Algoritmi aleatorii

- Introducere
- Fundamente
- Algoritmi Monte Carlo
- Algoritmi Las Vegas

Introducere

- Un algoritm aleatoriu poate fi privit ca un algoritm nedeterminist care are o distributie de probabilitate pentru fiecare alegere nedeterminista.
- O alta definitie: un algoritm aleatoriu este un algoritm determinista care are la intrare o secventa de biti alesi aleatoriu.
  - ⇨ Un algoritm aleatoriu poate fi vazut ca o multime de algoritmi deterministi, din care, pentru o intrare data, se alege unul intr-un mod aleatoriu
  - ⇨ Pentru o intrare data x, executiile algoritmului aleatoriu pot diferi in functie de seventa de biti alesi aleatoriu
  - ⇨ Aceste diferente se proiecteaza atat in complexitate cat si iesiri:
    - Timpul de executie sau iesirea pot fi consiedate ca variabile aleatorii

Introducere

- Un algoritm aleatoriu pentru care iesirile sunt considerate variabile aleatorii se numesc **algoritmi Monte Carlo**
  - ⇨ Iesirile nu sunt in mod necesar exacte sau corecte
  - ⇨ Probabilitatea de eroare poate fi coborata semnificativ prin executii repetate independente
- Un algoritm aleatoriu care intotdeauna (independent de alegerile aleatorii) calculeaza solutia corecta si numai complexitatea este considerata variabila aleatorie se numeste **algoritm Las Vegas**

Avantaje

- Algorimii aleatorii pot fi mai eficienti si mai usor de implementat
- Pot fi foarte utili pentru acele probleme pentru care nu stim alta abordare tractabila din punct de vedere practic
- O astfel de problema este testul de primalitate pentru care

  (i) Avem un algoritm aleatoriu cu complexitate timp polinomiala,

  (ii) Nu se cunsoc algoritmi deterministi polinomiali care sa rezolve problema, si

  (iii) Nu se stie daca problema este in P sau (sau chiar daca este NP-completa sau nu)
- Desi sunt putine probleme cunoscute pentru care algoritmii aleatorii s-au aplicat cu succes, deoarece aceste probleme sunt foarte importante, abordarile aleatorii au devenit un un instrument standard in multe domenii din informatica

Dezavantaje

- Asa cum am mai spus, cazurile in care s-au aplicat cu succes sunt putine
- De fapt cazul tipic este acela cand nu se cunoaste daca “randomizarea” ajuta sau nu
- De exemplu, nu se cunoaste niciun algoritm aleatoriu cu timp de executie polinomial pentru vreo problema NP-completa
- Intrebarea “este posibil sa convertim un algoritm aleatoriu intr-unul determinist fara sa platim un pret prea mare privind resursele?” este inca deschisa

Fundamente

- Consideram doar un caz particular de algoritmi aleatorii
  - ⇨ Algoritmul intreaba doar din cand in cand care e valoarea unor biti aleatorii pentru a-si putea continua calculul in functie de aceste valori
- Din punctul de vedere al teoriei probabilitatilor, un algoritm aleatoriu impreuna cu o intrare x determina un experiment probabilistic (***algoritmi probabilisti(ci)?***)
- Acest experiment este descris de spatiul

  $(S_{A,x}, Prob)$, unde

  $S_{A,x}$ = {C | C este o executie aleatorie a lui A pentru x}, si

  Prob o distributie de probabilitate peste $S_{A,x}$.
- Pentru fiecare algoritm aleatoriu A consideram o noua masura – numarul de biti aleatori utilizati

Fundamente

- Notam cu $Random_A(x)$ numarul maxim de biti aleatori utilizati de executiile aleatoare ale lui A pentru intrarea x
- Atunci, pentru $n \in N$,

$$Random_A(n) = \max\{Random_A(x) \mid x \text{ intrare de lungime } n\}$$

- Aceasta complexitate este importanta din urmatoarele motive:

(1) costa foarte mult sa simulezi intr-un mod rezonabil secvente aleatorii si acest cost creste odata cu lungimea secventei

(2) daca $Random_A(x)$ este marginita de o functie logaritmica, atunci numarul de executii aleatoare distincte pentru intrari de lungime n este marginit de $2^{Random_A(n)} \leq p(n)$, p un polinom.

