Computer Science & Engineering Department

Identificarea numerelor prime

Soare Ion-Alexandru,
coordonat de prof. dr.
Traian Rebedea

Universitatea Politehnica,
Bucuresti
Anul Universitar 2017-2018

Facultatea de Automatica si
Calculatoare, specializarea CTI
Grupa 323, Seria CD

Cuprins

Introducere

Definitie si apartenenta

Un numar **prim** este un numar natural, nenul, cu proprietatea ca singurii lui divizori pozitivi sunt 1 si el insusi. Studiul lor dateaza din Grecia antica (secolele al V-lea – al III-lea i.H.), in cadrul scolii pitagoreice, unde pasiunea fata de numere era sustinuta de misterul si perfectiunea ce le definea. Primele mentiuni relevante cu privire la acest subiect i se datoreaza lui **Euclid**, care a demonstrat ca exista o infinitate de numere prime[1], dar si ca orice numar se poate scrie, in mod unic, sub forma unui produs format din acestea (demonstratie cunoscuta sub numele de **Teorema Fundamentala a Aritmeticii**[2]). Primul algoritm de determinare a primalitatii a fost inventat de catre matematicianul grec **Eratosthenes,** in secolul al III-lea i.H., abordand o modalitate triviala, bazata pe generare si selectie[3]. Dupa mici progrese aduse de-a lungul anilor, matematicianul francez, **Pierre de Fermat**, a elaborat **Mica Teorema a lui Fermat**, ce avea sa stea la baza analizei primalitatii chiar si in prezent.

Utilitate

Numerele prime au fost vazute, pentru o lunga perioada de timp, drept concepte pur matematice, lipsite de utilitate in viata cotidiana. Ulterior, au devenit baza generarii cheilor publice in domeniul securitatii (**criptare**, spre exemplu: **algoritmul RSA**[4], Diffie-Hellman etc.), avand influente atat in analiza unor specii din natura (insectele **Cicada** din specia Magicicada, care modifica ciclul natural al pradatorilor prin aparitia lor la un interval prim de ani), cat si in viziuna artistilor (modificand metrica melodiilor, metaforizand comunicarea cu extraterestri in diverse opere literare etc.). Nici domeniul militar nu s-a lipsit de intrebuintarea numerelor prime. Matematicianul chinez **Sunzi Suanjing** a venit cu o

solutie la adresa problematicii organizarii armatelor, reusind, astfel, sa determine proportia optima de distribuire a soldatilor chinezi in razboi (**Lema Chineza a Resturilor**[5]).

Astfel, numerele prime dispun de o importanta considerabila in aplicabilitatea tehnologiei din domeniile contemporane, fiind considerate de mai bine de 2000 de ani drept un izvor de mister, perfectiune, dar si de nonconformism.

Prezentarea si implementarea solutiilor

Mica Teorema a lui Fermat

Unul dintre cei mai cunoscuti algoritmi de depistare a primalitatii il constituie **testul Fermat**. In anul 1640, Fermat i-a trimis o scrisoare prietenului sau, Frenicle De Bessy, enuntand Mica Teorema a lui Fermat:

> Consideram ca avem un numar p, prim, si a un numar intreg care nu e divizibil cu p, atunci a^{p-1} - 1 e divizibil cu p.[6]

Matematicianul francez a mai afirmat si ca i-ar fi trimis o demonstratie, daca nu ar fi fost prea lunga. Demonstratia a venit in 1736 prin **Euler**, fiind simplificata ulterior de **Gauss** (1801). Pentru demonstratie ar fi fost necesara teorema lui **Taylor** din 1717[7], sau teoria binomiala a lui **Blaise Pascal** din 1665[8].

Una dintre cele mai populare demonstratii este cea a cerceilor (**combinationala**)[9]. Voi particulariza pentru simplitatea intelegerii:

Consideram ca avem un alfabet binar (0, 1) si avem la dispozitie totalitatea cuvintelor de dimensiune 5. putem forma 2^5 combinatii posibile, si anume:

