Outline

Cuprins

1 Algoritmi nedeterminişti în general

Motivaţie

- abstractizarea spaţiului de stări este utilă în analiză
- algoritmii nedeterminişti 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 complexităţii problemelor
- noţiune preliminară pentru cea de algoritm probabilist

Algoritmi nedeterminişti intuitiv

- pentru anumite stări există mai multe posibilităţi de a continua calculul (presupunem că acestea sunt în număr finit)
- în consecinţă, pentru aceeaşi intrare, algoritmul poate da rezultate (configuraţii finale) diferite
- interpretarea angelică: algoritmul ghiceşte calea de execuţie care duce la rezultatul corect
- timpul de execuţie = timpul calculului care duce la rezultatul corect

Extensia limbajului

choose x in S;
– întoarce un element din S ales arbitrar
– timp de execuţie (uniform): $O(1)$

choose x in S s.t. B;
– întoarce un element din S care satisface condiţia B ales arbitrar
– timp de execuţie (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;
}

L = emptyList;
for (i = 0; i < 8; i = i+2)
  L.pushBack(i);
choose x in L 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 execuţie*

- un program nedeterminist rezolvă P dacă $\forall\, x \in P\ \exists$ un fir de execuţie care se termină şi a cărui configuraţie finală include $P(x)$

Exemplu: problema celor N regine

Input: o tablă de şah $n \times n$. *Output:* o aşezare 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 execuţie: $O(n^2)$
Detalii pe tablă.

Exemplu: problema celor N regine

```
$ alki tests/nqueens.alk --init="n |-> 4"
Program status: failure ;
State:

    b |-> [ 0, 2, -1, -1 ]
    i |-> 2
    j |-> 3
    n |-> 4
...
$ alki tests/nqueens.alk --init="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 sum{x | x in S} = M, dacă există.

*/

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 execuţie: $O(n)$, unde $n = S.size()$ (am presupus $T(\texttt{S} \setminus \texttt{singletonSet(x)}) = O(1)$).
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 determinişti
Fie $\sim$ o relaţie de de echivalenţă peste stări. De exemplu $\sigma \sim \sigma'$ dacă σ şi σ' codifică la fel o instanţă p a unei probleme P sau codifică la fel răspunsul $P(p)$.

Definition 1. Algoritmul A este echivalent cu algoritmul B (relativ la $\sim$) dacă:

1. $\langle A, \sigma_1\rangle \Rightarrow^* \langle \cdot, \sigma_1'\rangle$ şi $\sigma_1 \sim \sigma_2$ implică existenţa lui σ_2' a.î. $\langle B, \sigma_2\rangle \Rightarrow^* \langle \cdot, \sigma_2'\rangle$ şi $\sigma_1' \sim \sigma_2'$, şi
2. reciproc, $\langle B, \sigma_2\rangle \Rightarrow^* \langle \cdot, \sigma_2'\rangle$ şi $\sigma_1 \sim \sigma_2$ implică existenţa lui σ_1' a.î. $\langle A, \sigma_1\rangle \Rightarrow^* \langle \cdot, \sigma_1'\rangle$ şi $\sigma_1' \sim \sigma_2'$.

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

Execuţie exhaustivă a algoritmilor neteterminişti 1/2

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

```
$ alki choose.alk --search
Search results:
Solution no 1:
State:

    x1 |-> 1
...
Solution no 5:
State:

    x1 |-> 5
```

Execuţie exhaustivă a algoritmilor neteterminişti 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 choosest.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
```

2 Agoritmi nedeterminişti pentru probleme de decizie

Definiţie

Algoritmi nedeterminişti pentru probleme de decizie: exemplu
SAT *Instance*: O mulţime finită de variabile şi o formulă propoziţională F în formă normală conjunctivă.
Question: Este F adevărată pentru o anume 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 instanţă SAT

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

Execuţie nedeterministă

```
$ alki  sat.alk --init="n |-> 4"
Program status: failure ;
State:

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

Execuţie 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
```

Execuţie 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
...
```

Execuţie 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 (SSD3)

```
/*
Instance: O multime S de numere intregi, M numar intreg pozitiv.
Question: Exista o submultime S cu sum{x | 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
```

3 Algoritmi probabilişti

3.1 Variabilă aleatorie

Variabilă aleatorie: definiţie

Definition 3. O variabilă aleatorie este o funcţie X definită peste un câmp de evenimente Ω.

Exemple (considerăm numai variabile discrete)
1. $D2$ (două zaruri):

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

2. $SD2$ (suma a două zaruri):

- evenimentul aleatoriu: aruncarea a două zaruri

- $SD2$ întoarce suma numerelor de puncte de pe feţele superioare ale celor două zaruri.

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

   - evenimentul aleatoriu: alegerea unui număr i din mulţimea $\{1, 2, \ldots, n\}$, $n > 1$

   - CB întoarce $\max(\frac{i}{n}, \frac{n-i}{n})$, $i = 0, 1, \ldots, n$

Funcţia `random()`

`random(`n`)` - întoarce unumăr $x \in \{0, 1, \ldots, n-1\}$ ales aleatoriu uniform (i.e. cu probabilitatea $\frac{1}{n}$)
Timp de execuţie: $O(1)$
Experiment: fie programul

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

Starea finală obţinută prin execuţia programului:

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

Probabilităţile calculate experimental (e.g. $\frac{2448}{10000}$) sunt apropriate 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) + 1;
  s1 = float(i) / float(n);
  s2 = float(n - i) / float(n);
  if (s1 > s2 ) return s1;
  return s2;
}
```

