Numere prime. Algoritmi de identificare a numerelor prime

Alexandru Olaru - 321 CD

Facultatea de Automatică si Calculatoare

1 Introducerea Proiectului

1.1 Descrierea problemei rezolvate

Mulțimea numerelor prime este definită ca fiind o mulțime infinită de numere naturale ($\mathbb{N}$) mai mari decât 1, care au ca divizori doar numărul 1 și pe ele insele, neputând fi scrise ca produs de oricare alte numere naturale. Așadar, fiindu-ne oferită o listă de numere, ne propunem să verificăm dacă acestea sunt prime.

1.2 Aplicații practice

Acestea au aplicații practice, în mod special în domeniul matematicii. Cu ajutorul numerelor prime, avem posibilitatea de a descompune un număr natural în factori primi, lucru care ne ajută, totodată, să discutăm și despre divizibilitate, precum și să aflăm cel mai mic multiplu comun dintre două numere (CMMMC), respectiv cel mai mare divizor comun dintre două numere (CMMDC). De asemenea, numerele prime au întrebuințări și în domeniul criptografiei, decriptarea cheilor publice bazându-se pe dificultatea factorizării numerelor mari în factori primi. Securitatea programelor de criptare au la bază faptul că este foarte rapid sa înmulțesti două numere prime mari pentru a afla rezultatul, însa este foarte dificil si costisitor pentru un calculator de a realiza procedeul invers, și anume aflarea respectivelor două numere prime.

1.3 Specificarea algoritmilor aleși

Algoritmii aleși în cadrul acestui proiect vor fi următorii: Teorema lui Fermat de identificare a numerelor prime ("Fermat′s Little Theorem") și algoritmul de primalitate al lui Miller-Rabin.

1.4 Criteriile de evaluare pentru algoritmii aleși

Evaluarea celor doi algoritmi se va face prin intermediul mai multor teste,în felul următor: se vor citi datele din toate fișierele de test create, iar ceea ce va întoarce programul în urma rulării acestor teste va fi afișat în fișierul de output omolog fișierului cu datele de intrare. Fișierele de input vor fi formate dintr-o singură linie, care va conține 2 numere naturale pozitive, de tip "int": primul va fi numărul propriu-zis pentru care se va face verificarea, iar cel de-al doilea va reprezenta numărul de iterații care se vor realiza pentru a obține rezultatul. Atât numărul de iteratii din fiecare fișier de input, cât și dimensiunea numerelor ce urmează a fi verificate vor varia. În fișierele de output va fi tipărit, pentru fiecare număr care trebuie verificat, un mesaj în care va fi afișat explicit numărul și dacă este prim sau nu.

De asemenea, în cadrul ambilor algoritmi, avem posibilitatea de a alege un număr de iterații pentru fiecare set pe care îl testăm. De exemplu, dacă alegem un număr k de iterații, pentru fiecare element, acel "a" va fi generat de k ori, fiind de fiecare dată diferit. Așadar, algoritmii vor testa de mai multe ori pe același număr, acest lucru scăzând șansele ca rezultatul întors să fie eronat, deoarece ambele metode pot duce la ceva incorect. Din această cauză, pentru fiecare fișier de intrare se vor rula programele cu un număr scăzut de iterații și cu un număr mai mare de iterații, tocmai pentru a observa diferența de acuratețe dintre cele 2 rulări. Prin intermediul checkerului propus, se va verifica daca ce este afișat in fișierul de output este identic cu ce este scris in fișierul de referință. Așadar, în directorul "out" din arhivă se regăsesc fișierele de referință, în care este scrisă primalitatea reală a numerelor.

Testele vor fi generate de mâna, având ca scop utilizarea celor doi algoritmi pe mai multe categorii de numere, precum: numere prime mici, numere compuse mici, numere compuse foarte mari (ajungând și la ordinul milioanelor), dar și numere prime la fel de mari ca cele compuse. În acest sens, la fiecare 5 teste, se va crește ordinul pentru numerele ce urmează a fi testate (ex: numere din intervalul [1,100], [100, 10.000], [10.000, 10.000.000]).

