Tema Analiza Algoritmilor
Identificarea numerelor prime

Olteanu Maria-Teona 321CA

Facultatea de Automatica și Calculatoare
Universitatea Politehnica Bucuresti
maria_teona.olteanu@stud.acs.upb.ro

1 Introducere

1.1 Descrierea problemei rezolvate

Distingerea numerelor prime intregi de cele compuse si factorizarea celor compuse in factori primi reprezinta probleme esentiale in teoria numerelor. Acestea au captat atentia matematicienilor, iar si iar, timp de secole. Folosind cele mai cunoscute tehnici, putem identifica numere prime cu sute si chiar cu mii de cifre; insa, factorizarea acelorasi numere este imposibila la momentul actual pana si pentru cele mai puternice calculatoare.
Cu toate acestea, incepand cu secolul XX, intrebarile legate de detectia numerelor prime si factorizare au fost identificate drept probleme de importanta practica, si o parte fundamentala a matematicii aplicate. Aparitia sistemelor si algoritmilor criptografici care folosesc numere prime mari, precum RSA, a reprezentat principalul resort al dezvoltarii unor metode rapide si eficiente de identificare a numerelor prime.

Aplicatie practica: RSA, creat de Rivest, Shamir si Adleman in 1978, este un sistem de securitate in care fiecare persoana are o cheie publica si o cheie privata asociata. Mesajele pot fi criptate de oricine, folosind cheia publica a unei alte persoane, dar aceste mesaje pot fi decriptate doar de proprietarul cheii private. Ideea acestui criptosistem are la baza faptul ca este usor sa gasesti numere prime mari, dar greu sa factorizezi produsul lor. Cheia publica este compusa din 2 numere, unde unul este inmultirea a doua numere prime mari. Cheia privata este obtinuta de asemenea prin intermediul a celor doua numere prime.

Daca factorizarea ar fi usoara, spargerea criptosistemului RSA ar fi la fel. Chiar daca nu a fost demonstrat inca, se considera ca factorizare va ramane o problema grea. Asadar, presupunand ca am reusi sa factorizam numere de 300 de cifre, ar fi imposibil sa factorizam numere de 400 de cifre.

Siguranta criptari mizeaza in totalitate pe dimensiunea cheii. Daca dublam sau triplam dimensiunea acesteia puterea criptarii creste exponential. Ar trebui sa folosim intregi primi mari (cu dimensiuni de sute si chiar de peste o mie de biti)

pentru a rezista posibilelor atacuri ce folosesc arta factorizarii.
Asadar, ne confruntam cu sarcina de a gasi numere prime mari. Mai exact, avem nevoie de o metoda eficienta de a distinge intre numere prime si compuse. Daca reusim asta, trebuie doar sa cautam numere mari pana gasim unul prim. Fapt ce conduce la urmatoarea intrebare: Cum testam pentru a determina daca un numar este prim?

1.2 Specificarea solutiilor alese

In sectiunea aceasta, vom vorbi despre solutiile alese pentru gasirea numerelor prime, in speta algoritmii probalistici Fermat si Miller-Rabin.
Algoritmii probalistici sunt de obicei mai rapizi, dar nu sunt mereu precisi. Acestia determina de fapt daca un numar p indeplineste una sau mai multe conditii, pe care toate numerele prime le satisfac cu siguranta. Deci daca p nu satisface aceste conditii, p e un numar compus. Daca p satisface toate conditiile, probabil este prim, dar n nu trebuie neaparat sa fie prim.

Testul lui Fermat deriva din Mica Teorema a lui Fermat:

Theorem 1. *Daca p este numar prim, atunci* $a^{p-1} \equiv 1 \pmod{p}$*, pentru orice numar intreg* $0 < a < p$*, cu a si p coprime (cmmdc(a, p) = 1).*

Modificand teorema, vom obtine:

Proposition 1. *Daca* $a^{p-1} \not\equiv 1 \pmod{p}$*, pentru orice numar intreg* $0 < a < p$*, a si p coprime, atunci p este un numar compus.*

