Identificarea numerelor prime

Subțirică Florin-Ioan, 322CD

22 decembrie 2022

Rezumat Scopul acestei lucrări este o comparație între doi algoritmi populari de identificare a numerelor prime - Fermat, respectiv Miller-Rabin - care reprezintă de fapt un test probabilistic pentru a determina dacă un număr este compus sau probabil prim.

Keywords: Fermat · Miller-Rabin · numere prime · numere compuse · numere Carmichael

1 Introducere

1.1 Descrierea problemei rezolvate

Un **număr natural** p > 1 se numește prim dacă : p | ab atunci p | a sau p | b, unde a, b sunt numere naturale. Aceasta este o proprietate esențială a numerelor prime, iar cele două definiții sunt echivalente pentru inelul ($\mathbb{Z}$, +, ·). Un număr prim are exact doi divizori pozitivi: numărul 1 și numărul în sine. Acești divizori sunt improprii. Un număr prim este deci nefactorizabil. Cel mai mic număr prim este 2; în afară de 2 toate numerele prime sunt numere impare. Există o infinitate de numere prime, fapt demonstrat de Euclid în Antichitate.

Opusul noțiunii de număr prim este cel de număr compus. Un număr compus este un număr întreg pozitiv care are cel puțin un divizor pozitiv în afară de 1 și el însuși. Prin definiție, orice număr întreg mai mare de 1 este fie număr prim, fie număr compus. Se poate scrie ca produs de numere prime mai mici ca el, fiind deci factorizabil. Este astfel un multiplu al altor numere de modul mai mic decât el, acestea putând fi chiar prime.

Pe lângă cele două tipuri de numere, mai există și un al treilea tip de numere, numerele Carmichael, numite după matematicianul american Robert Carmichael. În teoria numerelor, un număr Carmichael este un număr compus n, care în aritmetica modulară satisface relația de congruență:

$$b^{n-1} \equiv 1$$

1.2 Exemple de aplicații practice pentru problema aleasă

1. Folosim și ne bazăm în mod constant pe numere prime pentru securitatea era cibernetică;
2. Folosită în criptare și decriptare;
3. Acestea sunt utilizate în generarea de coduri de corectare a erorilor utilizate în telecomunicații. Ele asigură faptul că mesajul este trimis și primit cu corecție automată;
4. Numerele prime acționează ca bază pentru crearea algoritmilor de criptare cu cheie publică;
5. Sunt folosite pentru tabelele de dispersie (Hash Tables);
6. Ele sunt, de asemenea, utilizate pentru generarea de numere pseudoaleatoare;
7. Numerele prime sunt, de asemenea, utilizate în proiectarea mașinilor cu rotor. Un număr este fie prim, fie coprim față de numărul de pe alt rotor dintr-un rotor. Acest lucru ajută la generarea ciclului complet înainte de a repeta orice poziție posibilă a rotorului;
8. Acestea mai sunt folosite în calcul în sistemul de criptare RSA.

1.3 Specificarea soluțiilor alese

Pentru a rezolva problema dată, am ales următorii algoritmi:
1. Metoda Fermat

Mica Teoremă a lui Fermat apare pentru prima dată într-o scrisoare scrisă de Fermatin 1640. A fost afirmată fără dovezi, deși se speculează că demonstrația lui Fermat s-a bazat pe teorema binomială. La aproape o sută de ani după ce Fermat a afirmat această teoremă, Euler a publicat prima demonstrație în Proceedings of the St.Petersburg Academy în 1736.

Theorem 1. *(Fermat's Little Theorem) Fie p un număr prim și a orice număr întreg cu proprietatea (a, p) = 1. Atunci*

$$a^{p-1} \equiv 1(modp)$$

Proof. Începem prin a enumera p-1 elemente diferite de zero ale $\mathbb{Z}_p$

$$1, 2, 3, ..., p-2, p-1 \tag{1}$$

Înmulțind fiecare membru al lui (1) cu un a $\in \mathbb{Z}_p$ fix, diferit de zero, obținem o nouă listă:

$$1a, 2a, 3a, ..., (p-2)a, (p-1)a \tag{2}$$

Deoarece $\mathbb{Z}_p$ este închis la înmulțire, fiecare membru al lui (2) este în $\mathbb{Z}_p$. În plus, fiecare membru al (2) este distinct. Deoarece produsele din $\mathbb{Z}_p$ sunt comutative și asociative, putem forma produsul elementelor din fiecare listă și obținem congruența

$$(p-1)! \cdot a^{p-1} \equiv (p-1)!(modp)$$

În cele din urmă, înmulțirea cu inversul lui (p-1)! dă rezultatul dorit.

