Identificarea numerelor prime

Iordache Tiberiu-Mihai, Grupa 322CD

Universitatea Politehnica din Bucureşti
Facultatea de Automatică şi Calculatoare

Abstract. Acest document este o analiză a algoritmilor de identificare a numerelor prime(Solovay-Strassen şi Frobenius) care prezintă performanţa acestora pe diferite seturi de date.

Keywords: Numere prime · Modulo

1 Introducere

1.1 Descrierea problemei rezolvate

Principala problemă pe care dorim sa o rezolvăm prin intermediul acestui document este identificarea numerlor probabil prime dintr-un set de date de intrare dat.

1.2 Exemple de aplicaţii practice pentru problema aleasă

Această problemă are foarte multe aplicaţii în viaţa de zi cu zi, cea mai întâlnită fiind criptarea datelor. Ştim foarte bine că putem înmulţii două numere prime foarte mari cu uşurinţă, însă procesul invers durează mult mai mult timp, ajutând astfel la o mai bună securizare a datelor noastre.

1.3 Specificarea soluţiilor alese

Pentru a rezolva problema prezentată am ales să folosesc algoritmii Solovay-Strassen şi Frobenius.

1.4 Specificarea criteriilor de evaluare alese pentru validarea soluţiilor alese

Testele vor fi concepute în aşa fel încât să prezinte într-o manieră cât mai obiectivă accurateţea agloritmilor, prin testarea numerelor compuse de diverse valori, şi mari şi mici, dar şi durata de execuţie a acestora, prin testarea unor numere din ce în ce mai mari.

2 Prezentarea soluţiilor

2.1 Descrierea modului în care funcţionează algoritmii aleşi

Solovay-Strassen[2] Acest algoritm determină rapid dacă un număr este compus sau posibil prim. Se folosesc două simboluri:

Simbolul Legendre

Fie p un număr impar prim şi a un număr întreg pozitiv, definim simbolul Legendre ca fiind:

$$(\frac{a}{p}) = \begin{cases} 0, & \text{dacă p este un divizor al lui a} \\ +1, & \text{dacă există un } k \text{ întreg astfel încât } k^2 = \text{a(mod p)} \\ -1, & \text{în rest} \end{cases}$$

Datorită unui rezultat al lui Euler[5], avem relaţia:

$$(\frac{a}{p}) = a^{\frac{p-1}{2}} \bmod \text{p}$$

Simbolul Jacobian

Este o generalizare a simbolului Legendre, unde n este un număr întreg pozitiv:

$$(\frac{a}{n}) = (\frac{a}{p_1})^{k_1} \cdot (\frac{a}{p_2})^{k_2} \cdot ..... \cdot (\frac{a}{p_n})^{k_n}$$

Pentru ca un număr să fie posibil prim, acesta trebuie să aibă simbolul Jacobian egal cu simbolul Legendre.

$$(\frac{a}{n}) = a^{\frac{n-1}{2}} \bmod \text{n}$$

Dacă această condiţie nu este satisfăcută, numărul este compus, iar programul se va oprii. Aşadar, va fi nevoie de mai multe iteraţii pentru a stabili cu un grad mai ridicat de certitudine dacă un număr este prim sau compus.

Pseudocodul algoritmului:

Algorithm 1: Jacobian(a, n)

Result: Valoarea simbolului Jacobian pentru un număr dat(a), în raport cu baza dată(n)

```
res = 1;
while a do
    if a < 0 then
        schimbă semnul lui a;
        verifică valoarea lui (-1/n);
        if n%4 == 3 then
            res = -res;
        end
    end
    while a par do
        împarte a la 2 pana devine impar;
        verifică valoarea lui (2/n);
        if n%8 == 3 sau n%8 == 5 then
            res = -res;
        end
    end
    schimbă(a, n);
    verifică valoarea lui (a/n) · (n/a);
    if a%4 == 3 sau n%4 == 3 then
        res = -res;
    end
    recalculează restul şi reia procesul;
    a% = n;
end
if n == 1 then
    return res;
end
return 0;
```

Algorithm 2: Solovay-Strassen(n, k)

Result: Adevărat dacă numărul este probabil prim, fals dacă numărul este compus

```
for i = 1 : k do
    generează un număr aleator a din intervalul [1, n-1];
    Jacobian(a, n);
    calculează rezultatul lui Euler a^((n-1)/2) mod n;
    if Jacobian ≠ euler_result then
        return false;
    end
end
return true;
```

Frobenius[3] Fie n un număr impar care nu este pătrat perfect.

