Identificarea numerelor prime

Maciuca Alexandru-Petru 324CA

Facultatea de Automatica si Calculatoare,
Universitatea POLITEHNICA Bucuresti
alexandru.maciuca@stud.acs.upb.

Abstract. Studiul numerelor prime a starnit mult interes printre marii matematicieni ai lumii, inca din timpul Antichitatii, dar care odata cu trecerea timpului a devenit din ce in ce mai neglijat. Aceasta lucrare exploreaza tematica numerelor prime cu scopul de a evidentia practicabilitatea si importanta acestora in viata noastra.

Matematica poate fii adesea vazuta ca o poarta spre gandirea pura, a carei influenta se revarsa in domenii precum istoria sau politica si ale carei blocuri de numere ajung sa sistematizeze intreaga noastra existenta.

Valentele unui numar prim se regasesc la nivelul unui numar natural, mai mare ca 1, care are exact doi divizori: numarul 1 si numarul in sine. Acesti divizori sunt improprii si deci un numar prim este nefactorizabil. Cel mai mic numar prim este 2, iar in afara de acesta, toate celelalte numere prime apartin spectrului numerelor impare. Opusul notiunii de numar prim este cel de numar compus

Cel mai mare numar prim descoperit pana acum este egal cu $2^{74207281}$-1, si a fost descoperit in anul 2016 in Missouri, depasind cu pana la 5 milioane de cifre pe precendetul sau.

Keywords: numere prime · divizori · impropriu · nefactorizabil · numar compus.

1 Introducere

1.1 Descrierea problemei rezolvate

Identificarea numerelor prime a pus la grea incercare cele mai sclipitoare minti incluzandu-l aici si pe Euclid care in anul 300 î.Hr. a demonstrat existanta unei infinitati de astfel de numere.

Problema cu care ne intalnim in ziua de astazi este gasirea unor noi numere prime. Suntem familiari cu conceptul de factorizare a unui numar, proces fezabil pentru numere mici(de ordinul a 10-15 digiti), dar aproape irealizabil atunci cand ne referim spre exemplu la 2 numere prime mari care se inmultesc.

Desi nu putem afirma sau infirma in legatura cu primalitatea unui numar mare(peste 20 digiti), exista metode care pana intr-un anumit punct pot determina aceasta caracterisitica pentru numere mai mici, pe care apoi le putem inmulti pentru a ajunge la numerele dorite.

1.2 Exemple de aplicatii practice pentru problema aleasa

Matematica moderna este guvernata de numere prime care au un rol foarte important in domenii precum securitatea, criptarea si decriptarea datelor. Astfel in anul 1978, Ron Rivest, Adi Shamir si Leonard Adleman vor pune bazele sistemului de criptare RSA.

Acest algoritm face posibila securizarea transmiterii de informatii in mediul online cu practicalitate imensa in domeniul platilor bancare. Pe scurt, in momentul achizitionarii unui produs cu plata online, dupa ce sunt transmise datele cardului, acestea sunt criptate si decriptate, pentru securitate cu ajutorul numerelor prime.

Importanta numerelor prime se reflecta masiv si in domeniul telecomunicatiilor, unde acestea faciliteaza transmiterea corecta a mesajelor. Numere prime apar si in functiile de hash folosite la tabelele de dispersie, unde ne dorim sa obtinem valori unice.

1.3 Specificarea solutiilor alese

Pentru a putea identifica numerele prime am ales sa implementez cei doi algoritmi Fermat si Miller-Rabin. Inainte de toate trebuie mentionat ca exista teste de primalitate care determina in mod concludent daca un numar este prim sau compus. Spunem despre acestea ca sunt metode deterministice in comparatie cu testele Fermat si Miller-Rabin care sunt metode probabilistice.

In ciuda clasificarii corecte a tuturor numerelor prime, cele din urma permit filtrarea unor numere compuse etichetate incorect ca fiind prime sau probabil prime, fapt ce denota latura probabilistica a algoritmilor.

