Tema Analiza Algoritmilor - Etapa 3

Andrei-Horia Ignat - 324CA

[1] Universitatea Politehnica din Bucuresti
[2] Facultatea de Automatica si Calculatoare

Abstract. Aceasta lucrare este orientata asupra verificarii conditiei de numar prim, pentru fiecare numar dintr-o secventa data si contine teoretizarea a doi algoritmi, Fermat si Miller-Rabin, implementarea si testarea acestora, analiza complexitatii precum si comparatia dintre ei.

Keywords: Numar prim · algoritmul Fermat · algoritmul Miller-Rabin · Numere Carmichael · Strong pseudoprimes · Ciurul lui Eratostene

Cuprins

1 Introducere

1.1 Descrierea problemei rezolvate

Problema rezolvata in cadrul acestei teme va fi cea a **identificarii numerelor prime** dintr-un set de numere date. Avand la dispozitie un vector de numere intregi pozitive, trebuie determinat care dintre acestea sunt prime.

1.2 Exemple de aplicatii practice pentru problema aleasa

Numerele prime au prezentat inca din antichitate un interes major asupra matematicienilor, acestea avand numeroase aplicatii in matematica, cu preponderenta in teoria numerelor, dar si in informatica, jucand un rol extrem de important in **RSA**, algoritm criptografic cu chei publice, prin generarea celor doua numere prime mari pe care se bazeaza intreaga criptare.

1.3 Specificarea solutiilor alese

Pentru a rezolva problema idenficarii numerelor prime, voi folosi doi algoritmi probabilisti, care au ca scop verificarea conditiei de numar prim, si anume **algoritmul Fermat** si **algoritmul Miller-Rabin**.

Ulterior, voi descrie functionalitatea acestora, voi calcula complexitatea pentru fiecare algoritm si voi prezenta diverse avantaje si dezavantaje.

1.4 Criterii de evaluare pentru solutia propusa

Intrucat ambii algoritmi sunt de tip probabilistic, am generat un set variat de teste pentru a asigura buna functionare a acestora.

Cele 10 teste create se impart in 4 categorii:

- 3 teste de dimensiune redusa ce contin numere mici, pentru a testa rapid corectitudinea algoritmilor si pentru a rezolva eventualele bug-uri.
- 3 teste de dimensiune medie cu numere foarte mari (unul contine inclusiv numere intre 10 000 000 si 50 000 000), pentru a valida algoritmii pe numere mari.
- 2 teste de dimensiune semnificativ mai mare ca a celor de pana acum (99 500 si 941 000 elemente), cu numere mai mici decat 500 000, pentru a verifica functionarea corespunzatoare pentru seturi mai mari de numere.
- 2 teste de dimensiune mare (in ultimul se gasesc 6 000 040 elemente), ce contin numere mari (intre 1 000 si 100 000 000)

2 Prezentarea solutiilor

2.1 Fermat primality test

2.1.1 Descrierea modului in care functioneaza algoritmul

Se bazeaza pe **mica teorema a lui Fermat**: Daca p este un numar prim, atunci $a^{p-1} \equiv 1 \pmod{p}$, pentru orice intreg a relativ prim cu p, $(a, p) = 1$.

Testam daca n este prim. Algoritmul genereaza valori random pentru baza a, din intervalul $[2, n-2]$ iar daca $(a, n) = 1$ (altfel n va fi compus) se calculeaza valoarea expresiei $a^{n-1} \pmod{n}$. Daca aceasta este diferita de 1, atunci n sigur este compus. In caz contrar, n va fi prim sau pseudoprim si vom relua algoritmul prin generarea unei noi valori pentru a.

Astfel, se va specifica o valoare k, care reprezinta de cate ori este repetat testul Fermat. Cu cat valoarea lui k este mai mare, cu atat sansele ca algoritmul sa functioneze corect (sa nu obtina numere pseudoprime) cresc. Cu toate acestea, exista o categorie de numere pentru care sunt sanse mari ca algoritmul sa nu functioneze: **numerele Carmichael**, numere compuse n, pentru care $a^{n-1} \equiv 1 \pmod{n}$, oricare ar fi a intreg, relativ prim cu n. Astfel, daca primin ca input un numar Carmichael, notat cu n si avem ghinionul de a genera doar numere a, relativ prime cu n, $(a, n) = 1$, atunci numarul n va fi considerat prim de catre acest algoritm.

