Proiectarea Algoritmilor

Curs 12 – Algoritmi aleatorii

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/

Obiective

- Definirea conceptului de algoritm aleator
- Algoritmi Las Vegas
- Algoritmi Monte Carlo
- Analiza algoritmilor aleatori

Algoritmi aleatorii

- Micșorăm timpul de rezolvare a problemei relaxând restricțiile impuse soluțiilor.
- Determinarea soluției optime:
  - Renunțăm la optimalitate (soluția suboptimală are o marjă de eroare garantată prin calcul probabilistic).
- Găsirea unei singure soluții:
  - Găsim o soluție ce se apropie cu o probabilitate măsurabilă de soluția exactă.

Algoritmi Las Vegas

- Găsesc soluția corectă a problemei, însă timpul de rezolvare nu poate fi determinat cu exactitate.
- Creșterea timpului de rezolvare → creșterea probabilității de terminare a algoritmului.
- Timp = ∞ → algoritmul se termina sigur (soluția e optimă).
- Probabilitatea de găsire a soluției creste extrem de repede astfel încât să se determine soluția corectă intr-un timp suficient de scurt.

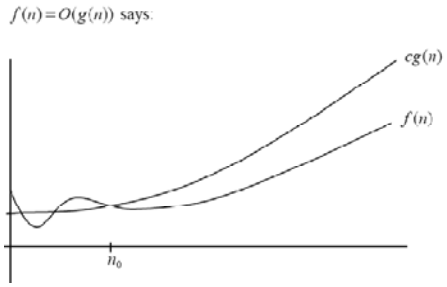
Algoritmi Monte Carlo

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

Reminder – complexitatea algoritmilor

$$O(g(n)) = \{ f(n) \mid \exists c > 0, \exists n_0 > 0, \forall n \geq n_0 : 0 \leq f(n) \leq cg(n) \}$$

- g(n) - limita asimptotică superioară pentru f(n)



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

Complexitatea algoritmilor Las Vegas

- Definiția 6.1: Algoritmii Las Vegas au complexitatea f(n) = O(g(n)) dacă $\exists$ c > 0 si n_0 > 0 a.i. $\forall$ n $\geq$ n_0 avem 0 < f(n) < c α g(n) cu o probabilitate de cel puțin 1 - $n^{-\alpha}$ pentru α > 0 fixat si suficient de mare.

Complexitate algoritmi Monte Carlo

- Definiția 6.1': Algoritmii Monte Carlo au complexitatea $f(n) = O(g(n))$ dacă $\exists\ c > 0$ si $n_0 > 0$ a.i.
  - $\forall n \geq n_0,\ 0 < f(n) \leq c\ \alpha\ g(n)$ 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 Las Vegas

- Problemă:
  - Capitolele unei cărți sunt stocate intr-un fișier text sub forma unei secvențe nevide de linii;
  - Fiecare secvență este precedată de o linie contor ce indică numărul de linii din secvență;
  - Fiecare linie din fișier este terminată prin CR, LF;
  - Toate liniile din secvență au aceeași lungime;
  - Fiecare secvență de linii conține o linie (titlul capitolului) ce se repetă si care apare in cel puțin 10% din numărul de linii al secvenței.
  - Secvențele sunt lungi.☺
- Cerință:
  - Pentru fiecare secvență de linii să se tipărească titlul capitolului (linia care se repetă).

Rezolvare "clasică"

Detectează_linii(fișier)

- ***Pentru fiecare*** Secv ∈ fișier
  - ***Pentru*** i ***de la*** 0 ***la*** dim(Secv)
    - ***Pentru*** j ***de la*** i + 1 ***la*** dim(Secv)
      - ***Dacă*** linie(i,Secv) = linie(j,Secv) ***atunci***
        - ***Întoarce*** (linie(i,Secv);

prelucrare secvență

Complexitate – $O(dim(Secv)^2)$

Algoritm Las Vegas pentru rezolvarea problemei

- Secțiunea "prelucrare secvență" se înlocuiește cu următoarea funcție:

- Selecție_linii(n,secv) // n = dim secv
  - ***Cât timp***(1) // mereu
    - i = random(0,n-1) // selectez o linie
    - j = random(0,n-1) // si încă una
    - ***Dacă*** i != j ***si*** linie(i,Secv) = linie(j,Secv) ***atunci*** // le compar
      - ***Întoarce*** linie(i,Secv) // am găsit linia

Analiza algoritmului Las Vegas (I)

- Notații:
  - n = numărul de linii din secvența curentă;
  - q = ponderea liniei repetate in secvență;
  - r = numărul de apariții al liniei repetate: r = n * q / 100;
  - m = numărul de pași necesari terminării algoritmului;
  - P_k = probabilitatea ca la pasul k să fie satisfăcută condiția de terminare a algoritmului;
  - P(m) = probabilitatea ca algoritmul să se termine după m pași.

Analiza algoritmului Las Vegas (II)

- Probabilitatea ca la pasul k linia i să fie una din liniile repetate este r / n.
- Probabilitatea ca la pasul k linia j să fie una din liniile repetate (diferită de i) este (r - 1) / n.
- Condiția de terminare: cele 2 evenimente trebuie să se producă simultan:

$P_k = r / n * (r-1) / n = q / 100 * (q / 100 – 1 / n)$

Analiza algoritmului Las Vegas (III)

- Probabilitatea ca algoritmul să NU se termine după m pași:
  - $\Pi_{k = 1 \rightarrow m}(1 - P_k) = \Pi_{k = 1 \rightarrow m} [1 – q / 100 * (q / 100 – 1 / n)] = [1 – q / 100 * (q / 100 – 1 / n)]^m$
- → $P(m) = 1 - [1 – q / 100 * (q / 100 – 1 / n)]^m$
- Pp: n > 100; q > 10%
- → $P(m) \geq 1 – [1 – q * (q – 1) / 10.000]^m$

Comparație timp de rulare

- q = 10%:
  - 3500 pași P = 1;
  - 1000 pași – P = 0,9988.
- q = 20%:
  - 1000 pași P = 1.
- q = 30%:
  - 500 pași P = 1.
- Varianta clasică: cazul cel mai defavorabil – 10000 pași.

