

# Proiectarea algoritmilor: Algoritmi nedeterminiști și probabiliști

Ștefan Ciobâcă, Dorel Lucanu

Faculty of Computer Science  
Alexandru Ioan Cuza University, Iași, Romania  
dlucanu@info.uaic.ro

PA 2016/2017

- 1 Algoritmi nedeterminiști în general
- 2 Algoritmi nedeterminiști pentru probleme de decizie
- 3 Algoritmi probabiliști
  - Variabilă aleatorie
  - Exemplu de algoritm Monte Carlo: Compoziționalitatea unui număr întreg
  - Algoritmi Las Vegas
  - Exemplu de algoritm Las Vegas:  $k$ -mediana
- 4 Material suplimentar

# Plan

- 1 Algoritmi nedeterminiști în general
- 2 Algoritmi nedeterminiști pentru probleme de decizie
- 3 Algoritmi probabiliști
  - Variabilă aleatorie
  - Exemplu de algoritm Monte Carlo: Compoziționalitatea unui număr întreg
  - Algoritmi Las Vegas
  - Exemplu de algoritm Las Vegas:  $k$ -mediana
- 4 Material suplimentar

# Motivație

- abstractizarea spațiului de stări este utilă în analiză
- algoritmi nedeterminiști vin cu un nivel de abstractizare în plus, care combină abstractizarea stărilor cu cea a programării procedurale
- ignoră detalii despre cum sunt create anumite structuri de date
- sunt utili în analiza complexității problemelor
- noțiune preliminară pentru cea de algoritm probabilist

# Algoritmi nedeterminiști intuitiv

- pentru anumite stări există mai multe posibilități de a continua calculul (presupunem că acestea sunt în număr finit)
- în consecință, pentru aceeași intrare, algoritmul poate da rezultate (configurații finale) diferite
- interpretarea angelică: algoritmul ghicește calea de execuție care duce la rezultatul corect
- timpul de execuție = timpul calculului care duce la rezultatul corect

# Extensia limbajului

`choose  $x$  in  $S$ ;` – întoarce un element din  $S$  ales arbitrar

Timp de execuție (uniform):  $O(1)$

`choose  $x$  in  $S$  s.t.  $B$ ;` – întoarce un element din  $S$  care satisface condiția  $B$  ales arbitrar

Timp de execuție (uniform):  $T(B)$

# Demo cu noile instructiuni

```
choose x1 in { 1 .. 5 };
```

```
$ alki choose.alk
```

```
State:
```

```
    x1 |-> 2
```

```
$ alki choose.alk
```

```
State:
```

```
    x1 |-> 4
```

# Demo cu noile instructiuni

```
odd(x) {  
    return x % 2 == 1;  
}  
choose x1 in { 1 .. 5 } s.t. odd(x1);  
choose x2 in { 1 .. 5 } s.t. odd(x2);
```

```
$ alki choosest.alk
```

State:

```
    x1 |-> 3
```

```
    x2 |-> 1
```

```
$ alki choosest.alk
```

State:

```
    x1 |-> 5
```

```
    x2 |-> 5
```



# Demo cu noile instructiuni

```

odd(x) {
  return x % 2 == 1;
}

s = emptySet;
for (i = 0; i < 8; i = i+2)
  s.pushBack(i);
choose x in s s.t. odd(x);

$ alki tests/failure.alk
Program status: failure ;
State:

i |-> 8
s |-> { 0, 2, 4, 6 }
x |-> 6

```

# Problemă rezolvată de un program nedeterminist

- un program nedeterminist are mai multe **fire de execuție**
- un **program nedeterminist rezolvă**  $P$  dacă  $\forall x \in P \exists$  un fir de execuție care se termină și a cărei configurație finală include  $P(x)$

## Exemplu: problema celor N regine

*Input:* o tablă de șah  $n \times n$ .

*Output:* o așezare a  $n$  piese de tip regină pe tablă a.î. nicio regină nu atacă o altă regină.

```

attacked(i, j, b) {
    attack = false;
    for (k = 0; k < i; ++k)
        if ((b[k] == j) || ((b[k]-j) == (k-i)) || ((b[k]-j) == (i-k)))
            attack = true;
    return(attack);
}

```

```

nqueens (n) {
    for (i = 0; i < n; ++i) {
        choose j in { 0 .. n-1 } s.t. ! (attacked(i, j, b));
        b[i] = j;
    }
}

```

Timp de execuție:  $O(n^2)$

Detalii pe tablă.

