Algoritmi Probabiliști

Algoritmi Probabiliști

- În timpul rezolvării unei probleme, putem ajunge la un moment dat în situaţia de a avea **de ales** între mai multe variante de continuare.

Algoritmi Probabiliști

- În timpul rezolvării unei probleme, putem ajunge la un moment dat în situaţia de a avea **de ales** între mai multe variante de continuare.
    - **se alege aleator una dintre variante**

Algoritmi Probabiliști

- În timpul rezolvării unei probleme, putem ajunge la un moment dat în situaţia de a avea **de ales** între mai multe variante de continuare.
    - **se alege aleator una dintre variante**
    - **la executări diferite ale unui algoritm probabilist, rezultatele sunt în general diferite.**

Algoritmi Probabiliști

Categorii

- **Monte Carlo**
- **Las Vegas**
- **Algoritmi numerici**

Algoritmi Monte Carlo

Algoritmi Monte Carlo

- Furnizează totdeauna un rezultat, care însă **nu este neapărat corect**
- **Probabilitatea ca rezultatul să fie corect crește pe măsură ce timpul disponibil crește**

Algoritmi Monte Carlo

Se consideră un vector cu *n* elemente distincte. Să se determine un element al vectorului care să fie mai mare sau egal cu mediana celor n numere din vector

- **n este foarte mare**
    - **timpul avut la dispoziţie este mic**

Algoritmi Monte Carlo - Mediana

v = -∞

Repetă fără a depăşi timpul disponibil:

- alegem aleatoriu x un element al vectorului
- v = maxim(v, x) = cel mai mare element ales

scrie v

Algoritmi Monte Carlo - Mediana

- Care este probabilitatea ca un element ales x să fie mai mic/mai mare decât mediana?

- **Care este probabilitatea ca răspunsul să fie corect/greșit după k încercări?**

Algoritmi Monte Carlo - Mediana

- Care este probabilitatea ca un element ales x să fie mai mic/mai mare decât mediana?

  - probabilitatea să fie mai mare sau egal $\geq 1/2$

  - probabilitatea să fie mai mic $< 1/2$

- **Care este probabilitatea ca răspunsul să fie corect/greșit după k încercări?**

Algoritmi Monte Carlo - Mediana

- **Care este probabilitatea ca răspunsul să fie greșit după k încercări?**

  - Probabilitatea ca **toate** cele k elemente alese (în timpul de rulare avut la dispoziţie) să fie mai mici decât mediana

P(v < mediana) ≤ $\left(\frac{1}{2}\right)^k$

Algoritmi Monte Carlo - Mediana

- Care este probabilitatea ca răspunsul să fie corect după k încercări?

  $1 - 1/2^k$

  - **Pentru k=20, această probabilitate este mai mare decât 0,999999**

Algoritmi Monte Carlo - Mediana

- Care este probabilitatea ca răspunsul să fie corect după k încercări?

  $$1 - 1/2^k$$

  - **Pentru k=20, această probabilitate este mai mare decât 0,999999**

- **$1 - (1-p)^k$ - corect**
  - $1-p$ = probabilitatea de a greși la un pas

Algoritmi Monte Carlo

Se consideră un vector cu *n* elemente. Să se determine dacă există un element majoritar în vector (cu frecvența > $n/2$)

- **Mai general – cu frecvența $>fr \cdot n$**

Algoritmi Monte Carlo – Element majoritar

v = -∞

Repetă fără a depăşi timpul disponibil:

- alegem aleator x un element al vectorului
- Calculam f = frecventa lui x
- Daca f>n/2 scrie DA; STOP

scrie NU

Algoritmi Monte Carlo – Element majoritar

Analiza

- **Problemă de decizie**
- Dacă scrie **DA, răspunsul este corect**
- Dacă scrie NU, este posibil ca să existe element majoritar (deci răspunsul să fie greșit)

Algoritmi Monte Carlo – Element majoritar

Analiza

- **Problemă de decizie**
- Dacă scrie DA, răspunsul este corect
- Dacă scrie NU, este posibil ca să existe element majoritar (deci răspunsul să fie greșit)
- Care este probabilitatea de a **răspunde greșit NU** după k încercări?

Algoritmi Monte Carlo – Element majoritar

Analiza

- **Problemă de decizie**
- Dacă scrie DA, răspunsul este corect
- Dacă scrie NU, este posibil ca să existe element majoritar (deci răspunsul să fie greșit)
- Care este probabilitatea de a **răspunde greșit NU** după k încercări?
  - ► Dacă există element majoritar, probabilitatea să nu îl găsească după k încercări este $\leq 1/2^k$

Algoritmi Monte Carlo

Algoritmul lui KARGER de determinare a unei tăieturi minime într-un graf

- Jon Kleinberg and Éva Tardos - Algorithm Design
- D. R. Karger, Global min-cuts in rnc, and other ramifications of a simple min-out algorithm. In Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA, 1993
- D. R. Karger, S. Clifford, A new approach to the minimum cut problem, Journal of the ACM. 43 (4), 1996 doi:10.1145/234533.234534.
- **https://en.wikipedia.org/wiki/Karger's_algorithm**

