Identificarea Numerelor Prime

Veis Onur Şener

Universitatea Politehnică din Bucureşti
Facultatea de Automatică şi Calculatoare
Calculatoare şi Tehnologia Informaţiei
CD 321

Abstract. Evaluarea performanţei algoritmilor de identificare a numerelor prime.

Keywords: Numere prime · criptare · Fermat · Miller-Rabin

1 Introducere

1.1 Descrierea problemei rezolvate

Numărul prim reprezintă numărul natural care este divizibil doar cu 1 şi cu numărul în sine (se iau în considerare numerele strict mai mari decât 1).
Se doreşte identificarea numerelor prime dintr-un set de date de intrare

Intrare	Ieşire
5	2
4 7 8 10 13	7 13

1.2 Exemplu aplicaţie practică

Numerele prime prezintă o mare utilitate în criptarea cheilor publice. Algoritmul RSA este considerat unul dintre cele mai bune metode de criptare a cheilor publice, în care criptarea şi decriptarea cheilor este diferită. Desigur că există o conexiune între două chei (privată şi publică), dar aceasta nu poate fi descoperită de cel mai rapid calculator, într-un timp rezonabil. Astfel, RSA reprezintă un sistem sigur. Acest algoritm se bazează pe înmulţirea a două numere prime, care depăşesc fiecare 100 de cifre.

1.3 Specificarea soluţiilor alese

În determinarea numerelor prime se utilizeaza doi algoritmi :

– **Fermat** : Mica Teoremă a lui Fermat afirmă că dacă un număr natural p este prim şi a este un număr natural mai mic ca p, astfel încât (p,a) = 1, atunci

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

– **Miller-Rabin** : Fie p un număr natural impar. Dacă

$$\mathbf{cmmdc(a,p) \neq 1, \forall a \in N, 1 < a < 2(\log_2 p)^2 \implies p - prim}$$

1.4 Criterii de evaluare pentru soluţia propusă

O metodă de verificare a celor doi algoritmi ar fi utilizarea unei liste formate din numere prime până la un N (de ordinul sutelor de mii) şi a unei liste formate din numere compuse până la N , cu ajutorul cărora se alcătuiesc seturi de date de intrare, luând Q, respectiv P, elemente din fiecare (cu Q,P $\leq$ 10.000).

2 Prezentarea soluţiilor

2.1 Descrierea modului în care funcţionează algoritmii aleşi

Algoritmul lui Fermat (algo.cpp)

- Iniţial se verifică excepţiile (algoritmul nu funcţionează pentru numere mai mici sau egale cu 4).
- Se alege un număr 'a' la întâmplare, între 2 şi n-2.
- Se calculează cel mai mare divizor comun între numerele 'a' şi 'n'.
- Dacă acesta este diferit de 1, înseamnă că numărul 'n' este compus (return false).
- Se calculeaza $a^{(n-1)} \mod n$ (utilizând funcţia modEficient).
- Dacă rezultatul este diferit de 1, înseamnă că numărul 'n' este compus (return false).
- Algoritmul se repetă de un anumit număr de iteraţii.
- Dacă se ajunge la finalul funcţiei, înseamnă că 'n' este prim (return true).

Algoritmul lui Miller-Rabin (algo2.cpp)

- Iniţial se verifică situaţiile speciale (algoritmul nu funcţionează pentru numere mai mici sau egale cu 5).
- Dacă 'n' este impar, se scrie $n - 1 = d * 2^r$, unde 'd' este număr impar.
- Dacă 'n' este par, înseamnă că nu este prim (return false).
- Se alege un număr 'a' la întâmplare, între 2 şi n - 2.
- Se calculează $b = a^d \mod n$.
- Dacă 'b' este 1 sau n - 1, înseamnă că 'n' are o probabilitate destul de mare să fie prim.
- Dacă 'b' este diferit de 1 şi n - 1, următoarele secvenţe de cod se repetă de 'r' ori sau până se ajunge la o condiţie de oprire.
- Se calculează noul $b = b^2 \mod n$.
- Dacă 'b' este 1, înseamnă că 'n' este compus (return false).
- Dacă 'b' este n - 1, înseamnă că 'n' are o probalitate destul de mare să fie număr prim şi se iese din buclă.
- Algoritmul se repetă de un anumit număr de iteraţii.
- Dacă se ajunge la finalul funcţiei, înseamnă că 'n' este prim (return true).