Expresia de mai sus este absoluta, este valabila pentru orice numar p (nu exista exceptii).
Insa, reciproca teoremei lui Fermat nu este mereu adevarata:

Proposition 2. *Daca* $a^{p-1} \equiv 1 \pmod{p}$*, pentru orice numar intreg* $0 < a < p$*, a si p coprime, atunci p este un numar prim.*

Exista numere compuse q, care verifica teorema pentru anumite valori ale lui a; aceste numere se numesc PSEUDOPRIME. Testul Fermat intoarce un rezultat fals-pozitiv pentru numere pseudoprime.
Exemplu: Pentru a = 2 si p = 341 = 11 * 31 (compus), dar $2^{340} \equiv 1 \pmod{341}$.
Spunem ca 341 este un pseudoprim Fermat (la baza 2).
De asemenea, trebuie sa avem in vedere ca exista numere naturale compuse q, astfel incat q | a^{q-1} - 1 pentru toate valorile lui a cu cmmdc(a, q) = 1. Astfel de numere sunt cunoscute drept numere CARMICHAEL(pseudoprime absolute).

Folosim Mica Teorema a lui Fermat ca un test pentru identificarea numerelor prime si nu drept o demonstratie. Daca un numar pica testul lui Fermat inseamna ca este compus, altfel exista o probabilitate ca numarul sa nu fie cu adevarat prim, ci doar pseudoprim.

O cale de a evalua probabilitatea ca p sa fie prim este sa incercam testul pentru mai multe baze a_1, a_2, a_3,.... Astfel, testam pentru mai multe baze pana cand numarul de iteratii ales se termina sau pana cand testul returneaza fals, adica numarul este compus. Daca numarul de iteratii se termina, atunci programul va marca numarul ca fiind probabil prim.

Algoritmul Miller-Rabin imbunatateste testul Fermat prin doua modificari; testeaza pentru mai multe baze a, alese aleatoriu si se foloseste de faptul ca p este prim doar daca radacinile ecuatiei $x^2 \equiv 1 \pmod{p}$ sunt $x_{1,2} = \pm 1$. In cazul acesta, daca un numar p trece testul Fermat, trebuie de asemenea sa verifice relatia $a^{(p-1)/2} = \pm 1$.

Algoritmul Miler-Rabin poate determina eficient ca orice numar Carmichael este compus, insa exista si sansa (destul de mica) ca acesta sa faca erori, sa identifice un numar compus ca fiind prim. Dar, spre deosebire de testul lui Fermat, probabilitatea de eroare a testului Miller-Rabin nu tine de numarul p pentru care vrem sa facem verificarea (nu exista input-uri nefavorabile), ci depinde de numarul de iteratii sau de norocul alegerii unor valori pentru baza a.

1.3 Criteriile de evaluare pentru solutia propusa

In evaluarea celor doua solutii propuse vom lua in considerare 2 criterii: acuratetea si timpul necesar rularii.
Pentru a determina si compara acuratetea acestora vom alcatui teste ce contin atat numere prime, cat si numere pseudoprime, Carmichael sau pseudoprime "puternice". In paralel, pentru a verifica output-ul acestora vom rula o functie care va identifica cu siguranta daca un numar este prim sau compus.
Pentru a evalua rapiditatea celor doi algoritmi, testele vor fi realizate in asa fel incat sa contina liste cu dimensiuni diferite de numere prime. Astfel, vom putea compara timpii de executie.
In cazul ambelor criterii, este important ca seturile de date sa contina si numere mari, cu peste 10 cifre, pentru a simula utilizarea testelor in practica.
In plus, in urma rezultatelor experimentale obtinute, am putea determina imbunatatiri sau optimizari pentru solutiile propuse.

2 Implementarea solutiilor

2.1 Descrierea modului in care functioneaza algoritmii alesi

2.1.1 Algoritmul Fermat

Vom determina daca un numar p dat este compus sau probabil prim, verificand expresia a^{p-1} mod p $\neq 1$, unde a este un numar intreg ales la intamplare, astfel incat $2 \leq a \leq p-2$. Daca expresia este adevarata atunci p este un numar COMPUS, altfel putem spune ca p este PSEUDOPRIM.
Testarea poate fi realizata folosind o singura baza a sau mai multe baze alese aleator.