Algoritmul lui KARGER

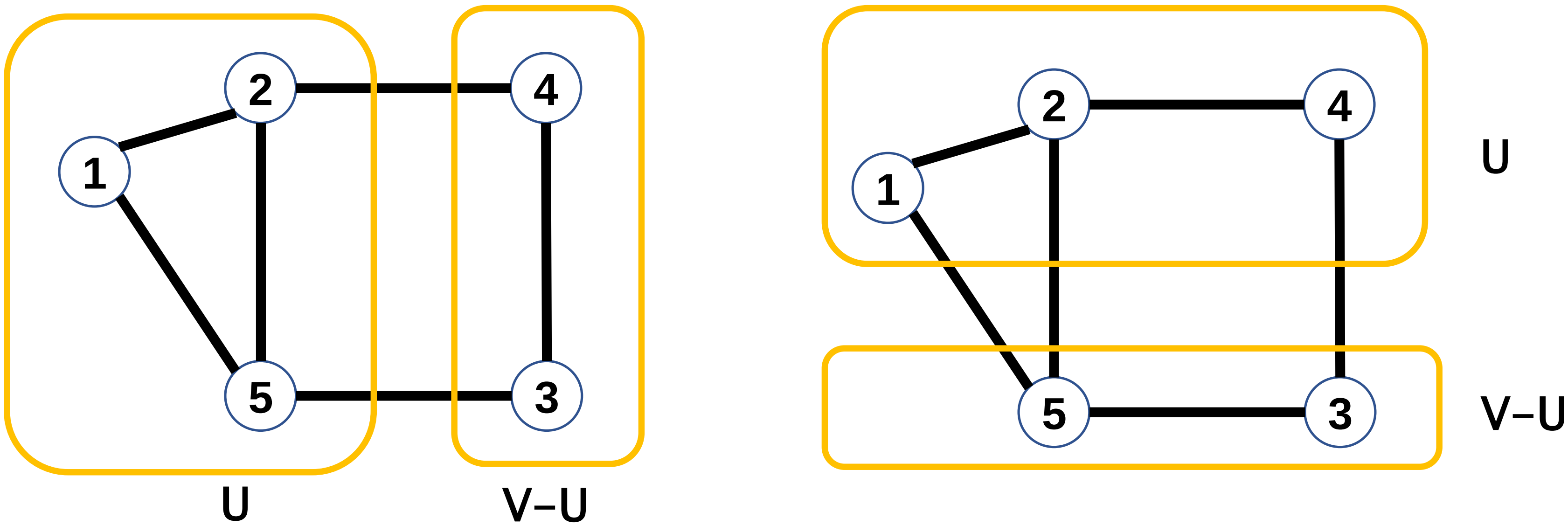
- **Tăietură într-un graf neorientat G=(V,E)**

  = partiționare a mulțimii vârfurilor (U,V-U)

  \- muchiile de la U la V-U sunt muchiile tăieturii

Cardinalul tăieturii = numărul de muchii ale tăieturii

Tăietură minimă = de cardinal minim



cardinal 2

cardinal 3

Algoritmul lui KARGER

- **Tăietură minimă într-un graf neorientat G=(V,E)**

  - **Soluție – bazată pe fluxuri maxime în rețele de transport**

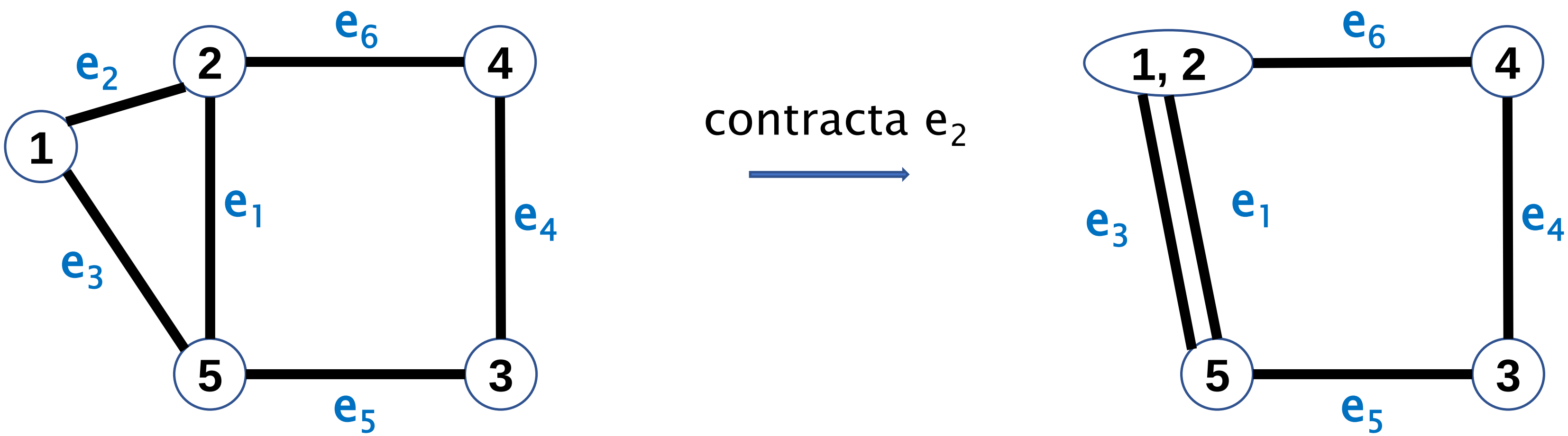
    - cu ajutorul fluxurilor putem determina s-t tăietură minimă într-un graf **orientat**, pentru s, t fixate (o s-t tăietură= tăietură (U,V-U) în care $s \in U$, $t \notin U$) – **Soluție**: transformăm graful în graf orientat cu toate capacitățile arcelor 1, repetam algoritmul pentru $t \in V$ (v. Jon Kleinberg and Éva Tardos - Algorithm Design)

