Curs de algebră pentru lotul de informatică

Ciobâcă Ştefan

Subiecte de discuție (1)

- Algoritmul lui Karatsuba (incalzire)
- Congruențe, proprietăți generale
- Algoritmul extins al lui Euclid
- Ecuatia dreptei (ax + by = c)
- Ecuatia $ax \equiv b \pmod{m}$
- Teorema chineză a resturilor

Subiecte de discuție (2)

- Teorema lui Fermat. Testul probabilist de primalitate. Numere Carmichael.
- Exponențiere rapidă
- Teorema Miller-Rabin. Testul Miller-Rabin

Algoritmul lui Karatsuba (1)

- Inmultirea scolareasca a doua numere mari (reprezentate pe n biti) are complexitate O(n^2) (n adunari de numere pe 2*n biti)
- Inmultirea Karatsuba este o abordare de tip divide et impera si complexitatea este redusa la O(n^(log3)) ~ O(n * sqrt(n))
- Se bazeaza pe impartirea celor doua numere (presupuse de dimensiuni egale si pare) in doua parti.

Algoritmul lui Karatsuba (2)

- Fie A si B partile in care este impartit primul numar si C, D partile din care este alcatuit cel de-al doilea numar.
- Atunci AB * CD = (Ax + B) * (Cx + D) = AC*x^2 + (AD+BC)*x + BD = AC*x^2 + x*((A + B) * (C + D) – AC – BD) + BD
- Observam ca pentru a efectua o inmultire a doua numere pe n biti se reduce la a efectua 3 inmultiri pe numere de n / 2 biti.

Algoritmul lui Karatsuba (3)

- In afara de cele 3 inmultiri, se mai efectueaza O(1) adunari si O(1) inmultiri cu x (care reprezinta de fapt shiftari in baza in care lucram)
- Complexitatea este deci data de formula:
- $T(1) = c$, c constanta
- $T(n) = 3 * T(N / 2) + O(n)$ (shiftarile si adunarile se fac in timp liniar).

Algoritmul lui Karatsuba (4)

- Rezolvarea recurentei se poate face calculand T(2^i), de exemplu
- Sau folosind teorema master (vezi Cormen, Leiserson & Rivest – Introducere in Algoritmi)
- Oricum ideea e ca T(n) = O(n^(log3)). Memoria consumata de o implementare corecta este de acelasi ordin. Încercați să implementați.

Congruențe (1)

- Relația de congruență a două numere
$$a \equiv b \pmod{n}$$
- Atenție! (mod n) se citeşte relativ la congruență (nu la b)
- Se mai notează:
$$a \equiv_m b$$
- Este o relație de echivalență (reflexivitate, tranzitivitate, simetrie)
- Prin definiție, a ≡ b (mod m) dacă şi numai dacă m divide a – b.

Congruențe (2)

- Fie $Z_n = \{0, 1, \ldots, n - 1\}$. Înzestrăm această mulțime cu înmulțirea modulo n şi adunarea modulo n. Orice număr întreg este ≡ cu un număr din Z_n (modulo n).
- Astfel, când vom face referire la o adunare sau la o înmulțire în Z_4, 3+5 va fi 0, iar 3*5 va fi 3.
- Dacă $a \equiv b \pmod n$ si $c \equiv d \pmod n$ atunci $a + c \equiv b + d \pmod n$.

Congruențe (3)

- Dacă $a \equiv b \pmod{n}$ atunci $ac \equiv bc \pmod{n}$.
- Dacă $ac \equiv bc \pmod{n}$ atunci $a \equiv b \pmod{n/cmmdc(c,n)}$
- cmmdc(c,n) este cel mai mare număr natural care divide atât c cât şi n
- Cam atât despre proprietățile teoretice

Algoritmul extins al lui Euclid (1)

- Pentru orice două numere întregi x şi y care nu sunt simultan 0, există a şi b întregi astfel încât ax + by = (x, y). Prin (x,y) se înțelege cmmdc(x, y).
- Algoritmul extins al lui Euclid găseşte cel mai mare divizor comun a două astfel de numere, precum şi a, b cu proprietatea de mai sus.

Algoritmul extins al lui Euclid (2)

- Să vedem cum funcționează mai întâi algoritmul simplu al lui Euclid
- Acesta se bazează pe o observația foarte simplă:
- Dacă x şi y nu sunt simultan 0, atunci (x,y) = (y, x mod y). Mai mult (x,y) = (|x|,|y|).
- Iterativ, algoritmul lui Euclid efectuează următorii paşi:

Algoritmul extins al lui Euclid (3)

- Se porneşte cu $r_{-1} = |x|$, $r_0 = |y|$. La fiecare pas, se aplică teorema împărții cu rest pentru numerele r_i şi r_{i-1}.
- $r_{-1} = r_0 * q_0 + r_1$ cu $r_i < r_{i-1}$ pentru $i = 1,n+1$
- $r_0 = r_1 * q_1 + r_2$
- …
- $r_{n-1} = r_n * q_n + r_{n+1}$
- La un moment dat, r_{n+1} va fi 0 şi algoritmul se va încheia cu răspunsul r_n.