Putem folosi reciproca Micii Teoreme a lui Fermat nu pentru a testa primalitatea, ci mai degrabă compoziția. Fiind n ¿ 2 impar, dacă putem găsi o bază relativ primă la n pentru care $a^{n-1} \neq 1$ (mod n), atunci n este în mod necesar compus.

Definition 1. *Fie a și n numere întregi cu (a, n) = 1. Atunci n este un pseudoprim la baza a dacă n este compus, totuși încă mai avem $a^{n-1} \equiv 1$ (mod n).*

Existența pseudoprimelor înseamnă că reciproca Micii Teoremei lui Fermat nu este adevărată. S-ar spera că pentru o anumită bază a, există doar un număr limitat de pseudoprime, ceea ce nu este adevărat. De fapt, există o infinitate de pseudoprime pentru baza 2. Baza 2 nu este singura bază "tulburată" de pseudoprime; fiecare bază are asociate infinite de pseudoprime. Mai rău, există numere compuse care sunt pseudoprime pentru fiecare bază posibilă. Aceste numere compuse "supărătoare" au fost studiate de Carmichael și sunt numite după el.

Definition 2. *Fie a și n numere întregi. Atunci n este un număr Carmichael dacă n este compus și $a^{n-1} \equiv 1$ (mod n) pentru orice a proprietatea (a, n) = 1.*

În 1912, Carmichael a presupus că există infinit multe numere Carmichael. Optzeci de ani mai târziu, Alford, Granville și Pomerance au demonstrat acest lucru. Deși numerele Carmichael apar mai puțin frecvent decât numerele prime, inteligența lor încă oferă o cantitate infinită de probleme în testarea compoziției folosind Teorema Mică a lui Fermat.

2. Metoda Miller-Rabin

Testul de primalitate Miller-Rabin este un test de primalitate: un algoritm care determină dacă un anumit număr este probabil să fie prim, similar testului de primalitate Fermat. Gary L. Miller a descoperit-o în 1976; Versiunea lui Miller a testului este deterministă, dar corectitudinea sa se bazează pe ipoteza Riemann nedovedită. Michael O. Rabin l-a modificat pentru a obține un algoritm probabilistic necondiționat în 1980.

Theorem 2. *(Miller-Rabin primality test). Să presupunem $n = 2^s d + 1$, dacă putem găsi un a astfel încât $a^d \not\equiv 1 \pmod n$ și*

$$a^{2^r d} \not\equiv -1(mod n)$$

pentru toți r, $0 \leq r \leq$ s-1, atunci n nu e prim.

Proof. Să presupunem că n este prim și n ¿ 2. Rezultă că n-1 este par și îl putem scrie ca $2^s d$, unde s și d sunt numere întregi pozitive și d este impar. Foreach a în $\mathbb{Z}_p$, fie

$$a^d \equiv 1(mod n)$$

sau

$$a^{2^r d} \not\equiv -1(mod n)$$

pentru câteva valori ale lui r, $0 \leq r \leq s-1$. Pentru a arăta că una dintre acestea trebuie să fie adevărată, putem folosi mica teoremă a lui Fermat, că pentru un număr prim n : $a^{n-1} \equiv 1$ (mod n). Dacă continuăm să luăm rădăcini pătrate ale lui a^{n-1}, vom obține fie 1, fie -1. Dacă obținem -1, atunci a doua egalitate este valabilă și este gata. Dacă nu obținem niciodată -1, atunci când am scos fiecare putere a lui 2, rămânem cu prima egalitate.

Numim un "martor" a pentru compunerea lui n. În caz contrar, a se numește un "mincinos puternic", iar n este un prim probabil puternic pentru baza a. Termenul "mincinos puternic" se referă la cazul în care n este compus, dar cu toate acestea ecuațiile sunt valabile așa cum ar fi pentru un număr prim.

Folosind pătratul repetat, timpul de rulare al acestui algoritm este $O(k * log^3 n)$, unde k este numărul de valori diferite ale lui a pe care noi îl testăm.

Eroarea făcută de testul de primalitate este măsurată prin probabilitatea ca un număr compus să fie declarat probabil prim. Cu cât se încearcă mai multe baze a, cu atât este mai bună acuratețea testului. Se poate demonstra că dacă n este compus, atunci cel mult $\frac{1}{4}$ dintre baze sunt "mincinoși puternici" pentru n. Ca o consecință, dacă n este compus, atunci rularea a k iterații ale testului Miller-Rabin va declara n probabil prim cu o probabilitate de cel mult 4^{-k}.