Incertitudinea oferita de cei doi algoritmi se poate diminua cu cat numarul de iteratii este mai mare, deoarece acestia se stabilizeaza direct proportional cu indicele de iteratie. Cu toate acestea caracterul probabilistic nu poate fi inlaturat complet!

1.4 Criteriile de evaluare pentru solutia propusa

In cadrul primei etape din procesul de testare voi analiza corectitudinea celor doi algoritmi verificand functionalitatea de baza. Acest lucru se va realiza introducand manual numere aleatoare de diferite dimensiuni.

Cea de a doua etapa va fi marcata de automatizarea procesului de testare. Astfel voi genera cu ajutorul *ciurului lui Eratostene* toate numerele prime din intervalul [2, 100.000]. Dat fiind caracterul probabilistic al algoritmilor, ne asteptam ca acestia sa nu ofere intotdeauna rezultatul adevarat din punct de vedere matematic, motiv pentru care voi intocmi o statistica de forma *nr.evaluate corect/nr. total evaluate*.

In etapa a treia voi introduce cateva teste speciale care vor contine in mare parte numere compuse menite sa induca in eroare algoritmii.(ex: *numere Carmichael*)

In etapa a patra voi incerca sa intocmesc un set de teste care sa contina un amestec de numere prime si numere complexe. Voi varia dimensiunea testelor cu scopul de a analiza eficienta si precizia fiecarui algoritm in parte.

In ultima etapa ne dorim sa vedem si cat de rapizi sunt cei doi algoritmi, motiv pentru care voi incerca sa monotorizez timpii fiecaruia in parte pentru acelasi set de date. Analizand idea de baza si structura acestora, este de asteptat ca testul Miller-Rabin sa aiba un timp mai bun decat cel al testului Fermat.

2 Prezentarea solutiilor

2.1 Descrierea modului in care functioneaza algoritmii alesi

Algoritmul Fermat este un test de primalitate ce verifica daca un numar este sau nu prim avand la baza *"Mica Teorema a lui Fermat"*.

Definitie:
Daca p este un numar prim si x este un numar intreg care nu este multiplu al lui p, atunci $x^{p-1} \equiv 1 \pmod{p}$

Definitie:
Fie n un intreg impar compus si x un intreg din intervalul [1,n-1]. Atunci n se numeste pseudo-prim in raport cu baza x daca $x^{p-1} \equiv 1 \pmod{p}$.

Intregul x se numeste mincinos Fermat(al calitatii de a fi prim) pentru n. Astfel de numere sunt considerate rare, iar un exemplu concret ar fi *numerele Carmichael*.

Fiecare numar Carmichael este produsul a cel putin trei numere prime, iar acestea sunt destul de rar intalnite, fiind descoperite pana in prezent doar 105212 exemple. Ele sunt capabile sa pacaleasca testul Fermat si poarta acest nume de pseudo-prime.

Algoritmul Fermat

Intrare: n = valoare pentru care testam primalitatea;
iter = numarul de iteratii al algoritmului;
Iesire: "numar compus" daca n este numar compus;
"probabil prim" daca algoritmul nu returneaza "numar compus";

Fermat(n, iter)

pentru i de la 1 la iter
alege aleator un intreg x din intervalul [2,n-2];
daca $x^{n-1} \not\equiv 1 \pmod{n}$ returneaza "numar compus"
returneaza "probabil prim"

Exemplu
n = 11
$\Rightarrow 2^{10} \equiv 1 \pmod{11}$
$\Rightarrow 3^{10} \equiv 1 \pmod{11}$
$\Rightarrow 4^{10} \equiv 1 \pmod{11}$
$\Rightarrow 5^{10} \equiv 1 \pmod{11}$
$\Rightarrow 6^{10} \equiv 1 \pmod{11}$
$\Rightarrow 7^{10} \equiv 1 \pmod{11}$
$\Rightarrow 8^{10} \equiv 1 \pmod{11}$
$\Rightarrow 9^{10} \equiv 1 \pmod{11}$
$\Rightarrow 10^{10} \equiv 1 \pmod{11}$

$\Longrightarrow$ numarul n este prim.