  - **Aplicații – probleme de conectivitate**

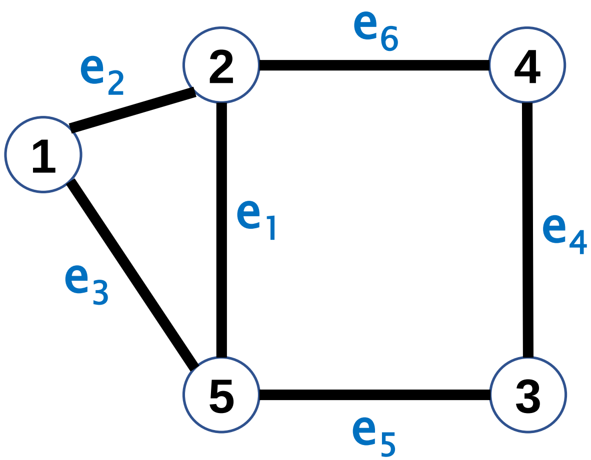
Algoritmul lui KARGER

- **Idee**
  - Se alege aleatoriu o muchie și se contractă (eliminând buclele) până când multigraful obținut are două vârfuri u și v
  - U = mulțimea vârfurilor contractate în u

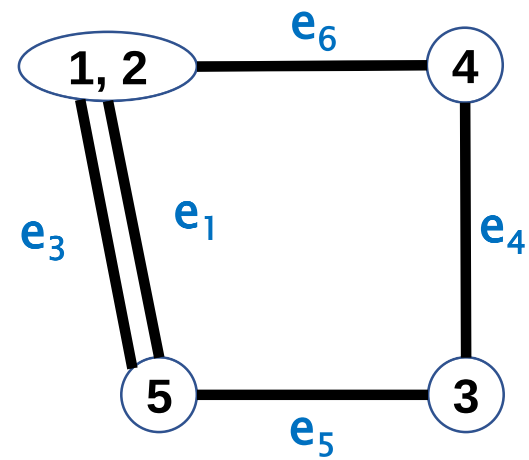
Algoritmul lui KARGER



Algoritmul lui KARGER

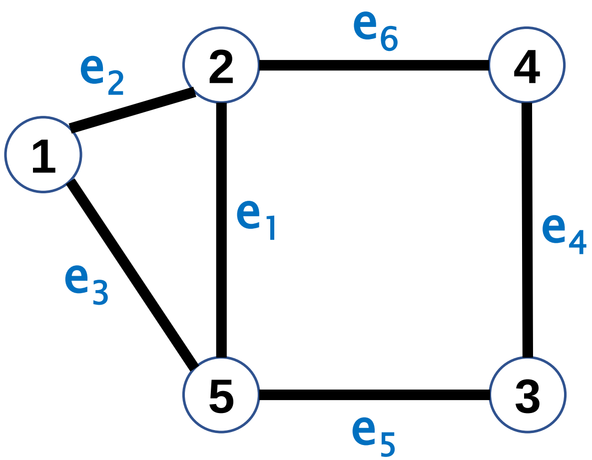


contracta e_2

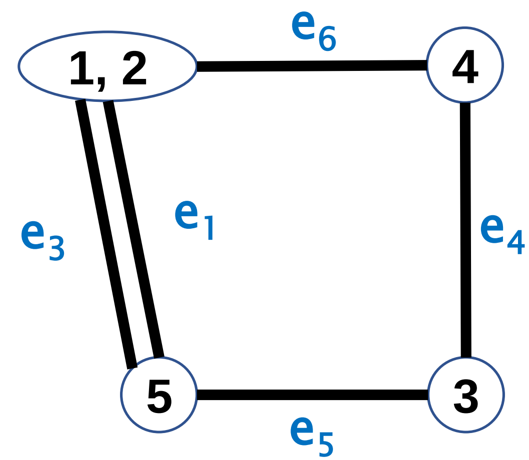


contracta e_5

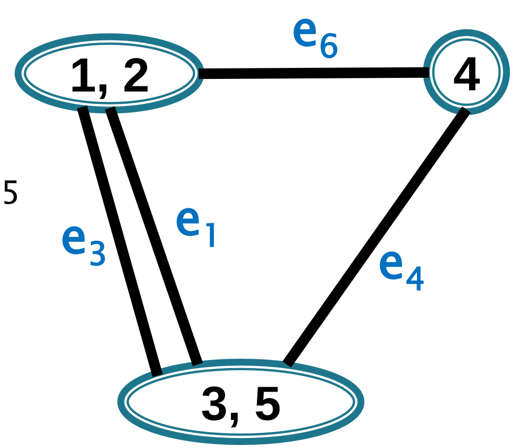
Algoritmul lui KARGER



contracta e_2

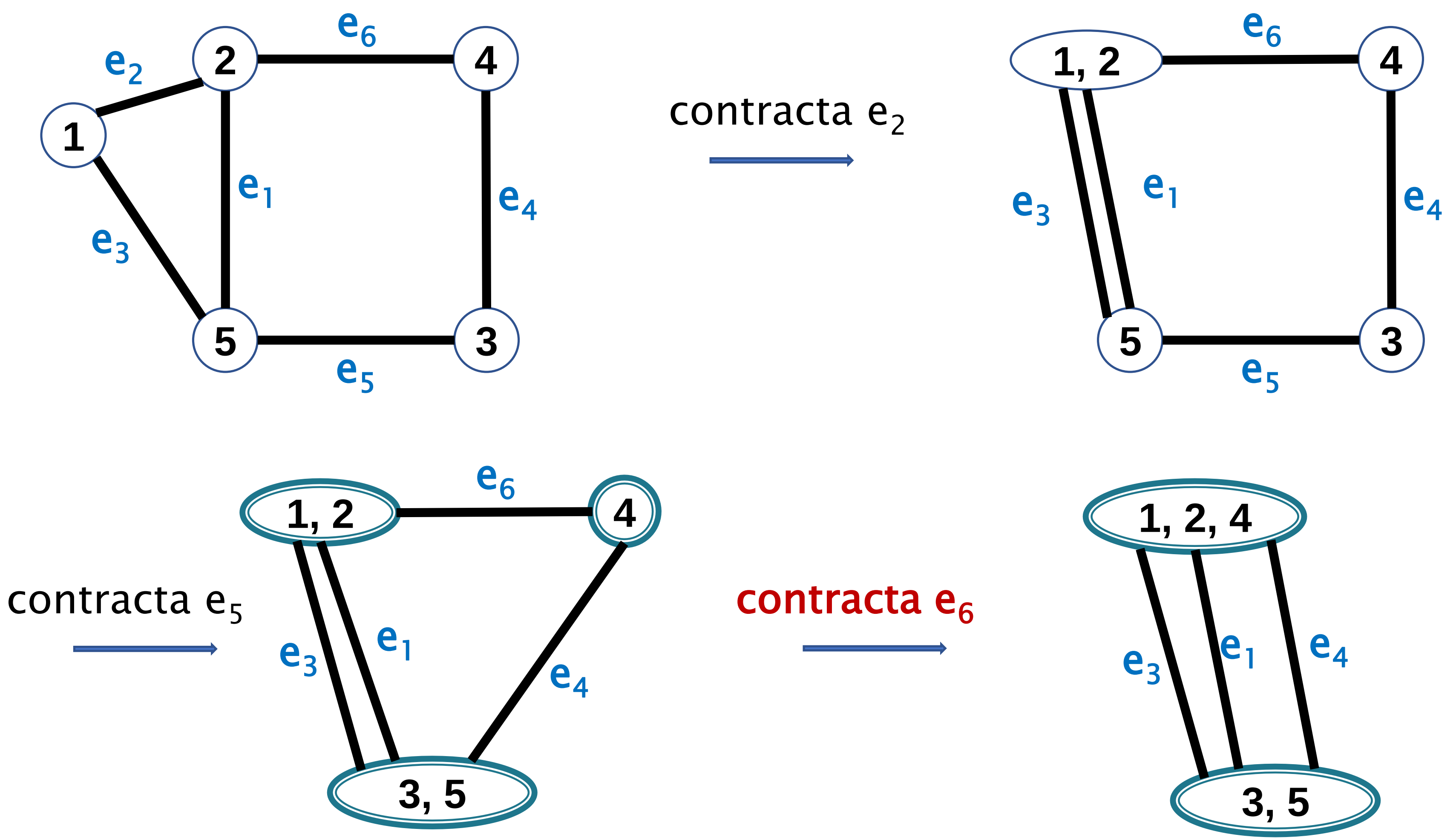


contracta e_5

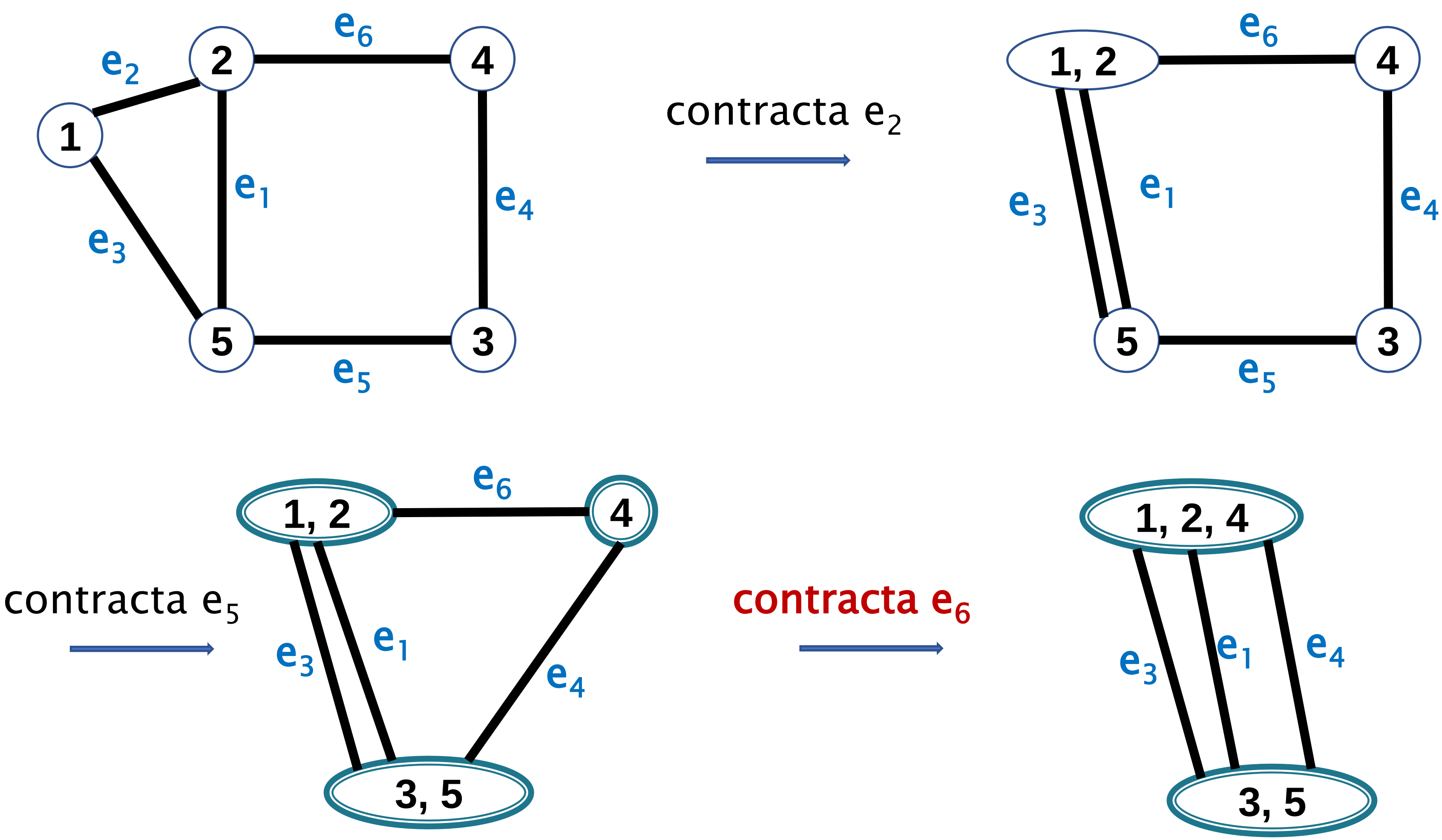


contracta e_6