2 Prezentarea soluțiilor

2.1 Descrierea modului de funcționare al algoritmilor

2.1.1 Fermat

Teorema lui Fermat de identificare a numerelor prime ("Fermat′s Little Theorem") este una dintre modalitățile de a verifica dacă un numar este prim sau nu. Metoda funcționează după următoarea idee: este dat un număr natural n, iar apoi, pentru un număr a ϵ $\mathbb{N}$, a ϵ (1 ; n-1), se va testa dacă restul împarțirii lui a^{n-1} la n (a^{n-1} mod n) este egal cu 1. Daca afirmația este adevărată, numărul n este prim. Implementarea acestei soluții va consta în construirea a 3 funcții: "power" - care va calcula a^{n-1} mod n, "greatestCommonDivisor" - pentru a returna "false" în cazurile în care a și n au cel mai mare divizor comun diferit de 1 și "isPrime" - care va returna "true/false" dacă numărul va fi neprim/prim. Această verificare va fi făcută de t ori, t fiind un număr natural primit ca parametru, reprezentând numărul de iterații.

```
FERMAT(n,t){
INPUT: odd integer n ≥ 3, # of repetition t
OUTPUT: PRIME or COMPOSITE
   for (i from 1 to t){
        Choose a random integer a s.t. 2 ≤ a ≤ n-2
        Compute   r = a^(n-1) mod n
        if ( r ≠ 1 )    return COMPOSITE
   }
   return PRIME
}
```

2.1.2 Miller-Rabin

Metoda lui Miller-Rabin va fi cel de-al doilea algoritm ales pentru a verifica proprietatea unui număr de a fi prim. Acesta se aseamănă ușor cu algoritmul lui Fermat, mergând pe principiul următor: se dă un număr natural n; se găsesc două numere d si r , astfel încat d $\cdot$ 2^r = n - 1; pentru un număr a, a ϵ (1 ; n-1), se va calcula restul împarțirii lui a^d la n (x = a^d mod n), iar dacă x este fie egal

cu 1, fie egal cu n - 1, numărul n este prim; în caz contrar, se va intra într-o buclă care se va opri atunci când d va fi egal cu n-1; în interiorul buclei, d se va dubla, restul x va lua o nouă valoare x = x^2 mod n, verificându-se mai apoi dacă x este egal cu 1, caz in care se iese din buclă, numărul nefiind prim și dacă x este egal cu n-1, caz în care numărul este prim. Vor fi implementate funcțiile: "power"- care va ajuta in calcularea lui x = a^d mod n, "millerRabin" - care va lua un număr a din intervalul specificat mai sus și va returna "true/false" în funcție de rest și funcția "isPrime" - care va găsi numerele r, d și va apela "millerRabin" pentru a decide în final dacă numărul este prim sau nu. Această verificare va fi făcută de t ori, t fiind un număr natural primit ca parametru, reprezentând numărul de iterații.

MILLER-RABIN(*n*,*t*){
INPUT: odd integer $n \geq 3$, # of repetition *t*
Compute *k* & odd *m* s.t. $n - 1 = m2^k$
for (*i* from 1 to *t*){
Choose a random integer *a* s.t. $2 \leq a \leq n - 2$
Compute $y = a^m \bmod n$
if ($y \neq 1$ or $y \neq n - 1$){
Set $j \leftarrow 1$
while($j \leq k - 1$ and $y \neq n - 1$){
Set $y \leftarrow y^2 \bmod n$
if ($y = 1$) return COMPOSITE
$j \leftarrow j + 1$
}
if ($y \neq n - 1$) return COMPOSITE
}
}
return PRIME
}

2.2 Analiza complexitații soluțiilor

2.2.1 Fermat

Funcția iterativă "power" din cadrul algoritmului Fermat calculează x = a^d mod n în O(log n). Datorită faptului că în cazul verificării propriu-zise nu se apeleaza decât funcția "power" si funcția "GreatestCommonDivisor(a, b)", având complexitatea de O(log (min(a, b))) ce poate fi neglijată, complexitatea finală va fi O(k·log n), unde k este numărul de iterații primit la input.

2.2.2 Miller-Rabin

La fel ca în cadrul primului algoritm menționat, funcția "power" calculează x = a^d mod n în O(log n). Totuși, diferența o face cealaltă funcție, numită "isPrime", care, făcând operația de radical în mod repetat, ajunge sa aibă o complexitate de O(log^2n). Așadar, complexitatea finală a acestei metode este O($k \cdot log^3$n), ceea ce face algoritmul lui Miller-Rabin mai ineficient din punct de vedere al timpului de execuție decât algoritmul lui Fermat, însă este puțin mai bine optimizat, așa cum vom vedea în următoarele paragrafe ale proiectului. K face referire, și în acest caz, la numărul de iterații.

