}{n}\right) (mod\ n);$$

tine sau nu pentru diferite valori ale bazei *a*, știind ca *a* si *n* sunt prime intre ele. Astfel, dacă alegem valori random pentru *a* si verificăm, odată ce am dat de un *a* pentru care nu este adevărată afirmația, putem deduce ca *n* nu este prim. Baza *a* se numește *Euler liar* dacă congruența este adevărată dar *n* este un număr compus. Aceasta este si problema algoritmului. In timp ce nu se întâmplă vreodată ca un număr *n* care este prim să fie introdus si algoritmul să returneze *compus*, este posibil ca un număr compus să fie trecut ca un număr prim. Eroarea este posibilă doar dacă *n* si *a* sunt prime intre ele.

In teoria numerelor, un număr întreg q este numit reziduu cuadratic modulo *n* dacă este congruent cu un pătrat perfect modulo *n*; dacă exista un număr întreg *x* astfel încât: $x^2 = q (mod\ n)$. *să* consideram cazul când *n* este un număr impar compus, produs a cel puțin doua numere prime. Fie $n = p^k m$, unde p este prim, k > 0 impar, si (p,m) = 1.

a) Orice număr $a < n$ poate să fie descompus in mod unic in $a = \langle \alpha, c \rangle$, unde $\alpha = a (mod\ p^k)$ si $c = a\ (mod\ m)$.

b) Sunt exact ½(p – 1) numere intre 0 si p care sunt reziduuri cuadratice modulo p.

Fie $0 < \alpha, \beta < p$, $0 < c < m$ cu α reziduu cuadratic modulo p si β non-reziduu. Evident:

$$\langle \alpha,c \rangle^{(n-1)/2} = \langle \beta,c \rangle^{(n-1)/2} = c^{(n-1)/2} \ (\text{mod } m)$$

Dacă $\langle \alpha,c \rangle^{(n-1)/2} \neq \langle \beta,c \rangle^{(n-1)/2} \ (\text{mod } n)$ atunci una nu aparține {1, -1}, astfel demonstrând ca *n* este compus. Altfel, ori:

$$\left(\frac{\langle \alpha,c \rangle}{n}\right) \neq \langle \alpha,c \rangle^{(n-1)/2} \ (\text{mod } n), \text{ ori}$$

$$\left(\frac{\langle \beta,c \rangle}{n}\right) \neq \langle \beta,c \rangle^{(n-1)/2} \ (\text{mod } n).$$

Deci nu se poate ca, simultan, un reziduu cuadratic si un non-reziduu modulo *p* să satisfacă ecuația:

$$a^{(n-1)/2} = \left(\frac{a}{n}\right) (mod\ n).$$

Așadar, cu o probabilitate de cel puțin 1/2 , când n este compus, algoritmul va fi corect.

Pseudocod:

Algoritm pentru Jacobi
1: //se omite cazul de baza
2: if a>n then
3: return **(a mod n)/n**
4: else
5: return **$(-1)^{((a-1)/2)((n-1)/2)}(a/n)$**

Algoritm pentru Solovay-Strassen:
1: Se alege un număr random a < n
2: if cmmdc**(a, n) > 1** then
3: return COMPUS
4: Se calculează **$a^{(n-1)/2}$** si **a/n** cu algoritmul pentru Jacobi
5: if **$(a/n) \neq a^{(n-1)/2}$** then
6: return COMPUS
7: else
8: return PRIM

2.2. Frobenius

Testul cuadratic Frobenius este un test probabilistic care determină dacă un număr este probabil prim. Testul folosește concepte din polinoame cuadratice si Algortimul Frobenius. A se nota ca mai exista si alt test Frobenius, însă acesta de fata restricționează ce polinoame pot fi folosite pe baza inputului dat. Este posibil ca un număr compus să treacă de acest test, acela numindu-se pseudoprim Frobenius.