2.2 Analiza complexităţii algoritmilor

Algoritmul lui Fermat (algo.cpp)

Funcţia __gcd are complexitate $\log n$.
Functia modEficient are complexitatea $\log n$ (în while, se împarte, la fiecare iteraţie, 'n' la 2 până ajunge la 0).
Astfel, algoritmul lui Fermat ajunge să aibă complexitatea :
T(n) = iteraţii * $(\log n + \log n) \in O(iteraţii * \log n)$

Algoritmul lui Miller-Rabin (algo2.cpp)

Bucla "while(d % 2 == 0)" are complexitatea $\log n$ ($d = n - 1$ şi se împarte cu 2 la fiecare iteraţie).
Functia modEficient are complexitatea $\log n$ (în while, se împarte, la fiecare iteraţie, 'n' la 2 până ajunge la 0).
Bucla cu "for", depinde de 'r', deci se repetă de $\log n$ ori.
Astfel, algoritmul lui Miller-Rabin ajunge să aibă complexitatea :
T(n) = $\log n$ + iteraţii * $(\log n + \log^2 n) \in O(iteraţii * \log^2 n)$

2.3 Principalele avantaje şi dezavantaje

Algoritmul lui Fermat (algo.cpp)

Unul dintre avantajele algoritmului lui Fermat este rapiditatea. Având o complexitate logaritmică de $O(iteraţii * \log n)$, reuşeşte să determine într-un timp mai scurt dacă un număr este prim sau nu, faţă de algoritmul în care se verifică cazul în care numărul are divizori proprii $O(\sqrt{n})$.

Totuşi, prezintă şi un dezavantaj. Există o probabilitate ca algoritmul să considere unele numere compuse ca fiind prime (numerele lui Carmichael). Acestea sunt numere compuse care satisfac ecuaţia $\mathbf{a^{p-1} \equiv 1} \pmod{\mathbf{p}}$, pentru orice $a < p$, cu proprietatea că (a,p) = 1. Astfel, pentru a determina că un număr Carmichael este compus, 'a' ar trebui să fie chiar unul dintre divizorii numărului. Deci, algoritmul are nevoie de un număr mare de iteraţii.

Algoritmul lui Miller-Rabin (algo2.cpp)

Un avantaj al algoritmului Miller-Rabin este faptul că determină corect soluţia, chiar şi în cazul numerelor pseudoprime (numere compuse, care nu sunt considerate compuse de unii algoritmi de testarea a numerelor prime). Astfel, acest algoritm reprezintă o extensie îmbunătăţită a algoritmului lui Fermat, abordând şi cazul numerelor Charmichael.

Însă, faptul că Miller-Rabin este un algoritm probabilistic reprezintă un dezavantaj. Conform cărţii "Prime Numbers: a computational perspective" de R. Crandall. şi C. Pomerance, algoritmul are o probabilitate de eşec mai mică de $1/4^k$, unde k reprezintă numărul de iteraţii. Deci pentru k = 2, există o şansă din 16 ca testul să ofere un răspuns greşit.

3 Evaluare

3.1 Set de teste folosite pentru validare

Generatorul de teste (generator.cpp)

Generatorul de teste utilizează un vector cu numere prime şi unul cu numere pseudoprime.
Acesta crează 25 de teste de intrare în fişierul "in" şi 25 de rezultate corecte în fişierul "out".
Primele 15 teste au o probabilitate foarte mică să conţină numere pseudoprime, iar următoarele 10 au o frecvenţă destul de mare de astfel de numere.

Checker (checker.sh)