00000

00001 00010 00100 01000 10000

00011 00110 01100 11000 10001

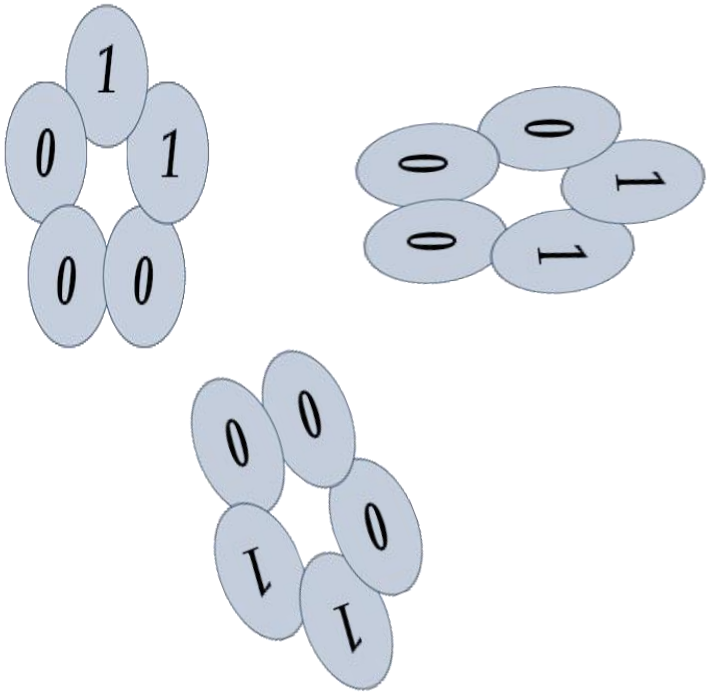
00111 01110 11100 11001 10011

10100 01010 00101 10010 01001

01011 10101 11010 01101 10110

01111 11110 11101 11011 10111

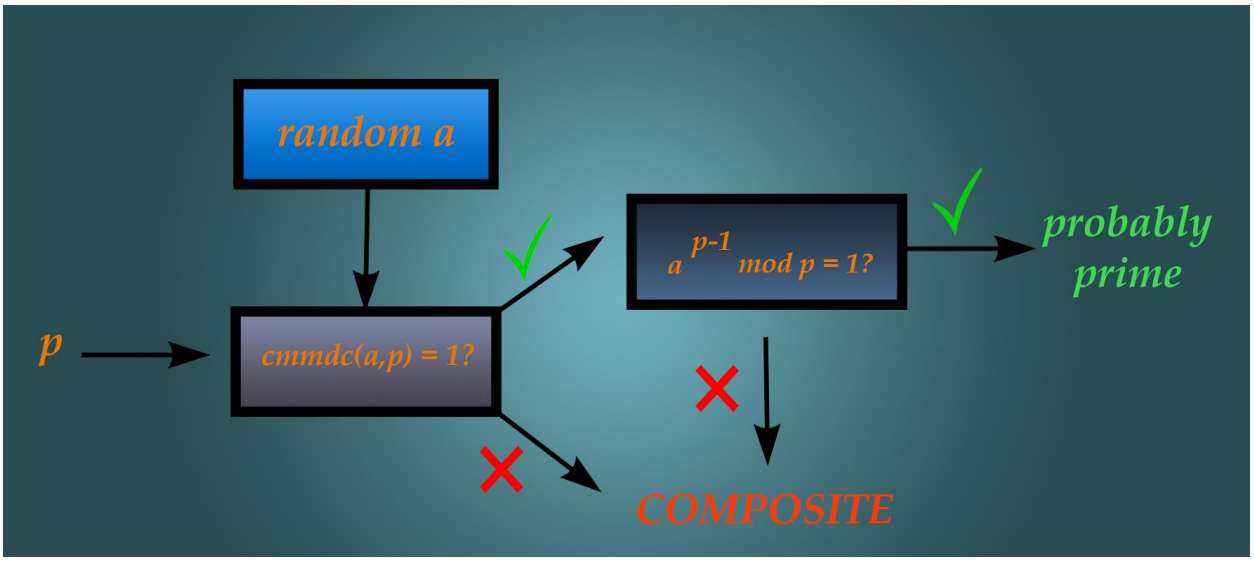
1111



Daca scapam de toate cuvintele ce nu contin ambele litere din alfabet (0000 si 1111), scadem exact cardinalul numarului de litere din alfabet, in cazul de fata 2. Vom ramane cu 2^5 - 2 combinatii posibile = 30, care este divizibil cu 5 (dimensiunea cuvantului). De asemenea, pe fiecare rotatie a unei serie de cuvinte (reprezentate mai sus pe linii), vom avea acelasi cercel, doar mutat din alta perspectiva (figura de mai sus), avand nevoie de 5 rotatii pentru a reveni la pozitia initiala. Asta inseamna ca totalitatea cuvintelor se vor imparti in grupuri egale de dimensiune p (in cazul de fata, 5). In final, ramanem cu 6 combinatii de dimensiune 5 => 6 * 5 = 30. Astfel, rezulta ca $2^5 - 2 = 1 \bmod 5$, deci $2^5 = 2 \bmod 5$. Generalizand, **pentru conditiile din ipoteza**, vom avea $a^p = a \bmod p$.

Testul Fermat

Asadar, s-a pus problema testarii primalitatii unui numar dat, cu o probabilitate ce tinde la 100, intr-un timp rezonabil pentru input-ul primit. Numerele care nu sunt prime se vor numi **compuse**, deoarece se pot compune din produsul mai multora. Ne vom folosi de proprietati pe care le au toate numerele prime, dar si unele (un procentaj foarte mic) numere compuse.

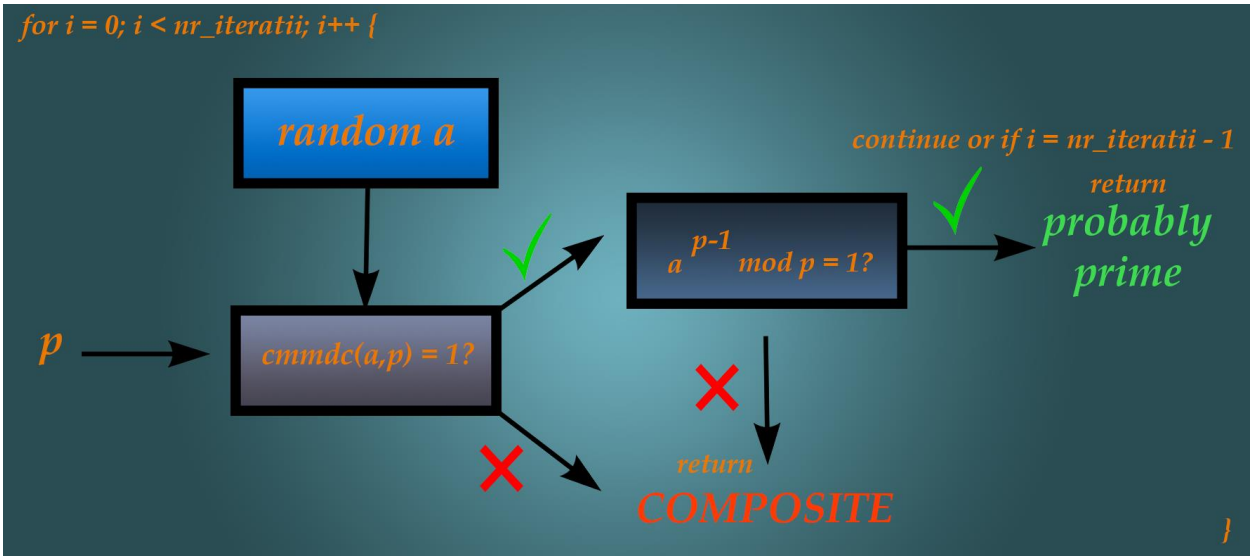


Avem input-ul **p** si un numar random **a** din intervalul (1, p). Ipoteza implica faptul ca a trebuie sa nu fie un **multiplu** de-al lui p pentru a functiona teorema. Vom echivala conditia cu instructiunea: cmmdc(a,p) = 1. Daca a si p sunt prime intre ele, se trece la urmatorul pas, altminteri, numarul este unul compus, deoarece prezinta un divizor. La urmatorul pas se executa **M**ica **T**eorema a lui **F**ermat, unde daca raspunsul este negativ, p este compus, iar daca este afirmativ, p este **probabil prim**. Acest lucru apare datorita faptului ca **MTF** nu este adevarat pentru totalitatea a-urilor generate si nici nu este stabilit un criteriu pentru determinarea lor. Aceste numere care duc la testarea pozitiva a primalitatii unui numar, fiind compuse, se numesc **Fermat liars** sau **Fermat fools**, deoarece ne pacalesc cu privire la primalitate.

Spre exemplu:

Pentru p = 341, vom considera a = 2. In urma testului va rezulta ca 341 va fi probabil prim. Daca a-ul generat va avea valoarea 3, testul ne va spune ca 341 este un numar compus. Cu toate astea, 341 poate fi impartit in factori ca 11 * 31, deci nu este un numar prim. Rezulta ca **a = 2** este un Fermat liar pentru numarul p = 341, in timp ce **a = 3** este un **Fermat witness** (arata comportamentul corect), iar **p = 341** este un **pseudoprim**.

Pentru asigurarea unei precizii cat mai ridicate, este inclus un numar de iteratii, ce reprezinta numarul de executii ale algoritmului.



Este demonstrat faptul ca numarul de fools trebuie sa divida numarul total de numere din care putem alege a-ul. Probabilitatea ca dupa T iteratii sa nu gasim niciun witness este mai mica decat $1 / 2^T$. Astfel, la 20 de iteratii, sansa de a **NU** gasi niciun witness este de aprox **1 la un milion**[10].