# Exemplu: problema celor N regine

```
$ alki tests/nqueens.alk -cINIT="n |-> 4"
```

```
Program status: failure ;
```

```
State:
```

```
  b |-> [ 0, 2, -1, -1 ]
```

```
  i |-> 2
```

```
  j |-> 3
```

```
  n |-> 4
```

```
...
```

```
$ alki tests/nqueens.alk -cINIT="n |-> 4"
```

```
State: i.e. success
```

```
  b |-> [ 1, 3, 0, 2 ]
```

```
  n |-> 4
```

## Exemplu: Submultime de sumă dată (SSD)

```
/*
Input: O multime S de numere intregi, M numar intreg pozitiv.
Output: O submultime S cu sumx | x in S = M.
```

```
*/

PM = 0;
/* choose a maximal size for the subset */
choose k in {1 .. S.size()};
/* try to choose at most k-1 elements */
for(i = 0; i < k-1; ++i) {
    choose x in S s.t. PM + x <= M;
    S = S \ singletonSet(x);
    PM = PM + x;
}
/* try to choose the k-th element, if needed */
if (PM != M)
    choose x in S s.t. PM == M;
```

Timp de execuție:  $O(n)$ , unde  $n = S.size()$ .

Detalii pe tablă.

# Exemplu: Submultime de sumă dată (SSD)

```
$ alki ssd.alk --init="S |-> {1, 3, 4, 7, 9}  M |-> 14"
State:
```

```
M |-> 14
PM |-> 14
S |-> { 1, 3, 4, 7, 9 }
i |-> 4
k |-> 5
x |-> 4
```

```
$ alki ssd.alk --init="S |-> {1, 3, 4, 7, 9}  M |-> 14"
Program status: failure ;
State:
```

```
M |-> 14
PM |-> 13
S |-> { 1, 3, 4, 7, 9 }
i |-> 4
k |-> 5
x |-> 9
```

## Reducerea la algoritmi determiniști

Fie  $\sim$  o relație de echivalență peste stări. De exemplu  $\sigma \sim \sigma'$  dacă  $\sigma$  și  $\sigma'$  codifică la fel o instanță  $p$  a unei probleme  $P$  sau codifică la fel răspunsul  $P(p)$ .

### Definition

Algoritmul  $A$  este **echivalent** cu algoritmul  $B$  (relativ la  $\sim$ ) dacă:

- ①  $\langle A, \sigma_1 \rangle \Rightarrow^* \langle \cdot, \sigma'_1 \rangle$  și  $\sigma_1 \sim \sigma_2$  implică existența lui  $\sigma'_2$  a.î.  
 $\langle B, \sigma_2 \rangle \Rightarrow^* \langle \cdot, \sigma'_2 \rangle$  și  $\sigma'_1 \sim \sigma'_2$ , și
- ② reciproc,  $\langle B, \sigma_2 \rangle \Rightarrow^* \langle \cdot, \sigma'_2 \rangle$  și  $\sigma_1 \sim \sigma_2$  implică existența lui  $\sigma'_1$  a.î.  
 $\langle A, \sigma_1 \rangle \Rightarrow^* \langle \cdot, \sigma'_1 \rangle$  și  $\sigma'_1 \sim \sigma'_2$ .

### Theorem

*Pentru orice algoritm nedeterminist  $A$  există un algoritm determinist  $B$  echivalent, care are complexitatea timp în cazul cel mai nefavorabil  $T_B(n) = O(2^{T_A(n)})$ .*

# Execuție exhaustivă a algoritmilor nedeterminiști 1/2

```
choose x1 in { 1 .. 5 };
```

```
$ ~/k-3.6/bin/krun tests/choose.alk -cINIT=".Map" --search
```

Search results:

Solution no 1:

State:

x1 |-> 1

...

Solution no 5:

State:

x1 |-> 5



# Execuție exhaustivă a algoritmilor nedeterminiști 2/2

```
odd(x) {
    return x % 2 == 1;
}
choose x1 in { 1 .. 5 } s.t. odd(x1);
choose x2 in { 1 .. 5 } s.t. odd(x2);
```

```
$ alki choosetest.alk --search
```

```
Search results:
```

```
Solution no 1:
```

```
State:
```

```
    x1 |-> 1
```

```
    x2 |-> 1
```

```
Solution no 2:
```

```
State:
```

```
    x1 |-> 1
```

```
    x2 |-> 3
```

```
...
```

```
Solution no 9:
```

```
State:
```

```
    x1 |-> 5
```

```
    x2 |-> 5
```

```
</state>
```

# Exemplu: problema celor N regine exhaustiv

```
$ alki nqueens.alk --init="n|->4" --search
```

Search results:

**Solution no 1:**

State: *i.e. success*

```
b |-> [ 1, 3, 0, 2 ]
```

```
n |-> 4
```

**Solution no 2:**

State: *i.e. success*

```
b |-> [ 2, 0, 3, 1 ]
```

```
n |-> 4
```

...

