# Proiect Analiza Algoritmilor
# Identificarea numerelor prime

Anana Girlea Catalina 323CAa

Universitatea Politehnica Bucuresti
Facultatea de Automatică si Calculatoare specializarea Calculatoare si tehnologia informatiei
anana.girlea@stud.isb.upb.ro

**Abstract.** Studiu comparativ privind identificarea numerelor prime dintr-un set de date folosind algoritmii lui Fermat si Miller-Rabin.

**Keywords:** Numere prime · Fermat · Miller-Rabin · Compararea a doi algoritmi.

## 1 Introducere

### 1.1 Descrierea problemei

Acesta este studiu comparativ privind algoritmii lui Fermat si Miller-Rabin pentru gasirea numerelor prime dintr-un set de date de intrate.

**Aplicabilitatea practica pentru gasirea numerelor prime:**

1) Criprografie - RSA - Se bazeaza pe teoria ca necesita prea mult timp si nu este practica factorizarea numerelor prime mari. Exista o cheie publica (public key) formata din produsul a doua numere prime mari care cripteaza un mesaj si o cheie secreta (secrete key) care contine alte doua numere prime mari care decripteaza mesajul. Functioneaza deoarece este foarte greu si consuma prea mult timp calculul cheii secrete.
2) Cercetare in domeniul matematicii
3) Random number generators
4) Micsora complexitatea computațională - Anumite biblioteci care fac calcule cu numere întregi mari sau polinoame folosesc reduceri modulo prime(lema lui Hensel)
5) Animatie - Ajuta ca animatiile din diferite parti ale unui website sa inceapa in momente diferite

### 1.2 Specificarea solutiilor alese

Pentru problema identificarii numerelor prime dintr-un set de date de intrare am ales sa compar algoritmul lui Fermat cu cel al lui Miller-Rabin.

Le-am ales pe acestea doua deoarece amandoua se bazeaza pe Mica Teorema a lui Fermat, iar cel de al doilea este o extensie a primului algoritm.

Inclusiv complexitatiile acestora semana: - teorema lui Fermat avand O(k log2n log log n) = $\tilde{O}$(k log2n), unde n este numarul pe care il testam si k este numarul de teste facute - teorema lui Miller-Rabin avand O(k log2n log log n log log log n) = $\tilde{O}$(k log2n), unde n este numarul pe care il testam si k este numarul de teste facute

### 1.3 Criteriile de evaluare pentru solutia propusa

Criteiile de evaluare pe baza carora voi studia comparativ cei doi algoritmi sunt complexitatea, timpul de rulare si eficienta acestora.

Voi realiza acest lucru prin crearea mai multor teste unde voi compara timpul de rulare, rezultatul obtinut si rata de succes a implementarilor. Se vor face 26 de teste cu date de intrare care sa faciliteze amandoua algoritmele, dar si sa puna piedici atacand slabiciunile fiecarei metode (numerele Carmichael pentru Fermat si numerele preudoprime pentru Miller-Rabin). Testele vor fi impartite in grupuri de cate 5, numarul de elemente si numerele propriu-zise crescand gradual:
- testele 1-10 au pana la 10 elemente,
- testele 11-15 au 10-25 elemente,
- testele 15-20 au 5-100 elemente,
- testele 21-24 au 200-600 elemente,
- testele 25 si 26 au 1000 si respectiv 2000 elemente.

## 2 Prezentarea solutiilor

### 2.1 Descrierea modului in care functioneaza algoritmii alesi

**Algoritmul lui Fermat**

Aceast algoritm se bazeaza pe Mica Teorema a lui Fermat, care face distinctia dintre numerele prime si cele comuse. Probabilitatea corectitudinii creste odata cu numarul de iteratii ale programului

Daca n este nu numar prim, atunci pentru oricare a cu proprietatea ca 1 < a < n-1

$$a^{n-1} \equiv 1 (mod\ n) \quad or \quad a^{n-1}\ \%\ n = 1$$

Implementarea algoritmului lui Fermat se face prin apelarea Mici Teoreme si anume, generarea unui numar aleator a cu proprietatea ca 1 < a < n-1 si calcularea $a^{n-1}$ % $n$. Daca rezultatul este diferit de 1, n este compus. Daca rezultatul ecuatiei este 1, se repeta recursiv testul pentru un alt numar aleator. Procesul acesta se repata de i ori cate iteratii avem.