[Nota Bene] Daca numarul n este unul compus, atunci acesta va reusi sa pacaleaca algoritmul Fermat, motiv pentru care trebuie mentionat ca rezultatul nu este intotdeauna cel corect!

Algoritmul Miller-Rabin este cel mai folosit test probabilistic pentru numere prime, cunoscut si sub numele de test pseudo-prim puternic.

Corectitudinea algoritmului lui Miller depinde de corectitudinea *Ipotezei Extinse a lui Riemann*.

Definitie:
Fie n un intreg de forma $2^r d + 1$, unde r si d sunt intregi pozitivi si d este un numar impar. Fie x un intreg din intervalul [0, n], atunci putem afirma despre n ca este un numar puternic probabil prim daca au loc urmatoarele:
a) $x^r \equiv 1 \pmod{n}$
b) Notam $p = 2^r d + 1$ astfel incat $x^p \equiv -1 \pmod{n}$ pentru r din intervalul [0,s)

Algoritmul Miller-Rabin

Intrare: n = valoare pentru care testam primalitatea;
iter = numarul de iteratii al algoritmului;
Iesire: "numar compus" daca n este numar compus;
"probabil prim" daca algoritmul nu returneaza "numar compus";

Miller-Rabin(n, iter)

scrie n-1 = $2^r d$, astfel incat d sa fie impar
pentru i de la 1 la iter
    alege aleator un intreg x din intervalul [2,n-2];
    calculeaza y = x^d (mod n)
    daca y $\neq$ 1 si y $\neq$ n-1, atunci
        j = 1;
        cat timp j <= r-1 si y $\neq$ n-1
            calculeaza y = y^2 (mod n);
            daca y = 1 atunci
                returneaza "numar compus";
            j = j + 1;
            daca y $\neq$ n - 1 atunci
                returneaza "numar compus";
returneaza "numar prim";

Exemplu

n = 13 iter = 2

1) Cautam d si r astfel incat n - 1 = d * 2^r
   $\Rightarrow$ d = 3 si r = 2
2) Efectuam 3 iteratii pentru algoritmul Miller-Rabin

Prima iteratie

- alegem un numar x din intervalul [2,n-2], x = 4
- x = 4^3 % 13 = 12
- x == n-1 $\Rightarrow$ *true*

A doua iteratie

- alegem un numar x din intervalul [2,n-2], x = 7
- x = 7^3 % 13 = 5
- x nu e nici 1, nici 12
- repetam algoritmul de r-1 ori
  a) x = (x * x) % 13 = (5 * 5) % 13 = 12
  b) x == n-1 $\Rightarrow$ *true*

Deoarece toate iteratiile au returnat *true* $\Rightarrow$ *numarul este prim*

[Nota Bene] Daca numarul n este unul compus, atunci acesta va reusi sa pacaleaca algoritmul Miller-Rabin, motiv pentru care trebuie mentionat ca rezultatul nu este intotdeauna cel corect!

Algoritmul Miller-Rabin este mai preferat fata de Fermat deoarece are o rata de succes mare dar nici acesta nu returneaza de fiecare data raspunsul corect!

2.2 Analiza complexitatii solutiilor

Algoritmul Fermat are complexitatea in **worst case** de $O(iter * \log^2 n)$, unde
n = numarul supus analizei
iter = numarul de iteratii pentru testul Fermat

Algoritmul este alcatuit din doua instructiuni repetitive, una pentru calcularea proprietatii *Fermat*, iar alta pentru realizarea celor *iter* numar de iteratii, rezultand astfel complexitatea de mai sus.