**Solution no 6:**

Program status: **failure** ;

State:

```
b |-> [ 3, 1, -1, -1 ]
```

```
i |-> 2
```

```
j |-> 3
```

```
n |-> 4
```

# Plan

- 1 Algoritmi nedeterminiști în general
- 2 Algoritmi nedeterminiști pentru probleme de decizie
- 3 Algoritmi probabiliști
  - Variabilă aleatorie
  - Exemplu de algoritm Monte Carlo: Compoziționalitatea unui număr întreg
  - Algoritmi Las Vegas
  - Exemplu de algoritm Las Vegas:  $k$ -mediana
- 4 Material suplimentar

# Definiție

- Activitatea unui algoritm nedeterminist pentru o problemă de decizie:
  - “se ghicește” o anumită structură  $S$
  - verifică dacă  $S$  satisface o proprietățile din *output*
  - dacă proprietatea este satisfăcută, se apeleaza instrucțiunea de terminare success, altfel instrucțiunea de terminare failure;
- Extensia limbajului:
  - `success;` – semnalează terminarea verificării (și a a algoritmului) cu succes
  - `failure;` – semnalează terminarea fără succes

# Algoritmi nedeterminiști pentru probleme de decizie: exemplu

SAT

*Instance:* O mulțime finită de variabile și o formulă propozițională  $F$  în formă normală conjunctivă. *Question:* Este  $F$  adevărată pentru o anumite atribuire de variabile? (i.e., este  $F$  satisfiabilă?)

```
// guess
for (i = 0; i < n; ++i) {
    choose z in {false, true};
    x[i] = z;
}
// check
if (f(x)) success;
else failure;
```

# Exemplu de instanță SAT

```
f(x) {  
    return (x[0] || x[1]) &&  
           (!x[0] || x[3] || x[2]) &&  
           (x[2] || !x[3]) &&  
           (!x[1] || !x[2] || x[3]);  
}
```

# Execuție nedeterministă

```
$ ~/k-3.6/bin/krun tests/sat.alk -cINIT="n |-> 4"
<k>
    failure ;
</k>
<state>
    i |-> 4
    n |-> 4
    x |-> [ false, true, false, true ]
    z |-> true
</state>
<stack>
    .List
</stack>
```

# Execuție nedeterministă

```
$ alki sat.alk --init="n |-> 4"
```

```
Program status: failure ;
```

```
State:
```

```
  i |-> 4
```

```
  n |-> 4
```

```
  x |-> [ true, true, false, false ]
```

```
  z |-> false
```

```
...
```

```
$ alki sat.alk --init="n |-> 4"
```

```
Program status: success ;
```

```
State:
```

```
  i |-> 4
```

```
  n |-> 4
```

```
  x |-> [ true, false, true, false ]
```

```
  z |-> false
```



# Execuție exhaustivă 1/2

```
$ alki sat.alk --init="n |-> 4" --search
```

```
Solution no 1:
```

```
Program status: failure ;
```

```
State:
```

```
  i |-> 4
  n |-> 4
  x |-> [ false, false, false, false ]
  z |-> false
```

```
...
```

```
Solution no 12:
```

```
Program status: success ;
```

```
State:
```

```
  i |-> 4
  n |-> 4
  x |-> [ false, true, false, false ]
  z |-> false
```

```
...
```

## Execuție exhaustivă 2/2

...

Solution no 16:

Program status: success ;

State:

i |-> 4

n |-> 4

x |-> [ true, true, true, true ]

z |-> true

# Exemplu: Submultime de sumă dată (SSD1)

```

/*
Instance: O multime S de numere intregi, M numar intreg pozitiv.
Question: Exista o submultime S cu sumx | x in S = M?

*/

PM = 0;
choose k in 1 .. S.size();
for(i = 0; i < k; ++i) {
    choose x in S;
    S = S \ singletonSet(x);
    PM = PM + x;
}
if (PM == M) success;
else failure;

```

## Exemplu: Submultime de sumă dată (SSD1)

```
$ alki ssd.alk --init="S |-> 1, 3, 4, 7, 9 M |-> 14"
```

```
Program status: failure ;
```

```
State:
```

```
M |-> 14
```

```
PM |-> 9
```

```
S |-> 1, 3, 4, 7
```

```
i |-> 1
```

```
k |-> 1
```

```
x |-> 9
```

```
alki ssd.alk --init="S |-> 1, 3, 4, 7, 9 M |-> 14"
```

```
Program status: success ;
```

```
State:
```

```
M |-> 14
```

```
PM |-> 14
```

```
S |-> 1, 9
```

```
i |-> 3
```

```
k |-> 3
```

```
x |-> 4
```

# Plan

- 1 Algoritmi nedeterminiști în general
- 2 Algoritmi nedeterminiști pentru probleme de decizie
- 3 Algoritmi probabiliști
  - Variabilă aleatorie
  - Exemplu de algoritm Monte Carlo: Compoziționalitatea unui număr întreg
  - Algoritmi Las Vegas
  - Exemplu de algoritm Las Vegas:  $k$ -mediana
- 4 Material suplimentar

# Plan

- 1 Algoritmi nedeterminiști în general
- 2 Algoritmi nedeterminiști pentru probleme de decizie
- 3 Algoritmi probabiliști
  - Variabilă aleatorie
  - Exemplu de algoritm Monte Carlo: Compoziționalitatea unui număr întreg
  - Algoritmi Las Vegas
  - Exemplu de algoritm Las Vegas:  $k$ -mediana
- 4 Material suplimentar

# Variabilă aleatorie: definiție

## Definition

O **variabilă aleatorie** este o funcție  $X$  definită peste un câmp de evenimente  $\Omega$ .