1.4 Specificarea criteriilor de evaluare alese pentru validarea soluțiilor

Metrici de performanță. Cele mai importante metrici la evaluarea performanței ale unui algoritm de identificare a numerelor prime sunt următoarele:
- timpului și memoria utilizată în timpul testării mai multor seturi de numere date;
- acuratețea programului (realizat prin testarea mai multor seturi de date), deoarece acești algoritmi au șanse (destul de mici) de a produce rezultate incorecte.

Testarea algoritmilor. După implementarea efectivă a algoritmilor, aceștia sunt testați cu ajutorul mai multor teste/date de intrare și se calculează valorile teoretice ale complexității timpului și memoriei folosite.

Fiecare algoritm va fi testat de mai multe ori pentru fiecare set de date, pentru a măsura o performanță medie pentru fiecare categorie și o medie generală mai precisă. Din acestea ar trebui să putem afla și acuratețea fiecărui algoritm.

Pentru a fi mai ușor de urmărit progresul algoritmilor la fiecare rulare, voi realiza tabele, care descriu dependența timp/memorie și acuratețe.

2 Prezentarea soluțiilor

2.1 Descrierea modului în care funcționeaza algoritmii aleși

Metoda Fermat:

Programul C care implementează metoda Fermat, utilizează o funcție modulo pentru a calcula puterea unei baze date la un exponent dat, modulo unui modul dat, apoi folosește acest rezultat pentru a determina dacă un anumit număr este prim sau nu. De asemenea, este folosit și un parametru suplimentar, k, care specifică de câte ori trebuie rulat testul (numărul de iterații). Dacă testul are succes de k ori, atunci numărul este determinat a fi prim.

Mai precis, programul începe prin a defini o funcție numită modulo care ia trei argumente: bază, exponent și modul. Această funcție calculează puterea bazei la exponent și modulo cu numărul dat. Utilizează o buclă while pentru a itera prin exponent și înmulțește rezultatul cu baza la exponentul curent. Rezultatul este apoi luat modulo cu numărul dat pentru a obține rezultatul final.

Apoi, programul definește o funcție numită isPrime care ia două argumente: n (numărul testat) și k (numărul de iterații). Această funcție folosește funcția modulo pentru a calcula puterea unui număr ales aleatoriu între 1 și n-1 la puterea n-1, modulo n. Dacă rezultatul nu este 1, atunci numărul testat nu este prim. Testul se execută de k ori și dacă are succes de k ori, atunci numărul este determinat a fi prim.

Metoda Miller-Rabin:

Următorul program este o implementare tot în C a testului de primalitate Miller-Rabin. Testul de primalitate Miller-Rabin este un algoritm folosit pentru a determina dacă un număr este sau nu prim. Acest algoritm se bazează pe observația că un număr prim trebuie să satisfacă anumite congruențe. Testul de primalitate Miller-Rabin este un test probabilistic, ceea ce înseamnă că nu este garantat să spună dacă un număr este sau nu prim, dar rata de succes e foarte mare.

Programul începe prin definirea unei funcții utilitare numită putere. Această funcție ia trei parametri, x (baza), y (exponent) și p (modul) și returnează x^y mod p. Această funcție este utilizată pentru a calcula "modular exponentiation", care este utilizată în testul de primalitate Miller-Rabin.

Apoi, programul definește funcția millerTest. Această funcție ia doi parametri, d (număr impar astfel încât d*2 = n-1) și n (numărul testat). Acesta calculează a^d mod n și verifică dacă acest rezultat este egal fie cu 1, fie cu n-1. Dacă este egal cu una dintre aceste valori, atunci numărul este probabil prim. Dacă nu este egal cu nici una dintre aceste valori, atunci programul va face pătratul lui x până când fie d ajunge la n-1, fie (x^2) mod n este egal cu 1, fie (x^2) mod n este egal cu n-1. Dacă niciuna dintre aceste condiții nu este îndeplinită, atunci numărul este compus.

În cele din urmă, programul definește funcția isPrime. Această funcție ia doi parametri, n și k. Parametrul n este numărul de testat, iar parametrul k este numărul de iterații. Funcția isPrime apelează funcția millerTest de k ori și returnează adevărat dacă numărul este (cel mai probabil) prim sau fals dacă numărul este compus.