Algoritmul lui KARGER

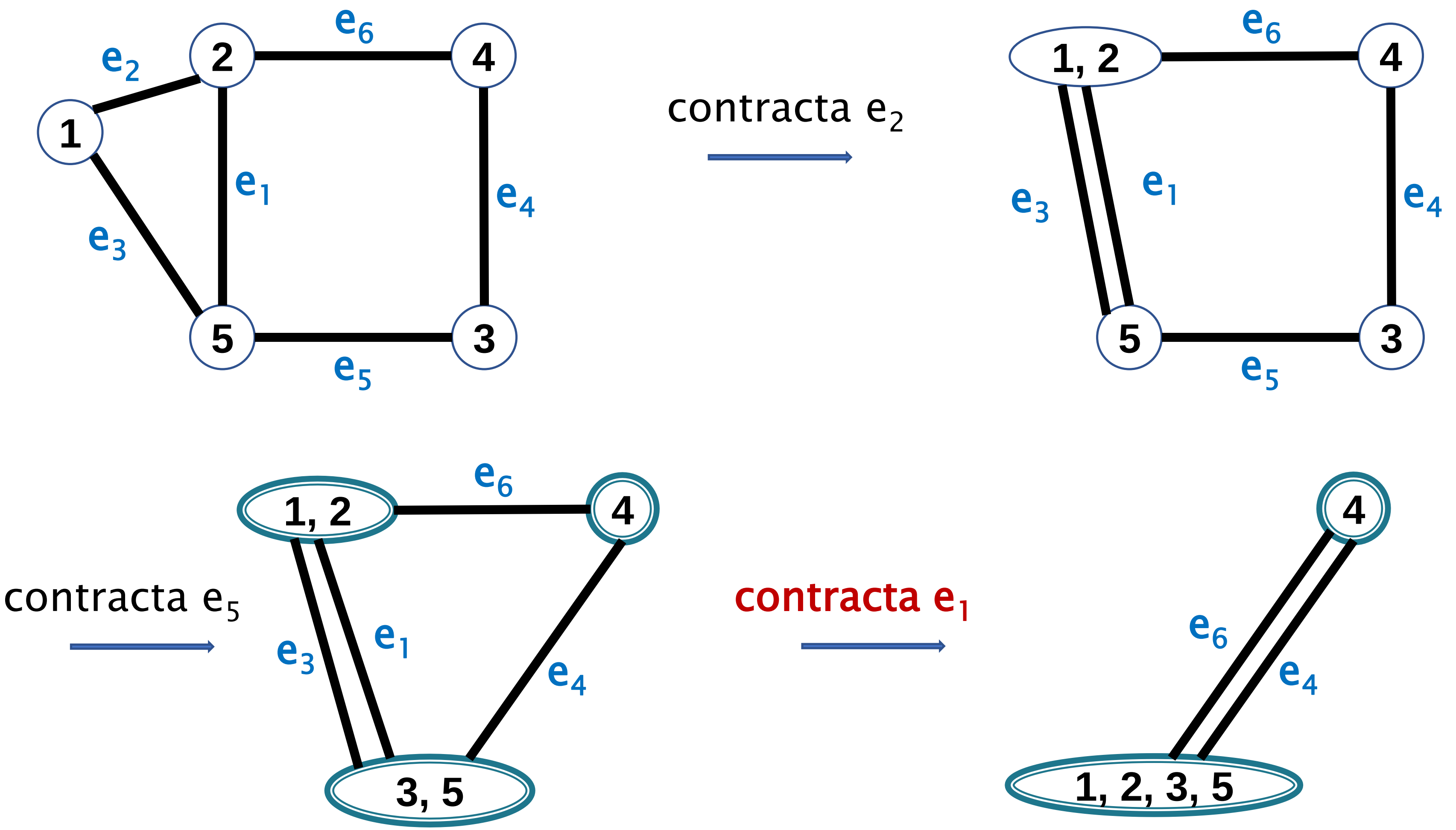


Algoritmul lui KARGER



Tăietura obținută are cardinal 3, algoritmul nu furnizează mereu o soluție optimă

Algoritmul lui KARGER



Algoritmul lui KARGER

- **Pseudocod – pentru o etapă** (se repetă de un număr de ori)

```
pentru v ∈ V executa M(v) ← {v}

cat timp |V(G)| >2 executa

     alege aleator o muchie e=uv

       contracta muchia e obtinand un supernod n_uv

       M(n_uv) ← M(u) ∪ M(v)

fie u,v cele doua varfuri ramase in V(G)

U ← M(u)

returneaza (U, V-U) – formata cu muchiile din E(G)
```

Algoritmul lui KARGER

Care este probabilitatea ca tăietura returnată de algoritm să fie minimă?

Algoritmul lui KARGER

