Temă la Analiza Algoritmilor

Dragne Lavinia-Ștefana - 324CA

[1] Universitatea Politehnica București, Facultatea de Automatica și Calculatoare
[2] lavinia.dragne@stud.acs.upb.ro

Abstract. În tema prezentată am ales să analizez comparativ 2 algoritmi probabilistici de identificare a numerelor prime și anume, Fermat și Miller-Rabin.

Keywords: Fermat · Miller-Rabin · Numere prime.

1 Introducere

1.1 Descrierea problemei rezolvate

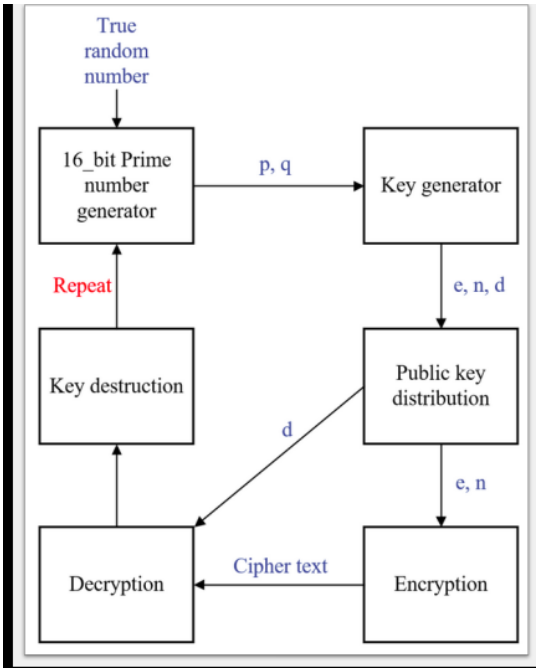
Fiind dat un set de date de intrare, problema își propune să identifice numerele prime din acel set.

1.2 Exemple de aplicații practice pentru problema aleasă

Algoritmii de identificare ai numerelor prime mari stau la baza schemelor criptografice. Cu un criptosistem cu cheie publică, putem cripta mesajele trimise între două persoane, astfel încât cineva care interceptează mesajele criptate să nu le poată decoda. Criptosistemul cu cheie publică RSA se bazează pe diferența contrastantă între ușurința de a găsi numere prime mari și dificultatea de a calcula produsul a 2 două numere de acest tip. Din fericire, aceste numere nu sunt prea rare, astfel încât este posibil să testăm numere întregi aleatorii de dimensiune adecvată până când găsim unul prim.

Numerele prime sunt, de asemenea, utilizate în:

- calcul pentru sume de control;
- tabele hash;
- generatoare de numere pseudorandom.



Mecanismul de criptare RSA [7]

1.3 Specificarea soluțiilor alese

Am ales pentru a rezolva această problema să analizez comparativ 2 algoritmi de identificare a numerelor prime: Fermat și Miller-Rabin. Ei sunt algoritmi probabiliști, de tip Monte Carlo. Aceștia au următoarele proprietăți: [2]

- Găsesc o soluție garantat corectă doar după un timp infinit de rezolvare.
- Probabilitatea ca soluția să fie corectă crește o dată cu timpul de rezolvare.
- Soluția găsită într-un timp acceptabil este aproape corectă.

Astfel, propabilitatea să afirmăm că un număr mare este într-adevăr prim sau nu, crește odată cu timpul de execuție.

1.4 Criteriile de evaluare pentru soluția propusă

Criteriile pe baza cărora voi compara cei 2 algoritmi vor fi complexitatea implementării cât și eficiența fiecăruia din punct de vedere al timpului de rulare, raportați la probabilitatea obținerii unui răspuns corect.
Pentru a testa corectitudinea algoritmilor, voi propune un set de teste cu numere specifice fiecărui algoritm și particularităților sale, ținând cont și de cazurile speciale, precum numerele Carmichael sau cele tari pseudoprime. Apoi, pentru a testa eficiența fiecăruia dintre ei, voi observa dacă probabilitatea de eroare

scade, pentru același set de date, și pentru ce timpi de rulare se întâmplă asta. Testele sunt construite astfel:

- Testele de la 1-5: contin un singur numar ce trebuie testat si se afla in intervalul [1, 1000000].
- Testele de la 6-10: numarul de numere testate este maxim 20, dintre care cel mult 9 sunt generate random si restul sunt sigur prime, fiind obtinute din Ciurul lui Eratostene. Numerele sunt din intervalul [1, 100000].
- Testele de la 11-15: sunt testate maxim 25000 numere, din intervalul [10000, 1000000], dintre care cel mult 5000 generate random.
- Testele de la 16-18: numarul de numere testate ajunge la 600000, maxim 10000 generate random, fiind din intervalul [100000, 10000000].
- Testele de la 19-25: sunt testate, pe rand, numerele de tip Carmichael, mai mici ca 10000, acestea reprezentand un caz special pentru algoritmul Fermat.