**Algorithm 1** isPrimeFermat

n = numarul care urmeaa sa fie testat
k = numaul de iteratii

1. trecem prin cazurile particulare n <= 4
2. while k > 0
3. generarea unui numar aleator a, unde a ∈ (2, n-2)
4. if $a^{n-1}$ $(mod\ n\ )! = 1$
5. return false − > numarul este compus
6. k = k - 1
7. return true − > numarul este prim

**Algoritmul lui Miller-Rabin**

Acest algoritm incorporeaza lemma lui Euclid, acre spune ca daca

$$x^2\ \%\ n = 1 \quad or \quad (x^2 - 1)\ \%\ n = 0 \quad or \quad (x-1)(x+1)\ \%\ n = 0$$

atunci n divide (x-1) sau (x+1), de unde rezulta ca $x\ \%\ = 1\ sau\ -1$ Daca n este numar prim, atunci pentru oricare a si i, cu proprietatile ca
$1 < a < n\text{-}1$ si $0 \leq a \leq r\text{-}1$

$$a^d\ \%\ n = 1 \quad or \quad a^{d2r}\ \%\ n = -1$$

Implementarea algoritmului lui Miller-Rabin se face prin verificarea daca n este impar, lucru din care rezulta ca n-1 este par. Un numar par poate fi scris ca $d * 2^s$, unde d este impar si $s < 0$. Se genereaza un numar aleator a pentru care calculam $a^{d2r}\ \%\ n$. Daca functia intoarce 1, numarul este prim. Procesul acesta se repata de i ori cate iteratii avem.

**Algorithm 2** isPrimeMiller-Rabin

n = numarul care urmeaa sa fie testat
k = numaul de iteratii

1. trecem prin cazurile particulare n <= 4
2. d = n -1
3. while d % 2 = 0
4. d = d / 2
5. for i − > k
6. if Miller-RabinTest = false
7. return false − > numarul este compus
8. return true − > numarul este prim

**Algorithm 3** Miller-RabinTest

n = numarul care urmeaa sa fie testat
d

1. generarea unui numar aleator a, unde a ∈ (2, n-2)
2. $x = a^d \ \% \ n$
3. if x este in extremele 1 sau n-1
4. return true $->$ numarul este prim
5. while d $\neq$ n-1
6. $x = (x \times x) \ \% \ n$
7. $d = d \times 2$
8. if x = 1
9. return false $->$ numarul este compus
10. if x = n-1
11. return true $->$ numarul este prim
12. return false $->$ numarul este compus

### 2.2 Analiza complexitatii solutiilor

**Algoritmul lui Fermat**

Pentru calcularea complextitatii algoritmului lui Fermat trebuie luate in considerare n numarul interogat si k numarul de iteratii. Prin calculul $a \times a \ (mod \ n)$ rezulta complexitatea $\mathcal{O}log^2n$. Se aplica operatia de inmultire modulara de aproximativ log n ori. Folosind aceasta metoda se calculeaza restul dupa fiecare iteratie dupa calculul patratului modula n $=> a^{n-1} \ (mod \ n) \ = \ \mathcal{O}(\log^2 n \times \log n)$. Din cele de mai sus rezulta ca:

$$\mathcal{O}(k \times log^2n \times log \ n) \ = \mathcal{O}(k \ log^3n)$$

**Algoritmul lui Miller-Rabin**

Pentru calcularea complextitatii algoritmului lui Miller-Rabin trebuie sa ne folosim exact ca mai sus de n numarul interogat si k numarul de iteratii. Se foloste aducerea la patrat in mod repetat, astfel rezultand complexitatea

$$\mathcal{O}(k \ log^3n)$$

Se poate optimiza algoritmul prin adaugarea unui algoritm de tip FFT(Fast Fourier transform). FFt este un algoritm care calculeaza transformata Fourier discreta (DFT) a unei secvente sau inversul acesteia. Acest tip de algoritm foloseste factorizarea matricei DFT intr-un produs de factori reali, astfel reusind sa reduca viteza de calcul si sa transforme o complexitatea fransformatei din $\mathcal{O}(n^2)$ in $\mathcal{O}(n \ log \ n)$, unde n reprezinta dimensiunea datelor. Daca folosim un astfel de algoritm, noua complexitate devine:

$$\mathcal{O}(k \times log^2n \times log \ log \ n \times log \ log \ log \ n) \ = \mathcal{O}(k \ log^2n)$$