Definim indexul Frobenius($Ind_F(n)$) ca fiind cel mai mic număr c din intervalul [-1,2,3,4,5,6...] astfel încât simbolul Jacobi $J(c/n) \neq 1$.

Fie z:

$$z = \begin{cases} 2 + \sqrt{c}, & c = -1, 2 \\ 1 + \sqrt{c}, & c \geq 3. \end{cases}$$

Putem afirma că n este Frobenius probabil prim dacă:

$$z^n \equiv \overline{z} \mod n.$$

Pseudocod:

Algorithm 3: Frobenius(n)

Result: Adevărat dacă numărul este probabil prim, fals dacă numărul este compus
verifică dacă n este pătrat perfect;
calculeaza c = indexul Frobenius;
calculează $z^n \mod n$ prin înmulţiri repetate;
if $z^n \mod n == \overline{z} \mod n$ **then**
  return true;
else
  return false;
end

2.2 Analiza complexităţii soluţiilor

Solovay-Strassen [4]

Pentru a calcula partea exponenţială ($a^{\frac{p-1}{2}} \mod n$) avem complexitatea de $O((\log_2 n)^3)$, iar pentru calculul simbolului Jacobian avem complexitatea de $O((\log_2 n)^2)$. Datorită faptului ca algoritmul Solovay-Strassen este repetat pentru o mai bună acurateţe, complexitatea sa finală va fi $O(k \cdot (\log_2 n)^3)$, unde k reprezintă numărul de iteraţii dat în input.

Frobenius [3]

Pentru a calcula indexul Frobenius ne folosim de funcția Jacobian de la algoritmul Solovay-Strassen, deci complexitatea va fi $O((\log_2 n)^2)$, iar pentru calcularea lui $z^n \mod n$ vom folosii înmulțiri repetate, care în total vor avea complexitatea $O(\log_2 n)$. Astfel, în final, algoritmul nostru va avea complexitatea $O((\log_2 n)^2)$.

2.3 Prezentarea principalelor avantaje şi dezavantaje pentru soluţiile luate în considerare

Algoritmul Solovay-Strassen are ca prim avantaj faptul că poate calcula cu o acuratețe foarte bună dacă un număr este probabil prim sau compus(mai ales daca rulăm pe un număr cât mai mare de iterații), însă aceasta vine și cu un dezavantaj, timpul de execuție crește odată cu creșterea numărului de iterații. Aceste iterații sunt necesare, deoarce fără ele programul are o probabilitate $\geq \frac{1}{2}$(Afirmația 10 din [1]) de a afirma că un număr compus este compus. Acest fapt reiese din implementarea verificării primalității unui număr, unde algoritmul folosește o variabilă aleasă aleator din intervalul [1, n-1]. Odată cu introducerea interațiilor, programul ajunge la o probabilitate de succes de $(1 - \frac{1}{2^t})$(Teorema 9 din [1]), unde t este numărul de iterații.

Referitor la algoritmul Frobenius, îl putem considera "opusul" algoritmului prezentat anterior. Acesta este mai eficient decât Solovay-Strassen, deoarce nu se folosește de variabile alese aletor, eliminând astfel nevoia de iterații. Cu toate acestea, algoritmul are un dezavantaj nedemonstrat încă. Frobenius poate întoarce prim pentru un număr compus, acest număr fiind denumit "Frobenius pseudoprime"(FPP)(pagina 4 din [3]). Spunem nedemonstrat, deoarce există o ipoteză care afirmă că nu ar exista numere Frobenius pseudoprime(deocamdată fiind doar afirmat că acestea sunt mai mari decât 2^{64}[3]).

3 Evaluare

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

Setul de teste a fost construit în formatul specificat: pe prima linie se va afla numărul de elemente din secvenţă, iar pe următoarea linie numerele care urmează să fie testate.