Testul Fermat pentru o singura baza
Testul Fermat pentru o singura baza este eficient din punct de vedere al complexitatii timpului, dar nu este precis. Dupa cum am precizat mai sus, exista input-uri care pot determina un rezultat fals-pozitiv.
Putem imbunatatii acest algoritm, verificand expresia pentru mai multe baze. Numarul de baze generate random va fi dat de numarul de iteratii.

Testul Fermat randomizat
Aplicam algoritmul de mai multe ori pentru baze diferite generate aleatoriu pana cand numarul de iteratii dat se termina sau pana cand expresia a^{p-1} mod p $\neq$ 1 este adevarata, adica p este COMPUS. Daca am testat pentru toate bazele (s-au executat toate iteratiile) atunci numarul va fi identificat ca PSEUDOPRIM (probabil prim).

2.1.2 Algoritmul Miller-Rabin
Vom determina daca un numar p dat este compus sau probabil prim, cautand o baza dintre cele generate random care este "martor" al proprietatii numarului p de a fi compus. Daca gasim un "martor", atunci numarul p este COMPUS. Altfel, daca numarul de iteratii s-a termina si nu am descoperit nici un "martor", p este PROBABIL PRIM.

Identificarea unui "martor" a:
Alegem o baza oarecare a, cu $2 \leq a \leq p-2$.
Calculam u si t astfel incat p - 1 = u * 2^t, unde u este un numar impar si t ≥ 1. Daca a^u mod p $\neq 1$ sau $a^{2^i u}$ mod p $\neq$ p - 1, cu i = 0,...,t (am gasit o radacina patrata $a^{2^i u} \neq \pm 1$), atunci baza a este "martor" al faptului ca numarul p este compus.

2.2 Analiza complexitatii solutiilor

2.2.1 Testul Fermat

```
function ModularExp(b, e, m)                          ▷ computes b^e % m
    result = 1
    while e do                                        ▷ O(loge)
        if e is odd then
            result = (result * b) % m
        end if
        e = e/2
        b = (b * b) % m
    end while
    return result
end function
```

Calculul operatiei (b * b) % m presupune o complexitate O($log^2 m$) si va fi executata de $log_2 e$ ori. Complexitatea algoritmului este O($log^2 m$ * $loge$) = O($log^2 m$ * $bit\_len^2(e)$) = O($log^2 m$).
Asadar, complexitatea pentru a calcula restul a^{p-1} mod p este:

$$\boxed{O(\log^2 p)}$$

```
function FermatTest(p, k)
    for i = 1 to k do                                 ▷ O(k)
        a = random(2, p − 2)
        if modularExp(a, p - 1, p) != 1  then         ▷ O(log^2 p)
            return COMPOSITE
        end if
    end for
    return PSEUDOPRIME
end function
```

Complexitatea testului Fermat pentru un numar intreg p dat si k iteratii este:

$$\boxed{O(k{*}\log^2 p)}$$

2.2.2 Testul Miller-Rabin

function WITNESS(a, p)
  let t and u be such that t $\geq$ 1, u is odd, p - 1 = u * 2^t
  $x_0 = modularExp(a, u, p)$
  for i = 1 to t **do** ▷ O(t) = O($logp$)
    $x_i = x_{i-1}^2$ % p ▷ O($log^2 p$)
    if x_i == 1 and $x_{i-1} \neq 1$ and $x_{i-1} \neq$ p - 1 **then**
      return TRUE
    end if
  end for
  if $x_t \neq 1$ **then**
    return TRUE
  else
    return FALSE
  end if
end function

Calculul operatiei x_{i-1}^2 % p presupune o complexitate O($log^2 p$) si va fi executata de $t \approx logp$ ori.
Asadar, complexitatea pentru algoritmului witness este:

$$\boxed{O(\log^2 p * logp) = \mathrm{O}(log^3 p)}$$

function MILLERRABINTEST(p, k)
  for i = 1 to k **do** ▷ O(k)
    $a \leftarrow random(2, p-2)$
    if witness(a, p) **then** ▷ O($log^3 p$)
      return COMPOSITE
    end if
  end for
  return PSEUDOPRIME
end function