⇨ Aceasta inseamna ca A poate fi simulat de un algoritm determinist si aceasta simulare se numeste "**derandomizare**".

Fundamente

- Pentru fieacare calcul aleatoriu C pentru intrarea x se poate determina probabilitatea de executie a acestui calcul:

  Prob $_{A,x}$ ***(C)*** = produsul probabilitatilor alegerilor aleatorii facute de-a lungul calculului C

- Probabilitatea ca A sa dea iesirea y pentru intrarea x este

  Prob(A(x)=y) := $\sum$ ($\text{Prob}_{A,x}$ (C) | C calculeaza y)

- Notam cu ***Time*** $_A$***(C)*** timpul necesar lui A sa execute calculul C

- ***Timpul mediu de executie*** este

  Exp-Time $_A$***(x)*** = M[Time] = $\sum_C \text{Prob}_{A,x}$ (C) · Time_A(C)

  Exp-Time $_A$***(n)*** = max{Exp-Time_A(x) | x intrare de lung. n}

- Ca deobicei, Exp-Time_A(n) este greu de calculat in practica, asa ca se prefera urmatoarea varianta mai simpla (cazul cel mai nefav.):

  Time $_A$***(x)*** = max{Time_A(C) | C calcul pentru intrarea x}

  Time $_A$***(n)*** = max{Time_A(x) | x intrare de lung. n}

Fundamente

- Calcule infinite sunt permise deoarece un calcul infinit nu reprezinta neaparat un ciclu infinit
  - ⇨ O noua decizie aleatorie poate conduce totdeauna la un calcul cu succes
- O posibilitate de a simplifica lucrurile: se determina complexitatea $Time_A$ din afara programului si va forta terminarea executiei daca depaseste acest timp si se va da iesirea “?”
- daca $Time_A(n)$ este rezonabil, atunci se poate restarta executia algoritmului in astfel de situatii

Algoritmi Las Vegas

- Prima definitie:

  Spunem ca algoritmul aleatoriu A, care rezolva problema P, este un ***algoritm aleatoriu Las-Vegas*** daca pentru orice intrare x,

  $$Prob(A(x) = P(x)) = 1$$

- A doua definitie:

Spunem ca algoritmul aleatoriu A, care rezolva problema P, este un ***algoritm aleatoriu Las-Vegas*** daca pentru orice intrare x,

$$Prob(A(x) = P(x)) \geq 1/2 \text{ si}$$

$$Prob(A(x) = \text{“?”}) = 1 - Prob(A(x) = P(x)) \leq 1/2$$

Exemplu de algoritm Las Vegas

- Quicksort aleatoriu

QSA(A)

intrare: $A = \{s[i] \mid i = 1, \ldots, n\}$

alege $i \in \{1, \ldots, n\}$ uniform aleatoriu

if $n = 1$ **then return** A

else

$A_< = \{ s[k] \mid s[k] < s[i] \}$

$A_= = \{ s[k] \mid s[k] = s[i] \}$

$A_> = \{ s[k] \mid s[k] > s[i] \}$

return $QSA(A_<)\ A_=\ QSA(A_>)$

Algoritmi Monte Carlo “one-sided-error”

- Consideram doar probleme de decizii
- Scriem $x \in L$ daca x are proprietatea ceruta de problema
  si $x \notin L$ daca x NU are proprietatea ceruta de problema
- Astfel problema de decizie devine o problema de apartenenta la un limbaj
- Un algoritm A se numeste ***algoritm Monte Carlo “one-sided-error”*** daca satisface conditiile:
  - pentru orice $x \in L$, $Prob(A(x) = \text{“da”}) \geq 1/2$
  - pentru orice $x \notin L$, $Prob(A(x) = \text{“nu”}) = 1$
- asadar, algoritmul nu intoarce niciodata raspuns “da” daca intrarea nu are proprietatea ceruta de problema

