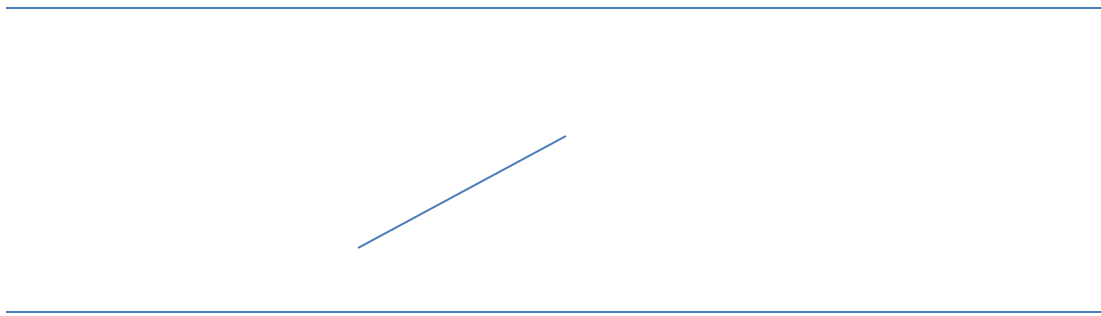


# PA - Curs 12 – Algoritmi aleatorii

# Calculul lui $\pi$

# Problema lui Buffon



# Problema lui Buffon

- $P = 2l/d\pi$
- $P = E(m)/n$
- $\pi = n2l/md$

- d



# Algoritmi Monte Carlo

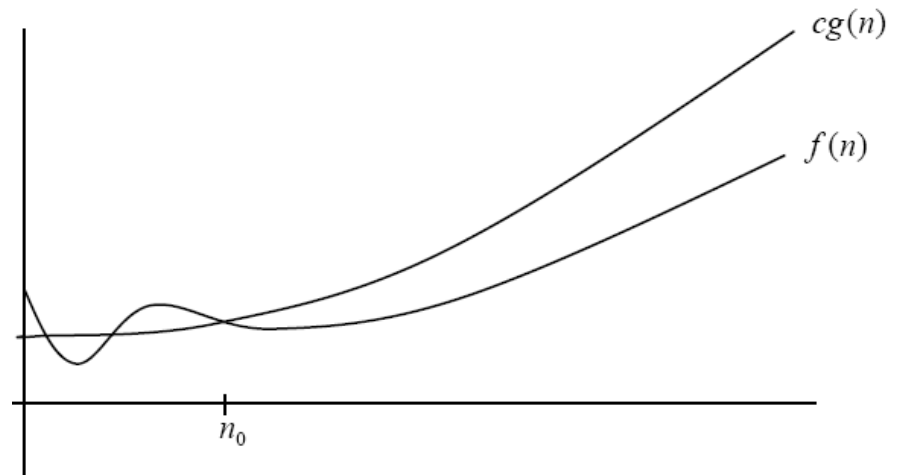
- Gasesc o solutie garantat corecta doar dupa un timp infinit de rezolvare
- Probabilitatea ca solutia sa fie corecta creste o data cu timpul de rezolvare
- Solutia gasita intr-un timp acceptabil este aproape corecta

# 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  $f(n)$

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



# Complexitatea algoritmilor Monte Carlo

- algoritm Monte Carlo are complexitatea  **$f(n) = \tilde{O}(g(n))$**

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  $1 - n^{-\alpha}$  pentru  $\alpha > 0$  fixat și suficient de mare
- Probabilitatea ca soluția determinată de algoritm să fie corectă este cel puțin  $1 - n^{-\alpha}$

# Algoritmi Las Vegas

- Gasesc solutia corecta a problemei, insa timpul de rezolvare nu poate fi determinat cu exactitate.
- Cresterea timpului de rezolvare => cresterea probabilitatii de terminare a algoritmului
- $T_{imp} = \infty \Rightarrow$  algoritmul se termina sigur
- Probabilitatea de gasire a solutiei creste suficient de repede incat sa se determine solutia intr-un timp suficient de scurt