### 2.3 Prezentarea principalelor avantaje si dezavantaje pentru solutiile luate in considerare

**Algoritmul lui Fermat**

Avantaje:
- Cu cat numarul iteratiilor este mai mare, cresc sansele ca algoritmul sa returneze ce trebuie.
- Rata de succes a depistarii numerelor prime este de 50%.
- Complexitatea este mai buna decat cea a algoritmilor clasici $O(n^2)$ *sau* $O(\sqrt{n})$

Dezavantaje:
- Aceasta metoda are o slabiciune si anume numerele Charmichael care sunt numare compuse dar care satiscafac relatia indiferent de cat de mare este numarul de iteratii. Aceste numere sunt rar intalnite (7 numere pana in 10.000: 561, 1105, 1729, 2465, 2821, 6601 si 8911)

**Algoritmul lui Miller-Rabin**

Avantaje:
- Numerele Charmichael nu mai prezinta o problema.
- Nu mai este necesar ca numarul de iteratii sa fie la fel de mare ca la algoritmul lui Fermat.
- Complexitatea este mai buna decat cea a algoritmilor clasici $O(n^2)$ *sau* $O(\sqrt{n})$

Dezavantaje:
- Numerele pseudioprime sunt numere compuse pentru care algoritmul nu returneaza mereu ce trebuie. Ele sunt de 4 ori mai putine decat numerele Charmichael

## 3 Evaluare

### 3.1 Descrierea modalitatii de construire a setului de teste folosite pentru validare

Am creat 2 fisiere cu cate 26 de teste folosite la ambi algoritmi, primul continand teste de input si celalt testele de output luate ca referinta.
Distributia testelor:

- Testele 1-5 au pana la 10 elemente in sir, aceste elemente fiind mai mici decat 1000
testul 1 contine 2 numere prime si 0 compuse
testul 2 contine 2 numere prime si 1 compus
testul 3 contine 1 numere prime si 4 compuse
testul 4 contine 4 numere prime si 3 compuse
testul 5 contine 1 numere prime si 0 compuse

- Testele 6-10 au pana la 10 elemente in sir, aceste elemente fiind mai mici decat 1000000
testul 6 contine 3 numere prime si 0 compuse
testul 7 contine 3 numere prime si 0 compus
testul 8 contine 4 numere prime si 1 compuse
testul 9 contine 1 numere prime si 0 compuse
testul 10 contine 7 numere prime si 0 compuse

- Testele 11-15 au 10-25 elemente in sir, aceste elemente fiind mai mici decat 1000000
testul 11 contine numere prime si 6 compuse
testul 12 contine 0 numere prime si 0 compus
testul 13 contine 10 numere prime si 1 compuse
testul 14 contine 15 numere prime si 0 compuse
testul 15 contine 25 numere prime si 0 compuse

- Testele 15-20 au 25-100 elemente in sir, aceste elemente fiind mai mici decat 1000000
testul 16 contine 15 numere prime si 10 compuse
testul 17 contine 50 numere prime si 0 compus
testul 18 contine 50 numere prime si 0 compuse
testul 19 contine 100 numere prime si 0 compuse
testul 20 contine 99 numere prime si 1 compuse

- Testele 21-24 au 200-600 elemente in sir, aceste elemente fiind mai mici decat 1000000
testul 21 contine 200 numere prime si 0 compuse
testul 22 contine 100 numere prime si 100 compus
testul 23 contine 500 numere prime si 0 compuse
testul 24 contine 600 numere prime si 0 compuse

- Testul 25 are 1000 elemente in sir, aceste elemente fiind mai mici decat 1000000
testul 25 contine 1000 numere prime si 0 compuse

- Testul 26 are 2000 elemente in sir, aceste elemente fiind mai mici decat 1000000
testul 26 contine 2000 numere prime si 0 compuse