Complexitatea testului Miller-Rabin pentru un numar intreg p dat si k iteratii este:

$$\boxed{O(k{*}\log^3 p)}$$

Folosind FFT pentru inmultire, complexitatea poate fi redusa, astfel incat:

$$\boxed{O(k{*}\log^2 p)}$$

2.3 Prezentarea principalelor avantaje si dezavantaje pentru solutiile luate in considerare

2.3.1 Algoritmul Fermat

Fiind un algoritm probabilistic, algoritmul Fermat este rapid, dar nu este mereu precis.
Un dezavantaj ar fi faptul ca exista multe numere compuse (Carmichael sau "pseudoprime puternice") care sunt identificate ca fiind prime de testul Fermat pentru o anumita baza, dar marcate ca fiind compuse pentru aceeasi baza de testul Miller-Rabin. Daca nu avem norocul de a alege o baza care sa fie factor al unui numar Charmichael, testul Fermat va intoarce un rezulatat fals-pozitiv. Cu toate acestea, probabilitatea ca acest test sa faca o eroarea pentru un numar mare generat random este aproape zero. Exista o infinitate de numere Carmichael dar acestea sunt rare: doar 255 de numere Charmichael mai mici decat 100,000,000.
Trebuie sa avem in vedere ca exista posibilitatea ca numerele propuse pentru testare sa nu fie alese aleatoriu. In acest caz, trebuie sa gasim o varianta mai buna pentru testare din punct de vedere al acuratetei.

Un avantaj ar fi acela ca atunci cand nu testam pentru numere Charmichael, testul Fermat intoarce rezultate fals-pozitive destul de rar. Daca testam pentru k iteratii, adica sunt alese k valori de baze la intamplare, probilitatea ca testul Fermat sa faca o eroare este de $\frac{1}{2^k}$. Astfel, pentru o valoare foarte mare a lui k, rata de eroare ar tinde spre zero.
De asemenea, asa cum am mentionat mai sus, algoritmul Fermat este mai rapid prin comparatie cu alti algoritmi utilizati pentru identificarea numerelor prime, mai aleas pentru testarea cu o singura baza.

2.3.2 Algoritmul Miller-Rabin

Un avantaj este faptul ca testul Miller-Rabin este mult mai precis. Spre deosebire de testul Fermat, probabilitatea ca testul Miller-Rabin sa intoarca un rezultat corect nu depinde de numarul p dat ca input pentru verificare.
Algoritmul Miller-Rabin este de asemenea probabilistic, deci exista sansa ca acesta sa faca erori, dar este foarte mica: $\frac{1}{4^k}$, unde k este numarul de iteratii. Deci, corectitudinea testului depinde exclusiv de numarul de iteratii sau de norocul generarii unor valori pentru baze.

Neoptimizat, algoritmul Miller-Rabin este mai lent decat algoritmul Fermat. Insa, in practica acesta este imbunatatit in asa fel incat, cei doi algoritmi ajung sa aiba aceeasi performanta.

Este important de mentionat si faptul ca daca testam pentru numere pe 64 de biti sau mai mici, putem implementa o varianta determinista a testului Miller-Rabin. In comparatie, acest test are rezultate mai bune atat din punct de vedere

al timpului de executie, cat si al preciziei. La acest moment, cea mai eficienta varianta determinista, presupune 3 iteratii pentru testarea numerelor pe 32 de biti si 7 iteratii pentru numere pe 64 de biti.

3 Evaluare

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

Testarea este impartita pe doua criterii: timp si acuratete.

3.1.1 Timpul de executie

Am creat 30 de teste pentru acest criteriu. Vrem sa evaluam si sa comparam timpul de executie al celor doi algoritmi, testand pentru diferite secvente de numere prime.

Primele 20 de teste contin secvente de numere prime de dimensiuni diferite dintr-o baza de date [4]. In 15 din acestea, fiecare secventa include doar numere cu acelasi numar de cifre. Scopul acestor teste este de a determia cum numarul de iteratii si dimensiunea numerelor afecteaza complexitatea temporala.