2.1.2 Analiza complexitatii solutiei

Calculul expresiei $a^{n-1} \pmod{n}$ va fi realizat in O(log n), pe baza faptului ca ridicarea la putere se calculeaza in O(log n) ($a^n = a^{\frac{n}{2}} \cdot a^{\frac{n}{2}}$, pentru n par si $a^n = a^{\frac{n-1}{2}} \cdot a^{\frac{n-1}{2}} \cdot a$, pentru n impar), iar intregul test de numar prim va avea complexitatea O(klog n) (k este numarul de repetari al testului Fermat), deci O(log n), pentru un singur numar n.

Intrucat inputul problemei este un vector V de dimensiune N, iar pentru fiecare element x din vector, programul dureaza O(log x), complexitatea totala a programului este un O(N log (max(V))), unde max(V) este cel mai mare element din vector.

Despre complexitatea spatiala, algoritmul Fermat aplicat unui singur numar foloseste O(1) spatiu. In problema, input-ul este reprezentat de un vector de dimensiune N, deci vom obtine O(N) complexitate spatiala.

2.1.3 Prezentarea principalelor avantaje si dezavantaje

Spre deosebire de algoritmii clasici de verificare a conditiei de numar prim, algoritmul Fermat are avantajul ca este mult mai rapid, intrucat ofera o complexitate mult mai buna, O(log n) in comparatie cu O(n).

Insa, acest algoritmul nu este atat de precis (pentru numere Carmichael de exemplu), iar pentru a asigura buna functionalitate a algoritmului, testul trebuie repetat de k ori. Astfel, acest k poate sa difere semnificativ de la un input la altul. Exista o reteta generala in alegerea lui k, demonstrata probabilistic, conform careia k va depinde de n. Dar daca vom construi algoritmul conform acestei premise, complexitatea temporala a acestuia va creste semnificativ (k nu va mai fi o constanta ci va depinde de n), altfel performanta sa va scadea, ceea ce nu ne dorim.

2.2 Miller-Rabin primality test

2.2.1 Descrierea modului in care functioneaza algoritmul

Este asemanator cu Fermat si functioneaza astfel: Se pleaca de la n, numarul care va fi testat si presupunand ca n este impar, mai mare decat 2 (altfel este compus), se gaseste factorul d impar, astfel incat $n-1 = 2^s \cdot d$. Ulterior se genereaza un numar random a, din intervalul $[2, n-2]$ si se calculeaza valoarea expresiei $x = a^d (\text{mod } n)$. Daca se obtine valoarea 1 sau $n-1$, atunci n va fi prim, sau pseudoprim. Daca nu, se dubleaza valoarea lui d si se recalculeaza expresia de mai sus (adica se calculeaza $x^2 (\text{mod } n)$) iar daca se obtine valoarea $n-1$, n va fi prim, sau pseudoprim. Continuam procedeul de dublare al lui d, pana se ajunge la valoarea $\frac{n-1}{2}$ inclusiv. Daca nu se obtine deloc valoarea $n-1$, n sigur va fi compus.

In cazul in care n va fi prim, sau pseudoprim, se repeta din nou procedeul pentru o alta valoare random a lui a. Din acelasi motiv ca la Fermat, se seteaza o valoare pentru k si se va repeta procedeul de k ori.

Algoritmul se bazeaza pe observatia ca daca, $x^2 \equiv 1 \pmod p$, cu x intreg si p prim, atunci $x \equiv 1 (\text{mod } p)$ sau $x \equiv -1 (\text{mod } p)$. Aceasta observatie aplicata succesiv in relatia data de mica teorema a lui Fermat conduce la algoritmul Miller-Rabin. Astfel, daca am avea $a^{n-1} \equiv 1 (\text{mod } n)$, cum $n = 1+2^s \cdot d$, atunci, pe baza observatiei anterioare, $a^{2^{s-1} \cdot d} \equiv 1 (\text{mod } n)$ sau $a^{2^{s-1} \cdot d} \equiv -1 (\text{mod } n)$. Daca se obtine valoarea -1 (valoarea care se atinge in algoritm tot dubland valoarea lui d), ne oprim. Daca se otine 1, se continua aplicarea observatiei anterioare si se obtine $a^{2^{s-2} \cdot d} \equiv 1 (\text{mod } n)$ sau $a^{2^{s-2} \cdot d} \equiv -1 (\text{mod } n)$. Verificam din nou ce valoarea se obtine si procedam identic. Cum d este impar, demonstratia se incheie cel tarziu la calcularea lui $a^d (\text{mod } n)$ (in algoritmul propriu-zis, asa cum am specificat mai sus, se incepe cu aceasta relatie, i se calculeaza valoarea, iar daca nu obtin 1 sau -1, se continua cu urmatoarele expresii de acelasi tip).