Care este probabilitatea ca tăietura returnată de algoritm să fie minimă?

- fixăm T=(U,V-U) o tăietură minimă cu mulțimea muchiilor F
- Calculăm probabilitatea ca algoritmul să returneze T

Algoritmul lui KARGER

Care este probabilitatea ca tăietura returnată de algoritm să fie minimă?

- fixăm T=(U,V-U) o tăietură minimă cu mulțimea muchiilor F

- Calculăm probabilitatea ca algoritmul să returneze T

returnează T ⇔ nu este contractată nicio muchie a tăieturii T (= nicio muchie din F)

Algoritmul lui KARGER

Care este probabilitatea ca tăietura returnată de algoritm să fie minimă?

- fixăm T=(U,V-U) o tăietură minimă cu mulțimea muchiilor F
- Calculăm probabilitatea ca algoritmul să returneze T

returnează T ⇔ nu este contractată nicio muchie a tăieturii T (= nicio muchie din F)

Care este probabilitatea ca la primul pas să fie contractată o muchie din F?

Algoritmul lui KARGER

Care este probabilitatea ca tăietura returnată de algoritm să fie minimă?

- fixăm T=(U,V-U) o tăietură minimă cu mulțimea muchiilor F

- Calculăm probabilitatea ca algoritmul să returneze T

returnează T $\Leftrightarrow$ nu este contractată nicio muchie a tăieturii T (= nicio muchie din F)

Care este probabilitatea ca la primul pas să fie contractată o muchie din F?

$$\frac{|F|}{|E(G)|} = \frac{|F|}{m}$$

Algoritmul lui KARGER

Care este probabilitatea ca tăietura returnată de algoritm să fie minimă?