Cu toate astea, numerele liars nu sunt cele mai mari probleme pe care le putem intampina in cadrul rularii testului Fermat. Exista unele numere composite, ce trec testul pentru orice a selectat (**cu conditia ca a sa fie relativ prim la p, adica cel mai mare divizor comun al lor sa fie 1**). Acestea se numesc **Carmichael numbers**[11] (spre exemplu, 1105 trece

toate testele de primalitate, fiind echivalentul a 5 * 13 * 17, in cazul in care conditia de relativism prim intre baza si numarul dat este pusa). Ca definitie echivalenta, un numar n este Carmichael daca, pentru orice baza b, $\mathbf{b^n \bmod n = b}$. Diferenta este una subtila si mai greu de sesizat, intrucat toate numerele prime vor indeplini aceasta conditie. Ceea ce este important este faptul ca numerele Carmichael sunt compuse, fiind singurele compuse care satisfac proprietatile respective.

Spre exemplu:

La parcurgerea algoritmului computational, a-ul este generat random cu valori cuprinse intre [2, p-1]. La primirea unui **p = 561** (cel mai mic numar Carmichael = 3 * 11 * 17), testul va returna true pentru orice valori diferite de numerele ce au cmmdc-ul comun diferit de 1 (adica un multiplu al unuia din cei 3 factori). Asadar, diferenta dintre un numar simplu compus, si unul Carmichael este faptul ca cele simple vor returna false pentru orice valoare random primita din intervalul stabilit, cu sau fara conditia divizorului comun, in timp ce Carmichael o sa returneze false **DOAR** pentru gcd(a,p) != 1.

Introducere Miller-Rabin

Numerele Carmichael au dus la necesitatea unor noi algoritmi de testare a primalitatii. In anul 1976, **Gary Miller**, profesor la Universitatea Carnegie Mellon din SUA a elaborat un test de primalitate bazat pe Ipoteza Extinsa Riemann[12], urmand sa fie imbunatatit de **Michael Rabin** in lucrarea *Probabilistic Algorithm for Testing Primality* din 1980[13]. In versiunea originala, algoritmul este unul **deterministic**, depinzand de corectitudinea **ERH** (fiind nedemonstrata pana in prezent, desi testele de-a lungul anilor ii confera credibilitate), devenind **unconditional probabilistic** in urma modificarilor lui Rabin. La fel ca cel al lui Fermat, testul Miller-Rabin se bazeaza pe parcurgerea unor egalitati care returneaza true pentru numerele

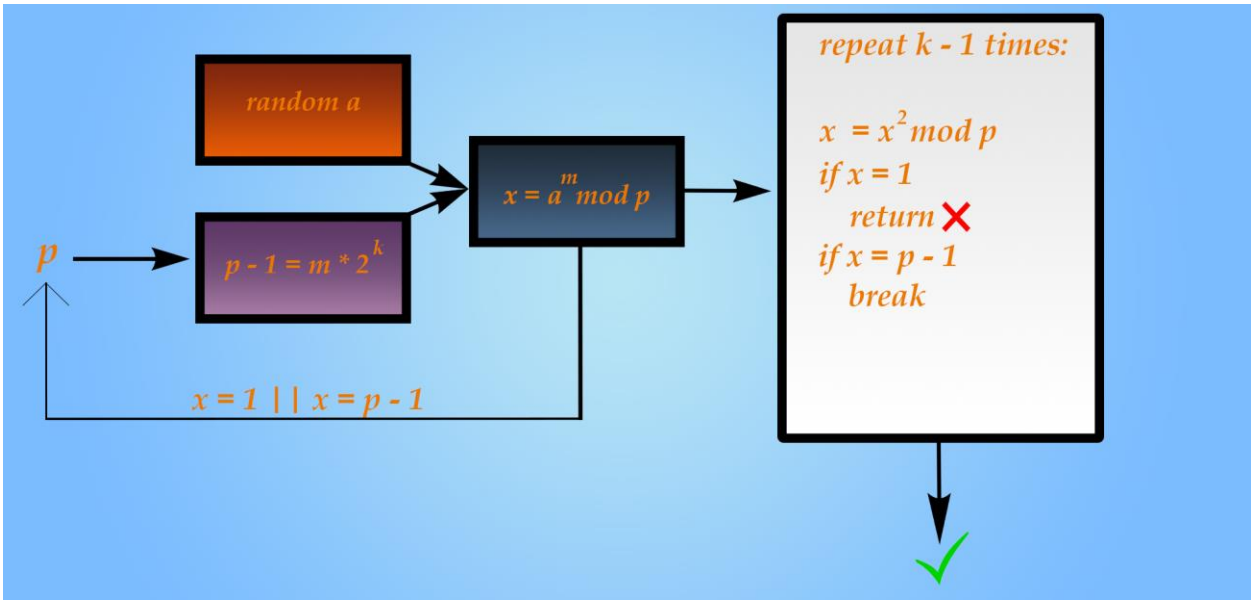
prime. La final, daca au fost trecute toate etapele, concluzionam ca numarul pentru care am testat este prim. Unul dintre conceptele ce sta la baza testului este Lema lui Euclid:

Consideram ca avem un numar p, **prim**. Daca p divide produsul de intregi a*b, p trebuie sa divida ori pe a, ori pe b.[14]

Raportat la subiect, lema de mai sus ne arata ca daca avem **x^2 mod p = 1**, inseamna ca **(x-1)(x+1) mod p = 0**, deci p divide unul dintre termenii x-1 si x+1 => x e congruent cu -1 sau cu 1 mod p.

Testul Miller-Rabin

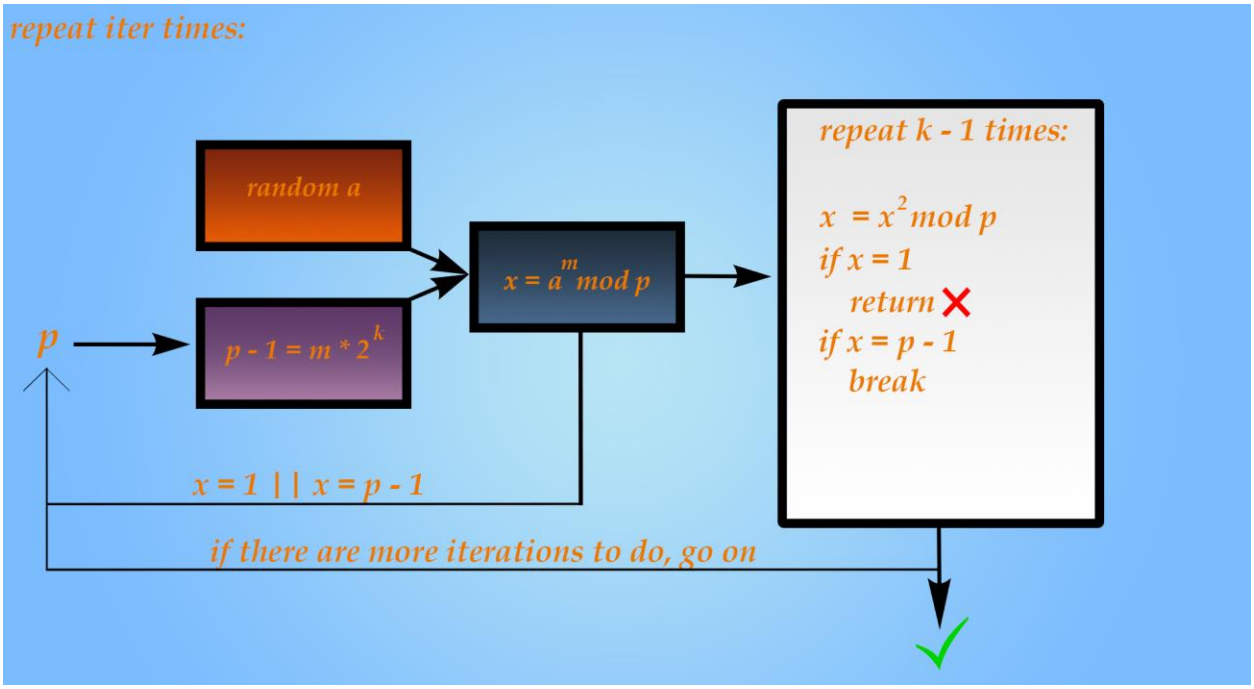
Avem numarul p si vrem sa depistam primalitatea acestuia.