Celelate 10 teste contin liste de numere prime de dimensiuni diferite generate random, pentru a simula utilizarea algoritmilor in practica. Acestea sunt realizate folosind un generator pseudorandom, "Mersenne Twister", si un algoritm care identifica cu siguranta daca un numar este prim sau nu.

3.1.2 Acuratete

Am implementat 25 de teste pentru acest criteriu. Vrem sa evaluam precizia algoritmilor, testand pentru numere Charmichael, "pseudoprime puternice" si intregi random.

Pentru 10 teste am construit liste de numere Charmichael sau "pseudoprime puternice" [5], astfel incat sa identificam de cate iteratii avem nevoie pentru ca testele sa aiba output-uri corecte.

In alte 5 teste, am creat atat secvente de numere intregi mici (e.g. primele 500000 de numere intergi), cat si de numere mari de 9 si 10 cifre. Vrem sa determinam daca dimensiunea numerelor reprezinta un factor in contextul probabilitatii de a produce rezultate fals-pozitive.

Restul testelor sunt construite pe baza unor liste de numere intregi de dimensiuni variate generate aleator cu ajutorul unui generator pseudorandom "Mersenne Twister". De asemenea, scopul acestor teste este de a reproduce utilizarea algoritmilor in industrie.

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

- **Laptop:** ASUS VivoBook S14 (X406UAR)
- **Procesor:** Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz 1.99GHz
- **Memorie disponibila:** 3,12 GB
- **Sistem de operare:** Windows 10 Pro

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

3.3.1 Timpul de executie

Pentru a determina timpul de executie al unui test, am rulat acel test de mai multe ori si am facut media rezultatelor.
In urmatoarele tabele este prezentat timpul de executie pentru fiecare test in milisecunde. Am organizat testele in tabele in functie de numarul de iteratii.

Trebuie mentionat si faptul ca testul Miller-Rabin determinist nu este influentat de numarul de iteratii.

Testare cu 2 iteratii.

Nr. test	**Nr. elemente**	**Nr. cifre element**	**Nr. iteratii**	**Fermat**	**Miller-Rabin**	**Miller-Rabin determinist**
12	1000	6	2	1.803	1.773	**0.734**
13	500	9	2	1.481	1.045	**0.737**
14	500	10	2	1.344	1.429	**0.935**
21	46748	random	2	104.214	99.187	**83.618**
22	47782	random	2	102.709	103.436	**78.197**
23	47142	random	2	100.185	100.342	**84.270**
24	46797	random	2	98.057	107.210	**83.993**
25	46864	random	2	99.186	103.267	**64.704**
26	230	random	2	0.447	0.705	**0.291**
27	227	random	2	0.441	0.461	**0.288**
28	280	random	2	0.564	0.569	**0.358**

Testare cu 3 iteratii.

Nr. test	**Nr. elemente**	**Nr. cifre element**	**Nr. iteratii**	**Fermat**	**Miller-Rabin**	**Miller-Rabin determinist**
19	600	≤ 4	3	1.037	1.057	**0.219**
20	22000	10	3	57.123	59.068	**38.643**
29	2358	random	3	6.288	6.600	**3.857**
30	47848	random	3	123.635	119.508	**78.384**

Testare cu 4 iteratii.

Nr. test	**Nr. elemente**	**Nr. cifre element**	**Nr. iteratii**	**Fermat**	**Miller-Rabin**	**Miller-Rabin determinist**
1	50	≤ 3	4	0.101	0.084	**0.015**
2	100	≤ 3	4	0.174	0.204	**0.033**
3	50000	≤ 6	4	113.194	115.087	**31.454**
4	50000	6 & 7	4	116.469	114.961	**40.080**
5	100000	≤ 7	4	225.943	233.297	**79.272**
6	1000	6	4	2.119	2.053	**0.552**
7	500	9	4	1.290	1.386	**0.718**
8	500	10	4	1.373	1.436	**0.612**
9	4000	7	4	9.717	10.183	**5.168**
10	2000	9	4	6.262	6.150	**2.631**
11	20500	10	4	62.390	69.216	**36.527**

Testare cu 8 iteratii.