# Complexitatea algoritmilor Las Vegas

- Un algoritm Las Vegas are complexitatea  **$f(n) = \tilde{O}(g(n))$**   
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

# Exemplu algoritm Las Vegas

- Problema:
  - Capitolele unei carti sunt stocate intr-un fisier text sub forma unei secvente nevide de linii
  - Fiecare secventa este precedata de o linie contor ce indica numarul de linii din secventa
  - Fiecare linie din fisier este terminata prin CR,LF
  - Toate liniile din secventa au aceeasi lungime
  - Fiecare secventa de linii contine o linie (titlul capitolului) ce se repeta si care apare in cel putin 10% din numarul de linii al secventei.
  - secventele sunt lungi
- Cerinta:
  - Pentru fiecare secventa de linii sa se tipareasca linia care se repeta

input	output
<pre>8 Adventures of Sherlock Holmes It is with a heavy heart that I take up my pen to write these last few words in which I shall ever record the singular gifts by which my friend Mr. Sherlock Holmes was distinguished. Adventures of Sherlock Holmes 5 Programming practice Real programmers use code generators! Programming practice Programming practice</pre>	<pre>Adventures of Sherlock Holmes Programming practice</pre>

# Rezolvare “clasica”

detecteaza\_linii(fisier)

pentru fiecare  $Secv \in \text{fisier}$

    Repeta  $i \leftarrow 0$  cat timp  $i < \text{dim}(Secv)$

        Repeta  $j \leftarrow i+1$  cat timp  $j < \text{dim}(Secv)$

            Daca  $(\text{linie}(i, Secv) = \text{linie}(j, Secv))$

                atunci  $\text{print}(\text{linie}(i, Secv); i=j=\text{dim}(Secv))$

} prelucrare  
secventa

complexitate –  $O(n^2)$

# Algoritm Las Vegas pentru rezolvarea problemei

- sectiunea “prelucrare secventa” se inlocuieste cu urmatoarea functie:
- `selectie_linii(n,Secv) //n=dim secv`

Repeta

`i ← random(0,n-1)`

`J ← random(0,n-1)`

Daca (`! # j` si `linie(i,Secv)=linie(j,Secv)`)

atunci return `linie(i,Secv)`

# Analiza algoritmului Las Vegas (I)

- Notatii:
  - $n$ =numarul de linii din secventa curenta
  - $q$ =procentul de linii repetate in secventa
  - $r$ =numarul de aparitii al liniei repetate:  $r=n*q/100$
  - $m$ =numarul de pasi necesari terminarii algoritmului
  - $P_k$ =probabilitatea ca la pasul  $k$  sa fie satisfacuta conditia de terminare a algoritmului
  - $P(m)$ = probabilitatea ca algoritmul sa se termine dupa  $m$  pasi

# Analiza algoritmului Las Vegas (II)

- probabilitatea ca la pasul  $k$  linia  $i$  sa fie una din liniile repetate este  $r/n$
- probabilitatea ca la pasul  $k$  linia  $j$  sa fie una din liniile repetate este  $(r-1)/n$
- conditia de terminare= cele 2 evenimente trebuie sa se produca simultan=>  
$$P_k = r/n * (r-1)/n = q/100 * (q/100 - 1/n)$$

# Analiza algoritmului Las Vegas (III)

- Probabilitatea ca algoritmul sa nu se termine dupa m pasi=

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

- $\Rightarrow P(m) = 1 - (1 - q/100 * (q/100 - 1/n))^m$
- $n > 100$ ;  $q > 10\%$
- $\Rightarrow P(m) \geq 1 - (1 - q(q-1)/10000)^m$