Algoritmul extins al lui Euclid (4)

- Este evident că r_n va fi răspunsul corect, deoarece pe toată durata execuției algoritmului cmmdc(r_{i-1},r_i) = cmmdc(r_i,r_{i+1}).
- Complexitatea este dată de teorema lui Lamé: $n + 1 \leq [\log_{10} y]$. (ideea de demonstrație: $r_i <= F_{n-i+2}$, unde F_i este termenul i din şirul lui Fibonacci)
- Cum se poate extinde algoritmul astfel încât să calculeze şi combinația liniară aferentă?

Algoritmul extins al lui Euclid (5)

- Putem presupune fără a pierde din generalitate că x şi y sunt pozitive.
- Pentru fiecare rest r_i, ținem şi o pereche $W_i=(u_i, v_i)$, cu proprietatea că $u_i * x + v_i * y = r_i$.
- La fiecare pas, pe lângă efectuarea împărțirii propriu-zise, se calculează şi $W_{i+1} = W_{i-1} – q_i * W_i$. Perechea W_n va fi perechea căutată.

Ecuația ax + by = c (1)

- Ne interesează soluțiile întregi ale ecuației atunci când a şi b nu sunt simultan 0.
- Ecuația ax + by = c admite soluții întregi dacă şi numai dacă (a,b) divide c. Demonstrațiile au tendința de a plictisi lotul de informatică, aşa că vom sări peste…
- Ştiind teorema de mai sus, putem găsi o soluție astfel:

Ecuația ax + by = c (2)

- Folosim algoritmul extins al lui Euclid pentru a găsi x_0, y_0 astfel încât $a*x_0 + b*y_0 = (a,b)$. Dacă înmulțim egalitatea de mai sus cu c/(a,b), obținem soluția căutată de forma $(x_0 * c / (a,b), y_0 * c / (a,b))$.
- Celelalte soluții se pot găsi foarte uşor (încercați să desenați o dreaptă pe o foaie liniată şi observați ce se întâmplă când (a,b) = 1 şi când (a,b) != 1)

Ecuația $ax \equiv b \pmod m$ (1)

- Se observă uşor că se reduce la o ecuație de forma celei discutate mai înainte astfel:
- $ax \equiv b \pmod m \Leftrightarrow$ m divide $ax - b \Leftrightarrow$ există y întreg astfel încât $my = ax - b \Leftrightarrow ax - my = b$
- Dacă găsim o soluție a acestei ecuații, fie ea x_0, atunci $x_0 \bmod m$ este soluție a ecuației $ax \equiv b \pmod m$.

Ecuația $ax \equiv b \pmod m$ (2)

- Ecuația în discuție admite exact (a,m) soluții în Z_m. Mai mult, dacă am găsit x0 o soluție din Z_m, atunci toate soluțiile au forma $x_0 + i*m/d$, unde d = (a,m), iar i ia valori între 0 şi d - 1.
- Corolar: Dacă (a,m) = 1, atunci ecuația $ax \equiv b \pmod m$ admite exact o soluție în Z_m.
- Încă o dată, demonstrațiile prea lungi plictisesc…, aşa că sărim peste ele.

Teorema chineză a resturilor (1)

- De ce chineză? Se spune că a fost o dată ca niciodată… un tip (de origine chineză) care când îşi impărțea orezul în grămezi de câte 3 boabe îi rămâneau 2 boabe, când îl împărțea în grămezi de câte 5 îi rămâneau 3 iar când îl împărțea în grămezi de câte 7 îi rămâneau 5. Se pune întrebarea: câte boabe avea tipul respectiv?

Teorema chineză a resturilor (2)

- Formal vorbind, este vorba de un sistem de ecuații de forma:
- $x \equiv b_1 \pmod{m_1}$
- $x \equiv b_2 \pmod{m_2}$
- …
- $x \equiv b_n \pmod{m_n}$
- Teorema chineză ne dă o condiție suficientă pentru existența unei soluții, oferind şi o cale de găsire a acesteia.

Teorema chineză a resturilor (3)

- Dându-se un sistem de forma celui de mai înainte, dacă $(m_i,m_j)=1$ ori de câte ori i != j, atunci sistemul respectiv are o soluție unică în $Z_{m1*m2*\ldots*mn}$.
- Demonstrația afirmației ne dă şi o cale de a găsi respectiva soluție.
- Fie $m = m_1 * m_2 * \ldots * m_n$. Fie $p_i = m / m_i$. Vom vedea în continuare cum se construieşte soluția dorită.