Nr. test	**Nr. elemente**	**Nr. cifre element**	**Nr. iteratii**	**Fermat**	**Miller-Rabin**	**Miller-Rabin determinist**
15	50000	≤ 6	8	161.238	165.832	**30.179**
16	1000	6	8	3.124	3.325	**0.624**
17	500	9	8	2.004	2.625	**0.705**
18	500	10	8	2.108	2.938	**0.794**

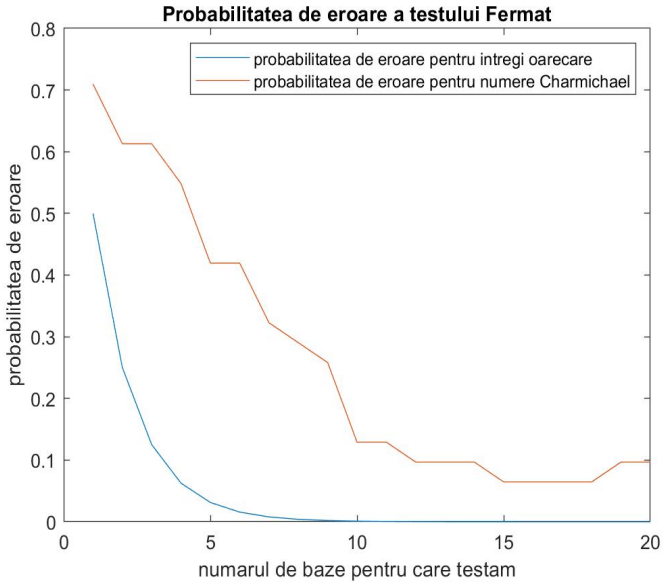
In urmatoarele doua tabele de mai jos, putem observa timpii de executie pentru teste diferite ce contin secvente de 1000 de numere prime.(urmarim numarul de cifre al tuturor elementelor din secventa)

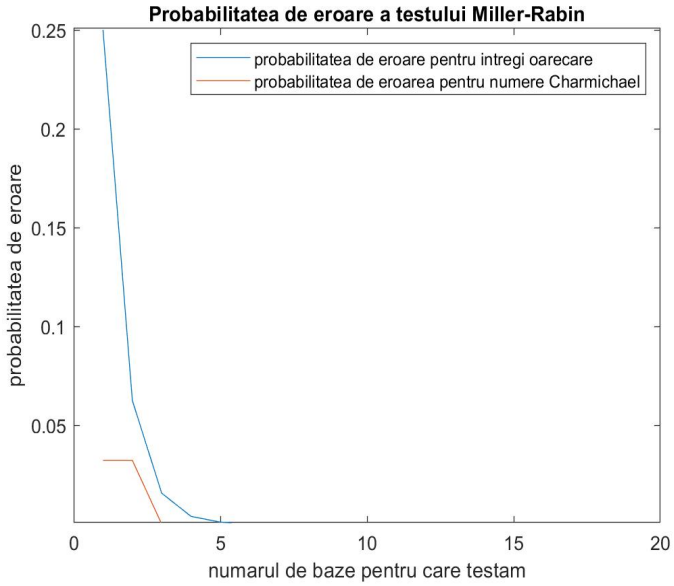
Nr. cifre element	**Nr. iteratii**	**Fermat**	**Miller-Rabin**
≤ 4	2	1.138	1.683
6	2	1.920	1.952
9	2	1.962	2.307
10	2	2.534	2.646
random	2	2.416	2.670

Nr. cifre element	Nr. iteratii	Fermat	Miller-Rabin
≤ 4	16	3.609	3.966
6	16	5.473	5.583
9	16	7.376	7.651
10	16	9.360	9.861
random	16	7.454	9.135

3.3.2 Acuratete

In graficele urmatoare vom ilustra probabilitatea de eroare a celor doua teste, care primesc ca input o lista de numere Charmichael:





In cazul acesta, testului Miller-Rabin in varianta determinista este 100% precis, nu face erori.