- fixăm T=(U,V-U) o tăietură minimă cu mulțimea muchiilor F

- Calculăm probabilitatea ca algoritmul să returneze T

returnează T ⇔ nu este contractată nicio muchie a tăieturii T (= nicio muchie din F)

Care este probabilitatea ca la primul pas să fie contractată o muchie din F?

$$\frac{|F|}{|E(G)|} = \frac{|F|}{m}$$

Algoritmul lui KARGER

- Probabilitatea ca la primul pas să fie aleasă o muchie din T este

$$\frac{|F|}{m}$$

- Probabilitatea ca la primul pas să **nu** fie aleasă o muchie din T este

$$1-\frac{|F|}{m}$$

$$\frac{|F|}{m} \square\ ?$$

Algoritmul lui KARGER

- Probabilitatea ca la primul pas să fie aleasă o muchie din T este

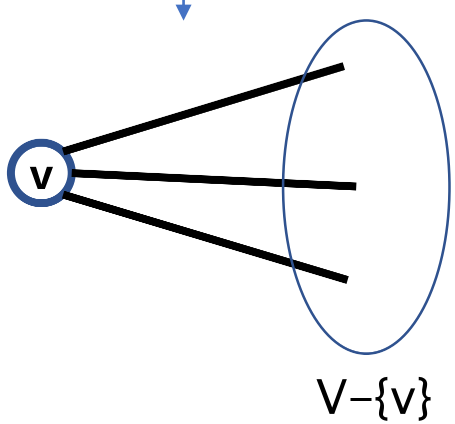
$$\frac{|F|}{m}$$

- Probabilitatea ca la primul pas să **nu** fie aleasă o muchie din T este

$$1-\frac{|F|}{m}$$

- $|F| \leq deg(v)$, $\forall v \in V$,

deoarece ({v}, V−{v}) este tăietură de cardinal deg(v)



Algoritmul lui KARGER

- Probabilitatea ca la primul pas să fie aleasă o muchie din T este

$$\frac{|F|}{m}$$

- Probabilitatea ca la primul pas să **nu** fie aleasă o muchie din T este

$$1-\frac{|F|}{m}$$

- $|F| \leq \deg(v), \forall v \in V$,

  deoarece $(\{v\}, V-\{v\})$ este tăietură de cardinal $\deg(v)$

- $$\sum_{v \in V} \deg(v) = 2m$$

Algoritmul lui KARGER

- Probabilitatea ca la primul pas să fie aleasă o muchie din T este

$$\frac{|F|}{m}$$

- Probabilitatea ca la primul pas să **nu** fie aleasă o muchie din T este

$$1-\frac{|F|}{m}$$

- $|F| \leq \deg(v)$, $\forall v \in V$,

  deoarece $(\{v\}, V-\{v\})$ este tăietură de cardinal $\deg(v)$

- $$\sum_{v\in V}\deg(v)=2m \quad \Rightarrow \quad n|F|\leq 2m \quad \Rightarrow \quad \frac{|F|}{m}\leq\frac{2}{n}$$

Algoritmul lui KARGER

- Rezultă că probabilitatea ca la primul pas să **nu** fie aleasă o muchie din T este

$$1-\frac{|F|}{m} \geq 1-\frac{2}{n}=\frac{n-2}{n}$$

Algoritmul lui KARGER

- **După primul pas** - orice tăietură din noul graf G_1 este tăietură și în G
- După ce este contractată prima muchie, dacă aceasta nu este din F, atunci F este tăietură minimă și pentru G_1 ($|V(G_1)|=n-1$)
  - $\Rightarrow$ supernodul n_{uv} are grad $\geq |F|$

    (deoarece $(\{n_{uv}\}, V-\{n_{uv}\})$ este tăietură in G_1 de cardinal $deg(n_{uv})$)
- Raționament similar $\Rightarrow$ probabilitatea ca a doua muchie aleasă să nu fie din F, **condiționată de faptul că prima nu a fost din F** este

$$\geq 1-\frac{2}{n-1}=\frac{n-3}{n-1}$$

Algoritmul lui KARGER

- **După i pași**
  - orice tăietură din noul graf G_i este tăietură și în G
  - Similar la pasul i+1, dacă până la acel pas nu a fost contractată nicio muchie din F, probabilitatea ca muchia aleasă la acest pas să nu fie din F

$$\geq 1 - \frac{2}{n-i} = \frac{n-i-2}{n-i}$$

Algoritmul lui KARGER

- **Formalizând**

  - Notăm A_i evenimentul: la pasul i nu este contractată o muchie din F

  - $Pr[A_{i+1} | A_1 \cap \ldots \cap A_i]$ = probabilitatea ca la pasul $i+1$ să nu fie contractată o muchie din F, condiționată de faptul că până la acel pas nu a fost contractată nicio muchie din F

  - Avem

$$Pr[A_{i+1} | A_1 \cap \ldots \cap A_i] \geq 1 - \frac{2}{n-i} = \frac{n-i-2}{n-i}$$