Testele 0, 5 au fost construite de mână, iar restul testelor cu ajutorul generatorului(test_generator.cpp).

Test 0: primele 10 numere naturale
Test 1: primele 100 numere naturale
Test 2: primele 1000 numere naturale
Test 3: primele 10000 numere naturale
Test 4: numere compuse formate prin a înmulţi repetat 4 cu 3
Test 5: numere compuse foarte mari(apropriîndu-se de 10^{47})
Test 6: primele 40 numere din şirul lui Fibonacci
Test 7: primele numere prime mai mici decât 1000, generate folosind ciurul lui Erastotene

3.2 Specificaţiile sistemului de calcul pe care au fost rulate testele

Hardware
-CPU: Intel® Core™ i5-8265U CPU @ 1.60GHz × 8
-RAM: 15,3 GiB RAM DDR4 2666MHz
-Storage: 256GiB SSD Kingston KC600

Software:
-OS: Ubuntu 20.04.1 LTS 64-bit
-GNOME Version: 3.36.3
-Versiune g++: 9.3.0

3.3 Ilustrarea rezultatelor evaluării soluţiilor pe setul de teste

Pentru fiecare test rulat, am calculat timpul de execuţie pentru fiecare număr din fişierul de intrare. După aceea, am calculat media testului ca fiind suma timpurilor de execuţie pentru fiecare număr împărţită la numărul de elemente din secvenţă.
Pentru algoritmul Solovay-Strassen s-au folosit 100 de iteraţii.
Am rulat fiecare test de 10 ori pentru o accurateţe cât mai bună. Astfel, în tabel se vor afla mediile fiecarui test, unitatea de masură folosită fiind nanosecunde.

Număr Test	Solovay-Strassen	Frobenius
0	2955.4	**62.2**
1	3832.6	**86.3**
2	3665.8	**117.4**
3	3809.4	**140.5**
4	23.9	**22.9**
5	**1433.9**	1938
6	6040.8	**204.4**
7	21779.4	**283.4**

3.4 Prezentarea valorilor obţinute pe teste

Rezultatele sunt pe măsura aşteptărilor, deoarce algoritmul Solovay-Strassen are de realizat un număr de iterații dat, acesta va avea un timp de execuție mai mare în majoritatea cazurilor.
Singura excepție fiind testul 5 în care sunt testate numai numere compuse, de valori cât mai mari, Solovay-Strassen dovedindu-se mai rapid decât Frobenius.
Ambii algoritmi afisează output-ul corect în toate fișierele de ieșire.

4 Concluzii

După toate informațiile acumulate despre acești doi algoritmi am ajuns la concluzia că ambii sunt destul de folositori.

Aș opta să folosesc algoritmul Solovay-Strassen pentru cazurile cand va fi nevoie să testez numere cât mai mari, unde timpul nu ar fi o problemă.
Când vine vorba de cel de-al doilea algoritm, Frobenius, as opta să il folosesc în cazurile în care este nevoie să testez primalitatea unui numar relativ mic, într-un timp cât mai scurt.

References

[1] Sam Buss. *Randomized Algorithms*. 2014. URL: bluehttps://www.math.ucsd.edu/~sbuss/CourseWeb/Math261C_2014S/2014_05_02_Marco_SolovayStrassen.pdf (visited on December 15, 2020).

[2] GeeksforGeeks. *Primality Test — Set 4 (Solovay-Strassen)*. 2020. URL: bluehttps://www.geeksforgeeks.org/primality-test-set-4-solovay-strassen/ (visited on December 14, 2020).

[3] Sergei Khashin. *Evaluation of the Effectiveness of the Frobenius Primality Test*. 2018. URL: bluehttps://arxiv.org/pdf/1807.07249v1.pdf (visited on December 14, 2020).

[4] Monica Perrenoud. *Randomized and Deterministic Primality Testing*. 2009. URL: bluehttps://algo.epfl.ch/_media/en/projects/bachelor_semester/randomized_and_deterministic_primality_testing.pdf (visited on December 14, 2020).

[5] *Testing for Prime Numbers: The Solovay-Strassen Algorithm*. URL: bluehttp://www-math.ucdenver.edu/~wcherowi/courses/m5410/ctcprime.html (visited on December 14, 2020).