Tema Analiza Algoritmilor - Etapa 1

Andrei-Horia Ignat - 324CA

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

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.

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.

2 Specificarea solutiilor alese

2.1 Fermat primality test

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 algoritmul nu va functiona niciodata: **numerele Carmichael**, numere compuse n, pentru care $a^{n-1} \equiv 1 \pmod{n}$, oricare ar fi a intreg, relativ prim cu n.

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), deci O(log n), pentru un singur numar.

2.2 Miller-Rabin primality test

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 a lui d, pana se ajunge la valoarea lui $\frac{n-1}{2}$ 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).

Complexitatea va fi O(klog n), 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 $n - 1 = 2^s \cdot d$.

3 Criterii de evaluare pentru solutia propusa

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

Initial voi crea vectori cu numere mici pentru a testa rapid corectitudinea algoritmilor si a rezolva eventualele bug-uri. Ulterior voi genera vectori de lungime variata si cu numere aleatoare, atat numere mari cat si numere mici comparand output-urile obtinute cu o metoda clasica de determinare a numerelor prime.

4 Referinte

1. GeeksforGeeks Fermat
2. GeeksforGeeks Miller-Rabin
3. Primality Tests Based on Fermat's Little Theorem
4. Introduction to Algorithms, Third Edition