UNIUNEA EUROPEANĂ

GUVERNUL ROMÂNIEI

Instrumente Structurale
2007-2013

Platformă de e-learning și curriculă e-content pentru învățământul superior tehnic

Proiectarea Algoritmilor

33. Algoritmul de tip Monte Carlo

Bibliografie

[1] C. Giumale – Introducere in Analiza Algoritmilor – cap. 6.1

[2] Cormen – Introducere in algoritmi – cap. 8.3

[3] http://www.soe.ucsc.edu/classes/cmps102/Spring04/TantaloAsymp.pdf

[4] http://www.mersenne.org/

Algoritmi Monte Carlo

- Găsesc o soluție a problemei care nu e garantat corectă (soluție aproximativă).
- Timp = ∞ → soluția corectă a problemei.
- Probabilitatea ca soluția să fie corectă crește o dată cu timpul de rezolvare.
- Soluția găsită într-un timp acceptabil este aproape sigur corectă.

Complexitate algoritmi Monte Carlo

- Definiția 6.1': Algoritmii Monte Carlo au complexitatea $f(n) = O(g(n))$ dacă $\exists\ c > 0$ și $n_0 > 0$ a.î.
  - $\forall n \geq n_0,\ 0 < f(n) \leq c\ \alpha\ g(n)$ cu o probabilitate de cel puțin $1\text{-}n^{-\alpha}$ pentru $\alpha > 0$ fixat si suficient de mare;
  - Probabilitatea ca soluția determinată de algoritm să fie corectă este cel puțin $1 - n^{-\alpha}$.

Exemplu algoritm Monte Carlo

- Problemă: testarea dacă un număr n dat este prim.
- Rezolvare “clasică”:
- Prim-clasic(n)
  - **Pentru** i **de la** 2 **la** sqrt(n)
    - **Dacă** (n mod i == 0) **întoarce** fals;
  - **Întoarce** adevărat

Complexitate:

O(sqrt(n))

Determinarea numerelor prime - complexitate

- Observație: pentru numere mari – operațiile nu mai durează O(1)!
- → Estimăm numărul de operații în funcție de numărul de biți pe care este exprimat numărul.
- → Prim_clasic – $O(2^{k/2})$ unde k = nr. de biți ocupat de n.

Complexitate nesatisfăcătoare!

- "On September 4, 2006, in the same room just a few feet away from their last find, Dr. Curtis Cooper and Dr. Steven Boone's CMSU team broke their own world record, discovering the 44th known Mersenne prime, $2^{32,582,657}$-1. The new prime at 9,808,358 digits is 650,000 digits larger than their previous record prime found last December."

- "On April 12th (2009) , the 46th known Mersenne prime, $2^{42,643,801}$-1, a 12,837,064 digit number was found by Odd Magnar Strindmo from Melhus, Norway! This prime is the second largest known prime number, a "mere" 141,125 digits smaller than the Mersenne prime found last August."

- As of October 2009, 47 Mersenne primes are known. The largest known prime number ($2^{43,112,609} - 1$) is a Mersenne prime. [Wikipedia]

http://www.mersenne.org

Algoritm aleator (I)

- Teorema 6.1 (mica teoremă a lui Fermat): Dacă n este prim → $\forall\ 0 < x < n$, $x^{n-1} \bmod n = 1$.

- Prim1(n,α) // detectează dacă n e număr prim
  - **Dacă** (n ≤ 1 **sau** n mod 2 = 0) **Întoarce** fals
  - Limit = limită_calcul(n,α) // numărul minim de pași pentru // soluția corectă cu P = 1 - $n^{-\alpha}$
  - **Pentru** i **de la** 0 **la** limit
    - x = random(1, n-1) // aleg un număr oarecare
    - **Dacă** (pow_mod(x,n) ! = 1) **Întoarce** fals // testez teorema // Fermat
  - **Întoarce** adevărat

Complexitate?

Algoritm aleator (II)

- Pow_mod(x,n) // calculează x^{n-1} mod n
  - r = 1 // restul
  - **Pentru** m **de la** n-1 **la** 0
    - **Dacă** (m mod 2 ≠ 0) // testez dacă puterea e pară sau nu
      - r = x * r mod n
    - x = (x * x) mod n // calculez x^2 mod n
    - m = m div 2 // înjumătățesc puterea
  - **Întoarce** r

Complexitate:

O(lg(n))

Algoritm aleator (III)

- Problemă: nu putem stabili cu exactitate care este limita de calcul:
  - Nu se poate estima pentru un număr compus n numărul de numere x, 2 < x < n pentru care nu se verifică ecuația;
  - Există numere compuse pentru care orice număr x < n și prim în raport cu n satisface ecuația lui Fermat (ex: nr. Carmichael → 561).
- → Nu știm cu exactitate câte numere sunt!
- → Nu putem calcula probabilitatea!