2.2.2 Analiza complexitatii solutiei

Complexitatea va fi O(klog n) (k este numarul de repetari al testului Miller-Rabin), deci O(log n), insa va fi semnificativ mai rapid decat Fermat, intrucat se calculeaza valoarea expresiilor $a^d (\text{mod } n)$, $a^{2 \cdot d} (\text{mod } n)$, pana la $a^{\frac{n-1}{2}} (\text{mod } n)$ (log d + s operatii, spre deosebire de log n in cazul Fermat), unde $n-1 = 2^s \cdot d$.

Deoarece problema primeste ca input un vector V de dimensiune N, complixatea totala, ca in cazul algoritmului Fermat, va fi O(N log (max(V))), unde max(V) este cel mai mare element din vector.

La fel ca in cazul Fermat, algoritmul Miller-Rabin are o complexitate spatiala de O(1), pentru un singur numar. Deci complexitatea spatiala totala, pentru intregul vector de dimensiune N, va fi O(N).

2.2.3 Prezentarea principalelor avantaje si dezavantaje

Adevarata putere a acestui algoritm consta in doua aspecte, legate de corectidutine (stabilitate) si performanta (timp de executie).

- Primul se refera la o mult mai buna stabilitate a algorimului, astfel ca o buna parte din numerele compuse, pentru care algoritmul Fermat afirma ca sunt prime, Miller-Rabin le va detecta ca fiind compuse. Practic sansele de eroare pentru acest algoritm sunt mult mai mici.
- Al doilea consta in valoarea lui k, adica de cate ori se repeta algoritmul, care, in general este mai redusa decat in cazul Fermat. Astfel, datorita acestui fapt, dar si din calculul efectiv al numarului de operatii pentru fiecare repetare (log d + s in comparatie cu log n), algoritmul Miller-Rabin va fi semnificativ mai rapid decat algoritmul Fermat.

In cazul Fermat, se poate afirma despre un numar pseudoprim ca este prim, lucru care se intampla si cu algoritmul Miller-Rabin, asa cum am precizat anterior. Aceasta categorie de numere poarta numele de **strong pseudoprimes**, insa ele sunt mult mai putine decat cele care dau astfel de erori in algoritmul Fermat.

3 Evaluare

3.1 Descrierea modalitatii de construire a setului de teste folosite

Creez cate un test de input si un fisier de tip ref, ce contine rezultatul corect, dupa modelul urmator:

make_single_test(numar_test, limita_inferioara, limita_superioara,
numar_prime, numar_compuse, numar_carmichael)

Fiecare test de input va contine numar_prime numere prime, numar_compuse numere compuse, numar_carmichael numere Carmichael (sau maximul de numere prime / compuse posibil dintr-un interval specificat), toate acestea fiind in intervalul [limita_inferioara, limita_superioara].

Folosesc **Ciurul lui Eratostene** pentru a genera toate numerele prime mai mici decat un numar dat si apoi le aleg pe cele mai mari decat o limita inferioara. Totodata, acest algoritm imi va da informatii si despre numerele compuse.

Astfel, voi alege random numar_prime din numerele prime generate de ciur, mai mari decat acea limita inferioara, iar pentru cele compuse si Carmichael procedez identic. Amestec numerele prime cu cele compuse din intervalul dorit, la care adaug si numere Carmichael (am creat o lista cu toate numerele Carmichael de maxim 6 cifre), fac o permutare random a acestei liste de numere, care va constitui input-ul dintr-un fisier. Testul din ref va contine lista cu numere prime generata de ciur.