$\frac{q}{m}$	10%	20%	30%
100	0.595083523989	0.979226618172	0.999888550595
300	0.933610966591	0.999991035592	0.999999999999
500	0.989115029844	0.9999999996132	1
1000	0.999881517425	1	1
1500	0.999998710321	1	1
2000	0.999999985962	1	1
2500	0.999999999847	1	1
3000	0.999999999998	1	1
3500	1	1	1

# Comparatie timp de rulare

- $q=10\%$  - 3500 pasi  $P=1$ ; 1000 pasi –  $P=0,9988$
- $q=20\%$  - 1000pasi  $P=1$
- $q=30\%$  500 pasi  $P=1$
- varianta clasica: cazul cel mai defavorabil – 10000 pasi

# Complexitate algoritm Las Vegas

- algoritm Las Vegas are complexitatea  $f(n) = \tilde{O}(g(n))$  daca  $\exists c > 0$  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
- aratam ca  $f(n) = \tilde{O}(\lg(n))$ 
  - $a = (1 - q(q-1))/10000$
  - calculam  $P(c\alpha \lg(n)) \geq 1 - n^{-\alpha}$  ( $c \geq \lg^{-1}(1/a)$ )

# Exemplu algoritm Monte Carlo

- Problema – testarea daca un numar  $n$  dat este prim
- rezolvare “clasica”
- prim-clasic( $n$ )

Pentru  $i \leftarrow 2$  la  $\sqrt{n}$  repeta

Daca  $(n \bmod i = 0)$  atunci return false

return true

# Determinarea numerelor prime - complexitate

- complexitate  $O(\sqrt{n})$
- pentru numere mari – operatiile nu mai dureaza  $O(1)$
- estimam numarul de operatii in functie de numarul de biti pe care este exprimat numarul
- =>prim\_clasic –  $O(2^{k/2})$  unde k este numarul de biti ocupat de n

# Complexitate nesatisfacatoare!

- “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,  $2^{232,582,657}-1$ . The new prime at [9,808,358 digits](#) is 650,000 digits larger than their previous record prime found last December.”
- <http://www.mersenne.org>

# Algoritm aleator (I)

- Teorema mică a lui Fermat:  
n este prim  $\Rightarrow \forall 0 < x < n, (x^{n-1} \bmod n) = 1$
- `prim1(n,α)`  
Daca  $(n \leq 1 \text{ sau } (n \bmod 2) = 0)$  atunci 'numar compus'  
`limit ← limita_calcul(n,α)`  
Pentru  $i \leftarrow 0$  la limit repeta  
     $x \leftarrow \text{random}(1, n-1)$   
    Daca  $(\text{pow\_mod}(x, n) \neq 1)$  atunci return false  
return true
- `pow_mod(x,n)`  
 $r \leftarrow 1$   
Pentru  $m \leftarrow n-1$  cat timp  $m > 0$ , pas:  $m \leftarrow m/2$   
    Daca  $(m \bmod 2 \neq 0)$  atunci  $r \leftarrow x * r \bmod n$   
     $x \leftarrow (x * x) \bmod n$   
return r

# Algorithm aleator (II)

- problema – nu putem stabili cu exactitate care este limita de calcul
  - nu se poate estima pentru un numar compus  $n$  numarul de numere  $x$ ,  $2 < x < n$  pentru care nu se verifica ecuati
  - exista numere compuse (Carmichael) pentru care orice numar  $x < n$  si prim in raport cu  $n$  satisface ecuati
- => nu stim cu exactitate cate numere sunt=> nu putem calcula probabilitatea



# Alta varianta de algoritm aleator

- **Teorema** Pentru orice numar prim ecuatia  $x^2 \bmod n = 1$  are exact 2 solutii:

$$x_1 = 1$$

$$x_2 = n - 1$$

- **Definitie** Fie  $n > 1$  si  $0 < x < n$  2 numere a.i.