Exemple (considerăm numai variabile discrete)

1.  $D2$  (două zaruri):

- experimentul aleatoriu: aruncarea a două zaruri
- $D2$  întoarce perechea formată ce reprezintă numerele de puncte de pe fețele superioare ale celor două zaruri

2.  $SD2$  (suma a două zaruri):

- experimentul aleatoriu: aruncarea a două zaruri
- $SD2$  întoarce suma numerelor de puncte de pe fețele superioare ale celor două zaruri.

3.  $CB$  ("chocolate bar", cazul discret):

- experimentul aleatoriu: alegerea unui număr  $i$  din mulțimea  $\{1, 2, \dots, n\}$ ,  $n > 1$
- $CB$  întoarce  $\max(\frac{i}{n}, \frac{n-i}{n})$

# Funcția random()

random( $n$ ) - întoarce un număr  $x \in \{0, 1, \dots, n - 1\}$  ales aleatoriu uniform (i.e. cu probabilitatea  $\frac{1}{n}$ )

Timp de execuție:  $O(1)$

Experiment: fie programul

```
a = [0, 0, 0, 0];
for (i = 0; i < 10000; ++i) {
    j = random(4);
    a[j] = a[j] + 1;
}
```

Starea finală obținută prin execuția programului:

State:

```
a |-> [ 2448, 2538, 2473, 2541 ]
i |-> 10000
j |-> 3
```

Probabilitățile calculate experimental (e.g.  $\frac{2448}{10000}$ ) sunt apropiate de cele teoretice (de fapt sunt aproximări ale acestora).



# Variabile aleatorii ca programe

```
D2()  
{  
    d[0] = random(6) + 1;  
    d[1] = random(6) + 1;  
    return d;  
}
```

```
SD2()  
{  
    d1 = random(6) + 1;  
    d2 = random(6) + 1;  
    return d1 + d2;  
}
```

```
CB(n)  
{  
    i = random(n) + 1;  
    s1 = i / n;  
    s2 = (n - i) / n;  
    if (s1 > s2 ) return s1;  
    return s2;  
}
```

# Variabilă aleatorie: media

Considerăm numai variabile aleatorii  $X$  discrete, care au ca valori o mulțime cel mult numărabilă  $x_1, x_2, \dots$  de numere reale.

$p_i = Pr(X = x_i)$  - probabilitatea ca  $X$  să ia valoarea  $x_i$

Valoarea medie a lui  $X$ :  $M(X) = \sum_i x_i \cdot p_i$

Proprietăți ale mediei:

$$M(X + Y) = M(X) + M(Y)$$

$$M(X \cdot Y) = M(X) \cdot M(Y)$$

( $X$  și  $Y$  independente)

# Media lui CB

①  $n$  impar:

– valorile posibile pentru  $CB$  sunt  $\left\{ \frac{k}{n} \mid k = n-1, n-2, \dots, \frac{n+1}{2} \right\}$ , fiecare cu probabilitatea  $\frac{2}{n-1}$

$$M(CB) = \sum_{k=\frac{n+1}{2}}^{n-1} \frac{k}{n} \frac{2}{n-1} = \frac{3n-1}{4n} < \frac{3}{4}$$

②  $n$  par:

– valorile posibile pentru  $CB$  sunt  $\left\{ \frac{k}{n} \mid k = n-1, n-2, \dots, \frac{n}{2} + 1 \right\}$ , fiecare cu probabilitatea  $\frac{2}{n-1}$ , și  $\frac{1}{2}$  cu probabilitatea  $\frac{1}{n-1}$

$$M(CB) = \sum_{k=\frac{n}{2}+1}^{n-1} \frac{k}{n} \frac{2}{n-1} + \frac{1}{2(n-1)} = \frac{3n-4}{4n-4} < \frac{3}{4}$$

**Obs.**  $\frac{3n-4}{4n-4} = \frac{3(n-1)-1}{4(n-1)}$ , de unde rezultă mediile (valorile așteptate) pentru  $n$  și  $n-1$  sunt aceleași dacă  $n$  este par.

# Aproximarea valorii medii prin rulări succesive

```
sum = 0.0;
for (j = 0; j < 100; ++j)
    sum = sum + CB(31);

m = sum / float(31);
```

Valori obținute cu două execuții: 0.766, 0.752.

