

Proiectarea Algoritmilor

Curs 13 - Algoritmi aleatorii



Bibliografie

- [1] C. Giumale – Introducere in Analiza Algoritmilor – cap. 6.1
- [2] Cormen – Introducere in algoritmi – cap. 8.3
- [3] <http://classes.soe.ucsc.edu/cms102/Spring04/TantaloAsymp.pdf>
- [4] <http://www.mersenne.org/>

Objective

- Definirea conceptului de algoritm aleatoriu
- Algoritmi Las Vegas
- Algoritmi Monte Carlo
- Analiza algoritmilor aleatorii

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 (găsirea soluției optime).
- $\text{Timp} = \infty \rightarrow$ algoritmul se termină sigur (soluția e optimă).
- Probabilitatea de găsim a soluției crește extrem de repede astfel încât să se determine soluția corectă într-un timp suficient de scurt.

Algoritmi Monte Carlo

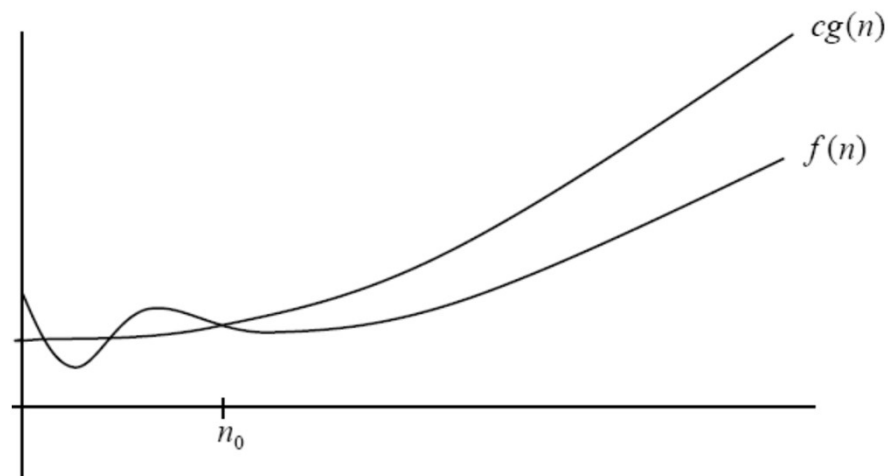
- Găsesc o soluție a problemei care nu e garantat corectă (soluție aproximativă).
- $T_{imp} = \infty \rightarrow$ soluția corectă a problemei.
- Probabilitatea ca soluția să fie corectă crește o dată cu timpul de rezolvare.
- Soluția găsită într-un timp acceptabil este aproape sigur corectă.

Reminder – complexitatea algoritmilor

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

$f(n) = O(g(n))$ says:

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



<http://classes.soe.ucsc.edu/cms102/Spring04/TantaloAsymp.pdf>

Complexitatea algoritmilor Las Vegas

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

Complexitate algoritmi Monte Carlo

- **Definiția 6.1'**: Algoritmii Monte Carlo au complexitatea $f(n) = O(g(n))$ dacă $\exists c > 0$ și $n_0 > 0$ astfel încât:
 - $\forall n \geq n_0, 0 < f(n) \leq c \cdot g(n)$ cu o probabilitate de cel puțin $1 - n^{-\alpha}$ pentru $\alpha > 0$ fixat și 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 într-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 și care apare în 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 \in fișier**
 - **Pentru i de la 0 la $\dim(\text{Secv})$**
 - **Pentru j de la $i + 1$ la $\dim(\text{Secv})$**
 - **Dacă $\text{linie}(i, \text{Secv}) = \text{linie}(j, \text{Secv})$ atunci**
 - **Întoarce ($\text{linie}(i, \text{Secv})$);**
- } prelucrare secvență

Complexitate – $O(\dim(\text{Secv})^2)$

Algoritm Las Vegas pentru rezolvarea problemei

- Secțiunea “prelucrare secvență” se înlocuiește cu următoarea funcție:
- **Selecție_linii(n,secv) // $n = \text{dim secv}$**
 - **Cât timp(1) // mereu**
 - $i = \text{random}(0, n-1)$ // selectez o linie
 - $j = \text{random}(0, n-1)$ // și încă una
 - **Dacă $i \neq j$ și $\text{linie}(i, \text{Secv}) = \text{linie}(j, \text{Secv})$ atunci**
// le compar
 - **Întoarce $\text{linie}(i, \text{Secv})$ // am găsit linia**

Analiza algoritmului Las Vegas (I)

● Notatii:

- n = numărul de linii din secvența curentă;
- q = ponderea liniei repetate în 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:

- $\prod_{k=1 \rightarrow m} (1 - P_k) = \prod_{k=1 \rightarrow m} [1 - q / 100 * (q / 100 - 1 / n)] = [1 - q / 100 * (q / 100 - 1 / n)]^m$

- $\rightarrow P(m) = 1 - [1 - q / 100 * (q / 100 - 1 / n)]^m$

- Pp: $n > 100$; $q > 10\%$

- $\rightarrow P(m) \geq 1 - [1 - q * (q - 1) / 10.000]^m$

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 $\exists c > 0$ și $n_0 > 0$ 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 și suficient de mare.
- Arătăm că $f(n) = O(\lg(n))$:
 - Notăm: $a = 1 - q * (q - 1) / 10.000$; $\rightarrow P(m) \geq 1 - a^m$
 - $P(c \alpha \lg(n))$ = probabilitatea ca algoritmul să se termine în $c \alpha \lg(n)$ pași;
 - $P(c \alpha \lg(n)) \geq 1 - a^{c \alpha \lg(n)} = 1 - n^{c \alpha \lg(a)} = 1 - n^{-c \alpha \lg(1/a)}$ pentru că $0 < a < 1$;
 - Dacă alegem $c \geq \lg^{-1}(1/a) \rightarrow P(c \alpha \lg(n)) \geq 1 - n^{-\alpha} \rightarrow$ algoritmul se termină în $\lg^{-1}(1/a) \alpha \lg(n)$ 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 $(n \bmod i == 0)$ întoarce fals;
 - Întoarce adevărat

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 $k = \text{nr. de biți ocupat de } n$.