Conform lemei lui Euler, enuntata anterior, stim ca p – 1 este par, in cazul in care p este prim, deci il putem scrie sub forma **2^k * m**. Daca p-1 nu se poate imparti la 2, rezulta ca p este par, deci nu este prim => testul s-a terminat. In continuare, ne vom folosi de o variabila **x** careia ii atribuim initial valoarea **a^m mod p**, unde a este un numar generat random din intervalul [2, p-2]. In cazul in care x-ul rezultat are valoarea 1 sau p -1, algoritmul se va intoarce la inceput. Daca nu, se va trece la pasul urmator,

unde vom utiliza si valoarea puterii lui 2 din descompunere, k. Asadar, se va executa de k-1 operatie $\mathbf{x = x^2 \bmod p}$, unde va returna composite daca rezultatul va fi 1, sau se va intoarce la inceputul loop-ului initial (inceputul programului) pentru valoarea p-1. Pentru orice alta valoare, se va parcurge numarul total de iteratii, urmand sa fie returnat true in cazul neintalnirii unor astfel de valori.

De asemenea, testul Miller-Rabin este unul probabilistic, ceea ce implica faptul ca exista sansa ca unele numere compuse sa fie detectate drept prime. Asemenea testului Fermat, a-urile care functioneaza corect poarta denumirea de witness, in timp ce a-urile ce implica contrariul detectarii primalitatii se numesc **strong liars**. De aceea, este adaugat suplimentar un numar de iteratii pentru a creste acuratetea algoritmului.



Probabilitatea de eroare a algoritmului este egala cu $\mathbf{4^{-iter}}$, ceea ce inseamna ca testul dispune intotdeauna de o sansa de succes de cel putin 75% (in cazul unei singure iteratii)[15], de $\mathbf{2^{-iter}}$ ori mai precis decat testul Solovay-Strassen, dezvoltat in 1977[16].

Exemplu strong liar:

Consideram ca vrem sa determinam primalitatea lui **p = 221**. Vom descompune p – 1 ca: 221 - 1 = 55 * 2^2 . Pentru **a = 174**, dupa o singura iteratie obtinem faptul ca 221 este prim. Daca vom folosi **a = 137**, dupa acelasi numar de iteratii, testul ne returneaza false, ceea ce inseamna ca 221 este compus (13 * 17), iar 174 este un strong liar pentru numarul 221, in timp ce 137 este un witness.

Complexitate si testare

Analiza Fermat

INPUT:

- un numar natural p > 3;
- un parametru k care reprezinta numarul de iteratii (parcurgeri).

OUTPUT:

- compus daca p este compus, altfel probabil prim.

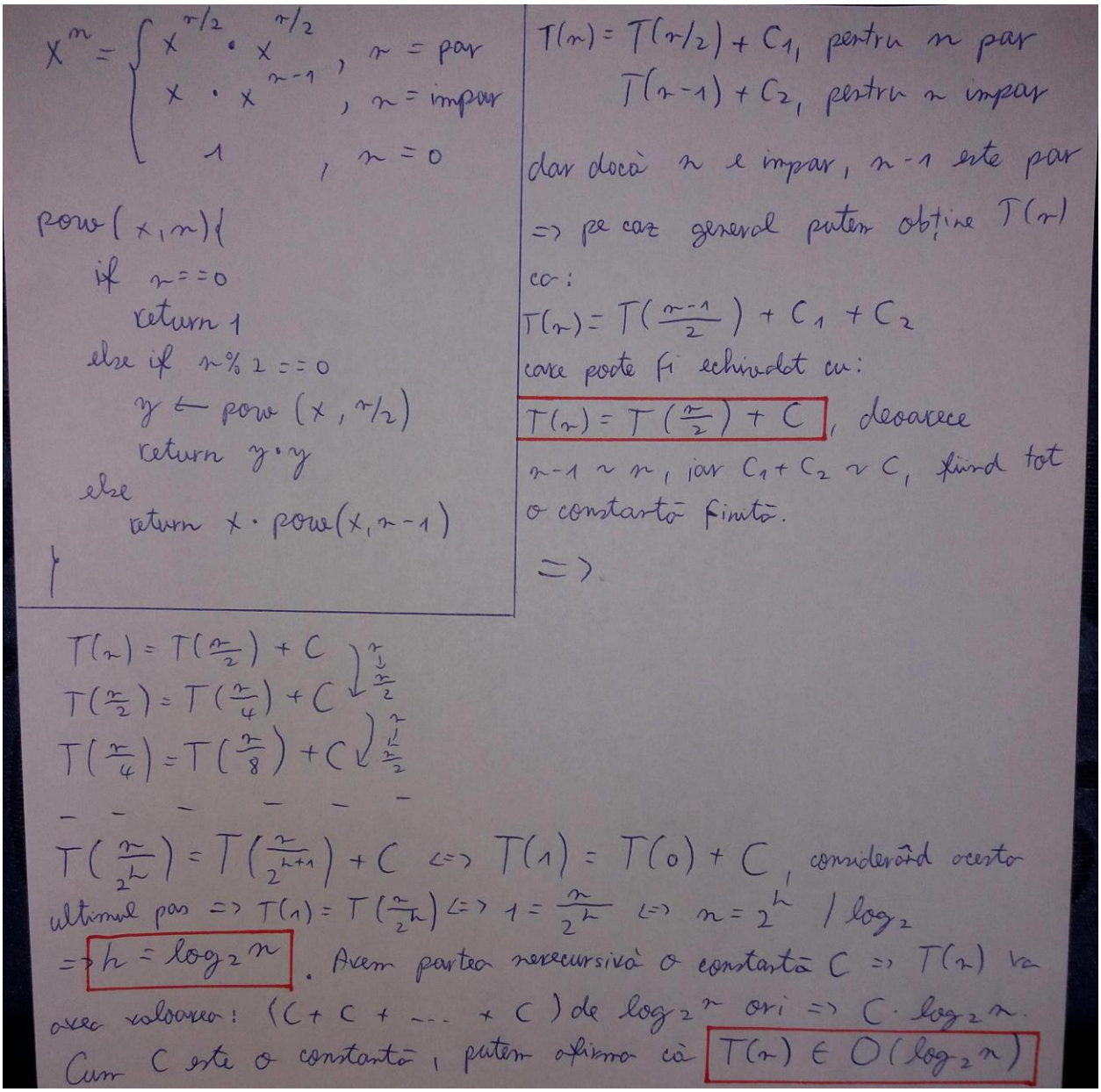
PSEUDOCOD:

```
Repeat k times:
      Generate random in interval [2, p-1]
      If a^(p-1) mod p != 1
            return composite
return probably prime
```