Teorema chineză a resturilor (4)

- Fie x_i o soluție a ecuației $p_i * x = b_i \pmod{m_i}$. Conform corolarului de la ecuația $ax \equiv b \pmod{m}$, ecuația considerată are o singură soluție în Z_{mi}. (de ce? $(p_i,m_i)=1$).
- Se poate arăta foarte uşor că suma $p_i * x_i$ pentru i de la 0 la n – 1 este soluție a ecuației (exercițiu pentru voi). De asemenea, se arată la fel de uşor că soluția este unică în Z_m.

Teorema chineză a resturilor (5)

- Exemplu:
- $x = 2 \pmod 3$ $m_1 = 3, p_1 = 5 * 7$
- $x = 3 \pmod 5$ $m_2 = 5, p_2 = 3 * 7$
- $x = 5 \pmod 7$ $m_3 = 7, p_3 = 3 * 5$
- $35x_1 = 2 \pmod 3$ => x1 = 1
- $21x_2 = 3 \pmod 5$ => x2 = 3
- $15x_3 = 5 \pmod 7$ => x3 = 5
- $x = 35 * 1 + 21 * 3 + 15 * 5 = 35 + 63 + 75 = 173$
- $x \% 3*5*7 = x \% 105 = 68$

Implementări

- Implementați:
  - 1) Algoritmul extins al lui Euclid
  - 2) Folositi 1) pentru a rezolva $ax + by = c$
  - 3) Folositi 2) pentru $ax \equiv b \pmod m$
  - 4) Folositi 3) pentru teorema chineza a resturilor
- ATENTIE LA NEATENTIE! In C/C++ şi Pascal operatorul %, respectiv mod nu întoarce restul corect dpdv matematic.

Teorema lui Fermat (1)

- Daca (a,p) = 1 si p este numar prim, atunci:
  $a^{p-1} \equiv 1 \pmod{p}$

Input: p >= 2

Output “prim” sau “compus”

Alege a aleator între 1 şi p – 1

Dacă (a,p) != 1 atunci output “compus” & halt

Dacă a^{p-1} mod p != 1 atunci output “compus” & halt

Output “prim” & halt

Teorema lui Fermat (2)

- Se poate demonstra că în cazul general, probabilitatea de eroare a testului Fermat este ~0.5. Totuşi există o categorie de numere, numite Carmichael, pentru care testul Fermat eşuează aproape de fiecare dată. Numerele Carmichael sunt numerele compuse care pentru orice a indeplinesc condiția din testul Fermat.

Exponențiere rapidă (1)

- Un lucru important în implementarea algoritmului probabilist bazat pe teorema lui Fermat este calculul lui a^{p-1} mod p.
- În general, cum calculam eficient a^x mod p? Foarte uşor. Ne bazăm pe scrierea în baza 2 a lui x.
- Fie $x = (x_0x_1x_2\ldots x_{31})$ scrierea în baza 2 a lui x. (x_i din {0, 1}). Atunci $x = x_0 + x_1 * 2 + x_2 * 2^2 + \ldots + x_i * 2^i + \ldots + x_{31} * 2^{31}$.

Exponențiere rapidă (2)

- Deci $a^x = a^{x0} \cdot a^{2*x1} \cdot a^{4*x2} \ldots a^{x31*2\hat{}31}$ = suma dupa i = 0 la 31 cu $x_i = 1$ din $a^{2\hat{}i}$.
- Algoritmul este următorul (complexitate O(logx)):
- z = a
- Result = 1
- While x != 0
  - If (x % 2 == 1) Result *= z, Result %= p
  - x /= 2
  - z *= z, z %= p

Teorema Miller-Rabin (1)

- Principalul dezavantaj al testul probabilist al lui Fermat este prezența numerelor de tip Carmichael, pentru care probabilitatea de eroare este foarte mare.
- Testul Miller-Rabin elimină acest dezavantaj. În plus, probabilitatea de eroare este de ¼.
- Se bazează pe următoarea teoremă:

Teorema Miller-Rabin (2)

- Fie n >= 3 număr impar. Fie $n - 1 = 2^s t$, cu t impar. Atunci n este prim ⇔ pentru orice a astfel încât (a,n) = 1

  $a^t \equiv 1 \pmod n$ sau există i între 1 şi s-1 astfel încât $a^{t*2\hat{}i} \equiv -1 \pmod n$.

Testul de primalitate care derivă din teorema de mai sus este evident.

Implementări

- Exponențiere rapidă a^x mod p
- Testul Fermat
- Testul Miller Rabin

Succes!