Variabilă aleatorie: media
Considerăm numai variabile aleatorii X discrete, care au ca valori o mulţime cel mult numărabilă $x_1, x_2, \ldots$ de numere reale.

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

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

Proprietăţi ale mediei:
$M(X + Y) = M(X) + M(Y)$
$M(X \cdot Y) = M(X) \cdot M(Y)$
(X şi Y independente)

Media lui CB

1. n impar:
   – valorile posibile pentru CB sunt $\left\{ \frac{k}{n} \,\middle|\, k = n-1, n-2, \ldots, \frac{n+1}{2} \right\}$, 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 $\left\{ \frac{k}{n} \,\middle|\, k = n-1, n-2, \ldots, \frac{n}{2}+1 \right\}$, fiecare cu probabilitatea $\frac{2}{n-1}$, şi $\frac{1}{2}$ 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 aşteptate) pentru n şi $n-1$ sunt aceleaşi dacă n este par.

Concluzie: $M(CB) < \frac{3}{4}$

Aproximarea valorii medii prin rulări succesive

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

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

Valori obţinute cu două execuţii: 0.745, 0.751.
Pe unele sisteme de operare, interpretorul Alk s termină cu eroare daca algoritmul include operaţii cu numere raţionale (float), provenite de la biblioteca utilizată.

Algoritmi probabilişti: definiţii
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 eşec poate fi oricât de mică dar cu un timp de execuţie mai mare.

2. algoritmi Las Vegas
   Nu produc niciodată rezultate incorecte, dar timpul poate varia de la o execuţie la alta.
   Alegerile aleatorii sunt făcute pentru a stabili timpul mediu de execuţie, care în esenţă este independent de intrare.

3.2 Exemplu de algoritm Monte Carlo: Compoziţionalitatea unui număr întreg

Compoziţionalitatea: motivaţie
Algoritm naiv pentru testarea primalităţii:

```
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);
```

Execuţie:

```
$ 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.

Compoziţionalitatea: domeniul problemei
Simbol Legendre:

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

unde p este prim; [1ex] 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.

Compoziţionalitatea: domeniul problemei

1. Dacă $n > 3$ şi este prim, atunci $(a|n) = (a/n)$, motiv pentru care în literatură de utilizează aceeaşi notaţie (de multe ori $\frac{a}{n}$).

2. Dacă $a \equiv b \pmod{n}$ atunci $(a|n) = (b|n)$.

3. $(a|n) = \begin{cases} 0 & \text{dacă } \gcd(a,n) \neq 1, \\ \pm 1 & \text{dacă } \gcd(a,n) = 1. \end{cases}$

4. $(ab|n) = (a|n)(b|n)$, deci $\left(a^2|n\right) = 1$ sau 0

5. $(a|mn) = (a|m)(a|n)$, deci $\left(a|n^2\right) = 1$ sau 0

Compoziţionalitatea: domeniul problemei

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

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 proprietăţi deduse

$(a \cdot 2|n) = (a|n)\cdot(2|n) = \begin{cases} -(a|n) & \text{dacă } n \equiv 3, 5 \pmod 8 \\ (n|a) & \text{altfel} \end{cases}$ [3ex] $(a|n) = \begin{cases} -(n|a) & \text{dacă } a \neq 0 \neq n \text{ şi } n \equiv a \\ (n|a) & \text{altfel} \end{cases}$

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 testeaza compoziţionalitatea,
Output: "composite" dacă n este compus, "maybe prime" altfel [2ex]

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 returneaza "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 puţin 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 testeaza compoziţionalitatea, un număr întreg pozitiv impar k ce reprezintă acurateţea 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 eşec 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 funcţia putere-modulo
De la domeniul problemei:

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

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;
}
```

Exerciţii
1. Să se determine timpul de execuţie pentru `power(a, n, p)`. [2ex] 2. Să se determine timpul de execuţie pentru `isComp(n)`. [2ex] 3. Să se determine timpul de execuţie pentru `isProbPrime(n, k)`.

3.3 Algoritmi Las Vegas

Timpul mediu de execuţie al algoritmilor probabilişti 1/2
Nu produc niciodată rezultate incorecte, dar timpul poate varia de la o execuţie la alta.
Alegerile aleatorii sunt făcute pentru a stabili timpul mediu de execuţie, care în esenţă este independent de intrare.