COMPLEXITATE:

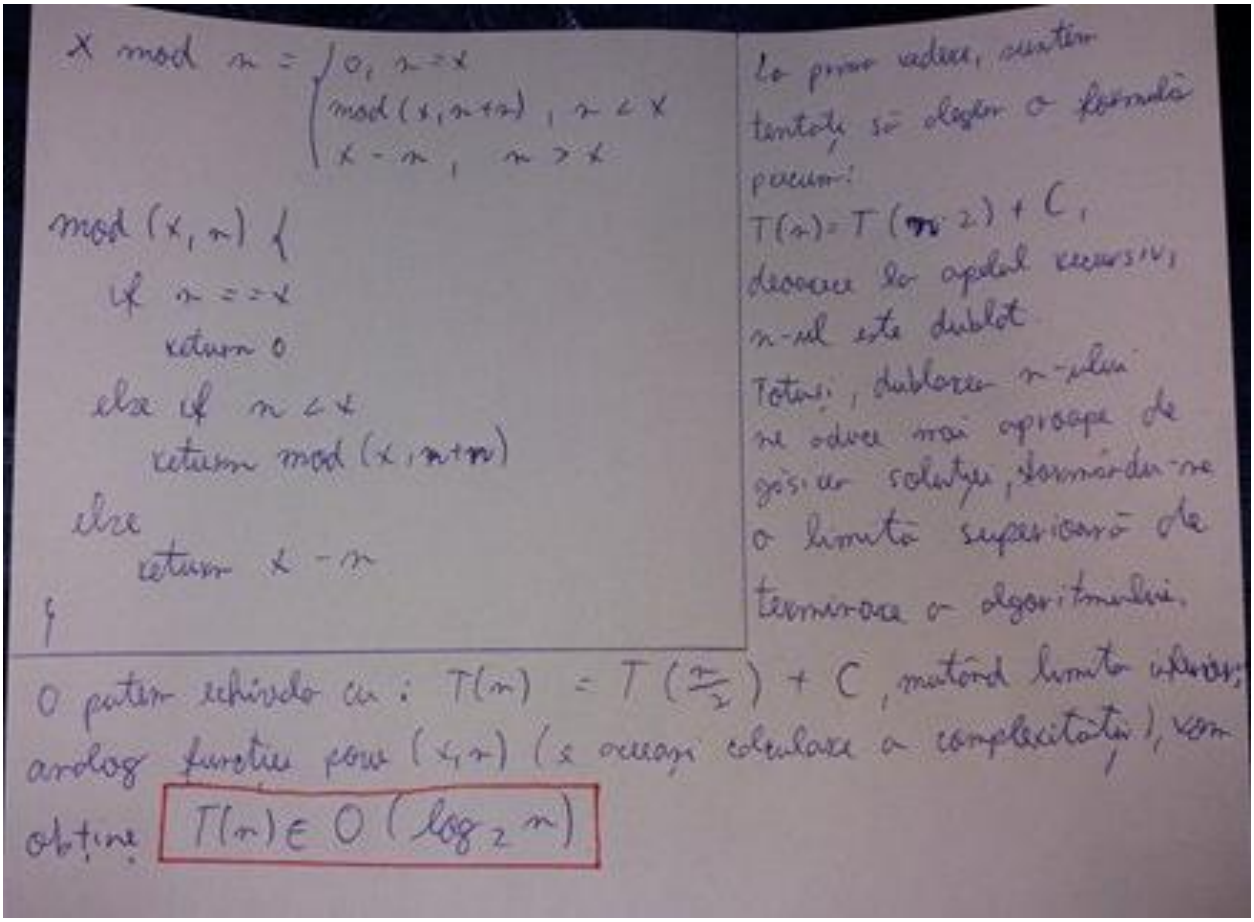
Functia **pow(x,n)** care ridica la puterea n numarul a are urmatorul pseudocod recursiv dispune de o complexitate scazuta daca este rezolvata

recursiv prin intermediul metodei *exponentiating by squaring*[17].



Am abordat 2 cazuri intrucat partea recursiva depinde de conditiile aplicate n-ului. Datorita paritatii lui n-1 in cazul in care n este impar, am putut reduce cele 2 situatii la una singura ce va determina rezultatul. Restul operatiilor sunt constante, avand complexitatea O(1), deci nu vor fi luate in calcul, datorita definitiei O-ului. Astfel, deducem cu ajutorul metodei iteratiei, ca functia pow va avea complexitatea O(logn).

Functia **mod(x,n)** care returneaza restul impartirii numarului x la n, prezinta aceeasi complexitate.



Analog ca mai sus, toate operatiile sunt neglijabile din punct de vedere al complexitatii, cu exceptia recursivitatii, unde am folosit, din nou, metoda iteratiei.

De asemenea, algoritmul ruleaza de un numar de iteratii = k, care va trebui inmultit cu complexitatea algoritmului, fiind echivalent cu o suma de complexitati ale algoritmului cu k termeni => k * T(n). T(n) final va fi egal cu inmultirea celor 2 complexitati aferente celor 2 functii utilizate, respectiv pow si mod, adica $(\log n)^2$. Prin urmare, testului Fermat ii corespunde o complexitate: **$T(n) = O(k * (\log n)^2)$**.

Implementare si testare

Implementarea este realizata in limbajul Java, datorita posibilitatii efectuarii testului pe o raza mai intinsa de numere (**up to 270 digits**), multumita clasei BigInteger[18]. Cu exceptia unor mici modificari si adaugarea timpului de rulare, algoritmul a fost preluat de aici. Fisierul cu implementarea este denumit FermatPrimality, iar drept fisiere de input exista 3 .txt, si anume:

- primes.txt – numere care vor intoarce intotdeauna prime(ultimul numar este testat pentru numarul de iteratii: 20, 50, 100, 500, 1000, 5000, pentru a evidentia variatia timpului de executie);
- carmichael.txt – fisier cu numere Carmichael, numere compuse ce vor returna prime daca cel mai mare divizor comun dintre baza aleasa si ele este diferit de 1 (numarul de iteratii aici este 1);
- composites.txt – numere compuse ce vor returna prime pentru Fermat liars, dar si unele din primes.txt cu ultima cifra inlocuita cu 2 pentru a determina corectitudinea pentru intrati de ordin mare.