Pseudocod:

Fie *n* un număr întreg pozitiv impar, iar ($b^2 + 4c \mid n) = -1$ si $(-c \mid n) = 1$. Fie B = 50000.
1: Se verifica dacă un număr prim mai mic sau egal cu min(**B, sqrt(n)**) îl divide pe n. Dacă da, stop, n este compus.
2: Se verifica dacă **sqrt(n)** aparține lui Z. Dacă da, stop, n este compus.
3: Se calculează **$x^{(n+1)/2}$ mod(n,$x^2 - bx - c$)**. Dacă nu aparține lui Z/nZ, stop, n este compus.
4: Se calculează **$x^{(n+1)}$ mod(n,$x^2 - bx - c$).** Dacă nu este congruent cu -*c*, stop, n este compus.
5: Fie **$n^2 - 1 = 2^r s$**, cu s număr impar. Dacă **x^s** nu este congruent cu **1 mod(n,$x^2 - bx - c$)**, si **x^{2js}** nu este congruent cu **-1 mod(n,$x^2 - bx - c$)**, pentru toți **0 <= j <= r – 2**, stop, *n* este compus.
Dacă algoritmul nu se oprește in pașii 1-5, atunci n este probabil prim.

2.3. Fermat

Testul Fermat de identificare a numerelor prime este un test probabilistic care determină dacă un număr este cel mai probabil prim. Mica teorema a lui Fermat spune ca dacă *p* este prim atunci:

$$a^{(p-1)} = \left(\frac{a}{p}\right) (mod\ p).$$

Dacă vrem să testam un număr *p*, putem alegem valori random ale lui a, nedivizibile de *p*, si vedem dacă se respectă relația data. Dacă egalitatea nu se respecta pentru o valoare a lui *a*, atunci *p* este compus. Este puțin probabil să fie respectată congruenta pentru un *a* random dacă *p* este compus.

O prima problema a acestui algoritm este faptul ca exista o infinitate de numere pseudoprime Fermat, care alcătuiesc cea mai importanta clasa de pseudoprime care provin din această teoremă. Dacă un număr întreg compus *x* divide $(a^{x-1}-1)$, aceste se numește pseudoprim Fermat pentru baza *a*. Cel mai mic pseudoprim pentru baza 2 este 341, fiind egal cu 11 * 31. Aceste trece de test întrucât $2^{340} = 1\ (mod\ 341)$.

O altă problemă mai serioasa este faptul că există o infinitate de numere Carmichael. Acestea sunt numere *n* pentru care toate valorile lui *a* cu **cmmdc(a, n) = 1**, sunt Fermat liars. Deși aceste numere sunt mult mai rare decât numerele prime, sunt destule astfel încât algoritmul in forma această să nu fie des folosit. Aceste numere vor trece mereu testul, indiferent de baza, astfel numărul de iterații este irelevant.

Pseudocod:

1: Se alege un număr *k* de iterații
2: Se repeta de *k* ori pașii 3-5:
3: Se alege un număr a, 2 <= a <= n – 2
4: if $\mathbf{a^{n-1} \neq 1\ (mod\ n)}$ then
5: return COMPUS
6: Dacă nu se întoarce niciodată COMPUS, atunci n este cel mai probabil prim.

3. Comparație[7]

Următorul tabel prezinta rezultatele in timp pentru diverse numere pentru algoritmii Solovay-Strassen si Fermat. Timpul dat este in milisecunde, iar algoritmi au fost implementați in Java. Din păcate, pentru Solovay-Strassen nu am putut folosi numere extrem de mari, întrucât nu am reușit să calculez simbolul Jacobi folosind clasa BigInteger din Java.