Pe unele sisteme de operare, interpretorul Alk s termină cu eroare daca algoritmul include operații cu numere raționale (float), provenite de la biblioteca utilizată.

# Definiții

Exista două puncte de vedere:

## ① algoritmi Monte Carlo

Pot produce rezultate incorecte cu o mică probabilitate.

Dacă un astfel de algoritm este executat de mai multe ori peste alegeri independente alese aleatoriu de fiecare dată, probabilitatea de eșec poate fi oricât de mică dar cu un timp de execuție mai mare.

## ② algoritmi Las Vegas

Nu produc niciodată rezultate incorecte, dar timpul poate varia de la o execuție la alta.

Alegerile aleatorii sunt făcute pentru a stabili timpul mediu de execuție, care în esență este independent de intrare.

# Plan

- 1 Algoritmi nedeterminiști în general
- 2 Algoritmi nedeterminiști pentru probleme de decizie
- 3 Algoritmi probabiliști
  - Variabilă aleatorie
  - Exemplu de algoritm Monte Carlo: Compoziționalitatea unui număr întreg
  - Algoritmi Las Vegas
  - Exemplu de algoritm Las Vegas:  $k$ -mediana
- 4 Material suplimentar

# Compoziționalitatea: motivație

Algoritm naiv pentru testarea primalității:

```
isPrime(x) {  
    if (x < 2) return false;  
    for (i= 2; i <= x / 2; ++i)  
        if (x % i == 0) return false;  
    return true;  
}  
b = isPrime(15061 * 15073);
```

Execuție:

```
$ time ../../bin/alki ../../miscelanea/prime.alk  
State:  
    b |-> false  
real 0m16.113s
```

Programul a rulat mai mult de 16 secunde pe un Mac 2,4 GHz Intel Core 2 Duo.

# Compoziționalitatea: domeniul problemei

## Simbol Legendre:

$$(a/p) = \begin{cases} 0 & \text{dacă } a \equiv 0 \pmod{p}, \\ 1 & \text{dacă } a \not\equiv 0 \pmod{p} \text{ și există } x: a \equiv x^2 \pmod{p}, \\ -1 & \text{dacă } a \not\equiv 0 \pmod{p} \text{ și nu există un astfel de } x. \end{cases}$$

unde  $p$  este prim;

## Simbol Jacobi:

$$(a|n) = (a/p_1)^{\alpha_1} (a/p_2)^{\alpha_2} \cdots (a/p_k)^{\alpha_k},$$

unde  $n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$  este un număr întreg pozitiv impar descompus în factori primi.



# Compoziționalitatea: domeniul problemei

- ❶ Dacă  $n > 3$  și este prim, atunci  $(a|n) = (a/n)$ , motiv pentru care în literatură de utilizează aceeași notație (de multe ori  $\frac{a}{n}$ ).
- ❷ Dacă  $a \equiv b \pmod{n}$  atunci  $(a|n) = (b|n)$ .
- ❸ 
$$(a|n) = \begin{cases} 0 & \text{dacă } \gcd(a, n) \neq 1, \\ \pm 1 & \text{dacă } \gcd(a, n) = 1. \end{cases}$$
- ❹  $(ab|n) = (a|n)(b|n)$ , deci  $(a^2|n) = 1$  sau  $0$
- ❺  $(a|mn) = (a|m)(a|n)$ , deci  $(a|n^2) = 1$  sau  $0$

# Compoziționalitatea: domeniul problemei

- 5 Dacă  $m$  și  $n$  sunt coprime, atunci

$$(m|n)(n|m) = (-1)^{\frac{m-1}{2} \cdot \frac{n-1}{2}} = \begin{cases} 1 & \text{dacă } n \equiv 1 \pmod{4} \text{ or } m \equiv 1 \pmod{4}, \\ -1 & \text{dacă } n \equiv m \equiv 3 \pmod{4} \end{cases}$$

6  $(-1|n) = (-1)^{\frac{n-1}{2}} = \begin{cases} 1 & \text{dacă } n \equiv 1 \pmod{4}, \\ -1 & \text{dacă } n \equiv 3 \pmod{4}, \end{cases}$

7  $(2|n) = (-1)^{\frac{n^2-1}{8}} = \begin{cases} 1 & \text{dacă } n \equiv 1, 7 \pmod{8}, \\ -1 & \text{dacă } n \equiv 3, 5 \pmod{8}. \end{cases}$

## Alte proprietăți deduse

$$(a \cdot 2|n) = (a|n) \cdot (2|n) = \begin{cases} -(a|n) & \text{dacă } n \equiv 3, 5 \pmod{8} \\ (n|a) & \text{altfel} \end{cases}$$

$$(a|n) = \begin{cases} -(n|a) & \text{dacă } a \neq 0 \neq n \text{ și } n \equiv a \equiv 3 \pmod{4} \\ (n|a) & \text{altfel} \end{cases}$$