Pentru fiecare input din fisierele de mai sus, putem regasi output-ul in folderul *testing/fermat/*, respectiv fisierele *primes_results.txt*, *carmichael_results.txt*, *composites_results.txt*. Numarul de iteratii standard a fost stabilit 20, pentru a avea sansa de a nu intalni un witness egala cu 1 la un milion.

la o alta testare pe aceleasi input-uri, output-urile pot fi diferite pentru fisierele carmichael.txt (deoarece a si p nu au conditia de a fi relativ prime, pentru a putea trece testul in unele cazuri) si composites.txt (pentru strong liars)

Analiza Miller-Rabin

INPUT:

- un numar natural p > 3;
- un parametru k care reprezinta numarul de iteratii (parcurgeri).

OUTPUT:

- compus daca p este compus, altfel probabil prim.

PSEUDOCOD:

```
write p – 1 = 2^k * m
repeat iter times:
  generate a [2, p − 2]
  x ← a^m mod p
  if (x == 1 || x == p – 1)
    continue
  repeat k − 1 times:
    x ← x^2 mod p
    if (x == 1)
      return composite
    if (x == p – 1)
      break
  return composite
return probably prime
```

COMPLEXITATE:

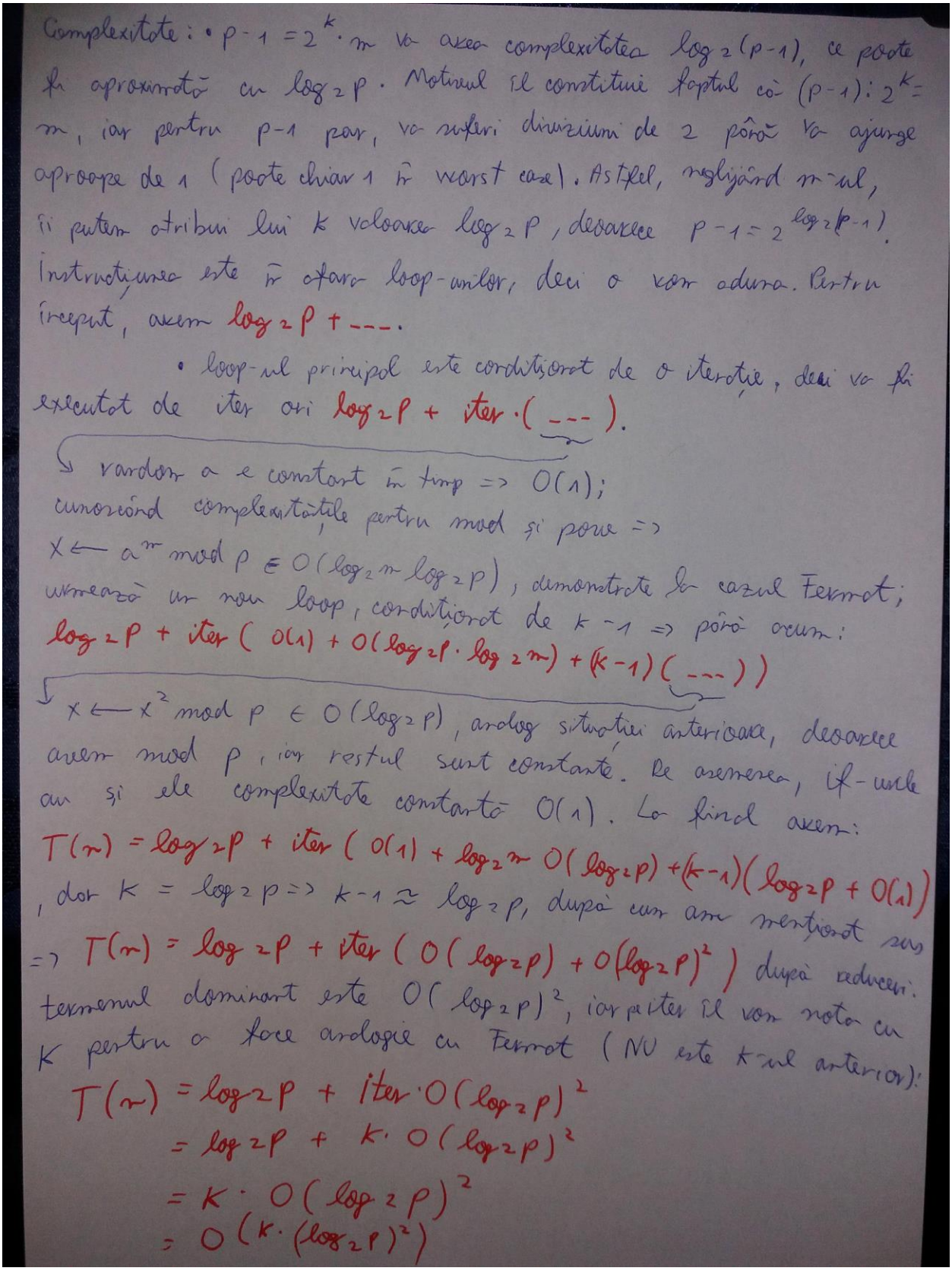
Avem acum cunoscute complexitatile functiilor pow si mod, iar singurul lucru ramas este analiza treptata a complexitatii de-a lungul algoritmului. Vom alege de fiecare data cand se poate cazul nefavorabil (**worst case**), deoarece rezultatul este dat sub forma lui **Big O**[19], iar diferentele sunt nesemnificative fata de cazul mediu pentru a influenta considerabil complexitatea.

Implementare si testare

De asemenea, pentru a pastra consecventa, algoritmul pentru Miller-Rabin este realizat tot in Java. Datorita existentei metodei prestabilite din clasa BigInteger (public boolean isProbablePrime(int certainty)), care foloseste insusi testul Miller-Rabin, a fost utilizata direct implementarea data (in fisierul MillerRabinPrimality). O versiune a algoritmului poate fi vazuta in fisierul MillerRabinExtended, ce prezinta o preluare din Python, luata de aici(acesta nu este executabil, deoarece reprezinta ce se afla in spatele metodei din Java). Dupa diferite studii realizate de matematicieni la sfarsitul secolului al XX-lea, inceput de secol XXI (Pomerance, Wagstaff, Sorenson, Webster etc.), a fost trasa concluzia ca, in functie de dimensiunea p-ului, poate fi aleasa o multime de numere fixa pentru a, astfel incat testul sa fie determinist in situatia data:

p < 2,047, a = 2;
p < 1,373,653, a = 2 si 3;
p < 9,080,191, a = 31 si 73;
p < 25,326,001, a = 2, 3, si 5;
p < 3,215,031,751, a = 2, 3, 5, si 7;
p < 4,759,123,141, a = 2, 7, si 61;
p < 1,122,004,669,633, a = 2, 13, 23, si 1662803;
p < 2,152,302,898,747, a = 2, 3, 5, 7, si 11;
p < 3,474,749,660,383, a = 2, 3, 5, 7, 11, si 13;
p < 341,550,071,728,321, a = 2, 3, 5, 7, 11, 13, si 17;
p < 3,825,123,056,546,413,051, a = 2, 3, 5, 7, 11, 13, 17, 19, si 23;
p < 18,446,744,073,709,551,616 = 264, a = 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, si 37;
p < 318,665,857,834,031,151,167,461, a = 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, si 37;
p < 3,317,044,064,679,887,385,961,981, a = 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, si 41.