### 3.2 Mentionati specificatiile sistemului de calcul pe care ati rulat testele

MacBookPro
Procesor: 2,7 GHz Dual-Core Intel Core i5
Memorie: 8 GB 1867 MHz DDR3

### 3.3 Ilustrarea, folosind grafice/tabele, a rezultatelor evaluarii solutiilor pe setul de teste

Se testeaza timpului de rulare pentru ficare algoritm.
Pentru prima rulare a programului:

| Nr Test | Timp Alg Fermat (s) | Timp Alg Miller-Rabin (s) | Nr de elemente in test |
|---|---|---|---|
| 1 | 0.224723 | 0.203088 | 2 |
| 2 | 0.231627 | 0.256371 | 3 |
| 3 | 0.148160 | 0.152951 | 5 |
| 4 | 0.570069 | 0.600908 | 7 |
| 5 | 0.216577 | 0.260785 | 1 |
| 6 | 0.824528 | 0.910986 | 3 |
| 7 | 0.932673 | 1.015149 | 3 |
| 8 | 0.762175 | 0.894678 | 5 |
| 9 | 0.367551 | 0.454372 | 1 |
| 10 | 2.087519 | 2.453699 | 7 |
| 11 | 1.491505 | 1.676322 | 10 |
| 12 | 0.000105 | 0.000176 | 10 |
| 13 | 3.631776 | 4.252229 | 11 |
| 14 | 5.713794 | 6.311925 | 15 |
| 15 | 9.407785 | 10.434639 | 25 |
| 16 | 4.417080 | 4.858124 | 25 |
| 17 | 19.009061 | 20.979798 | 50 |
| 18 | 17.693352 | 19.375461 | 50 |
| 19 | 36.327941 | 40.165779 | 100 |
| 20 | 36.221707 | 40.814007 | 100 |
| 21 | 75.885295 | 83.835874 | 200 |
| 22 | 36.286535 | 41.777781 | 200 |
| 23 | 187.016150 | 205.734353 | 500 |
| 24 | 213.641974 | 236.528057 | 600 |
| 25 | 231.244493 | 248.148860 | 1000 |
| 26 | 748.340968 | 853.886363 | 2000 |

**Fig. 1.** Tabel Timp Fermat si Miller-Rabin

Comparam timpul scos de cei doi algoritmi, prima data pentru primele 16 teste (1-16) si apoi pentru urmatoarele 10 (17-26):

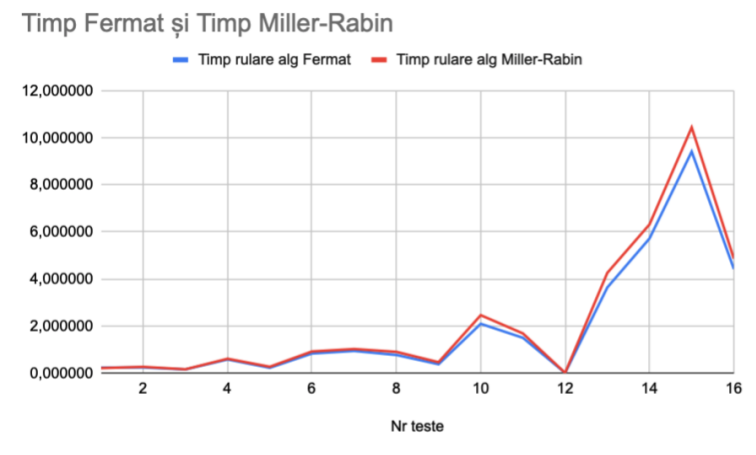


**Fig. 2.** Pentru testele 1-16

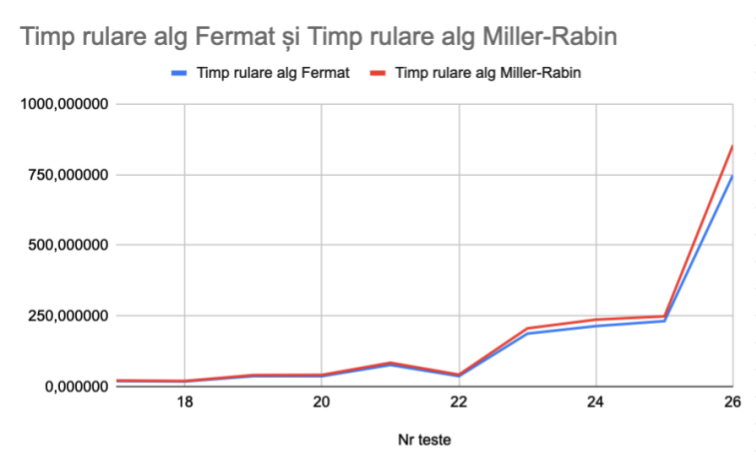


**Fig. 3.** Pentru testele 17-26

Pentru a doua rulare a programului:

| Nr Test | Timp Alg Fermat (s) | Timp Alg Miller-Rabin (s) | Nr de elemente in test |
|---|---|---|---|
| 1 | 0.205973 | 0.185387 | 2 |
| 2 | 0.239202 | 0.229612 | 3 |
| 3 | 0.147458 | 0.143449 | 5 |
| 4 | 0.564077 | 0.546947 | 7 |
| 5 | 0.213322 | 0.211552 | 1 |
| 6 | 0.906712 | 0.817469 | 3 |
| 7 | 0.930359 | 0.904357 | 3 |
| 8 | 0.781330 | 0.735122 | 1 |
| 9 | 0.374538 | 0.361081 | 5 |
| 10 | 2.108052 | 2.169281 | 1 |
| 11 | 1.520779 | 1.577710 | 7 |
| 12 | 0.000109 | 0.000134 | 10 |
| 13 | 3.700358 | 3.849068 | 10 |
| 14 | 5.709292 | 5.554193 | 11 |
| 15 | 9.558078 | 9.825775 | 15 |
| 16 | 4.436820 | 4.532341 | 25 |
| 17 | 19.395570 | 19.712360 | 25 |
| 18 | 18.428605 | 18.001494 | 50 |
| 19 | 37.533284 | 35.281140 | 50 |
| 20 | 38.738213 | 36.978253 | 100 |
| 21 | 79.762183 | 81.172871 | 100 |
| 22 | 37.779061 | 37.803629 | 20 |
| 23 | 194.118912 | 194.236072 | 500 |
| 24 | 224.628821 | 225.223950 | 600 |
| 25 | 237.279222 | 222.548589 | 1000 |
| 26 | 775.792984 | 776.029770 | 2000 |

**Fig. 4.** Tabel Timp Fermat si Miller-Rabin

Comparam timpul scos de cei doi algoritmi, prima data pentru primele 16 teste (1-16) si apoi pentru urmatoarele 10 (17-26):