## De la domeniul problemei la algoritm

```
jacobi(a, n)
{
    j = 1;
    while (a != 0) {
        while (a % 2 == 0) { // a is even
            a = a / 2;
            if (n % 8 == 3 || n % 8 == 5) j = 0-j;
        }
        swap(a, n);
        if (a % 4 == 3 && n % 4 == 3) j = 0-j;
        a = a % n;
    }
    if (n == 1) return j;
    else return 0;
}
```

# Algoritmul Solovay-Strassen: descriptiv

Input: un număr întreg pozitiv impar  $n$  pentru care se testează compoziționalitatea,

Output: "composite" dacă  $n$  este compus, "maybe prime" altfel

- 1 alege aleatoriu  $a$  în  $[2, n - 1]$
- 2  $x = (a|n)$
- 3 dacă  $x == 0$  sau  $a^{(n-1)/2} \not\equiv x \pmod{n}$  atunci returnează "composite"
- 4 altfel returnează "maybe prime"

# Algoritmul Solovay-Strassen în Alk

```
isComp(n)
{
    a = random(n-3) + 2;
    if (gcd(a, n) != 1) return "composite";
    x = jacobi(a, n);
    if (x < 0) x = x + n;
    if (x != power(a, (n-1)/2, n)) return "composite";
    return "maybe prime";
}
```

# Algoritmul Solovay-Strassen: demo

```
m1[0] = 15061 * 15073;
```

```
m1[1] = isComp(m1[0]);
```

```
m2[0] = 15061;
```

```
m2[1] = isComp(m2[0]);
```

Rezultat:

```
$ time ../../bin/alki compos.alk
```

State:

```
    m1 |-> [ 227014453, "composite" ]
```

```
    m2 |-> [ 15061, "may be prime" ]
```

```
real 0m0.916s
```

```
...
```

Mai puțin de o secundă pentru două teste!

# Algoritmul Solovay-Strassen ca test de primalitate cu o anumită probabilitate

Input: un număr întreg pozitiv impar  $n$  pentru care se testează compoziționalitatea,

un număr întreg pozitiv impar  $n$  ce reprezintă acuratețea

Output: "composite" dacă  $n$  este compus, "probably prime" altfel

```
isProbPrime(n, k) {  
    while (k > 0 && isComp(n) != "composite")  
        --k;  
    if (k == 0) return "probably prime";  
    return "composite";  
}
```

Probabilitatea de eșec este  $2^{-k}$ .

(O justificare poate fi găsită în Richard M. Karp. An introduction to randomized algorithms. Discrete Applied Mathematics 34 (1991) 165-201.)



# Algoritmul Solovay-Strassen ca test de primalitate: demo

```
m3[0] = 1031267;  
m3[1] = isProbPrime(m3[0], 100);
```

Rezultat:

```
$ time ../../bin/alki compos.alk
```

State:

```
      m3 |-> [ 1031267, "probably prime" ]  
real 0m22.945s
```

Testul cu algoritmul naiv:

```
$ time ../../bin/alki ../miscelanea/prime.alk
```

State:

```
      d |-> true  
real 17m56.410s
```

# Algoritm eficient pentru funcția putere-modulo

De la domeniul problemei:

$$a^n \pmod{p} = \begin{cases} 1 & \text{dacă } n = 0, \\ a \pmod{p} & \text{dacă } n = 1, \\ (a^{\frac{n}{2}} \pmod{p}) * (a^{\frac{n}{2}} \pmod{p}) \pmod{p} & \text{dacă } n \% 2 == 0, \\ (a * a^{n-1} \pmod{p}) \pmod{p} & \text{dacă } n \% 2 == 1, \end{cases}$$

la algoritm:

```
power(a, n, p)
  x = 1;
  while (n > 0)
    if (n % 2 == 0) {
      a = (a * a) % p ;
      n = n / 2;
    }
    else {
      x = (a * x) % p;
      n = n - 1;
    }
  return x;
}
```

# Exerciții

1. Să se determine timpul de execuție pentru `power(a, n, p)`.
2. Să se determine timpul de execuție pentru `isComp(n)`.
3. Să se determine timpul de execuție pentru `isProbPrime(n, k)`.