Algoritmul Miller-Rabin are complexitatea in **worst case** de $O(iter * \log^3 n)$, unde
n = numarul supus analizei
iter = numarul de iteratii pentru testul Miller-Rabin

Varianta neoptimizata a algoritmului prezinta patru structuri repetitive care genereaza complexitatea de mai sus. Folosirea unui algoritm polinomial precum **FFT** poate eficientiza testul Miller-Rabin rezultand o complexitate de $O(iter * \log^2 n)$

2.3 Prezentarea principalelor avantaje si dezavantaje pentru solutiile luate in considerare

Algoritmul Fermat

Din punct de vedere strict teoretic, un avantaj evident al acestui algoritm este reprezentat de complexitatea sa relativ mica raportata la numarul de iteratii, acesta fiind mai bine optimizat decat Miller-Rabin in forma initiala.

Tot legat de complexitate, in cazul testarii unui numar care nu este prim, in cel mai bun caz, rezultatul poate fi obtinut in O(1).

Algoritmul este scurt si rapid de inteles, deoarece nu are multe conditii de verificat, ceea ce conduce la un cost infim si la un timp de rulare mai bun.

Un dezavantaj important al algoritmilor probabilistici este legat de incertitudinea cu privinta la corectitudinea rezultatului returnat.

Algoritmul Miller-Rabin

In varianta optimizata, algoritmul Miller-Rabin rivalizeaza cu Fermat din punct de vedere al complexitatii. Datorita modului de filtrare a numerelor pare mai mari ca 3 si al mecanismului mai bun de verificare a numerelor prime acesta este favorizat in industrie.

Analog, un dezavantaj al testului Miller-Rabin este determinat de caracterul probabilistic al acestuia. Cu toate acestea, sansa de a returna un rezultat eronat

este de $1/4^k$, unde k este numarul de iteratii, ceea ce este multumitor de cele mai multe dati.

3 Evaluare

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

Testele sunt in numar de 40 si au fost construite astfel :

Am folosit un generator in C++ care se bazeaza pe *ciurul lui Eratostene*, acesta fiind un algoritm determinist de verificare a primalitatii unui numar prim.

Testele mele se bazeaza pe un interval de valori, iar algoritmul specificat mai sus se foloseste de valori anterioare, motiv pentru care a trebuit mereu sa generez toate numerele din intervalul [2, limita superioara a intervalului dorit], iar mai apoi am aplicat o functie de filtrare care elimina numerele mai mici decat limita inferioara a intervalului dorit.

Testele variaza atat in volumul de numere, cat si in dimensiunea acestora.

Doar algoritmul **Miller-Rabin** a trecut acest test.

3.2 Mentionati specificatiile sistemului de calcul pe care ati rulat testele (procesor, memorie disponibila)

Testele au fost rulate pe un sistem de calcul cu urmatoarele caracteristici :

Procesor : AMD Ryzen 7 4800HS
Placa video : GeForce GTX 1650TI 4 GB GDDR6
Memorie RAM : 16GB
Sistem de operare : Windows 10 PRO

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

In primul rand, algoritmii nu sunt la fel de eficienti motiv pentru care, testul *Fermat* are nevoie de mai multe iteratii decat testul *Miller-Rabin* pentru a intoarce un rezultat cat mai aproape de cel corect.

Totusi, pentru a testa intr-un mod congruent ambii algoritmi, voi pastra numarul de iteratii identic.

Timpii returnati nu depind de natura rezultatului (corect / gresit)

