Tema la Analiza Algoritmilor
- etapa 3 -

Pavăl Bogdan-Costin

322CA Facultatea de Automatică și Calculatoare, UPB
bogdan_costin.paval@stud.acs.upb.ro

1 Introducere

1.1 Descrierea problemei rezolvate

Problema rezolvată în această analiză este "Identificarea numerelor prime". Aceasta primește un set de numere întregi pozitive și găsește toate numerele prime din secvență.

1.2 Exemple de aplicații practice pentru problema aleasă

În zilele noastre se cunosc, în special, algoritmi care recunosc întotdeauna corect un număr prim, iar cu probabilitate mică ar putea declara eronat că anumite numere compuse sunt de fapt prime. Acesta este în sine un fapt foarte important, pentru că toţi algoritmii importanţi de criptografie cu cheie publică se bazează în construirea cheii pe generarea de numere prime, care la rândul ei are la bază testarea primalităţii. Asta înseamnă de fapt că toți cei care folosesc RSA - o metodă foarte răspândită de criptare (de exemplu atât Netscape, cât şi Internet Explorer folosesc această metodă pentru anumite comunicaţii criptate prin Internet) au şansa să folosească o cheie care poate fi "spartă" uşor, pentru că nu e compusă din produsul a două numere prime. Dar, folosind tehnica amplificării, se poate reduce probabilitatea acestui eveniment la o valoare suficient de mică pentru a face algoritmul foarte folositor în practică.

1.3 Specificarea soluțiilor alese

Algoritmii aleși de mine sunt Solovay-Strassen și Lucas, care se ocupă cu determinarea caracterului de primalitate al unui număr. Primul algoritm este de tip probabilistic și determină dacă un număr este compus sau probabil prim, iar al doilea algoritm, Lucas, este o metodă mai teoretică de a determina dacă un număr este prim.

1.4 Specificarea criteriilor de evaluare alese pentru validarea soluțiilor

Pentru a pune în evidență diferențele și eficiența acestor algoritmi, testarea trebuie facută pe un set variat de teste. Astfel, am generat numere aleatorii cu ajutorul site-ului random.org pentru a forma cele 25 de teste de intrare pe care se realizează testarea algoritmilor. Acestea au trei tipuri de dificultate și conțin cel mult 100.000 de numere mai mici decât 10.000. Două dintre cele 25 de teste conțin doar numere prime pentru a se observa comportamentul algoritmilor pe acest caz, dar și dacă sunt identificare corect numerele 100% prime.

2 Prezentarea soluțiilor

2.1 Descrierea modului în care funcționează algoritmii aleși

Solovay-Strassen

Selectăm un număr n pentru a îi testa primalitatea și un număr aleatoriu a care se află în intervalul [2, n-1] și calculăm jacobianul (a/n). Dacă n este un număr prim atunci jacobianul va îndeplini condiția (i) a lui Euler. Dacă nu satisface condiția dată, atunci n este număr compus și programul se va opri. La fel ca orice alt test de primalitate probabilistică, acuratețea sa este direct proporțională cu numărul de iterații, iar pentru a obține rezultate mai precise, se rulează testul pentru mai multe iterații.

$$(a/p) = a^{((p-1)/2)} \% p \quad \textbf{Condition (i)}$$

Algoritmul nu garantează că returnează mereu răspunsul corect. Dacă n nu este găsit ca fiind un număr compus după câțiva pași, atunci n poate fi declarat număr probabil prim. Astfel, numărul n se numește pseudoprim Euler-Jacobi.

```
Algorithm for Solovay-Strassen:
Step 1    Pick a random element a < n
Step 2    if gcd(a, n) > 1 then
Step 3        return COMPOSITE
Step 4    end if
Step 5    Compute a^(n - 1)/2 using repeated squaring
          and (a/n) using Jacobian algorithm.
Step 6    if (a/n) not equal to a^(n - 1)/2 then
Step 7        return composite
Step 8    else
Step 9        return prime
Step 10   endif
```

Lucas

În teoria numerelor computaționale, Testul Lucas este un test de primalitate pentru numere naturale și poate testa primalitatea oricărui tip de număr. Testul Lucas pornește de la Mica Teoremă a lui Fermat, și anume: dacă p este prim și a este un număr întreg, atunci a^p este congruent cu a (mod p).

Testul lui Lucas spune că un număr pozitiv n este prim dacă există un număr întreg a (1 < a < n) astfel încât:

$$a^{n-1} \equiv 1 \pmod{n}$$

Și pentru fiecare factor prim q din (n - 1), atunci:

$$a^{(n-1)/q} \not\equiv 1 \pmod{n}$$

Acest algoritm este destul de complicat și ineficient în comparație cu alte teste de primalitate. Principalele probleme sunt găsirea factorilor n-1 și alegerea potrivită a termenului a.

```
lucasTest(n):
If n is even
    return composite
Else
   Find all prime factors of n-1
   for i=2 to n-1
      pick 'a' randomly in range [2, n-1]
      if a^(n-1) % n not equal 1:
          return composite
      else
          // for all q, prime factors of (n-1)
          if a^(n-1)/q % n not equal 1
             return prime
   Return probably prime
```

2.2 Analiza complexității soluțiilor

Solovay-Strassen

Operația de calculare a jacobianului se realizează în complexitate O($log^2 n$).