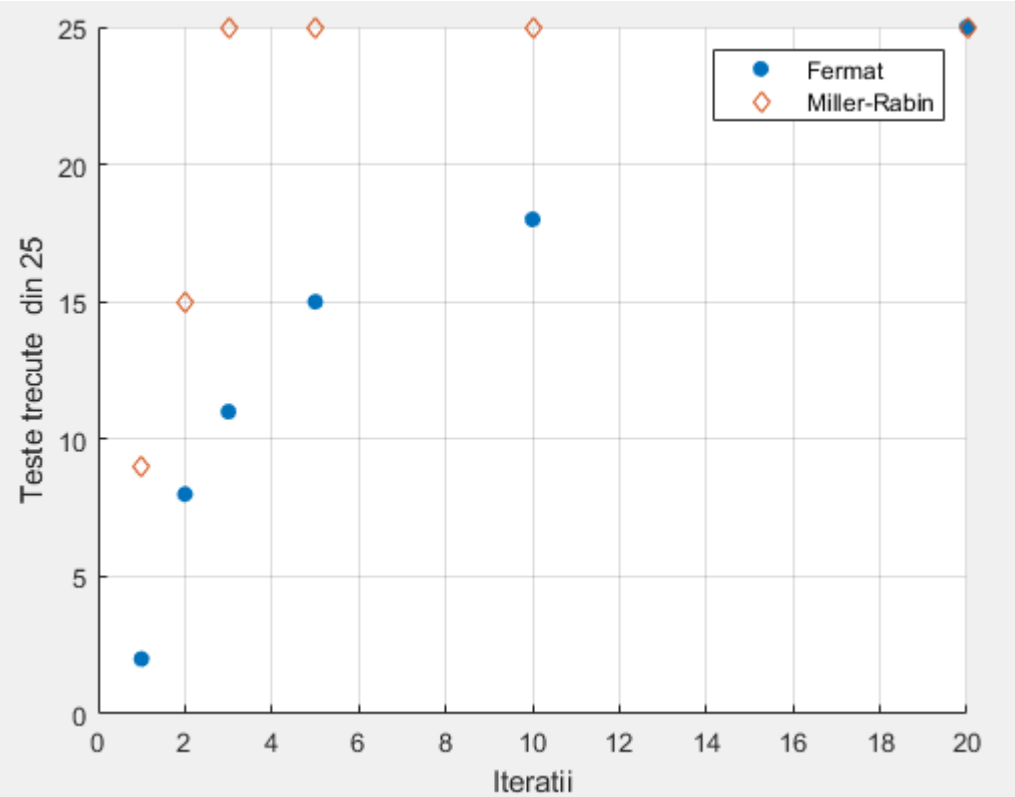
După rularea checker-ului trebuie introdus numărul de iteraţii pentru cei doi algoritmi.
Checker-ul compară fişierele de ieşire cu cele din "out" şi dacă nu există nicio diferenţă afişează "OK" şi "FAIL" în caz contrat.

3.2 Specificaţii sistem de calcul

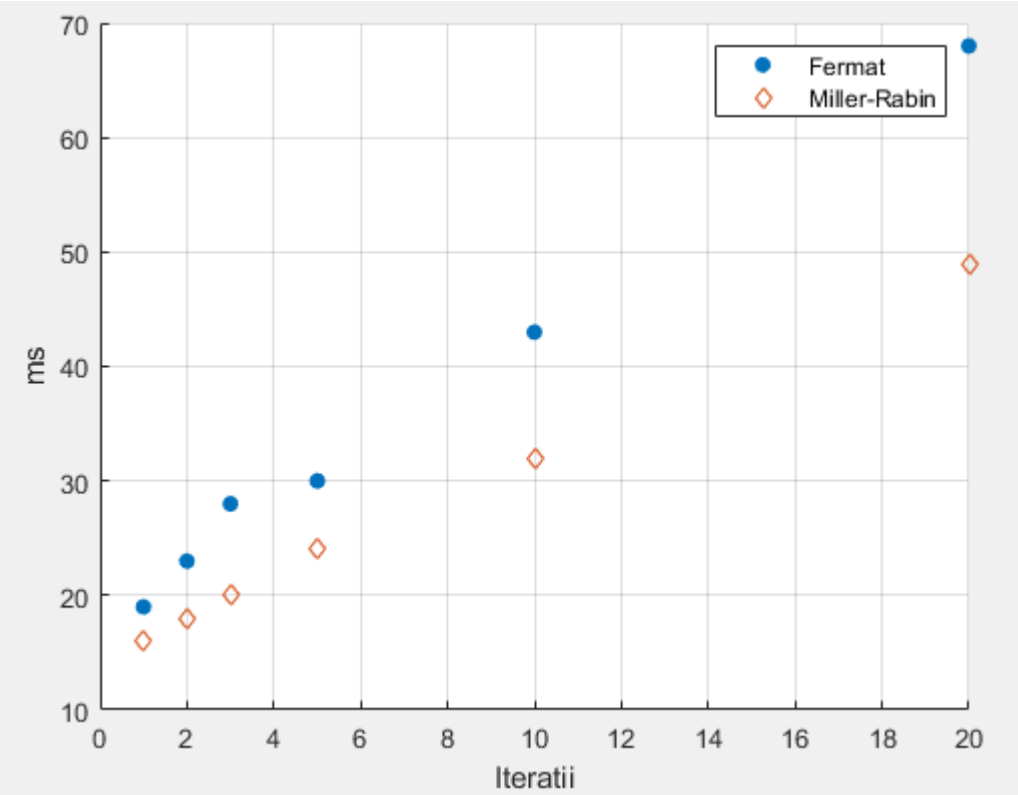
Tema a fost realizată pe un laptop cu procesor Intel(R) Core(TM) i3-5005U 2.00GHz şi memorie de 4GB, DDR3, 1600MHz.