# Plan

- 1 Algoritmi nedeterminiști în general
- 2 Algoritmi nedeterminiști pentru probleme de decizie
- 3 **Algoritmi probabiliști**
  - Variabilă aleatorie
  - Exemplu de algoritm Monte Carlo: Compoziționalitatea unui număr întreg
  - **Algoritmi Las Vegas**
  - Exemplu de algoritm Las Vegas:  $k$ -mediana
- 4 Material suplimentar

# Timpul mediu de execuție al algoritmilor probabiliști 1/2

Notății:

$prob_{A,x}(C)$  = probabilitatea cu care algoritmul  $A$  execută calculul  $C$  pentru intrarea  $x$

$time_{A,x}(C)$  = timpul necesar lui  $A$  ca să execute calculul  $C$  pentru intrarea  $x$  (un pic diferit față de cazul determinist)

## Timpul mediu de execuție al algoritmilor probabiliști 2/2

timpul mediu de execuție a lui  $A$  pentru intrarea  $x$  este

$$\text{exp-time}(A, x) = M[\text{time}_{A,x}] = \sum_C \text{prob}_{A,x}(C) \cdot \text{time}_{A,x}(C).$$

$\text{time}_{A,x}$  este variabilă aleatorie.

timpul mediu de execuție a lui  $A$  în cazul cel mai nefavorabil este

$$\text{exp-time}(A, n) = \max\{\text{exp-time}(A, x) \mid \text{size}(x) = n\}$$

Dacă  $A$  este subînțeles din context, atunci scriem numai  $\text{exp-time}(n)$  ( $\text{exp-time}(x)$ ) în loc de  $\text{exp-time}(A, n)$  (resp.  $\text{exp-time}(A, x)$ ).

# Plan

- 1 Algoritmi nedeterminiști în general
- 2 Algoritmi nedeterminiști pentru probleme de decizie
- 3 Algoritmi probabiliști
  - Variabilă aleatorie
  - Exemplu de algoritm Monte Carlo: Compoziționalitatea unui număr întreg
  - Algoritmi Las Vegas
  - Exemplu de algoritm Las Vegas:  $k$ -mediana
- 4 Material suplimentar

## $k$ -mediana: problema

### Definition

Fie  $S$  o listă cu  $n$  elemente dintr-o mulțime univers total ordonată.

$k$ -mediana este cel de-al  $k$ -lea element din lista sortată a elementelor din  $S$ .

Presupunem  $S$  memorată într-un tablou.

Considerăm următoarea problemă:

*Input*      un tablou  $(a[i] \mid 0 \leq i < n)$  și un număr  $k \in \{0, 1, \dots, n-1\}$ ,  
*Output*      $k$ -mediana



## $k$ -mediana: descriere algoritm

Se alege un pivot  $x$  din  $a[0..n-1]$ .

Partiționează tabloul  $a$  în jurul lui  $x$ : se permută elementele tabloului astfel încât  $a[j] = x$  și

$$(\forall i)(i < j \implies a[i] \leq x) \wedge (i > j \implies a[i] \geq x)$$

care este echivalent cu:

$$(\forall i)(i < j \implies a[i] \leq a[j]) \wedge (i > j \implies a[i] \geq a[j])$$

- ①  $j = k \implies$  problema este rezolvată
- ②  $j < k \implies$  caută  $k$  în  $a[j+1..n-1]$
- ③  $j > k \implies$  caută  $k$  în  $a[0..j-1]$

# Partiționare Lomuto

```
partition(out a, p, q)
{
    pivot = a[q];
    i = p - 1;
    for (j = p; j < q; ++j)
        if (a[j] <= pivot) {
            i = i + 1;
            swap(a, i, j);
        }
    swap(a, i+1, q);
    return i + 1;
}

swap(out a, i, j) {
    if (i != j) {
        temp = a[i];
        a[i] = a[j];
        a[j] = temp;
    }
}
```

# Analiza partiționării Lomuto

## Corectitudine:

Invariantul buclei for:

$$i < j \wedge (\forall k)(p \leq k \leq i \implies a[k] \leq pivot) \wedge (i < k < j \implies a[k] > pivot)$$

După for:

invariantul și  $j = q$ , care implică  $a[i + 1] > pivot$

După ultimul swap:

$a[i + 1] = pivot$  și

$$(\forall k)(p \leq k \leq i \implies a[k] \leq pivot) \wedge (i < k < q \implies a[k] > pivot).$$

Număr comparații:  $q - p$

# Partiționare aleatorie

Pivotul este ales aleatoriu din  $a[p..q]$ :

```
randPartition(out a, p, q) {  
    if (p < q) {  
        i = p + random(q - p);  
        swap(a, i, q);  
        return partition(a, p, q);  
    }  
}
```

## $k$ -mediana: algoritm Las Vegas

```
randSelectRec(out a, p, q, k)
{
    j = randPartition(a, p, q);
    if (j == k) return a[j];
    if (j < k) return randSelectRec(a, j+1, q, k);
    return randSelectRec(a, p, j-1, k);
}

randSelect(out a, k)
{
    return randSelectRec(a, 0, a.size()-1, k);
}
```

## randSelect: analiza 1/3

$\text{exp-time}(n, k)$  - timpul mediu pentru a găsi  $k$ -mediana într-un tablou de lungime  $n$

$$\text{exp-time}(n) = \max_k \text{exp-time}(n, k)$$

Pentru că analizăm timpul mediu în cazul cel mai nefavorabil, presupunem apelul recursiv alege tot timpul subtabloul mai mare

Reamintim că  $M(CB) < \frac{3}{4}$

Rezultă că dacă se împarte aleatoriu în două un tablou de lungime  $n$ , lungimea medie a celui mare subtablou este cel mult  $\frac{3}{4}n$ .