2.2 Analiza complexitații soluțiilor

Complexitatea metodei Fermat utilizată în testarea primalității este O(logn) - O(1) pentru isPrime la o singură rulare și O(logn) pentru modulo. Aceasta înseamnă că timpul necesar pentru efectuarea testului crește logaritmic cu numărul de cifre din numărul testat. Timpul necesar pentru efectuarea testului este proporțional cu logaritmul numărului testat. Însă, deoarece se folosesc k iterații, complexitatea algoritmului devine O(k*logn).

Complexitatea algoritmului Miller-Rabin este $O(k * log^3 n)$, unde k este numărul de runde și n este numărul de biți din numărul testat. Acest lucru se datorează faptului că pentru fiecare rulare, algoritmul trebuie să efectueze o operație care necesită un timp O(log3 n). La fiecare rulare, algoritmul trebuie să verifice dacă numărul generat aleatoriu a este un martor pentru primalitatea lui n. Un martor este un număr care poate demonstra că n nu este un număr prim. Algoritmul face acest lucru verificând dacă numărul a îndeplinește anumite condiții. Acest lucru se realizează prin calcularea unui set de "modular exponentiations", care necesită timp $O(log^3 n)$. Complexitatea algoritmului este $O(k * log^3 n)$ deoarece trebuie să efectueze "modular exponentiations" de k ori pentru a verifica dacă numărul este prim.

2.3 Prezentarea principalelor avantaje și dezavantaje pentru soluțiile luate în considerare

Metoda Fermat
Pro:

1. O metodă relativ rapidă și eficientă pentru a testa primalitatea unui număr.
2. Nu necesită factorizarea numărului pentru a-l testa, ceea ce este un proces care consumă mai mult timp.
3. Este relativ ușor de implementat, ceea ce îl face potrivit pentru utilizarea în aplicații precum cele de criptografie.

Cons:

1. Nu este un test perfect, deoarece poate da ocazional un fals pozitiv (adică, declararea unui număr compus (numere Carmichael) ca fiind prim).
2. Nu este potrivit pentru utilizarea în aplicații în care este necesar un grad mai mare de precizie, cum ar fi aplicații bancare și financiare.
3. Nu este un test determinist, deoarece există o mică probabilitate de un rezultat fals pozitiv.

Metoda Miller-Rabin
Pro:

1. Este relativ simplu de implementat și poate fi folosit cu orice număr de baze.
2. Este mai eficientă decât alte teste de primalitate, în special pentru un număr mare.
3. Este foarte rapid și poate testa rapid numere mari.

Cons:

1. La fel ca metoda Fermat, Miller-Rabin este o metodă probabilistică și nu este garantată corectitudinea în toate cazurile.
2. Poate fi afectat de factori precum stabilitatea hardware și software.

3 Evaluare

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

Format date intrare:
Pe prima linie, N (int - numărul de elemente din secvență)
Pe următoarea linie N numere întregi (reprezentabile pe 32 biți)

Format date ieșire:
Pe prima linie, - numărul de elemente prime
Pe a doua linie se va afla secvența propriu-zisă de numerele prime extrase din secvența originală.

Restricții:

$$1 \leq N \leq 10^6$$

Baza de date care conține datele de intrare pentru testarea algoritmilor conține 20 astfel de teste. Testele sunt alcătuite astfel:
- primele 5 teste conțin numai numere compuse - testele 6-10 conțin numere Carmichael, numerele care au cea mai mare probabilitate de a "păcăli" algoritmii - testele 11-15 conține numere prime, care verifică practic dacă algoritmii determină ceea ce trebuie - iar ultimele 5 teste conțin un mix între primele 15 teste

Pentru a-mi ușura analiza, am ales ca testul 16 să fie un mix între testele 1, 6 și 11 pentru a determina mai ușor dacă rezultatul este cel așteptat sau nu. Idem pentru testele 17 (T2 + 7 + 12), 18 (T3 + 8 + 13), 19 (T4 + 9 + 14), 20 (T5 + 10 + 15).

Ulterior, pentru fiecare test în parte am construit output-ul de referință după care se va determina dacă algoritmul a determinat răspunsul corect.