Intrucat numerele Carmichael nu mai reprezinta o amenintare pentru testul Miller-Rabin, input-urile au ramas doar fisierele primes.txt si composites.txt, ale caror output-uri pot fi gasite in folderul *testing/miller_rabin/*, in fisierele *composites_results.txt*, respectiv *primes_results.txt*. Numarul de iteratii ales este tot 20, pentru a putea face o comparatie prematura a timpului de rulare al celor doua implementari.



deci **T(n) = O(k * (logn)^2)**.

Analiza comparativa

Asadar, ambii algoritmi denota o caracteristica probabilistica de aflare a primalitatii unui numar dat ca input (in ciuda incercarii lui Miller de a-l face determinist prin intermediul teoriilor lui Riemann **nedemonstrate**). Mai mult, dispun de aceeasi complexitate temporala, respectiv **$O(k * (\log n)^2)$**, evitand rularea intr-un timp polinomial.

Diferentele dintre cei doi algoritmi ar putea sa nu fie considerate semnificative avand in vedere perioada de timp necesara trecerii de la unul la altul. Testul Fermat se bazeaza pe Mica Teorema a lui Fermat, enuntata in secolul al XVII-lea si demonstrata 100 de ani mai tarziu, pe cand testul Miller-Rabin a fost elaborat in 1980. Probabilismul celor doua teste este legat prin prezenta pseudoprimelor, care pot sa apara la fiecare dintre ele. Totusi, sansa de a returna valoarea adevarata difera. Probabilitatea de reusita este de minim 50% in cadrul testului Fermat, in timp ce Miller-Rabin poate dezvalui output-ul asteptat intr-o proportie de cel putin 75%. In plus, testul Fermat va pica pentru numerele Carmichael, care, desi sunt mai putin numeroase decat numerele prime, trebuie luate in considerare.

Cardinalul multimii numerelor prime mai mici decat un numar n este aproximativ egal **cu $\pi(n) = n / \log(n)$**, pe cand cardinalul numerelor Carmichael mai mici decat un numar n poate fi reprezentat dupa tabelul urmator:

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	...	21
$C(10^n)$	0	0	1	7	16	43	105	255	646	1547	3605	8241	19279	44706	105212	246683	...	20138200

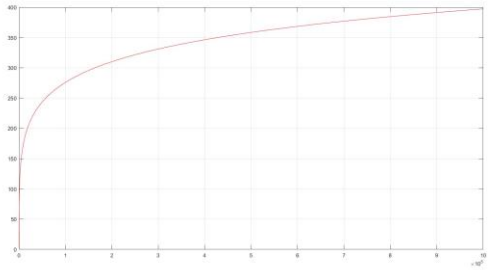
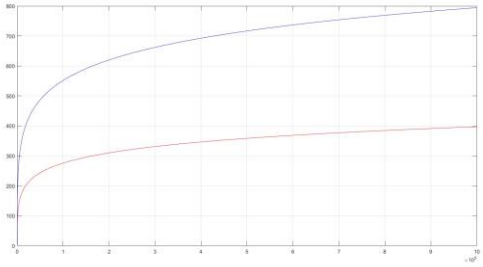
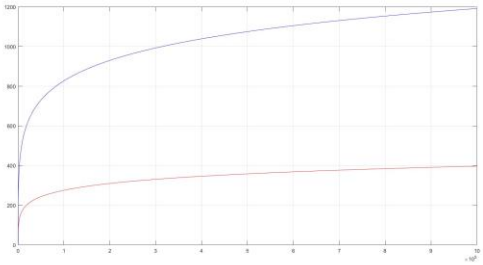
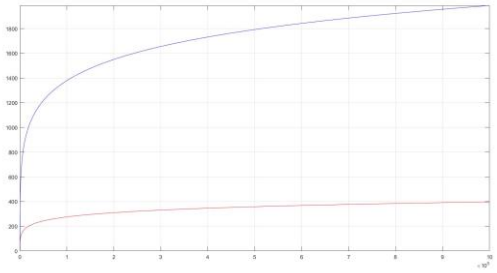
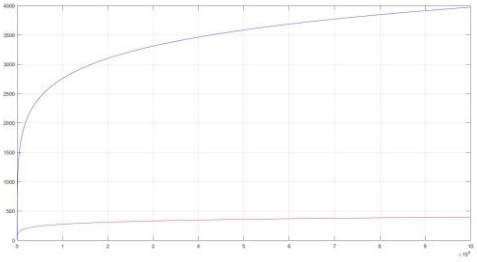
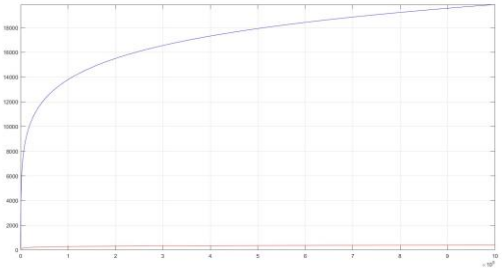
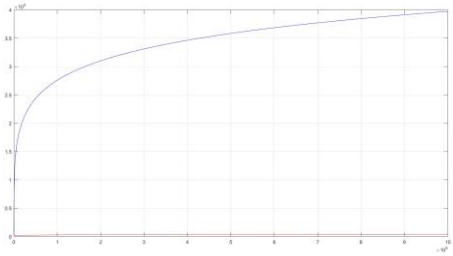
C(x) = numarul de Carmichaels mai mici sau egale decat un x

Numarul de iteratii poate juca un rol crucial in corectitudinea acestora. Cu cat numarul de iteratii este mai mare, cu atat raspunsul asteptat va fi si cel dorit. Miller-Rabin isi va dubla eficienta fata de Fermat pentru

fiecare incrementare a numarului de iteratii, ceea ce il face superior. Desi ambele dispun de aceeasi complexitate (aproximativa), conform output-urilor din fisiere, testul Miller-Rabin se executa mai rapid, datorita posibilitatilor multiple de intoarcere in bucla principala (x = 1 || x = p – 1, break-ul din al 2-lea loop etc.), ce sporesc rapiditatea parcurgerilor algoritmului. Se poate observa ca pentru un input de genul **pow(10,270)**, pentru k = 5000, testul Fermat ajunge la 10 secunde, in timp ce Miller-Rabin are timp aproximativ constant pentru k = 20 pana la 5000. Acest lucru, pentru numere mai mici, poate fi datorat si particularitatilor de care dispune Miller-Rabin, prezentate **aici**. Asadar, **testul Miller-Rabin isi mareste acuratetea prin numarul de iteratii ridicat, avand posibilitatea de a sfida timpul de executie suplimentar**. Din pacate, chiar si in prezenta unei memorii superioare si a unei viteze de procesare pe masura, nu vom putea sti niciodata daca a fost returnat rezultatul corect, deoarece eroarea este diminuata, dar nu eliminata complet. De aceea se prefera executarea de mai multe ori asupra unui input, ori trecerea la un algoritm determinist (exemplu: AKS[20]).