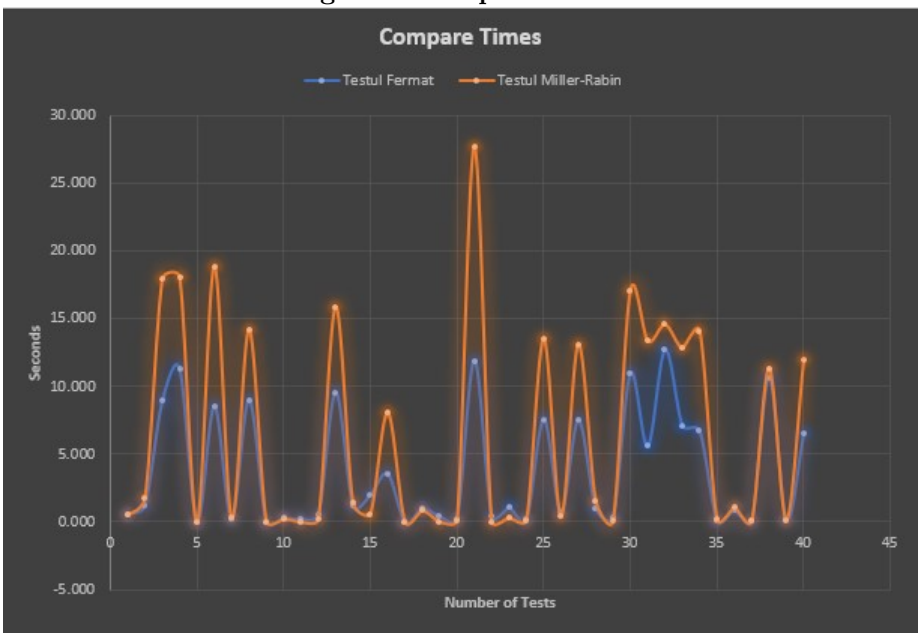
Figura 1 Fermat Times

Numar Test	Timp (secunde)
Testul 1	0.600
Testul 2	1.239
Testul 3	8.930
Testul 4	11.277
Testul 5	0.016
Testul 6	8.521
Testul 7	0.235
Testul 8	9.023
Testul 9	0.019
Testul 10	0.272
Testul 11	0.228
Testul 12	0.556
Testul 13	9.524
Testul 14	1.171
Testul 15	2.003
Testul 16	3.556
Testul 17	0.002
Testul 18	0.999
Testul 19	0.424
Testul 20	0.202
Testul 21	11.899
Testul 22	0.433
Testul 23	1.087
Testul 24	0.161
Testul 25	7.554
Testul 26	0.771
Testul 27	7.495
Testul 28	1.000
Testul 29	0.288
Testul 30	10.944
Testul 31	5.592
Testul 32	12.692
Testul 33	7.118
Testul 34	6.704
Testul 35	0.148
Testul 36	0.896
Testul 37	0.092
Testul 38	10.691
Testul 39	0.192
Testul 40	6.490

Figura 2 Miller-Rabin Times

Numar Test	Timp (secunde)
Testul 1	0.512
Testul 2	1.761
Testul 3	17.933
Testul 4	18.062
Testul 5	0.006
Testul 6	18.875
Testul 7	0.294
Testul 8	14.196
Testul 9	0.044
Testul 10	0.222
Testul 11	0.011
Testul 12	0.231
Testul 13	15.885
Testul 14	1.389
Testul 15	0.596
Testul 16	8.065
Testul 17	0.003
Testul 18	0.830
Testul 19	0.011
Testul 20	0.141
Testul 21	27.702
Testul 22	0.022
Testul 23	0.344
Testul 24	0.140
Testul 25	13.520
Testul 26	0.396
Testul 27	13.116
Testul 28	1.590
Testul 29	0.092
Testul 30	17.103
Testul 31	13.365
Testul 32	14.576
Testul 33	12.853
Testul 34	14.117
Testul 35	0.161
Testul 36	1.102
Testul 37	0.060
Testul 38	11.349
Testul 39	0.156
Testul 40	11.951

Figura 3 Compare Times



In doilea rand, am intocmit o statistica care calculeaza procentajul de numere prime gasite in intervalul dat.

Deoarece algoritmul *Fermat* nu este la fel de precis precum *Miller-Rabin*, ne asteptam ca acesta sa returneze procentaje mai mari, gasind astfel ca fiind prime, numere compuse.