2.3 Prezentarea principalelor avantaje si dezavantaje pentru soluțiile luate in considerare

2.3.1 Fermat

Unul dintre avantajele acestui algoritm este faptul că are un timp de executare foarte scazut. Tocmai din această cauză, reprezintă o modalitate foarte rapidă de a verifica primalitatea numerelor. Acest algoritm este utilizat deseori in practică, fiind folosit ca metodă de filtrare, de exemplu, în algoritmii criptografici de generare a cheilor publice RSA.

Principalul dezavantaj al acestui algoritm este dat de existența numerelor compuse "Carmichael", cu următoarea proprietate: pentru orice număr a < n, unde n este data de intrare pentru care se va efectua verificarea (în acest caz, un număr "Carmichael"), atât cel mai mare divizor comun dintre a și numărul nostru n, cât și a^{n-1} mod n sunt egale cu 1. Din această cauză, algoritmul Fermat poate întoarce și rezultate eronate.

2.3.2 Miller-Rabin

Un avantaj al acestui algoritm este reprezentat de faptul că funcționează cel puțin la fel de bine ca Fermat′s Little Theorem în momentul în care primeste ca date de intrare numere destul de mari, ce se apropie de ordinul sutelor de mii (dar nu îl ating), întorcand rezultate corecte. Datorită eficienței și preciziei sale, și acesta este folosit în aplicații practice în domeniul criptografiei. Un plus pe care îl are față de primul algoritm este faptul că probabilitatea de a întoarce un rezultat incorect, primind ca date de intrare numere "Carmichael" este mult mai mică în raport cu algoritmul Fermat, acest lucru datorându-se logicii după care funcționeaza (aplicarea repetată a operațiior radical).

De asemenea, așa cum vom testa și în următoarea secțiune a proiectului, dacă datele de intrare sunt constituite din numere mai mari de ordinul sutelor de mii, cele două programe vor avea probleme în a recunoaște numerele prime, afișând un rezultat corect doar pentru cele neprime. Acest lucru reprezintă un

dezavantaj major in cadrul ambilor algoritmi, însa aceștia sunt ușor modelați pentru a putea fi folosiți într-un mod eficient în practică.

3 Evaluare

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

În fișierul "in" se află datele de intrare (testele.in), iar in fisierul "out" se află testele de referință, în care este regăsită primalitatea reală a numerelor.

Testele au fost generate de mâna, în așa fel încât să se realizeze verificarea pe mai multe categorii de numere: numere mici si mari, atât compuse, cât și prime(ajungând și la ordinul milioanelor). În acest sens, la fiecare 5 teste, a fost crescut ordinul pentru numerele ce urmează a fi testate (ex: numere din intervalul [1,100], [100, 10.000], [10.000, 10.000.000]).

În acest sens, testele sunt următoarele:

Testele 1-5 conțin numere mai mici decât 100, fiecare cu câte o singură iterație: 1, 5, 24, 11, 97.

Testele 6-10 conțin numere mai mici decât 10.000 , fiecare cu câte 5 iterații: 210, 811, 4999, 6024, 9949.

Testele 11-15 conțin numere mai mici decât 100.000: 12.157, 12.158 (ambele cu câte 10 iterații), 27.299, 45.827 (ambele cu o singură iterație), 99.991 (cu 100 de iterații).

Testele 16-20 conțin numere mai mari decât 100.000: 100.002 (o singură iterație), 101.807, 104.729 (ambele cu 100 de iterații), 1.047.299 (cu 10 iterații), 9.999.991 (cu 100 de iterații).