2 Prezentarea soluțiilor

2.1 Descrierea modului în care funcționează algoritmii aleși

Algoritmul lui Fermat

Are la bază implicația inversă (care nu este întotdeauna adevărată) a Micii Teoreme a lui Fermat și anume:

Dacă $a^{(n-1)} \equiv 1 \pmod{n}$ pentru un a cu $a \not\equiv 0 \pmod{n}$, atunci n este prim.

Dacă pentru un anumit a ∈ [1, n), relația nu este verificată, atunci n este compus. Deci procentul de a afirma, cu siguranță, că n este prim, crește odată cu testarea relației anterioare, pentru cât mai multe numere naturale a. Totuși, există niște numere, numite numere Carmichael (sau absolut pseudoprime Fermat), pentru care, pentru orice valoare a lui a din intervalul [1,n), relația este satisfăcută și totuși numerele Carmichael sunt numere compuse. De aceea, nu funcționează întotdeauna folosirea unui număr foarte mare de iterații, dacă n este de tip Carmichael.

Ideea implementării: Se alege un numar random a și se calculează $a^{(n-1)} \pmod{n}$. Dacă $a^{(n-1)} \equiv 1 \pmod{n}$, n este posibil să fie prim și repetăm testul pentru un alt a, în caz contrar returnăm că n este compus. [3]

Mai jos am prezentat un pseudocod pentru implementarea algoritmului:

```
INPUT: un numar n > 2, impar, un parametru k > 1 (numarul de iteratii)
OUTPUT: un raspuns privind primalitatea lui n
        1. Pentru i de la 1 la k
                a) Alege aleator un intreg a cu 2 <= a <= (n - 2)
                b) Calculeaza r = a^(n-1) mod n
                c) Daca r!=1 atunci returneaza "numar compus" si se opreste
        2. Returneaza "numar prim"
```

Pseudocod Fermat [5]

Algoritmul Miller - Rabin

Acest algoritm este cunoscut și sub numele de testul de tare pseudo-primalitate. Se bazează pe următoarea teoremă:

Dacă n este număr prim și b număr întreg nenul, atunci una din următoarele afirmații este adevărată:

- sau $b^t \equiv 1 \pmod n$
- sau există r $\in \{0, ..., s - 1\}$ astfel încât: $b^{2^r t} \equiv -1 \pmod n$

Dacă un număr n îndeplinește una din condiții, atunci este numit pseudoprim tare în raport cu b. În practică, se calculează succesiv b^t, b^{2t}, b^{4t}, ... până se obține valoarea 1. Dacă n este prim, atunci ultima valoare diferită de 1 trebuie să fie -1.

Ideea implementării: Pentru a verifică dacă un număr impar n este prim sau nu, folosind acest test, scriem întâi n - 1 = 2^st unde t este impar și alegem la întamplare o baza b, 1 < b < n. Dacă n nu trece testul Miller pentru această bază, atunci el este compus. Altfel, n este prim sau "tare pseudoprim" cu baza b și repetăm algoritmul pentru o nouă bază. [2]

Mai jos am prezentat un pseudocod pentru implementarea algoritmului:

```
INPUT: un numar n > 2 impar, un parametru k > 1 (numarul de iteratii)
OUTPUT: un raspuns privind primalitatea lui n
        1. Se descompune (n - 1) = (2^s) * t, cu t impar
        2. Pentru i de la 1 la k
                2.1. Se alege aleator un numar intreg a, cu 2 <= a <= (n - 2)
                2.2. b = a^m (mod n)
                2.3. Daca b = 1 (mod n) returneaza "n este prim" si se opreste
                2.4. Pentru i = 0 : (s - 1)
                        a) Daca b = -1 (mod n) atunci returneaza " n este prim" si se opreste
                                               altfel b = b^2 (mod n)
        3. Returneaza "n nu este prim" si se opreste
```

Pseudocod Miller-Rabin [5]