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

3.4.1 Timpul de executie

Avand in vedere complexitatile algoritmilor si rezultatele experimentale, putem spune cu siguranta ca marirea numarului de iteratii, creste timpul de executie.

Cele doua teste au timpi de executie destul de apropiati, cea mai mare diferenta inregistrata fiind de aproximativ 6 milisecunde. Dar per total, diferentele sunt destul de mici intre testul Fermat si Miller-Rabin.
Prin comparatie, testul Fermat devine in medie mai rapid decat testul Miller-Rabin, cu cat crestem numarul de baze utilizate (numarul de iteratii). Pentru 2, 3 si 4 iteratii, exista teste care au fost executate mai repede cu ajutorul algoritmului Miller-Rabin, insa pentru 8 iteratii, toate testele Fermat au timpi de executie mai mici.

Dupa cum se poate observa in ultimele doua tabele, dimensiunea numarului dat pentru testare afecteaza complexitatea temporala. Cu cat este mai mare numarul, cu atat creste timpul de executie, indiferent de numarul de iteratii.

Trebuie evidentiat si faptul ca varianta determinista a testului Miller-Rabin este semnificativ mai rapida, decat cele doua teste in varianta lor neoptimizata. In unele cazuri, testul Miller-Rabin determinist este cu peste 50% mai eficient.

3.4.2 Acuratete

Pentru ambele teste, cu cat crestem numarul de iteratii, cu atat imbunatatim corectitudinea acestora.
Cu toate acestea, testul Fermat este mai putin precis decat testul Miller-Rabin, mai ales pentru numere Charmichael.

Al doilea grafic ilustreaza faptul ca numarul de iteratii este factorul care influnteaza in totalitate probabilitatea erorii, ci nu input-ul. Pe cand, pentru testul Fermat exista input-uri nefavorabile.

Pentru testele care contin doar secvente de numere generate random, algoritmii nu a facut nici o eroare.

4 Concluzii

Sunt de parere ca in cazul in care testam numera alese aleatoriu, este putin probabil sa alegem un numar Charmichaeal. Testul Fermat va returna rar rezultate fals-pozitive. Exista in continuare programe de criptare, cum ar fi PGP, care utilizeaza algoritmul Fermat, avand in vedere ca numere primite ca input sunt generate random.
Insa, daca ne ingrijoreaza posibilitatea mica de a testa pentru un numar Charmichael, testul Miller-Rabinar este o alternativa buna.

In urma rezultatelor obtinute, putem spune ca testul Fermat este mai rapid decat testul Miller-Rabin, iar testul Miller-Rabin mai precis decat testul Fermat. In varianta lor optimizata, algoritmii au aceleasi complexitati, de aceea se prefera in general utilizarea testului Miller-Rabin.

In plus, pentru ca in cadrul acestei lucrari, testam numai pentru numere pe 32 de biti putem utiliza o varianta determinista a testului Miller-Rabin. Acest test are cele mai bune rezultate din punct de vedere al eficientei si al acuratetei.
Insa, trebuie sa tinem cont de faptul ca probabilitatea de eroarea afecteaza orice test care identifica daca un numar foarte mare este prim sau compus, indiferent daca este determinist sau nu.

Nu in ultimul rand, este important sa mentionam ca in practica acesti doi algoritmi sunt optimizati si sunt folositi pentru testarea numerelor intregi de dimensiuni mari, pe 1024 de biti sau 2048 de biti. De aceea, rezultatele experimentale acumulate, ar putea sa nu demonstreze adevaratul potential al celor doi algoritmi.

Bibliografie

1. Kenneth H. Rosen: Elementary Number Theory, Sixth Edition, Pearson
2. Thomas H. Cormen, Charles E. Leiserson, Ronald Rivest, Clifford Stein: Introduction to Algorithms, Third Edition, MIT Press
3. LNCS Homepage, http://www.springer.com/lncs. Last accessed 1 Oct 2021
4. Primes, https://primes.utm.edu/lists/small/millions/. Last accessed 20 Nov 2021
5. OEIS, https://oeis.org/. Last accessed 20 Nov 2021