Pradigma abundenta de martori (abundance of witnesses)

- Un martor pentru x (de fapt pentru $x \in L$) este un sir y pentru care exista un algritm polinomial (eficient) care avand la intrare perechea (x,y), dovedeste ca $x \in L$
- De exemplu, daca x este un numar intreg si L este multimea numerelor compuse, atunci orice factor propriu y al lui x este un martor pentru apartenenta lui x la L
- Pentru multe probleme, dificultatea de a gasi martori vine din faptul spatiul de cautare a martorilor este foarte mare
- Dar daca sunt multi martori in acel spatiu, atunci putem cauta la intamplare (aleatoriu)

Pradigma abundenta de martori

- Notatii:
  - ⇨ CanW(x) = multimea de candidati pentru calitatea de martor pentru x
  - ⇨ Witness (x) = multimea martorilor pentru x
- Abordarea este eficienta daca multimea Witness(x) $\subseteq$ CanW(x) are cardinalitatea proportionala cu cea a lui CanW(x)

Pradigma abundenta de martori: exemplu

- Consideram $x, y \in \{0,1\}^n$
- Un numar prim p este martor pentru $x \neq y$ daca

  Number(x) mod p ≠ Number(y) mod p
- CanW = multimea numerelor prime din $\{1, 2, \ldots, n^2\}$
- Stim ca avem $|CanW(x)| \approx n^2/ (\ln n^2)$
- Cel mult n-1 elemente din CanW(x) nu sunt martori pentru $x \neq y$
- Urmatorul algoritm

  Intrare: $x, y \in \{0,1\}^n$

  Alege uniform $p \in \{1, 2, \ldots, n^2\}$

  s = Number(x) mod p

  q = Number(y) mod p

  if q ≠ s **then return** “da” **else return** “nu”

Raspunde corect (“da” $\Rightarrow x \neq y$) cu probabilitatea $\frac{|CanW(x)|-n+1}{|CanW|}$

Testul de primalitate

- Pentru reprezentarea lui p sunt necesari $\lceil \log_2 p \rceil$ biti
- Algoritmul trivial care verifica, pentru orice $a \in \{1,..., p - 1\}$, daca a este factor a lui n are complexitatea exponentiala in $n = \log_2 p$
- Daca n = 100, atunci acest algoritm nu poate fi executat
- nici cautarea in spatiul restrans {2, 3, . . . , [sqrt(p)]} nu schimba substantial situatia
- Algoritmul aleatoriu utilizeaza abundenta de martori
- Daca $p = a \cdot b$, atunci a si b sunt martori pentru compozitionalitatea lui p
- Daca a si b sun primi, atunci avem exact doi martori …
- Asadar ne trebuie o alta definitie pentru primalitate …

Testul de primalitate

Teorema lui Fermat

pentru orice numar p si orice orice a $\in$ {l,..., p - 1},

$a^{p-1} = 1 \pmod p$

- De unde obtinem:

  p este prim daca si numai daca $a^{(p-1)/2} \bmod p \in \{-1, 1\}$ pentru orice a $\in$ {l,..., p - 1}

- Rezulta ca b $\in$ {l,..., p - 1} este martor pentru compozitionalitatea lui p daca $b^{(p-1)/2} \bmod p \notin \{-1, 1\}$

- Cati martori avem acum?