Se poate observa in output-uri faptul ca, desi se trece de la un input mai mic la unul mai mare, timpul de executie scade. Acest lucru se datoreaza alegerii aleatoare a bazei. Intrucat baza se alege din intervalul [2, p-2], nu vom sti ce valoare va fi aleasa. Fireste, timpul de executie pentru a = 2 va fi mult mai mic fata de unul pentru a = p - 2.

Timpul de rulare in functie de numarul de iteratii poate fi vizualizat in graficele de mai jos (linia rosie pentru Fermat la k = 1 si Miller-Rabin la k orice k, restul pentru Fermat la k variant dupa figura explicativa de mai jos):

$k = 1$	$k = 2$
$k = 3$	$k = 5$
$k = 10$	$k = 50$
$k = 100$	$n = 1.000.000$

Concluzii

Concluzie

In ciuda erorilor de care pot dispune cei doi algoritmi, timpul lor de rulare este mai mic decat al tuturor celorlalte teste de primalitate (urmatorul fiind Baillie-PSW care are $O((\log n)^3)$). Testul Miller-Rabin garanteaza o pondere mai mare de reusita fata de Fermat, fiind ajutat si de timpul de executie scazut la crestarea semnificativa a numarului de iteratii (astfel, se

fac mai multe iteratii la acelasi cost). In plus, Miller-Rabin nu returneaza probably prime pentru numere Carmichael, ceea ce-l plaseaza deasupra testului Fermat. Mai mult, acesta poate duce la generarea factorilor in care se imparte un numar compus[21]. Pe arhitecturi slabe se pot face mai multe executii de Fermat/Miller-Rabin, insa este de preferat o metode determinista daca circumstantele ne permit asta.

In concluzie, testele Fermat si Miller-Rabin constituie o modalitate eficienta din punct de vedere al complexitatii cu privire la problematica primalitatii unui set de numere (Fermat fiind inferior din punct de vedere al acuratetei si al eficientei temporale, dar conceptul ce-i sta la baza existand de mai bine de 3 secole), insa rezultatul poate fi inselator.

Mentiuni

Aici sunt prezentate detalii structurale ale proiectului. Mai precis, ce a fost surprins in cadrul fiecarui titlu, pentru a putea evidentia, succint, continutul lucrarii.

Introducerea:

- definirea numerelor prime;
- domeniile in care pot fi utilizate;
- istorie sumara a analizei lor de-a lungul anilor.

Prezentarea si implementarea solutiilor:

- introducerea algoritmilor: contextul aparitiei, ce stau la baza lor,
- explicarea algoritmilor cu ajutorul unor desene si exemple;
- Mica Teorema a lui Fermat: definitie, istorie, demonstratie;
- Lema lui Euler;

- punerea in lumina a vulnerabilitatilor: input-uri care intorc rezultatul gresit la output (mereu sau doar candva), cat si probabilitatile de reusita.

Complexitate si testare:

- analiza pseudocodurilor si calcularea complexitatilor algoritmilor;
- calcularea complexitatilor functiilor pow si mod;
- testarea pentru seturi de date predefinite, cat si explicarea rezultatului in diferit situatii (numar schimbat de iteratii/input specific);
- grafice pentru modificarea numarului de iteratii (cu referinta pentru k = 1, desenata cu rosu);
- posibilitatea particularizarii testului Miller-Rabin (generarea fixa a bazelor in functie de dimensiunea numarului care trebuie testat);
- analiza comparativa: situatiile in care este preferat un algoritm sau altul, evidentiand avantajele si dezavantajele fiecaruia, dar si cand ar putea fi folosite altele cu caracter determinist.

Concluzii:

- concluzia.

Referinte

Animated explanation for Fermat Test. Retrieved from https://www.khanacademy.org/computing/computer-science/cryptography/random-algorithms-probability/v/fermat-primality-test-prime-adventure-part-10

Big O. Retrieved from http://bigocheatsheet.com/

BigInteger class from Oracle. Retrieved from https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html

Bishop E. Robert analysis over Fermat's Little Theorem. Retrieved from http://www.math.uchicago.edu/~may/VIGRE/VIGRE2008/REUPapers/Bishop.pdf

Bobby Kleinberg's introduction to Miller-Rabin Test. (2010). Retrieved from https://www.cs.cornell.edu/courses/cs4820/2010sp/handouts/MillerRabin.pdf

Carmichael Numbers. Retrieved from http://www.maths.lancs.ac.uk/jameson/carpsp.pdf

Chinese Remainder Theorem. Retrieved from https://crypto.stanford.edu/pbc/notes/numbertheory/crt.html

Comparative analysis for primality tests. Retrieved from http://www.cmi.ac.in/~shreejit/primality.pdf

Euclid's infinite prime numbers proof. Retrieved from https://primes.utm.edu/notes/proofs/infinite/euclids.html

Euclid's Lemma. Retrieved from https://artofproblemsolving.com/wiki/index.php?title=Euclid%27s_Lemma

Exponentiation by squaring. Retrieved from https://www.rookieslab.com/posts/fast-power-algorithm-exponentiation-by-squaring-cpp-python-implementation

Fermat's Little Theorem animated proof. Retrieved from https://www.youtube.com/watch?v=OoQ16YCYksw

Fermat's source inspiration. Retrieved from https://stackoverflow.com/questions/4027225/implementation-of-fermats-primality-test

Fundamental Theorem of Arithmetic. Retrieved from https://gowers.wordpress.com/2011/11/18/proving-the-fundamental-theorem-of-arithmetic/

Jeff Suzuki, Factorizing a number with Miller-Rabin Test. Retrieved from https://www.youtube.com/watch?v=S3vNCMTEXQM

Michael O. Rabin, Probabilistic Algorithm for Testing Primality. (1977). Retrieved from https://www.sciencedirect.com/science/article/pii/0022314X80900840

Miller-Rabin's algorithm source. Retrieved from http://rosettacode.org/wiki/Miller%E2%80%93Rabin_primality_test

Pascal's Binomial Theorem. Retrieved from https://www.shmoop.com/polynomial-equations/Pascal-and-Binomial-Theorem.html

RSA public key, cryptography algorithm. Retrieved from http://searchsecurity.techtarget.com/definition/RSA

Sieve of Eratosthenes. Retrieved from http://www.geeksforgeeks.org/sieve-of-eratosthenes/

Taylor's 1717 theorem. Retrieved from https://www.math.hmc.edu/calculus/tutorials/taylors_thm/

The Extended Riemann Hypothesis (ERH). Retrieved from https://www.staff.uni-mainz.de/pommeren/Cryptology/Asymmetric/3_Prim/ERH.pdf

The Solovay-Strassen Test for primality. Retrieved from http://www.math.uconn.edu/~kconrad/blurbs/ugradnumthy/solovaystrassen.pdf