$$x^{n-1} \bmod n \neq 1 \text{ sau}$$

$$x^2 \bmod n \neq 1, x \neq 1 \text{ si } x \neq n-1$$

$x$  este numit *martor al divizibilitatii* lui  $n$

# Algoritmul Miller-Rabin

- $\text{prim2}(n, \alpha)$

Daca ( $n \leq 1$  sau  $n \bmod 2 = 0$ ) atunci return false

$\text{limit} \leftarrow \text{limita\_calcul}(n, \alpha)$

Pentru  $i \leftarrow 0$  la  $\text{limit}$  repeta

$X \leftarrow \text{random}(1, n-1)$

Daca  $\text{martor\_div}(x, n)$  atunci return false

return true

# Algoritmul Miller-Rabin (II)

- `martor_div(x,n)`

`r ← 1; y ← x;`

Pentru `m ← n-1` cat timp `m > 0`, pas `m ← m/2` repeta

    Daca `(m mod 2 ≠ 0)` atunci `r ← y*r mod n`

`y ← y*y mod n`

`z ← y`

    Daca `(y=1 si z≠1 si z≠n-1)` atunci return 1

return `r ≠ 1` //mica teorema

# calcularea numarului de pasi

## Teorema

Pentru orice numar  $n$ , impar si compus exista cel putin  $(n-1)/2$  martori ai divizibilitatii lui  $n$

$x$ =element generat la un pas al algoritmului

$P(x)$ =probabilitatea ca  $x$  sa fie martor al divizibilitatii.

$$P(x) \geq (n-1)/2 * 1/(n-1) = 0.5$$

$$P_{\text{incorect}}(n) = \prod_{1 \rightarrow \text{limit}} (1 - P(x)) \leq 1/2^{\text{limit}}$$

$$\Rightarrow P_{\text{corect}}(n) = 1 - 2^{-\text{limit}} = 1 - n^{-\alpha}$$

$$\Rightarrow \text{limit} = \alpha \log n$$

$$\Rightarrow \text{Complexitatea} = \tilde{O}(\lg^2 n)$$

$$\text{in functie de numarul de biti } k \Rightarrow \text{complexitatea} = \tilde{O}(k^2)$$

# Exemplu de utilizare practica

- QUICKSORT( $A, p, r$ )
  - **if**  $p < r$
  - **then**  $q \leftarrow \text{PARTITION}(A, p, r)$
  - QUICKSORT( $A, p, q - 1$ )
  - QUICKSORT( $A, q + 1, r$ )
- PARTITION( $A, p, r$ )
  - $x \leftarrow A[r]$
  - $i \leftarrow p - 1$
  - **for**  $j \leftarrow p$  **to**  $r - 1$ 
    - **do if**  $A[j] \leq x$ 
      - **then**  $i \leftarrow i + 1$ 
        - » exchange  $A[i] \leftrightarrow A[j]$
      - exchange  $A[i + 1] \leftrightarrow A[r]$
  - return  $i + 1$

# Exemplu de utilizare practica (II)

- problema quicksort – cazul defavorabil – datele de intrare sunt sortate in ordine inversa
- Complexitate quicksort:  $O(n^2)$
- folosind algoritmi aleatori eliminam acest caz

# Quicksort-aleator

- RANDOMIZED-QUICKSORT( $A, p, r$ )
  - **if**  $p < r$
  - **then**  $q \leftarrow$  RANDOMIZED-PARTITION( $A, p, r$ )
  - RANDOMIZED-QUICKSORT( $A, p, q - 1$ )
  - RANDOMIZED-QUICKSORT( $A, q + 1, r$ )
- RANDOMIZED-PARTITION( $A, p, r$ )
  - $i \leftarrow$  RANDOM( $p, r$ )
  - exchange  $A[r] \leftrightarrow A[i]$
  - **return** PARTITION( $A, p, r$ )

# Bibliografie

1. curs preluat din Introducere in Analiza Algoritmilor – C. Giumale, Ed. Polirom capitolul 6.1 – p 291-301
2. slide-urile 25-27 v. Cormen, Leiserson, Rivest – Introducere in Algoritmi, Cap. 7