2.2 Analiză complexității soluțiilor

Algoritmul Fermat

Pentru a calcula un $a^{(n-1)}$ se poate folosi pătratul și multiplicarea repetată, care necesită una sau două înmulțiri modulare pe bit de n. Deoarece numărul de biți este logn, obținem un factor de logn. În general, pentru calcularea eficientă a lui a^x se folosește scrierea în baza 2 a lui x. Se va calcula $a^n = a^{n/2} * a^{n/2}$, pentru n par și a^n = a * $a^{(n-1)/2} * a^{(n-1)/2}$, pentru n impar. O înmulțire modulară poate fi făcută prin adăugarea unor copii în biți ale unui factor. Fiecare adăugare folosește cel mult de două ori mai mulți biți decât are n, deci vor fi O (logn) pași. Este astfel evident că această metodă de exponențiere modulară

se poate face în timp polinomial și complexitatea este în esență O(logn x log^2n) = O(log^3n), pentru fiecare valoare a lui a, testată. Concluzionăm că complexitatea în timp a algoritmului Fermat este O(k x log^3n) = O(klog^3n), unde k este numărul de numere a pe care le testăm (numărul de iterații dat în input). [1], [8]

Algoritmul Miller-Rabin

Complexitatea algoritmului Miller-Rabin poate fi determinată din pseudocodul său. Algoritmul se bazează, întocmai ca și Fermat, pe exponențierea rapidă pentru a^m mod n. Vedem că în acest algoritm, inmulțirea a două numere este întotdeauna O(logn x logn) sau O(log^2n) complexitate temporală, unde n poate fi reprezentat pe logn biți. Pentru orice a putem compune a^m mod n din O(logn) înmulțiri. Putem să-l spargem pe m în puteri ale lui 2 și apoi să continuăm algoritmul. De exemplu, dacă dorim să ridicăm, să zicem, la puterea 19 modulo n, trebuie să realizăm doar următoarele modulo n înmulțiri, și anume: a×a,a^2×a^2, a^4×a^4, a^{16}×a^2 și a^{18}×a.

Este astfel evident că această metodă de exponențiere modulară se poate face în timp polinomial și complexitatea este în esență O(logn x log^2n) = O(log^3n), pentru fiecare valoare a lui a, testată. Această metodă de înmulțire prin creșterea la puteri de 2 se numește exponențiere modulară prin pătrare repetată.

Deoarece această exponențiere are complexitatea dominantă în algoritmul de mai sus, concluzionăm că complexitatea în timp a algoritmului Miller-Rabin este O(k x log^3n) = O(klog^3n), unde k este numărul de baze pe care le testăm (numărul de iterații dat în input). [8]

2.3 Prezentarea principalelor avantaje și dezavantaje pentru soluțiile luate în cosiderare

Algoritmul Fermat:

Avantaje:

- Principalul avantaj al acestuia este complexitatea temporală comparativ mai bună de O(logn) cu cea de O(n) sau O($\sqrt{n}$) a algoritmilor clasici de identificare a numerelor prime.
- Se poate demonstra că, în cazul general, probabilitatea de eroare a testului Fermat este 0.5. Așadar avem șanse de cel puțin 50% să depistăm dacă numărul este compus. [6] Șansele cresc odată cu creșterea numărului k de iterații, dar aceasta influențează și performanță temporală a algoritmului.

Dezavantaje:

- Principalul dezavantaj al testului probabilist al lui Fermat este prezența numerelor de tip Carmichael (numere absolut pseudoprime Fermat), pentru care probabilitatea de eroare este foarte mare. Deși numerele Carmichael sunt compuse, algoritmul Fermat le va returna ca fiind prime. Deci în cazul

lor nu funcționează folosirea unui număr cât mai mare de iterații, pentru determinarea primalității corecte. Totuși aceste numere sunt rare (există 250.000 de numere Carmichael mai mici decât 10^{16}) și doar 7 mai mici ca 10000: 561, 1105, 1729, 2465, 2821, 6601, 8911.
– Nu putem stabili cu exactitate care este limita de calcul – nu se poate estima pentru un număr compus n numărul de numere x, x ∈ [2, n] pentru care nu se verifică ecuația.

Algoritmul Miller-Rabin:

Avantaje:

– Elimină principalul dezavantaj al testului Fermat de primalitate și anume, prezența numerelor Carmichael.
– Probabilitatea de eroare scade de la 1/2 la 1/4, deci șansele ca primalitatea obținută să fie cea corectă sunt de minim 75% spre deosebire de 50% la Fermat, deci implicit este nevoie și de un număr mai mic de iterații, astfel testul Miller-Rabin este mai stabil decât testul Fermat.