3.2 Specificatiile sistemului de calcul pe care am rulat testele

Sistemul de calcul pe care am rulat testele are urmatoarele specificatii:

- Procesor: AMD Ryzen 7 4800H
- Memorie: 16 GB RAM

3.3 Rezultatele evaluarii solutiilor pe setul de teste

```
horia@TUF-FA506IV:~/facultate/an II/sem 1/aa/PrimeNumbers/etapa2$ make run_tests
g++ -Wall -Wextra -O2 algo1.o algo2.o main.o -o main
./main
----------------------------------------------------------------------
---------------------- TEMA AA - ETAPA 2 - 2020 ----------------------
----------------------------------------------------------------------
----------------------------- FERMAT -----------------------------
test0                                                   PASSED [5/5]
time taken                                              186 micro
test1                                                   PASSED [5/5]
time taken                                              113 micro
test2                                                   PASSED [5/5]
time taken                                              928 micro
test3                                                   PASSED [5/5]
time taken                                              7793 micro
test4                                                   PASSED [5/5]
time taken                                              36224 micro
test5                                                   PASSED [5/5]
time taken                                              297965 micro
test6                                                   PASSED [5/5]
time taken                                              119254 micro
test7                                                   PASSED [5/5]
time taken                                              637387 micro
test8                                                   PASSED [5/5]
time taken                                              1631521 micro
test9                                                   PASSED [5/5]
time taken                                              26427462 micro
----------------------------------------------------------------------
FERMAT POINTS                                                 [50/50]
-------------------------- MILLER_RABIN --------------------------
test0                                                   PASSED [5/5]
time taken                                              264 micro
test1                                                   PASSED [5/5]
time taken                                              45 micro
test2                                                   PASSED [5/5]
time taken                                              209 micro
test3                                                   PASSED [5/5]
time taken                                              861 micro
test4                                                   PASSED [5/5]
time taken                                              4097 micro
test5                                                   PASSED [5/5]
time taken                                              32935 micro
test6                                                   PASSED [5/5]
time taken                                              26223 micro
test7                                                   PASSED [5/5]
time taken                                              134981 micro
test8                                                   PASSED [5/5]
time taken                                              329108 micro
test9                                                   PASSED [5/5]
time taken                                              4179574 micro
----------------------------------------------------------------------
MILLER RABIN POINTS                                           [50/50]
----------------------------------------------------------------------
TOTAL POINTS                                                [100/100]
horia@TUF-FA506IV:~/facultate/an II/sem 1/aa/PrimeNumbers/etapa2$
```

3.4 Prezentarea valorilor obtinute pe teste

Corectitudinea algoritmilor a fost demonstrata prin setul de teste creat. Se observa si timpul de executie, exprimat in microsecunde, pentru fiecare algoritm, aplicat pe toate cele 10 teste.

Astfel, pentru a trece toate testele, am aplicat algoritmul Fermat de 50 ori pentru fiecare numar din secventa data ca input, primele 9 teste avand un timp de executie sub 1.7 secunde (dintre care primele 8 chiar sub 1 secunda), iar ultimul test, 26 secunde.

Pentru Miller-Rabin, a fost suficient sa repet algoritmul doar de 6 ori pentru a trece toate testele, obtinandu-se o performanta mult mai buna, ultimul test durand doar 4 secunde, iar restul, sub 0.3 secunde.

4 Concluzii

In urma analizei amanuntite a celor doi algoritmi, consider ca algoritmul Miller-Rabin ii este superior algortimului Fermat, atat din punctul de vedere al corectitudinii (numarul de input-uri pe care ar putea obtine rezultate eronate este semnificativ mai redus) cat si al performantei (timpul de executiei este mult mai mic).

Asadar, avand in vedere comparatia algoritmilor, in cazul unei probleme practice in care as fi nevoit sa verific daca un numar este prim, as folosi algoritmul Miller-Rabin in marea majoritate a cazurilor, pentru a avea o siguranta mai mare a unui rezultat corect si o viteza de raspuns a programului marita.

Totusi, necesitatea algoritmului Miller-Rabin se observa mai ales pentru numere mari, astfel ca pentru a verifica daca un numar relativ mic este prim, este suficient sa folosesc algoritmul Fermat. Astfel codul va fi mai simplu de redactat, deci cel mai probabil il pot scrie intr-un timp mai scurt, iar performanta nu va scadea prea mult, asa cum se observa si din primele 3 teste.

5 Bibliografie

1. GeeksforGeeks Fermat
2. GeeksforGeeks Miller-Rabin
3. RSA
4. Primality Tests Based on Fermat's Little Theorem
5. Primality tests from CP-Algorithms
6. Fermat and Miller-Rabin primality tests
7. Introduction to Algorithms, Third Edition