Operația exponențială de la pasul 5 din algoritm se realizează în complexitate O($log^3 n$). Complexitatea finală este dată de suma celor două complexități menționate anterior, fiind aplicate pentru k iterații (k = valorile diferite a pe care se face testarea):

$$\text{O(k * } log^2 n) + \text{O(k * } log^3 n) = \text{O(k * } log^3 n)$$

Lucas

Găsirea factorilor primi ai numărului n - 1 se realizează în complexitate O($\sqrt{n}$).

Operația exponențială din algoritm se execută pentru toți factorii primi ai lui n - 1 găsiți (numărul total notat cu k), astfel această secvență se realizează în complexitate O(k * $log^3 n$). Complexitatea finală este dată de suma celor două complexități menționate anterior, fiind aplicate pentru n - 2 iterații (cum a $\in$ [2, n - 1], înseamnă că au loc n - 2 operații):

$$O(\sqrt{n}) + O(k * (n - 2) * log^3 n) = O(k * (n - 2) * log^3 n)$$

2.3 Prezentarea principalelor avantaje și dezavantaje pentru soluțiile luate în considerare

Avantajele generale pentru ambii algoritmi sunt: se poate găsi primalitatea chiar și a numerelor mari, iar dacă un număr este prim, atunci algoritmii cu siguranță vor returna că numărul este prim. Vorbind despre dezavantaje comune, în cazul celor doi algoritmi trebuie găsite valori potrivite pentru a.

Solovay-Strassen

Avantajul major al algoritmului Solovay-Strassen este timpul de execuție, fiind polinomial comparativ cu dimensiunea testelor. Cu cât se realizează mai multe iterații, cu atât crește exactitatea rezultatului algoritmului, dar astfel crește timpul de execuție. Fiind un algoritm de tip probabilistic, nu se garantează că rezultatul este corect 100% în cazul unui număr probabil prim.

Lucas

Avantajul major al algoritmului Lucas este stabilitatea mult mai bună, având o rată de eroare mai mică. Adică un număr compus pe care algoritmul Solovay-Strassen l-ar identifica număr prim, algoritmul Lucas îl returnează ca fiind compus. Acest avantaj este tras în jos de dezavantajul timpului de execuție, fiind semnificativ mai mare decât timpul de la algoritmul Solovay-Strassen (comparație facută pe același set de teste). Chiar dacă teoretic algoritmul poate testa primalitatea numerelor mari, din cauza timpului mare de execuție, returnarea rezultatului durează foarte mult sau poate chiar să nu fie primit (există situații în care rularea nu se termină). Având la bază găsirea factorilor primi ai lui n - 1, acest lucru influențează semnificativ complexitatea, fiind mai mare decât cea de la algoritmul anterior.

3 Evaluare

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

În funcție de fișierele de intrare numite de forma "testX.in" aflate într-un folder numit "in", se va genera fișierul de ieșire "test.out" la rularea fiecărui test de intrare. Detalii despre cele 25 de teste pe care se face testarea:

- primele (1 - 5) sunt de tip easy (se verifică pe teste de bază, având maxim 15 numere mici, pentru a se observa dacă programul returnează răspunsul corect);
- următoarele (6 - 19) sunt medii, având 1.000-7.000 numere din intervalul [0, 10.000);
- ultimele (20 - 25) sunt de tip hard (testarea se face pe un număr mare de elemente - până la 100.000 numere, iar două teste conțin doar numere prime).

Checker-ul pune conținutul fiecărui test de intrare în fișierul "./test.in" și afișează rezultatul fiecărui program în fișierul "./test.out". Acest fișier se compară cu conținutul fișierului corespunzator din directorul out, ce conține testele corecte.

Scopul testelor variate este de a testa algoritmii pe diferite tipuri de intrare și pentru a putea compara eficiența acestora din punct de vedere al duratei execuției. Apoi, se poate forma o concluzie privind situațiile potrivite de aplicare a fiecărui algoritm.

3.2 Specificațiile sistemului de calcul pe care au fost rulate testele (procesor, memorie disponibilă)

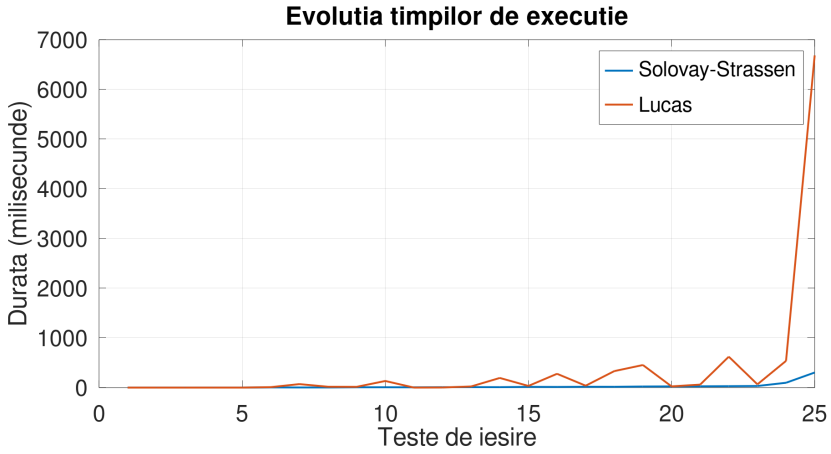
Teste au fost rulate pe unitatea cu specificațiile:

- procesor: Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz 1.99 GHz;
- memorie RAM: 8.00 GB (din care 7.85 GB utilizabili);
- memorie: 500 GB (din care 320 GB liberi).