Complexitate algoritm Las Vegas

- Algoritmii Las Vegas au complexitatea $f(n) = O(g(n))$ dacă $\exists$ $c > 0$ si $n_0 > 0$ a.i. $\forall$ $n \geq n_0$ avem $0 < f(n) < c \alpha g(n)$ cu o probabilitate de cel puțin $1 - n^{-\alpha}$ pentru $\alpha > 0$ fixat si suficient de mare.
- Arătăm că $f(n) = O(\lg(n))$:
  - Notăm: $a = 1 – q * (q - 1) / 10.000$;
  - $1 – P(c \alpha \lg(n))$ = probabilitatea ca algoritmul să nu se termine in $c \alpha \lg(n)$ pași;
  - $P(c \alpha \lg(n)) \geq 1 - a^{c \alpha \lg(n)} \rightarrow 1 – P(c \alpha \lg(n)) \leq a^{c \alpha \lg(n)} = n^{c \alpha \lg(a)} = n^{-c \alpha \lg(1/a)}$ pentru că $0 < a < 1$;
  - Dacă alegem $c \geq \lg^{-1}(1/a) \rightarrow 1 – P(c \alpha \lg(n)) \leq n^{-\alpha} \rightarrow P(c \alpha \lg(n)) \geq 1 - n^{-\alpha} \rightarrow$ algoritmul se termină in $\lg^{-1}(1/a) \lg(n)$ pași cu o probabilitate $\geq 1 - n^{-\alpha} \rightarrow$ (definiție) $f(n) = O(\lg(n))$.

Exemplu algoritm Monte Carlo

- Problemă: testarea dacă un număr n dat este prim.
- Rezolvare “clasică”:

Complexitate: O(sqrt(n))

- Prim-clasic(n)
  - ***Pentru*** i ***de la*** 2 ***la*** sqrt(n)
    - ***Dacă*** (n mod i == 0) ***întoarce*** fals;
  - ***Întoarce*** adevărat

Determinarea numerelor prime - complexitate

- Observație: pentru numere mari – operațiile nu mai durează O(1)!
- → Estimăm numărul de operații in 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 47th 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."
- http://www.mersenne.org

Algoritm aleator (I)

- Teorema 6.1 (mica teoremă a lui Fermat): Dacă n este prim → ∀ 0 < x < n, x^{n-1} mod 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 si prim in raport cu n satisface ecuația lui Fermat (ex: nr. Carmichael → 561).
- → Nu știm cu exactitate cate numere sunt!
  → Nu putem calcula probabilitatea!

Altă variantă de algoritm aleator

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

  $x_1 = 1$ SI $x_2 = n - 1$.

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

Algoritmul Miller-Rabin

- Prim2(n,α)
  - ***Daca*** (n == 2) ***Intoarce*** adevarat
  - ***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 ***si*** z ≠ 1 ***si*** z ≠ n-1) // verific teorema 6.2
      - ***Întoarce*** 1
  - ***Întoarce*** r != 1 // mica teoremă Fermat

Complexitate:

O(lg(n))

Calcularea numărului de pași

- Teorema 6.3: Pentru orice număr n, impar si compus există cel puțin (n-1) / 2 martori ai divizibilității lui n.
- Caz neinteresant: număr prim pentru ca 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 lgn; →după α lgn pași $P_{corect}(n) \geq 1 - n^{-\alpha}$;
- → Complexitate: $O(lg^2 n)$ →in funcție de numărul de biți k → Complexitate: $O(k^2)$

Exemplu de utilizare practică

- QUICKSORT(*A*, *p*, *r*)
  - **if** *p* < *r*
  - **then** *q* ← PARTITION(*A*, *p*, *r*)
  - QUICKSORT(*A*, *p*, *q* - 1)
  - QUICKSORT(*A*, *q* + 1, *r*)
- PARTITION(*A*, *p*, *r*)
  - *x* ← *A*[*r*]
  - *i* ← *p* - 1
  - **for** *j* ← *p* **to** *r* - 1
    - **do if** *A*[*j*] ≤ *x*
      - **then** *i* ← *i* + 1
        - exchange *A*[*i*] ↔ *A*[*j*]
    - exchange *A*[*i* + 1] ↔ A[r]
  - return i + 1

Cazul defavorabil?
Complexitatea?

Exemplu de utilizare practică (II)

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

Quicksort-aleator

- RANDOMIZED-QUICKSORT(*A*, *p*, *r*)
  - **if** *p* < *r*
  - **then** *q* ← RANDOMIZED-PARTITION(*A*, *p*, *r*)
  - RANDOMIZED-QUICKSORT(*A*, *p*, *q* - 1)
  - RANDOMIZED-QUICKSORT(*A*, *q* + 1, *r*)
- RANDOMIZED-PARTITION(*A*, *p*, *r*)
  - *i* ← RANDOM(*p*, *r*)
  - exchange *A*[*r*] ↔ *A*[*i*]
  - **return** PARTITION(*A*, *p*, *r*)

INTREBĂRI?