Număr	Timp S-S	Timp F
3	1.102708	2.867105
41	1.576442	2.872557
68	0.001925	3.153527
15485863	1.671061	3.169244
15485862	0.002245	3.440911
141650939	1.751567	3.162828
593441861	1.772415	2.930612
693441255	1.738417	3.307162
941083987	2.016179	3.016891

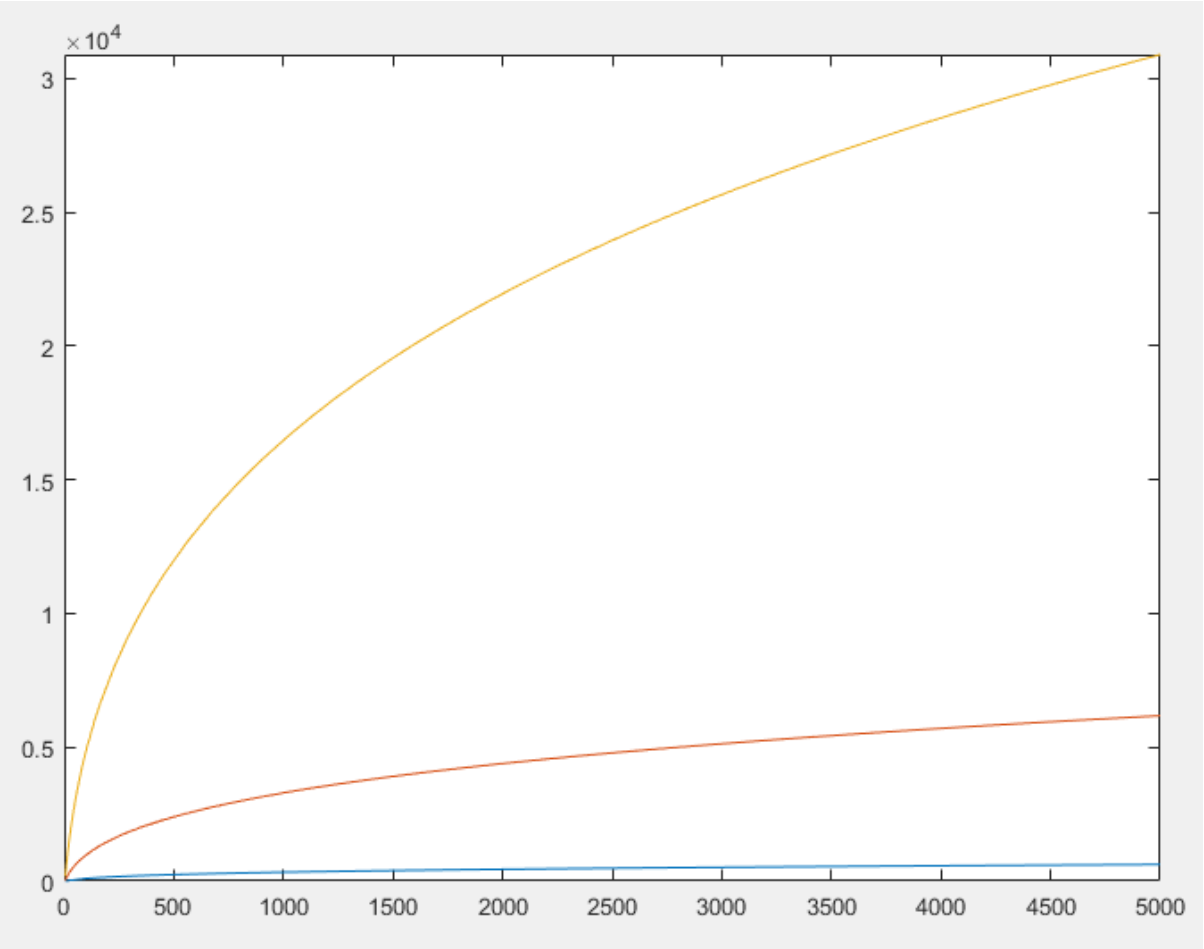
Timpul dat este exprimat in milisecunde.

Chiar si pentru aceste numere se poate observa o diferență de timp intre cele doua metode de a verifica numere prime. Cel mai clar se poate vedea aceasta diferență in testarea cu numere pare, unde Solovay-Strassen este net superior, excluzând imediat acel rezultat.