3.2 Specificațiile sistemului de calcul pe care am rulat testele (procesor, memorie disponibilă)

System: ASUS TUF Gaming A15 FA506IU

Hardware

- CPU: AMD Ryzen 7 4800H 2.9 GHz nominal, overclock 4.2 GHz
- GPU: NVIDIA GeForce GTX 1660 Ti Mobile 6GB GDDR6
- RAM: 16GB DDR4 @ 3200MHz
- Storage: 1TB SSD Kingston

Software

- OS: Arch Linux $x86_64$
- Kernel: $6.0.12 - arch1 - 1$
- IDE: VisualStudio Code $x86_64$, code $1.74.2 - 1$

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

În cele două tabele de mai jos am expus rezultatele unei sigure rulări a celor două programare folosind toate datele de intrare în același timp:

Fermat	Test	Calificativ	Timp	Memorie
	1	Reușit	0.020000ms	1.707031kB
	2	Reușit	0.019000ms	1.707031kB
	3	Reușit	0.014000ms	1.707031kB
	4	Reușit	0.032000ms	1.707031kB
	5	Reușit	0.009000ms	1.707031kB
	6	Picat	0.009000ms	1.707031kB
	7	Picat	0.035000ms	1.707031kB
	8	Reușit	0.005000ms	1.707031kB
	9	Reușit	0.003000ms	1.707031kB
	10	Reușit	0.003000ms	1.707031kB
	11	Reușit	10.872000ms	1.707031kB
	12	Reușit	12.297000ms	1.707031kB
	13	Reușit	23.843000ms	1.707031kB
	14	Reușit	14.169000ms	1.707031kB
	15	Reușit	5.760000ms	1.707031kB
	16	Picat	10.892000ms	1.707031kB
	17	Picat	12.388000ms	1.707031kB
	18	Reușit	13.187000ms	1.707031kB
	19	Reușit	14.213000ms	1.707031kB
	20	Reușit	5.779000ms	1.707031kB

Miller-Rabin	Test	Calificativ	Timp	Memorie
	1	Reușit	0.061000ms	1.679688kB
	2	Reușit	0.063000ms	1.679688kB
	3	Reușit	0.014000ms	1.679688kB
	4	Reușit	0.030000ms	1.679688kB
	5	Reușit	0.031000ms	1.679688kB
	6	Reușit	0.005000ms	1.679688kB
	7	Reușit	0.010000ms	1.679688kB
	8	Reușit	0.005000ms	1.679688kB
	9	Reușit	0.004000ms	1.679688kB
	10	Reușit	0.010000ms	1.679688kB
	11	Reușit	0.134000ms	1.679688kB
	12	Reușit	0.449000ms	1.679688kB
	13	Reușit	0.463000ms	1.679688kB
	14	Reușit	0.163000ms	1.679688kB
	15	Reușit	0.204000ms	1.679688kB
	16	Reușit	0.160000ms	1.679688kB
	17	Reușit	0.172000ms	1.679688kB
	18	Reușit	0.304000ms	1.679688kB
	19	Reușit	0.180000ms	1.679688kB
	20	Reușit	0.241000ms	1.679688kB

Următorul tabel expune o analiză medie în urma rulării de 7 ori a celor două programe:

Metodă	Nr. total teste (nr. teste input * nr. rulări program)	Nr. teste picate (nr. teste input picate * nr. rulări program)	Acuratețe (rata succes medie)	Timp de execuție mediu per test	Memorie utilizată mediu per test
Fermat	140	28	80%	6.66872142ms	1.60993314kB
Miller-Rabin	140	0	100%	0.9275ms	1.58258928kB

3.4 Interpretarea, succintă, a valorilor obținute pe teste

Din rezultatele trecute în tabelele de mai sus putem interpreta astfel:

1. Metoda Fermat este mult mai rapidă pe testele care conțin numere compuse / Carmichael, însă are acuratețe mult mai mică.
2. Metoda Miller-Rabin este puțin mai lentă pe testele care conțin numere compuse / Carmichael, însă are acuratețe mult mai mare (maximă pe aceste teste).
3. Metoda Fermat este mai lentă atunci când este vorba de identificarea numerelor prime, pe când Miller-Rabin este cu mult mai rapidă.

4 Concluzii

1. Testele de primalitate sunt folosite pentru a determina dacă un anumit număr este prim sau nu.
2. Două metode cunoscute de testare a primalității unui număr / set de numere sunt Metoda Fermat și Metoda Miller-Rabin.

3. Complexitatea de timp a teoremei de primalitate a lui Fermat este O(k*logn).
4. Complexitatea de timp a testului de primalitate Miller-Rabin este $O(k * log^3 n)$.
5. Fermat este mai rapid pe numere compuse / Carmichael, dar mai expus erorilor, pe când Miller-Rabin este mult mai rapid pe numere prime și mult mai puțin expus greșelilor per total.

Bibliografie

1. https://www.geeksforgeeks.org/prime-numbers/
2. https://en.wikipedia.org/wiki/Prime_number
3. https://en.wikipedia.org/wiki/Composite_number
4. https://oeis.org/wiki/Carmichael_numbers
5. https://www.researchgate.net/publication/348899360_Methods_of_Primality_Testing