Dezavantaje:

– La fel ca și Fermat, și Miller-Rabin are numere pseudoprime pentru care, deși sunt compuse, algoritmul le declara "probabil prime". Densitatea apariției lor este totuși de 4 ori mai mică, pentru un n dat, decât în cazul testului Fermat. [4]

3 Evaluare

3.1 Descrierea modalității de construire a setului de teste folosite pentru validare

Am creat câte un test de input cu un vector de numere ce urmează a fi testate și un fișier de ref pe care îl folosesc pentru validarea rezultatelor. Testele de input sunt în număr de 25 și sunt folosite pentru ambii algoritmi.

Primele 18 teste sunt generate în același mod: Se amestecă m numere generate random din intervalul [p, n] cu (max_size - m) numere prime generate cu ajutorul Ciurului lui Eratostene. Variabilă max_size reprezintă dimensiunea setului de date de la acel test, iar m, p și n sunt numere date ca și paramtri generatorului.

Numerele prime sunt adăugate pe poziții aleatoare în vector, pe baza unui factor "fact", generat random. Frecvența lor este crescută, "fact" înjumătățindu-se la fiecare nouă adăugare. De asemenea, numerele prime sunt alese atât de la începutul Ciurului lui Eratostene, cât și de la mijloc și final, variind astfel dimensiunea lor.

Testele sunt construite astfel:

– Testele de la 1-5: conțin un singur număr ce trebuie testat și se află in intervalul [1, 1000000].
  a) Testul 1 și testul 2 conțin numere în intervalul [1, 100].
  b) Testul 3 conține un număr în intervalul [500, 10000].
  c) Testul 4 conține un număr în intervalul [5000, 100000].
  d) Testul 5 conține un număr în intervalul [50000, 1000000].

– Testele de la 6-10: numarul de numere testate este maxim 20, dintre care cel mult 9 sunt generate random și restul sunt sigur prime, fiind obținute din Ciurul lui Eratostene. Numerele sunt din intervalul [1, 100000].
  a) Testele 6 și 7 conțin 5 numere, dintre care 3 random și restul prime în intervalul [1, 100].
  b) Testul 8 conține 10 numere, dintre care 5 random și restul prime în intervalul [10, 1000].
  c) Testul 9 conține 15 numere, dintre care 7 random și restul prime în intervalul [100, 10000].
  d) Testul 10 conține 20 numere, dintre care 9 random și restul prime în intervalul [100, 100000].

– Testele de la 11-15: sunt testate maxim 25000 numere, din intervalul [10000, 1000000], dintre care cel mult 5000 generate random.
  a) Testele 11 și 12 conțin 25 numere, dintre care 5 random și restul prime în intervalul [10000, 1000000].
  b) Testul 13 conține 250 numere, dintre care 50 random și restul prime în intervalul [20000, 990000].
  c) Testul 14 conține 2500 numere, dintre care 500 random și restul prime în intervalul [30000, 980000].
  d) Testul 15 conține 25000 numere, dintre care 5000 random și restul prime în intervalul [40000, 970000].

– Testele de la 16-18: numărul de numere testate ajunge la 600000, maxim 10000 generate random, fiind din intervalul [100000, 10000000].
  a) Testul 16 conține 100000 numere, dintre care 10000 random și restul prime în intervalul [500000, 1000000].
  b) Testul 17 conține 500000 numere, dintre care 10000 random și restul prime în intervalul [100000, 1000000].
  c) Testul 18 conține 600000 numere, dintre care 10000 random și restul prime în intervalul [100000, 1000000].

– Testele de la 19-25: sunt testate, pe rand, numerele de tip Carmichael, mai mici ca 10000, acestea reprezentand un caz special pentru algoritmul Fermat.

3.2 Specificațiile sistemului de calcul pe care am rulat testele:

Procesor: Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz
Memorie: 16GB RAM

3.3 Ilustrarea, folosind grafice/tabele, a rezultatelor evaluării soluțiilor pe setul de teste.

Pentru generarea timpilor am folosit biblioteca <chrone>.

Nr. test	Timp Fermat (microsecunde)	Timp Miller-Rabin (microsecunde)	Diferenta de timp (Fermat - Miller-Rabin) (microsecunde)	Numarul de numere testat	Numarul de numere random	Limita superioara pentru cel mai mare numar testat
1.	7	4	3	1	0	100
2.	4	3	1	1	0	100
3.	6	5	1	1	0	10000
4.	1	1	0	1	0	100000
5.	7	7	0	1	0	1000000
6.	10	10	0	5	3	100
7.	6	5	1	5	3	100
8.	26	25	1	10	5	1000
9.	41	41	0	15	7	10000
10.	76	75	1	20	9	100000
11.	121	120	1	25	5	1000000
12.	169	135	34	25	5	1000000
13	1112	1085	27	250	50	1000000
14.	13673	13028	645	2500	500	1000000
15.	140054	137890	2164	25000	5000	1000000
16.	677578	675991	1587	100000	10000	1000000
17	3918602	3877472	41130	500000	10000	1000000
18.	4462095	4217520	244575	600000	10000	1000000
19.	4	2	2	1	0	10000
20.	5	2	3	1	0	10000
21.	3	2	1	1	0	10000
22.	2	2	0	1	0	10000
23.	4	2	2	1	0	10000
24.	3	2	1	1	0	10000
25.	9	2	7	1	0	10000