Teorema

Fie p un numar impar astfel incat (n-1)/2 este impar (i.e., p = 3 (mod 4)). Daca p este compus atunci cel putin jumatate dintre elementele {l,..., p - 1} sunt martore pentru compozitionalitatea lui p

SSSA "Simplyfied Solovay Strassen Algorithm"

intrare: un numar impar p astfel incat (p-1)/2 este impar

alege aleatoriu uniform un $a \in \{1,..., p - 1\}$

calculeaza $A = a^{(p-1)/2} \mod p$

if $A \in \{-1, 1\}$

then return ("prim") // insucces (nu)

else return ("compus") //succes (da)

Teorema

SSSA este un algoritm aleatoriu Monte Carlo "one-sided-error" cu timp polinomial pentru recunoasterea numerelor compuse p pentru care p si (p-1)/2 sunt prime.

Paradigma probelor aleatorii

- O proba aleatorie dintr-o populatie este adesea reprezentativa pentru intreaga populatie
- Aceasta afirmatie se bazeaza pe:

Fapt 1. Orice variabila aleatoare presupune cel putin o valoare care nu e mai mica decat media si cel putin o valoare care nu e mai mare decat media.

Claim 2. Daca un obiect ales aleatoriu dintr-un univers satisface proprietatea cu o probabilitate pozitiva, atunci exista un obiect in univers care satisface acea proprietate.

- Desi par simple si evidente, cele doua fapte au o putere surprinzatoare, conducand de multe ori la cele mai bune abordari pentru unele probleme dificile

Resturi patratice

- Un numar a este **rest patratic modulo p** daca exista $x \in Z_p$ astfel incat $a = x^2 = x \cdot x \pmod p$.
- Problema pe care o consideram este: dat un numar prim p, sa se gaseasca $a \in Z_p$ care NU este rest patratic mod p.
- Nu exista algoritm deterministic polinomial pentru aceasta problema
- In schimb, problema complementara este foarte simpla: se ia un $a \in \{1, 2,...,p-1\}$, se calculeaza $b = a^2 \bmod p$, si intoarce b
- Vom prezenta un algoritm Las Vegas, cu timp polinomial, care se bazeaza pe paradigma probelor aleatorii.
- Avem urmatoarele doua fapte:

(A) Dat un numar prim p si un $a \in Z_p$, este posibil sa decidem daca a este rest patratic mod p in timp polinomial.

(B) Pentru orice numar prim p, exact jumatate din elementele lui Z_p sunt resturi patratice.

a^b mod p poate fi calculat eficient

- Considerm b = Number($b_k b_{k-1} \ldots b_1 b_0$)
- A calcula a^b este echivalent cu a calcula

$$a^{b_0 \cdot 2^0} a^{b_1 \cdot 2^1} a^{b_2 \cdot 2^2} \ldots a^{b_k \cdot 2^k}$$

Daca $b_i = 0$ atunci $a^{b_i \cdot 2^i} = 1$

si daca $b_i = 1$ atunci $a^{b_i \cdot 2^i} = a^{2^i}$

si algoritmul este

intrare: a, b, p cu b = Number($b_k b_{k-1} \ldots b_1 b_0$)

C $\leftarrow$ a ; D $\leftarrow$ 1 ;

for i $\leftarrow$ 1 **to** k **do**

if $b_i = 1$ **then** D $\leftarrow$ D $\cdot$ C mod p

C $\leftarrow$ C $\cdot$ C mod p

return D

Criteriul lui Euler

Teorema

Fie $a \in Z_p$.

1) Daca a este rest patratic mod p, atunci $a^{(p-1)/2} = 1 \pmod p$.
2) Daca a NU este rest patratic mod p, atunci $a^{(p-1)/2} = -1 \pmod p$.

- Utilizand criteriul Euler si algoritmul de calcul al lui a^b , se poate decide efiecient daca a este rest patratic mod p.

Teorema

Daca p este un numar prim impar, atunci exact jumatate dintre elementele lui Z_p sunt resturi patratice mod p.

Algoritm Las Vegas pentru resturi patratice

intrare: un numar prim p

alege aleator uniform $a \in \{1,..., p - 1\}$

$X \leftarrow a^{(p-1)/2}$ (cu alg. eficient descris mai devreme)

if $X = p - 1$ **then return** a

else return “Incercare nereusita”