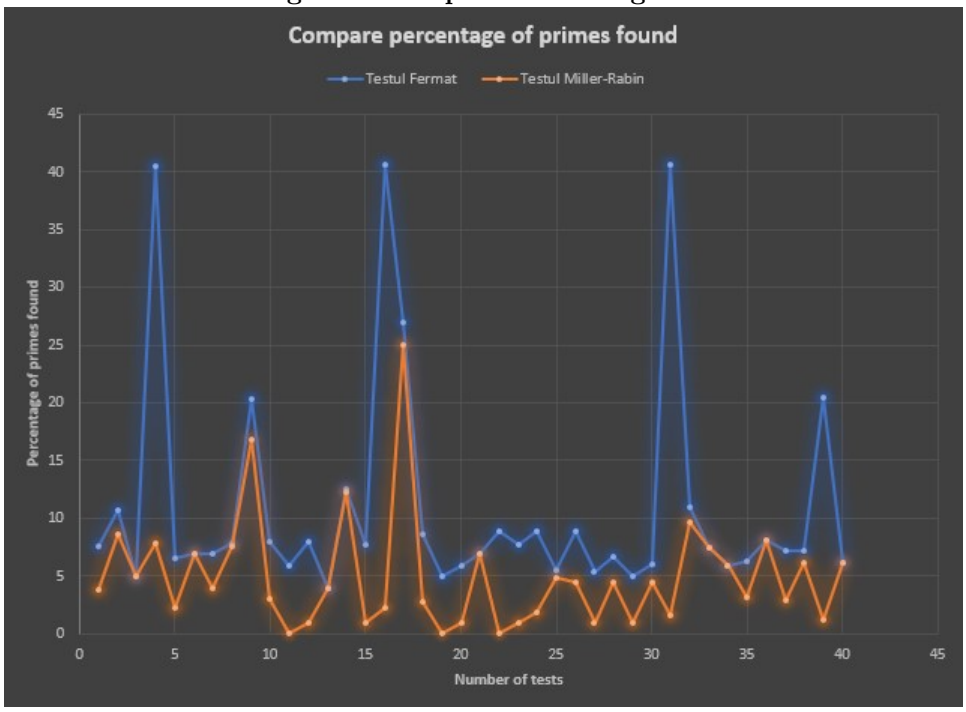
Figura 4 Fermat Percentage of primes found

Numar Test	Percentage of primes(%)
Testul 1	7.547
Testul 2	10.678
Testul 3	4.950
Testul 4	40.536
Testul 5	6.521
Testul 6	6.930
Testul 7	6.930
Testul 8	7.779
Testul 9	20.300
Testul 10	7.920
Testul 11	5.940
Testul 12	7.920
Testul 13	3.960
Testul 14	12.520
Testul 15	7.641
Testul 16	40.603
Testul 17	27.000
Testul 18	8.582
Testul 19	4.950
Testul 20	5.940
Testul 21	6.930
Testul 22	8.910
Testul 23	7.641
Testul 24	8.910
Testul 25	5.494
Testul 26	8.910
Testul 27	5.394
Testul 28	6.693
Testul 29	4.950
Testul 30	6.039
Testul 31	40.627
Testul 32	10.993
Testul 33	7.492
Testul 34	5.940
Testul 35	6.289
Testul 36	8.091
Testul 37	7.142
Testul 38	7.199
Testul 39	20.479
Testul 40	6.093

Figura 5 Miller-Rabin Percentage of primes found

Numar Test	Percentage of primes(%)
Testul 1	3.773
Testul 2	8.609
Testul 3	4.950
Testul 4	7.862
Testul 5	2.173
Testul 6	6.930
Testul 7	3.960
Testul 8	7.592
Testul 9	16.800
Testul 10	2.970
Testul 11	0.000
Testul 12	0.990
Testul 13	3.960
Testul 14	12.290
Testul 15	0.996
Testul 16	2.269
Testul 17	25.000
Testul 18	2.794
Testul 19	0.000
Testul 20	0.990
Testul 21	6.930
Testul 22	0.000
Testul 23	0.996
Testul 24	1.890
Testul 25	4.895
Testul 26	4.395
Testul 27	0.990
Testul 28	4.395
Testul 29	0.990
Testul 30	4.395
Testul 31	1.586
Testul 32	9.592
Testul 33	7.492
Testul 34	5.940
Testul 35	3.144
Testul 36	8.091
Testul 37	2.857
Testul 38	6.139
Testul 39	1.149
Testul 40	6.093