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 Magнар 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]
- As of **October 2014**, 48 Mersenne primes are known. The largest known prime number $2^{57,885,161} - 1$ is a Mersenne prime.
- **January 7, 2016**: the discovery of the largest known prime number, $2^{74,207,281}-1$. The prime number has 22,338,618 digits -- almost 5 million digits longer than the previous record prime number



Algoritm aleatoriu (I)

- **Teorema 6.1 (mica teoremă a lui Fermat):** Dacă n este prim $\rightarrow \forall 0 < x < n, x^{n-1} \bmod n = 1$.
- **Prim1(n, α)** // detectează dacă n e număr prim
 - Dacă ($n \leq 1$ sau $n \bmod 2 = 0$) **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
 - $x = \text{random}(1, n-1)$ // aleg un număr oarecare
 - Dacă ($\text{pow_mod}(x, n) \neq 1$) **Întoarce** fals // testează teorema
// Fermat
 - **Întoarce** adevărat

Complexitate?



Algoritm aleatoriu (II)

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

Complexitate:
 $O(\lg(n))$

Algoritm aleatoriu (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 \rightarrow 561).
- \rightarrow Nu știm cu exactitate câte numere sunt!
- \rightarrow Nu putem calcula probabilitatea!

Altă variantă de algoritm aleatoriu

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

$$x_1 = 1 \quad \text{ȘI} \quad x_2 = n - 1.$$

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

Algoritmul Miller-Rabin

- **Prim2(n, α)**
 - **Dacă** ($n \leq 1$ **sau** $n \bmod 2 = 0$) **Întoarce** fals
 - $\text{limit} = \text{limita_calcul}(n, \alpha)$
 - **Pentru** i **de la** 0 **la** limit
 - $x = \text{random}(1, n-1)$
 - **Dacă** ($\text{martor_div}(x, n)$) **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 $(m \bmod 2 \neq 0)$ // putere impară
 - `r = y * r mod n`
 - `z = y` // salvez valoarea lui x
 - `y = y * y mod n` // calculez $y^2 \bmod n$
 - Dacă $(y = 1 \text{ și } z \neq 1 \text{ și } z \neq n-1)$ // verific teorema 6.2
 - Întoarce 1
 - `m = m div 2` // înjumătățesc puterea
 - Întoarce `r != 1` // mica teoremă Fermat ($x^{n-1} \bmod n \neq 1$)

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 ($P_{\text{corect}}(n) = 1$)!
- **Caz interesant:** număr compus (impar) ($P_{\text{corect}}(n) = ?$):
- 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;
- $P(x) \geq (n-1) / 2 * 1 / (n-1) = 0.5$;
- $P_{\text{incorect}}(n) = \prod_{1 \rightarrow \text{limit}} (1 - P(x)) \leq 1/2^{\text{limit}}$;
- $\rightarrow P_{\text{corect}}(n) \geq 1 - 2^{-\text{limit}} = 1 - n^{-\alpha} \rightarrow \text{limit} = \alpha \lg(n)$; \rightarrow după $\alpha \lg(n)$ pași $P_{\text{corect}}(n) \geq 1 - n^{-\alpha}$;
- \rightarrow **Complexitate: $O(\lg^2(n))$** \rightarrow în funcție de **numărul de biți k** \rightarrow **Complexitate: $O(k^2)$**



Exemplu de utilizare practică

- Quicksort(A, st, dr)
 - Dacă $st < dr$
 - $q \leftarrow \text{Partiție}(A, st, dr)$
 - Quicksort(A, st, q - 1)
 - Quicksort(A, q + 1, dr)

Cazul
defavorabil?

- Partiție(A, st, dr)
 - $x \leftarrow A[dr]$
 - $i \leftarrow st - 1$
 - Pentru j de la st la dr - 1
 - Dacă $A[j] \leq x$
 - $i \leftarrow i + 1$
 - Interschimbă $A[i] \leftrightarrow A[j]$
 - Interschimbă $A[i + 1] \leftrightarrow A[dr]$
 - Întoarce $i + 1$

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 aleatorii eliminăm acest caz.

Quicksort-aleatoriu

- Quicksort-Randomizat(A , st , dr)
 - **Dacă** $st < dr$
 - $q \leftarrow \text{Partiție-Randomizată}(A, st, dr)$
 - Quicksort-Randomizat(A , st , $q - 1$)
 - Quicksort-Randomizat(A , $q + 1$, dr)
- Partiție-Randomizată(A , st , dr)
 - $i \leftarrow \text{Random}(st, dr)$
 - **Interschimbă** $A[dr] \leftrightarrow A[i]$
 - **Întoarce** Partiție(A , st , dr)



INTREBĂRI?