3.2 Specificațiile sistemului de calcul pe care au fost rulate testele

-CPU: Intel Core i5 4460 @ 3.20GHz
-RAM: 8GB DDR3 1600 MHz
-Storage: HDD 1TB
-Operating System: Windows 10 Pro

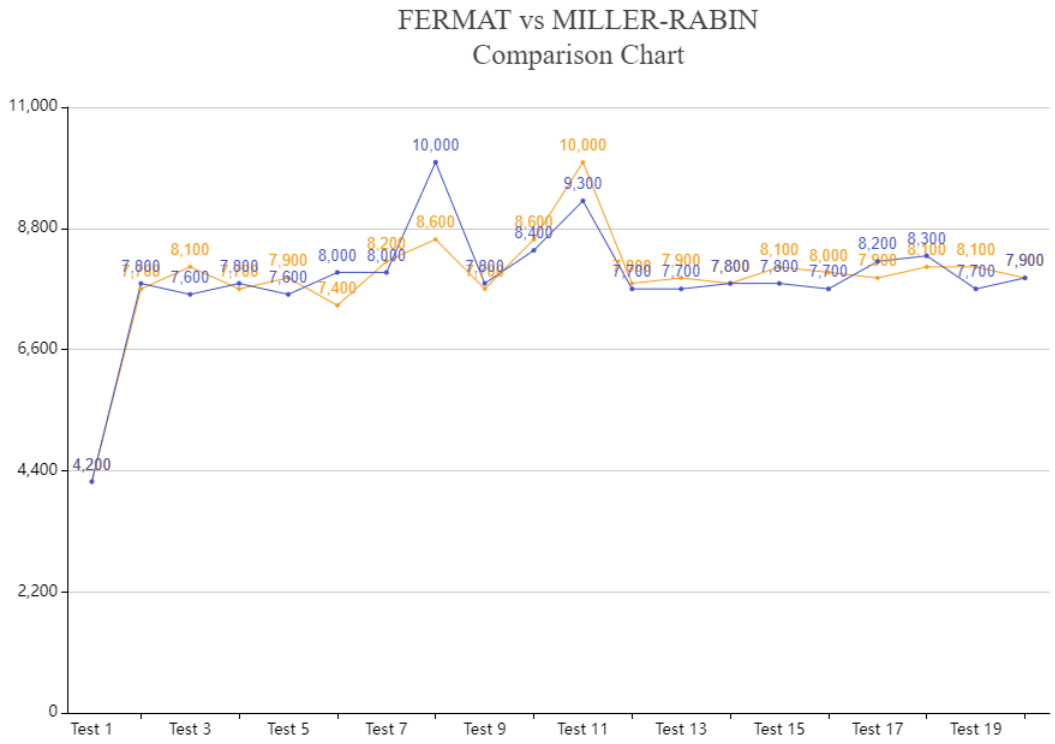
3.3 Ilustrarea rezultatelor evaluării soluțiilor pe setul de teste

Nr test	Fermat	Miller-Rabin
1 (1 iterație)	4200	4200
2 (1 iterație)	7700	7800
3 (1 iterație)	8100	7600
4 (1 iterație)	7700	7800
5 (1 iterație)	7900	7600
6 (5 iterații)	7400	8000
7 (5 iterații)	8200	8000
8 (5 iterații)	8600	10000
9 (5 iterații)	7700	7800
10 (5 iterații)	8600	8400
11 (10 iterații)	10000	9300
12 (10 iterații)	7800	7700
13 (1 iterație)	7900	7700
14 (1 iterație)	7800	7800
15 (100 iterații)	8100	7800
16 (1 iterație)	8000	7700
17 (100 iterații)	7900	8200
18 (100 iterații)	8100	8300
19 (10 iterații)	8100	7700
20 (100 iterații)	7900	7900

Tabelul reprezentat mai sus ilustrează timpul de execuție (în nanosecunde) ale celor doi algoritmi, pe fiecare test în parte. Având în vedere faptul că cele două programe primesc aceleași date de intrare si întorc același rezultat pentru fiecare, putem observa faptul că, pentru un număr mic de iterații, algoritmul Miller-Rabin este cel puțin mai eficient în ceea ce privește timpul de rulare, decât algoritmul lui Fermat.

Așa cum va fi prezentat în continuare, ambii algoritmi întorc un rezultat eronat atunci când numărul care urmează a fi verificat este in zona sutelor de mii. Testarea acestora a fost realizată fiind oferit un număr mai ridicat de iterații (100), cu toate acestea, rezultatul fiind incorect.