Notaţii:

$prob_{A,x}(C)$ = 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 faţă de cazul determinist)

Timpul mediu de execuţie al algoritmilor probabilişti 2/2
timpul mediu de execuţie a lui A pentru intrarea x este $\textit{exp-time}(A, x) = M[time_{A,x}] = \sum_C prob_{A,x}(C) \cdot time_{A,x}(C)$.

$time_{A,x}$ este variabilă aleatorie.

timpul mediu de execuţie a lui A în cazul cel mai nefavorabil este $\textit{exp-time}(A, n) = max\{\textit{exp-time}(A, x) \mid \textit{size}(x) = n\}$

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

3.4 Exemplu de algoritm Las Vegas: k-mediana

k-mediana: problema

Definition
Fie S o listă cu n elemente dintr-o mulţime univers total ordonată. k-mediana este cel de-al k-lea element din lista sortată a elementelor din S. În alte cuvinte, k-mediana este un element $x \in \{a[0], \ldots, a[n-1]\}$ cu proprietăţile $|\{i \mid 0 \leq i < n \wedge a[i] < x\}| < k$ şi $|\{i \mid 0 \leq i < n \wedge a[i] \leq x\}| \geq k$ (dacă toate lelementele din S sunt distincte, atunci avem egalitate în ultima relaţie).

Presupunem S memorată într-un tablou. Considerăm următoarea problemă:

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

Evident, orice algoritm de sortare rezolvă problema de mai sus. Deoarece cerinţele pentru selecţie sunt mai slabe decât cele de la ordonare, se pune firesc întrebarea dacă există algoritmi mai performanţi decât cei utilizaţi la sortare.

k-mediana: descriere algoritm
Se alege un pivot x din $a[0..n-1]$.
Partiţionează tabloul `a` în jurul lui x: se permută elemenetele tabloului astfel încât $a[j] = x$ şi

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

care este echvalent cu:

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

Dacă $j = k$ atunci problema este rezolvată. Dacă $j < k$ atunci cel de-al k-lea cel mai mic element trebuie căutat în subtabloul `a[j+1..n]`, iar dacă $j > k$ atunci cel de-al k-lea cel mai mic element trebuie căutat în subtabloul `a[1..j-1]`.

Partiţionare 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 partiţionării Lomuto
Corectitudine:
Invariantul buclei for:
$i < j \wedge (\forall k)(p \leq k \leq i \implies a[k] \leq pivot) \wedge (i < k < j \implies a[i] > pivot)$
După for:
invariantul şi $j = q$, care implică $a[i+1] > pivot$
După ultimul `swap`:
$a[i+1] = pivot$ şi
$(\forall k)(p \leq k \leq i \implies a[k] \leq pivot) \wedge (i < k < q \implies a[i] > pivot)$.

Număr comparaţii: $q - p$

Partiţionare 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

exp-time(n, k) - timpul mediu pentru a găsi k-mediana într-un tablou de lungime n

exp-time$(n) = \max_k$ *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 1. *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 fracţionară de elemente păstrate la nivelul j.
X_j variabila aleatorie care dă lungimea celui mare subtablou la nivelul j.
Avem: $L_i = n \prod_{j=1}^{i} P_j$, $P_j = \frac{X_j}{n}$ (presupunem că la fiecare apel recursiv se alege subtabloul cel mai lung), $M(P_j) = M\left(\frac{X_j}{n}\right) = \frac{M(X_j)}{n} \leq \frac{\frac{3}{4}n}{n} = \frac{3}{4}$,
$P_1, \ldots, P_n$ sunt independente,
$M(L_i) = M(n \prod_{j=1}^{i} P_j) = n \prod_{j=1}^{i} M(P_j) \leq \left(\frac{3}{4}\right)^i n$
Acum lema e demonstrată.

randSelect: analiza 3/3

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

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

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

4 Material suplimentar

Un algoritm determinist liniar pentru k-mediană

1. grupează tabloul în $(n|5)$ grupe de 5 elemente şi calculează mediana fiecărei grupe;
2. calculează recursiv mediana medianelor p
3. utilizează p ca pivot şi 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
Notaţii: $T(n,k)$ 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 puţin $(3|10)$ din tablou este $\leq p$ şi că cel puţin $(3|10)$ din tablu este $\geq p$, pasul recursiv ia cel mult $T((7n|10))$.
Însumând obţinem:

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

(similar teoremei de master).

Un algoritm determinist liniar: analiza 2/2
Demonstrarea afirmaţiei "cel puţin $(3|10)$ din tablou este $\leq p$ şi că cel puţin $(3|10)$ din tablou este $\geq p$": Fie $g = (n|5)$. Cel puţin $\lceil (g|2) \rceil$ dintre grupuri (cele cu mediana $\leq p$) au cel puţin trei elemente $\leq p$. Rezultă că numărul de elemente $\leq p$ este cel puţin $3\lceil (g|2) \rceil \geq (3n|10)$. Analog pentru numărul de elemente $\geq p$.

Comparaţi experimental cei doi algoritmi pentru mediană.