Figura 6 Compare Percentages



3.4 Prezentarea succinta a valorilor obtinute pe teste. Daca apar valori neasteptate, incercati sa oferiti o explicatie)

Dupa cum se poate observa din figurile 1, 2 si 3 timpii de rulare ai algoritmului *Miller-Rabin* sunt in medie mai mari decat cei ai algoritmului *Fermat*, datorita complexitatii ridicate a celui dintai.

De asemenea trebuie mentionat faptul ca am folosit o varianta neoptimizata a algoritmului Miller-Rabin, cu toate acestea dat fiind numarul in plus de verificari si calcule pe care testul de primalitate le efectueaza, rezultatele par a fii in concordanta cu asteptarile.

Lucrurile devin interesante odata cu introducerea figurilor 4,5 si 6 care prezinta cateva discrepante majore intre cei doi algoritmi pe anumite teste.

Rezultatele surprinzatoare pot fi explicate prin faptul ca folosim acelasi numar de iteratii pentru a compara output-ul generat de fiecare algoritm in parte. In medie, algoritmul *Fermat* ar avea nevoie de un factor de amplificare al numarului de iteratii de 5,10 sau poate chiar 50 pentru a concura cu output-ul generat de testul *Miller-Rabin.*(conform testelor proprii)

4 Concluzii

Algoritmii concureaza intre ei, ambii fiind folositi in practica. In urma unei serii de testari pe seturi de date de diferite dimensiuni, am ajuns la concluzia ca algoritmul *Miller-Rabin* intoarce raspunsul corect mai des fata de algoritmul *Fermat*.

Desi toate semnele il indica pe testul *Miller-Rabin* ca fiind cel net superior, nu trebuie scapat de sub vedere faptul ca in industrie **OpenPFGW** foloseste doar *Fermat* pentru testare, de asemenea **Pretty Good Privacy** se bazeaza tot pe *Fermat*.

As alege algoritmul *Miller-Rabin* deoarece, in varianta optimizata, timpii de executie sunt comparabili cu ai algoritmului *Fermat*, insa probabilitatea ca raspunsul intors de acest algoritm sa fie corect este mult mai mare.

Bibliografie

1. http://easymathbeyondprimenumbers.com/importance-of-prime-numbers-in-everyday-life/
October 28, 2021

2. https://math.stackexchange.com/questions/43119/real-world-applications-of-prime-numbers
October 28, 2021

3. https://research-repository.griffith.edu.au/bitstream/handle/10072/42664/73794_1.pdf?sequence=1
October 29, 2021

4. http://fs.unm.edu/EnciclopediaNumerelor.pdf
October 29, 2021

5. https://theconversation.com/why-do-we-need-to-know-about-prime-numbers-with-millions-of-digits-89878
November 1, 2021

6. https://www.britannica.com/science/Fermats-theorem
November 1, 2021

7. https://crypto.stanford.edu/pbc/notes/numbertheory/millerrabin.html
November 1, 2021

8. https://www.cmi.ac.in/ shreejit/primality.pdf
November 2, 2021

9. https://www.geeksforgeeks.org/primality-test-set-2-fermet-method/
November 17, 2021

10. https://www.geeksforgeeks.org/primality-test-set-3-miller-rabin/
November 17, 2021

11. https://math.stackexchange.com/questions/379592/computing-the-running-time-of-the-fermat-primality-test
December 5, 2021

12. https://www.baeldung.com/cs/miller-rabin-method
December 11, 2021

13. https://codeforces.com/blog/entry/50202
December 12, 2021