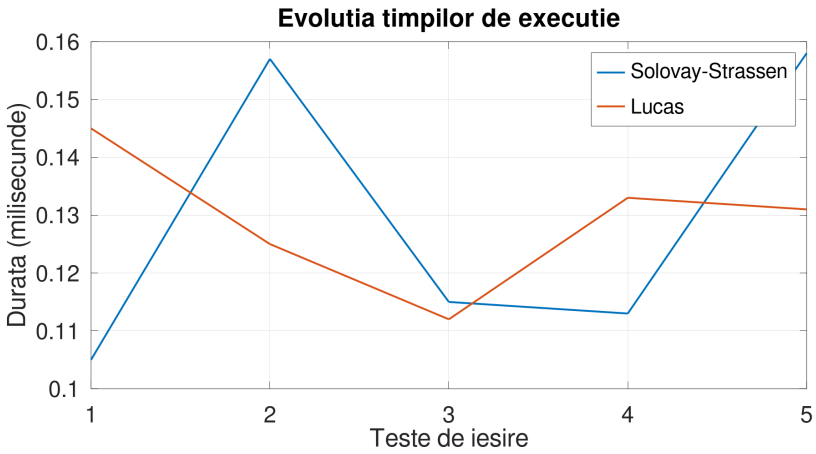
3.3 Ilustrarea, folosind grafice / tabele, a rezultatelor evaluării soluțiilor pe setul de teste

Timpii de execuție sunt exprimați în microsecunde.

Nr. test	timp Solovay-Strassen	timp Lucas
1	105	145
2	157	125
3	115	112
4	113	133
5	158	131
6	4.090	9.172
7	2.845	70.322
8	2.845	16.935
9	6.507	15.342
10	7.042	133.189
11	5.591	255
12	8.304	2.033
13	9.548	22.890
14	8.804	194.033
15	14.803	33.621
16	12.970	277.627
17	16.864	38.285
18	16.244	331.935
19	21.933	454.102
20	24.264	24.304
21	25.871	60.180
22	28.648	621.977
23	32.266	65.769
24	97.266	537.694
25	303.189	6.683.547
Media	26.020	383.754



Primele 5 teste având timpi mici de execuție, am reprezentat și un grafic doar pentru aceste teste:



3.4 Prezentarea valorilor obținute pe teste

Din al doilea grafic și din tabelul anterior, se poate observa că la primele 5 teste, cei doi algoritmi au timpi de excuție asemănători, fiind teste cu un număr mic de elemente. Testele 6 - 14 se executa într-un timp crescător (pentru că si testele devin mai mari), fiind rulate intre 2.000 microsecunde si 9.000 microsecunde. La Lucas nu poate fi găsită vreo regulă, uneori crește foarte mult, alteori scade. Testul 11 conține numere cu o singură cifră, iar algoritmul Lucas s-a descurcat într-un timp extrem de scurt față de Solovay-Strassen. La următoarele, timpii la Solovay-Strassen cresc constant, pe când la Lucas, asemănător ca la testele precedente, nu se respectă nicio regulă.

Așa cum arată și media prezentă pe ultima linie a tabelului, se poate concluziona că la o rulare pe un număr de 25 de teste variate, cu un număr variat de numere diverse, alese aleatoriu, algoritmul Solovay-Strassen are media timpilor de execuție de 15 ori mai mică decât media timpilor de la algoritmul Lucas.

4 Concluzii

În urma analizei facute, aș folosi într-o implementare algoritmul Solovay-Strassen datorită avantajelor pe care acesta le oferă (timpul mic de rezolvare, acuratețea) și aș încerca să îmbunătățesc și să reduc dezavantajele pe care acest algoritm le are. Dacă aș avea o situație în care știu de la început că urmează să lucrez doar cu numere mici, atunci aș alege algoritmul Lucas, pentru că am observat că acesta se descurcă foarte bine în această situație.

5 Bibliografie

- https://www.cs.cmu.edu/ mihaib/articole/complex/complex-html.html
- https://www.geeksforgeeks.org/primality-test-set-4-solovay-strassen/
- https://www.geeksforgeeks.org/lucas-primality-test/
- https://math.dartmouth.edu/ carlp/PDF/lucastalk.pdf
- https://arxiv.org/pdf/2006.08444.pdf
- https://www.wikiwand.com/en/Solovay%E2%80%93Strassen_primality_test