Limite teste

3.4 Prezentarea, succintă a valorilor obținute pe teste. Dacă apar valori neașteptate încercați să oferiți o explicație.

Prin cele 25 teste alese se poate observa corectitudinea algoritmilor. Am ales să folosesc 10 iterații, atât pentru algoritmul Fermat cât și pentru Miller-Rabin, pentru că acesta este cel mai mic număr care îmi permitea trecerea tuturor celor 25 teste. Dacă se scade numărul de iterații la 5, algoritmul Miller-Rabin tot va returna rezultate pozitive pe toate testele, dar Fermat va da fail pentru testele 17 și 18, acestea fiind și cele mai mari inputuri.

Se observă că toți timpii de rulare sunt mai mari pentru algoritmul Fermat, pentru un vector cu 600000 numere mai mici că 1000000, diferența între Fermat și Miller-Rabin ajungând la 244575 microsecunde, aproximativ 0.25 secunde.

Algoritmul lui Fermat are o limită de timp de maxim 4.46 secunde, în timp ce pentru același număr de iterații, algoritmul Miller-Rabin se execută în doar 4.21 secunde.

De asemenea, pentru primele 13 teste, ce conțin vectori de input de dimensiune maxim 25 elemente, diferențele dintre cei 2 algoritmi sunt mici, maxim 35 microsecunde. Deci, pentru puține numere testate algoritmii se comportă similar.

Timpii pot varia de la o rulare la alta, deoarece testele se bazează și pe o generare de numere random. De asemenea, pot varia în funcție de sistemul de calcul pe care se testează.

4 Concluzii

4.1 Precizați în urmă analizei făcute, cum ați aborda problema în practică; în ce situații ați opta pentru una din soluțiile alese.

Analizând în amănunt cei 2 algoritmi se ajunge la concluzia că deși ambii sunt de tip probabilistic și determină primalitatea unui număr cu un anumit procent de eroare, sunt superiori din punct de vedere al complexității temporale ($O(k log^3 n)$) (k fiind numărul de iterații) față de algoritmii clasici de determinare a numerelor prime, cei mai eficienți având o complexitatea de $O(\sqrt{n})$.

Deci, dacă este nevoie în practică de stabilirea primalității unui număr mare (cu peste 10 cifre), astfel de numere fiind folosite în criptografie, într-un timp cât mai scurt, asumându-se o anumită probabilitate de eroare (de cel mult 50%) se poate alege unul dintre cei 2 algoritmi analizați.

Algoritmul Fermat poate fi folosit mai ales în cazul verificării numerelor mai mici, până în 10000, existând doar 7 numere Carmichael pe care acesta va da sigur fail.

Algoritmul Miller-Rabin este superior lui Fermat, atât din punct de vedere al timpului de execuție (pentru testul Miller-Rabin este nevoie de un număr mai mic de iterații k pentru determinarea primalității, dar cu aceeași rată de eroare, față de cazul testului Fermat), cât și al corectitudinii (numărul de input-uri pe care Fermat funcționează este mai mic din cauza numerelor Carmichael).

Deci, eu aș alege să folosesc în practică algoritmul Miller-Rabin pentru a obține într-un timp cât mai scurt, un rezultat cu o rată de eroare cât mai mică.

References

1. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms
2. https://www.mta.ro/masterinfosec/files/resources/math.pdf
3. https://www.math.uaic.ro/~criptografie/curs_2020/slides_crypto3_2020_h.pdf
4. http://home.sandiego.edu/~ dhoffoss/teaching/cryptography/10-Rabin-Miller.pdf
5. http://math.ucv.ro/~dan/courses/carteAlg.pdf
6. http://campion.edu.ro/arhiva/www/arhiva2009/papers/paper34.pdf
7. https://www.mdpi.com/2079-9292/9/2/246/htm
8. https://www.cmi.ac.in/~ shreejit/primality.pdf
9. https://www.geeksforgeeks.org/primality-test-set-2-fermet-method/
10. https://www.geeksforgeeks.org/primality-test-set-3-miller-rabin/

Ultima accesare a referințelor am făcut-o în data de 17 decembrie 2020.