Mai jos, este prezentat un grafic ce compară timpii de rulare dintre cei doi algoritmi, pe același set de date. Linia portocalie reprezintă performanțele algoritmului Fermat, iar cea albastră, pe cele ale algoritmului Miller-Rabin:



3.4 Prezentarea valorilor obținute pe teste

Testele 1-5 : numere mai mici decat 100:
1) 1, cu 1 iteratie. - Afișeaza corect cu ambii algoritmi. (NEPRIM)
2) 5, cu 1 iteratie. - Afișeaza corect cu ambii algoritmi. (PRIM)
3) 24, cu 1 iteratie. - Afișeaza corect cu ambii algoritmi. (NEPRIM)
4) 11, cu 1 iteratie. - Afișeaza corect cu ambii algoritmi. (PRIM)
5) 97, cu 1 iteratie. - Afișeaza corect cu ambii algoritmi. (PRIM)

Testele 6-10 : numere mai mici decat 10.000:
6) 210, cu 5 iteratii. - Afișaza corect cu ambii algoritmi. (NEPRIM)
7) 811, cu 5 iteratii. - Afișaza corect cu ambii algoritmi. (PRIM)
8) 4999, cu 5 iteratii. - Afișeaza corect cu ambii algoritmi. (PRIM)
9) 6024, cu 5 iteratii. - Afișeaza corect cu ambii algoritmi. (NEPRIM)
10) 9949, cu 5 iteratii. - Afișeaza corect cu ambii algoritmi. (PRIM)

Testele 11-15 : numere mai mici decat 100.000:

11) 12157, cu 10 iteratii. - Afișeaza corect cu ambii algoritmi. (PRIM)
12) 12158, cu 10 iteratii. - Afișeaza corect cu ambii algoritmi. (NEPRIM)
13) 27299, cu 1 iteratie. - Afișeaza corect cu ambii algoritmi. (PRIM)
14) 45827, cu 1 iteratie. - Afișeaza corect cu ambii algoritmi. (PRIM)
15) 99991, cu 100 iteratii. - Afișeaza incorect cu ambii algoritmi. (ESTE PRIM, SE AFIȘEAZA NEPRIM)

Testele 16-20 : numere mai mari decat 100.000:
16) 100002, cu 1 iteratie. - Afișeaza corect cu ambii algoritmi. (NEPRIM)
17) 101807, cu 100 iteratii. - Afișeaza incorect cu ambii algoritmi. (ESTE PRIM, SE AFIȘEAZA NEPRIM)
18) 104729, cu 100 iteratii. - Afișeaza incorect cu ambii algoritmi. (ESTE PRIM, SE AFIȘEAZA NEPRIM)
19) 1047299, cu 10 iteratii. - Afișeaza corect cu ambii algoritmi. (NEPRIM)
20) 9999991, cu 100 iteratii. - Afișeaza incorect cu ambii algoritmi. (ESTE PRIM, SE AFIȘEAZA NEPRIM)

După cum am mentionat și mai sus, ambii algoritmi întorc un rezultat eronat atunci când numărul care urmează a fi verificat este în zona sutelor de mii. Testarea acestora a fost realizată fiind oferit un număr mai ridicat de iterații (100), cu toate acestea, rezultatul fiind incorect.

4 Concluzii

În urma analizei făcute asupra celor doi algoritmi, se remarcă utilitatea lor practică în numeroase activități din mediul criptografic. Personal, din ce mi-a fost demonstrat din testele si rezultatele anterioare, aș opta pentru folosirea algoritmului Fermat in condițiile in care testarea va fi efectuată pe un set de numere mai mici, care nu necesită un număr ridicat de iterații, deoarece am observat că acesta rulează într-un timp mai scurt decât algoritmul Miller-Rabin pentru astfel de date de intrare. Pe de altă parte, aș utiliza cel de-al doilea algoritm pentru numere mai mari, dacă scopul ar fi să obțin rezultate mai precise, indiferent de timpul de rulare al programului. Datorită faptului că algoritmul Miller-Rabin rezolvă această problemă în ceea ce privește numerele "Carmichael", acest lucru îl face să fie mai folosit în practică decât Fermat.

5 Referințe

https://www.geeksforgeeks.org/primality-test-set-3-miller-rabin/?ref=rp — ultima accesare: 16.12.2021

https://www.geeksforgeeks.org/primality-test-set-2-fermet-method/ — ultima accesare: 16.12.2021

https://crypto.stackexchange.com - ultima accesare: 30.10.2021