# randSelect: analiza 2/3

## Lemă

*Lungimea medie a tabloului după  $i$  apeluri recursive este cel mult  $\left(\frac{3}{4}\right)^i n$ .*

Demonstrarea lemei.

$L_i$  variabila aleatorie care dă lungimea tabloului după  $i$  apeluri.

$P_j$  variabila aleatorie care dă partea fracționară de elemente păstrate la nivelul  $j$ .

$X_j$  variabila aleatorie care dă lungimea celui mare subtablou la nivelul  $j$ .

Avem:  $L_i = n \prod_{j=1}^i P_i$ ,  $M(P_i) = M\left(\frac{X_j}{n}\right) = \frac{M(X_j)}{n} \leq \frac{\frac{3}{4}n}{n} = \frac{3}{4}$ ,

$P_1, \dots, P_n$  sunt independente,

$M(L_i) = M(n \prod_{j=1}^i P_i) = n \prod_{j=1}^i M(P_i) \leq \left(\frac{3}{4}\right)^i n$

Acum lema e demonstrată.

## randSelect: analiza 3/3

La nivelul  $i$ , numărul de operații este liniar, pp.  $\leq aX_i + b$

Fie  $r \leq n$  numărul de apeluri recursive. Timpul mediu:

$$\begin{aligned}
 \text{exp-time}(n) &= M \left( \sum_{i=1}^r (aY_i + b) \right) \\
 &= \sum_{i=1}^r M(aY_i + b) \\
 &\leq \sum_{i=1}^n (aM(Y_i) + b) \\
 &\leq \sum_{i=1}^n \left( a \left( \frac{3}{4} \right)^i n + b \right) \\
 &= an \sum_{i=1}^n \left( \frac{3}{4} \right)^i + bn \\
 &\leq 4an + bn \\
 &= O(n)
 \end{aligned}$$



# Plan

- 1 Algoritmi nedeterminiști în general
- 2 Algoritmi nedeterminiști pentru probleme de decizie
- 3 Algoritmi probabiliști
  - Variabilă aleatorie
  - Exemplu de algoritm Monte Carlo: Compoziționalitatea unui număr întreg
  - Algoritmi Las Vegas
  - Exemplu de algoritm Las Vegas:  $k$ -mediana
- 4 Material suplimentar

# Un algoritm determinist liniar pentru $k$ -mediană

- 1 grupează tabloul în  $(n/5)$  grupe de 5 elemente și calculează mediana fiecărei grupe;
- 2 calculează recursiv mediana medianelor  $p$
- 3 utilizează  $p$  ca pivot și separă elementele din tablou
- 4 apelează recursiv pentru subtabloul potrivit (în care se află  $k$ -mediana)

Manuel Blum, Robert W. Floyd, Vaughan Pratt, Ronald L. Rivest, and Robert E. Tarjan. Linear time bounds for median computation, 1972.

# Un algoritm determinist liniar: analiza 1/2

Notății:  $T(n, k)$  timpul pentru cazul cel mai nefavorabil pentru  $k$ -mediana,  
 $T_n = \max_k T(n, k)$

Pasul 1:  $O(n)$

Pasul 2:  $T(n/5)$

Pasul 3:  $O(n)$

pasul 4: presupunând că cel puțin  $(3|10)$  din tablou este  $\leq p$  și că cel puțin  $(3|10)$  din tablu este  $\geq p$ , pasul recursiv ia cel mult  $T((7n|10))$ .

Însumând obținem:

$$\begin{aligned} T(n) &\leq cn + T((n|5)) + T((7n|10)) \\ &\leq cn + c(n|5) + T((n|5^2)) + T((7n|5 \cdot 10)) + c(7n|10) + T((7n|5 \cdot 10)) + T((7n|5 \cdot 10)) + \dots \\ &\leq \dots \\ &= O(n) \end{aligned}$$

(similar teoremei de master).

## Un algoritm determinist liniar: analiza 2/2

Demonstrarea afirmației "cel puțin  $(3|10)$  din tablou este  $\leq p$  și că cel puțin  $(3|10)$  din tablou este  $\geq p$ ":

Fie  $g = (n|5)$ .

Cel puțin  $\lceil (g|2) \rceil$  dintre grupuri (cele cu mediana  $\leq p$ ) au cel puțin trei elemente  $\leq p$ .

Rezultă că numărul de elemente  $\leq p$  este cel puțin  $3\lceil (g|2) \rceil \geq (3n|10)$ .  
Analog pentru numărul de elemente  $\geq p$ .

Comparați experimental cei doi algoritmi pentru mediană.