In ceea ce privește complexitatea temporala, algoritmul Fermat are o complexitate de $O(k(\log n)^3)$, unde k este numărul de iterații. In principiu, fiecare număr n este reprezentat in $\log n$ biți, deci înmulțirea se va face într-un timp $(\log n)^2$. Dar, pentru fiecare va trebui să facem si operația mod. De exemplu, dacă am vrea $a^{19} \bmod n$, putem face următoarele înmulțiri mod n: a.a, $a^2.a^2$, $a^4.a^4$, $a^{16}.a^2$, si $a^{18}.a$. Este astfel evident ca operația se va putea efectua in timp exponențial, având in esență o complexitate $O((\log n)^3)$. Se poate, folosind Fast Fourier Transformation, să reducă complexitatea algoritmului la $O(k(\log n)^2(\log\log n)(\log\log\log n))$.

Intr-un mod asemănător, Solovay-Strassen are o complexitate temporala de $O(k(\log n)^3)$, unde k este numărul de iterații. Aceasta complexitate rezultă din parți diferite ale algoritmului: găsirea cmmdc, calcularea simbolului lui Jacobi, si, in final, calcularea puterilor lui a. Respectiv: $O((\log n)^2) + O((\log n)^2) + O((\log n)^3)$.

In următorul grafic se poate observa cum funcția $k(\log n)^3$ creste in funcție de diferite numere de iterații:



De sus in jos liniile reprezintă: $50*(\log n)^3$, $10*(\log n)^3$ si, respectiv, $1*(\log n)^3$. Astfel, numărul de iterații este extrem de important, când vine vorba de timpul de execuție al algoritmilor.

4. Generarea de numere prime[7]

Numere prime random sunt des folosite in criptosisteme, cum ar fi cel RSA, pe care il vom considera mai târziu.

Pentru orice număr pozitiv întreg *x*, se definește $\Pi(x)$ ca numărul de numere prime $p \leq x$. Din *teorema numerelor prime* știm ca:

$$\frac{x}{\ln(x)} < \Pi(x) < 1.26\,\frac{x}{\ln(x)}.$$

Ca o consecință, să presupunem ca vrem să generam un număr prim random într-o mulțime oarecare {m + 1, ..., 2m}. Numărul de numere prime in acest set este:

$$\Pi(2m) - \Pi(m) > \frac{2m}{\ln(2m)} - 1.26\,\frac{m}{\ln(m)} = \Omega\left(\frac{m}{\log(m)}\right).$$

Deci exista o constanta oarecare $c > 0$, astfel încât, pentru un *m* suficient de mare:

$$\frac{2m}{\ln(2m)} - 1.26\,\frac{m}{\ln(m)} \geq c\left(\frac{m}{\log(m)}\right).$$

Probabilitate ca un număr random din intervalul dat să fie prim este astfel $\Omega\left(\frac{1}{\log(m)}\right)$ si ne permite să facem algoritmi pentru generarea random de numere prime, cum ar fi următorul:

1: **loop**
2: Se ia un *a* random din *{m* + 1, . . . , 2*m}*
3: **for** *i* = 1 to *k* **do**
4: *Solovay − Strassen*(*a*)
5: **end for**
6: **if** *Solovay − Strassen*(*a*) =“prim”*k* times **then**
7: return *a*
8: **end if**
9: **end loop**

Numărul așteptat de iterații al buclei este *O(log m)*. Timpul așteptat de rulare este *O(k(log m)⁴)*. Când algoritmul se oprește, numărul găsit este un prim din {m + 1, ..., 2m} cu o probabilitate $\geq 1 - \frac{1}{2^k}$.

5. Sistem de criptare RSA

Sistemul de criptare RSA este un sistem de criptare cu cheie publica creat de Rivest, Shamir si Adleman in 1978.