Altă variantă de algoritm aleator

- Teorema 6.2: Pentru orice număr prim n ecuația $x^2 \bmod n = 1$ are exact 2 soluții:

  $x_1 = 1$ ȘI $x_2 = n - 1$.

- Definiție 6.2: Fie $n > 1$ și $0 < x < n$ două numere a.î. $x^{n-1} \bmod n \neq 1$ sau $x^2 \bmod n \neq 1$, $x \neq 1$ și $x \neq n - 1$. X se numește martor al divizibilității lui n.

Algoritmul Miller-Rabin

- Prim2(n,α)
  - **Dacă** (n ≤ 1 **sau** n mod 2 = 0) **Întoarce** fals
  - limit = limita_calcul(n,α)
  - **Pentru** i **de la** 0 **la** limit
    - x = random(1,n-1)
    - **Dacă** (martor_div(x,n)) **Întoarce** fals
  - **Întoarce** adevărat

Complexitate?

Algoritmul Miller-Rabin (II)

- martor_div(x,n) // determină dacă x e
  // martor al divizibilității lui n
  - r = 1; y = x;
  - **Pentru** m **de la** n-1 **la** 0 // puterea
    - **Dacă** (m mod 2 ≠ 0) // putere impară
      - r = y * r mod n
    - z = y // salvez valoarea lui x
    - y = y * y mod n // calculez y^2 mod n
    - **Dacă** (y = 1 **și** z ≠ 1 **și** z ≠ n-1) // verific teorema 6.2
      - **Întoarce** 1
    - m = m div 2 // înjumătățesc puterea
  - **Întoarce** r ≠ 1 // mica teoremă Fermat (x^{n-1} mod n ≠ 1)

Complexitate:

O(lg(n))

Calcularea numărului de pași

- Teorema 6.3: Pentru orice număr n, impar și compus există cel puțin (n-1) / 2 martori ai divizibilității lui n.
- Caz neinteresant: număr prim pentru că oricum algoritmul întoarce adevărat ($P_{corect}(n) = 1$)!
- Caz interesant: număr compus (impar) ($P_{corect}(n) = ?$):
- x = element generat la un pas al algoritmului ($0 < x < n$);
- P(x) = probabilitatea ca numărul **x** generat din cele n-1 posibilități să fie martor al divizibilității;
- $P(x) \geq (n-1) / 2 * 1 / (n-1) = 0.5$;
- $P_{incorect}(n) = \Pi_{1->limit}(1 - P(x)) \leq 1/2^{limit}$;
- → $P_{corect}(n) \geq 1-2^{-limit} = 1 - n^{-\alpha}$ → limit = α lg(n); →după α lg(n) pași $P_{corect}(n) \geq 1 - n^{-\alpha}$;
- → Complexitate: $O(lg^2(n))$ → în funcție de numărul de biți k → Complexitate: $O(k^2)$

Exemplu de utilizare practică

- Quicksort(A, st, dr)
  - **Dacă** st < dr
    - q ← Partiție(A, st, dr)
    - Quicksort(A, st, q - 1)
    - Quicksort(A, q + 1, dr)

- Partiție(A, st, dr)
  - x ← A[dr]
  - i ← st - 1
  - **Pentru** j **de la** st **la** dr – 1
    - **Dacă** A[j] ≤ x
      - i ← i + 1
      - **Interschimbă** A[i] ↔ A[j]
  - **Interschimbă** A[i + 1] ↔ A[dr]
  - **Întoarce** i + 1

Cazul defavorabil?

Complexitate ?

Exemplu de utilizare practică (II)

- Problema Quicksort – cazul defavorabil – datele de intrare sunt sortate în ordine inversă.
- Complexitate Quicksort: $O(n^2)$.
- Folosind algoritmi aleatori eliminăm acest caz.

Quicksort-aleator

- Quicksort-Randomizat(A, st, dr)
  - **Dacă** st < dr
    - q ← Partiție-Randomizată(A, st, dr)
    - Quicksort-Randomizat(A, st, q - 1)
    - Quicksort-Randomizat(A, q + 1, dr)
- Partiție-Randomizată(A, st, dr)
  - i ← Random(st, dr)
  - **Interschimbă** A[dr] ↔ A[i]
  - **Întoarce** Partiție(A, st, dr)