3.3 Rezultate evaluare soluţii

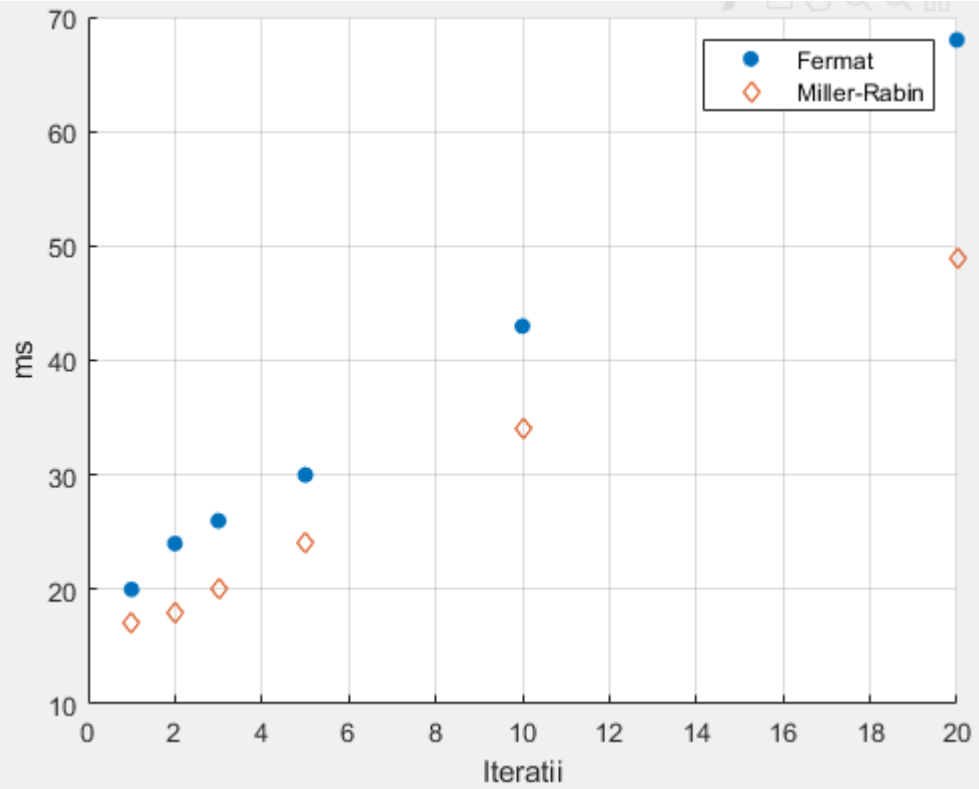
Graficul de mai jos cuprinde comportamentul celor doi algoritmi pe setul de date de intrare din fişierul "in".



Graficul de mai jos prezintă timpul de rurale al intregului set de date de intrare ”test0.in” pentru ambii algoritmi (test fără pseudoprime).



Graficul de mai jos prezintă timpul de rurale al intregului set de date de intrare "test24.in" pentru ambii algoritmi (test cu pseudoprime).



3.4 Prezentare valori obţinute

Din primul grafic se poate observa că algoritmul lui Miller-Rabin reuşeşte să rezolve corect toate testele de la un număr de 3 iteraţii.
Până la 10 iteraţii algoritmul lui Fermat reuşeste să treacă doar testele fără pseudoprime. În cazul celor cu pseudoprime, acesta are nevoie de 20 de iteraţii pentru a le efectua corect.
Pentru graficele 2 şi 3 nu se identifică diferenţe notabile. Se observa în ambele situaţii că algoritmul lui Miller-Rabin este mai rapid decât cel al lui Fermat.

4 Concluzii

În urma mai multor teste, se poate observa că algoritmul Miller-Rabin este mai eficient, reuşind să descopere numerele prime din 3-5 iteraţii, pe când algoritmul Fermat are nevoie de cel puţin 20 de iteraţii în cazul testelor cu pseudoprime.
Algoritmul lui Fermat este utilizat atunci când se doreşte o filtrare rapidă, de exemplu, în criptare.

Algoritmul lui Miller-Rabin, reprezentând o metodă rapidă de verificare a numerelor, este utilizat în librăriile algoritmului RSA şi OpenSSL (bibliotecă pentru aplicații care asigură comunicarea prin internet).

5 Referinţe

1. David Wells : Prime Numbers: The Most Mysterious Figures in Math. 1st edition. Wiley. New Jersey(2005).
2. Paulo Ribenboim : The Little Book of Bigger Primes. 2nd edition. Springer. New York (2004)
2 Richard Crandall, Carl B. Pomerance : Prime Numbers: a computational perspective. 2nd edition. Springer. New York (2005).