Sa presupunem ca avem 2 jucători, Ana si Andrei. Andrei vrea să ii trimită un mesaj Anei. Mesajul trebuie să rămână cu orice preț secret, iar căile de comunicare dintre ei nu sunt sigure. O soluție ar fi să folosească o cheie secretă, dar dacă aceasta trebuie să fie împărtășită pe aceleași cai, poate si aceasta să fie furata. Dacă folosim un RSA, cheile de criptare si decriptare sunt diferite si aceasta problema nu se întâmplă.

Conceptul:

1. Ana alege doua numere prime foarte mari, *p* si *q* ($\approx$ 500 bits long).
2. Ana setează *n* = *pq* si alege un e random din $\mathbb{Z}^*_{\varphi(n)}$, unde $\varphi(n) = \varphi(p)\varphi(q) = (p-1)(q-1)$, funcția lui Euler.
3. Ana calculează *d* din $\mathbb{Z}^*_{\varphi(n)}$ astfel încât d * e mod $\varphi(n)$ = 1.
4. Cheia publica *(n, e)* este cunoscuta de toata lumea si poate fi folosita pentru criptare, iar cheia privata (n, d) este folosita pentru decriptare si este cunoscuta doar de Ana.

Trebuie să acest protocol să fie sigur. Un posibil atac ar fi acela de a se calcula $\varphi(n)$ din *n*, si apoi de a se calcula *d* din *n*. Întrebarea este: se poate calcula $\varphi(n)$ din n?

Se poate arata ca acest calcul este la fel de greu ca factorizarea lui *n*, unde *n* = *pq*, si este acceptat in general ca aceasta nu se poate realiza într-un timp rezonabil, adică polinomial, cu tehnologia curenta. Dar, cum am precizat si mai devreme in aceasta lucrare, acest task este, prin comparație, ușor pentru calculatoarele cuantice: folosind algoritmul lui Shor, se pot afla factorii primi ai unui număr întreg in timp polinomial. Așadar, dacă se va putea construi un calculator cuantic la scala mare, criptosistemul RSA va fi spart.

6. Concluzie

In concluzie, algoritmii pentru identificarea numerelor prime sunt extrem de importante, nu numai in matematica pură, dar mai ales in sistemele pe care le folosim in fiecare zi. As dori să precizez ca, deși in „jocul" anterior, Ana folosea un număr de aproximativ 500 de biți lung, in zilele noastre, minimul recomandat ar fi de 2048. Agenția NSA spune ca un număr pe 1024 se poate sparge într-un timp *rezonabil*, tehnologia de astăzi încă nu poate atinge 2048-bit, dacă, desigur, cheia privată a fost aleasă corespunzător.

Legat de algoritmii prezentați, se poate observa cât de mult am avansat in acest domeniu. Îmi dau seama de cate vulnerabilități are algoritmul lui Fermat, si de ce nu este foarte folosit in forma lui pura, dar crearea lui semnifică un punct foarte important in istoria științelor. Intre cele doua, Solovay-Strassen este net superior, dar pentru operatii mai uzuale, sau, de ce nu, pentru un student care începe să studieze, algoritmul lui Fermat este mai folositor, drept dovadă ca eu nu am putut, într-un timp rezonabil *(polinomial :D)*, să fac ca algoritmul S-S să funcționeze in Java pe numere foarte mari.

7. REFERINTE:

[1] https://www.revolvy.com/main/index.php?s=Primality%20test&uid=1575

[2] A Simplified Quadratic Frobenius Primality Test by Martin Seysen

[3] Primality Tests Based on Fermat's Little Theorem, Manindra Agrawal,IIT Kanpur ICDCN, IIT Guwahati

[4]https://www.revolvy.com/main/index.php?s=Solovay%E2%80%93Strassen%20primality%20test

[5] http://www.geeksforgeeks.org/primality-test-set-4-solovay-strassen/

[6] https://www.revolvy.com/main/index.php?s=Fermat%20primality%20test&uid=1575

[7] University of Tokyo: Advanced Algorithms, summer 2010, Francois Le Gall.