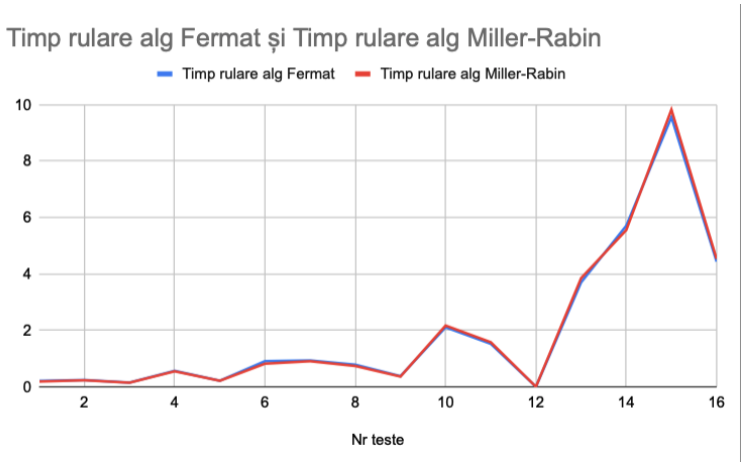


**Fig. 5.** Pentru testele 1-16

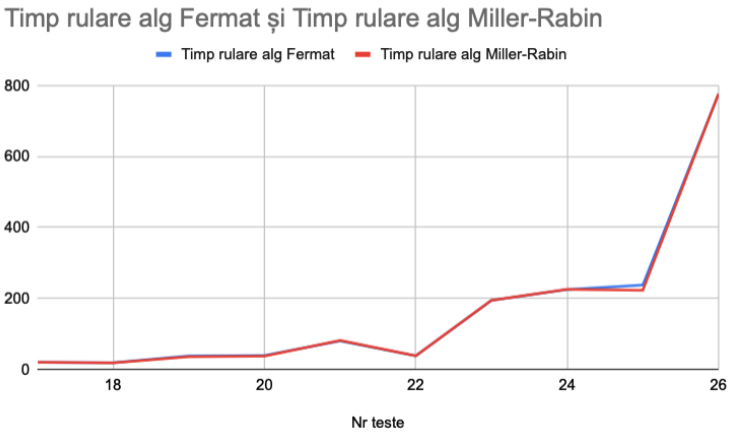


**Fig. 6.** Pentru testele 17-26

### 3.4 Prezentarea, succinta a valorilor obtinute pe teste

Analizand cele 26 de teste se observa faptul ca difernta dintre timpii de rulare ai celor doi algoritmi sunt foarte mici, dar algoritmul lui Fermat pare sa aiba un mic avantaj la prima rulare. Am rulat programul inca o data, ide data aceasta diferentele erau mult mai mici, de aproximativ 0.1s in avantajul algoritmului lui Miller-Rabin. Din grafice si tabele reiese ca timpii variaza de la o rulare la alta.

Aceste valori au fost calculate cand k = 1000000. Am ales un k cu valoare mare pentru ca odata cu cresterea numarului iteratiilor, creste si sansa ca raspunsul returnat sa fie corect.

## 4 Concluzii

Amandoua algoritmele sunt mai bune decat metodele clasice de determinare a primalitatii unui numar, din punct de vedere al complexitatilor:

$$\mathcal{O}(k\ log^3 n) < \mathcal{O}(n^2) sau \mathcal{O}(\sqrt{n}$$

Din punct de vedere al numerlor care prezinta probleme, algoritmul lui Miller-Rabin este mai stabil. Numerele preudoprime sunt de 4 ori mai putine decat numerlele Charmichael. Inca un avantaj al algoritmului lui Miller-Rabin este ca nu este necesar ca numarul de iteratii sa fie la fel de mare ca la cel al lui Fermat. Din analiza tabeleor reiese ca timpii tuturor testelor variaza de la o rulare la alta.
Luand in cosiderare toate punctele de mai sus, as opta sa sa folosesc in practica algoritmul lui Miller-Rabin deoarece este mai stabil, numerele pseudoprime fiind mai rare si timpul rularii poate scadea datorita faptului ca nu necesita la fel de multe iteratii.

## References

1. Why are primes important in cryptography?, https://stackoverflow.com/questions/439870/why-are-primes-important-in-cryptography. Last accessed 16 Dec 2021
2. Fermat's little theorem, https://www.geeksforgeeks.org/fermats-little-theorem/. Last accessed 16 Dec 2021
3. Primality Test — Set 2 (Fermat Method), https://www.geeksforgeeks.org/primality-test-set-2-fermet-method/?ref=lbp. Last accessed 16 Dec 2021
4. Primality Test — Set 3 (Miller–Rabin), https://www.geeksforgeeks.org/primality-test-set-3-miller-rabin. Last accessed 16 Dec 2021
5. C++ Program to Perform Fermat Primality Test, https://www.tutorialspoint.com/cplusplus-program-to-perform-fermat-primality-test. Last accessed 16 Dec 2021

6. C++ Program to Implement the Rabin-Miller Primality Test to Check if a Given Number is Prime, https://www.tutorialspoint.com/cplusplus-program-to-implement-the-rabin-miller-primality-test-to-check-if-a-given-number-is-prime. Last accessed 16 Dec 2021
7. Fermat Primality Test, https://www.baeldung.com/cs/fermat-primality-test Last accessed 16 Dec 2021
8. Primality Test: Miller-Rabin Method, https://www.baeldung.com/cs/miller-rabin-method Last accessed 16 Dec 2021
9. LaTeX:Symbols, https://artofproblemsolving.com/wiki/index.php/LaTeX:Symbols Last accessed 16 Dec 2021
10. Jean Gallier and Jocelyn Quaintance : Notes on Primality Testing And Public Key Cryptography, February 27, 2019 https://www.cis.upenn.edu/ jean/RSA-primality-testing.pdf