Algoritmul lui KARGER

- Rezultă că probabilitatea ca algoritmul să returneze F (să nu aleagă la niciun pas o muchie din F) este

  $= Pr[A_1] \cdot Pr[A_2 | A_1] \cdot \ldots \cdot Pr[A_{n-2} | A_1 \cap \ldots \cap A_{n-3}]$

$$\geq \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{n-2} \cdot \ldots \cdot \frac{2}{4} \cdot \frac{1}{3} = \frac{2}{n(n-1)} = \frac{1}{C_n^2} = \binom{n}{2}^{-1}$$

- $O(n^2)$ încercări – păstrăm tăietura cea mai mică obținută

- Karger – Stein - optimizare

Algoritmul lui KARGER

- Amintim relația

$$\boxed{1-p \le e^{-p}}$$

- După k repetări, probabilitatea de eșec va fi (inegalitatea Bernoulli):

$$\boxed{(1-p)^k \le e^{-pk}}$$

Algoritmul lui KARGER

$$\left(1-p\right)^k \le e^{-pk}, \quad p = \frac{1}{C_n^2}$$

- După C_n^2 încercări probabilitatea de eșec este ≤ e^{-1}
- După $C_n^2 \ln(n)$ încercări probabilitatea de eșec este $\le e^{-\ln(n)} = \dfrac{1}{n}$
- După $r \cdot C_n^2 \ln(n)$ încercări probabilitatea de eșec este $\le \dfrac{1}{n^r}$

Algoritmi Monte Carlo

Algoritmul lui SCHÖNING pentru 3-SAT

- n variabile +negații
- $E = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ – presupunem satisfiabilă
- **Clauze C_i disjunctive – cu cel mult 3 literali**
    - **$O(2^n m)$ – încercând toate cele 2^n asocieri posibile**

Algoritmul lui SCHÖNING pentru 3-SAT

Algoritmul lui SCHÖNING pentru 3-SAT (vers1)

O etapă (se repetă de un număr de ori):

1. Asociază aleatoriu valori variabilelor $x=(x_1,\dots, x_n)$
2. Repetă de n ori
   - Dacă toate clauzele sunt satisfăcute STOP
   - Alege aleatoriu o clauză C nesatisfăcută
   - Alege aleatoriu o variabilă din C și modifică-i valoarea (true↔false)

Algoritmul lui SCHÖNING pentru 3-SAT

Probabilitatea de succes?

Algoritmul lui SCHÖNING pentru 3-SAT

Probabilitatea de succes?

- Fie $v = (v_1,...,v_n)$ o asociere de valori pentru variabilele **pentru care expresia este adevărată** (soluție corectă)

- Studiem pe parcursul algoritmului

  dist(x,v)=numărul de poziții pe care x și v diferă

  (distanța Hamming)

(pentru a determina probabilitatea ca x să devină egal cu v)

Algoritmul lui SCHÖNING pentru 3-SAT

Probabilitatea de succes?

- **O primă analiză:**

  "Scenariu"

  - Inițial: **dist(x, v) ≤ n/2**
  - La o repetare distanța d(x, v) scade cu 1

Algoritmul lui SCHÖNING pentru 3-SAT

Probabilitatea de succes?

- **O primă analiză:**
  - Fie **A evenimentul**: după pasul 1 **dist(x,v) ≤ n/2**

$$\Pr[A] \geq ?$$

Algoritmul lui SCHÖNING pentru 3-SAT

Probabilitatea de succes?

- **O primă analiză:**
  - Fie **A evenimentul**: după pasul 1 **dist(x,v) ≤ n/2**

$$\Pr[A] \geq \frac{1}{2}$$

- **Simetrie: numărul de șiruri x cu dist(x,v) = k**

= $$C_n^k = C_n^{n-k}$$

= numărul de șiruri x cu dist(x,v) = n-k

Algoritmul lui SCHÖNING pentru 3-SAT

Probabilitatea de succes?

- **O primă analiză:**
  - La **o trecere** prin ciclul repeat **probabilitatea ca** dist(x,v) să scadă cu 1 este $\geq$

Algoritmul lui SCHÖNING pentru 3-SAT

Probabilitatea de succes?

- **O primă analiză:**
  - La o trecere prin ciclul repeat **probabilitatea ca** dist(x,v) să scadă cu 1 este $\geq$ **1/3**
  - **sunt cel mult n/2 poziții pe care x și v diferă**
  - **⇒ probabilitatea de succes p este**

$$p \geq \Pr[A] \cdot \Pr[succes \mid A] \geq$$

Algoritmul lui SCHÖNING pentru 3-SAT

Probabilitatea de succes?

- **O primă analiză:**
  - La o trecere prin ciclul repeat **probabilitatea ca** dist(x,v) să scadă cu 1 este $\geq$ **1/3**
  - **sunt cel mult n/2 poziții pe care x și v diferă**
  - **⇒ probabilitatea de succes p este**

$$p \geq \Pr[A] \cdot \Pr[succes \mid A] \geq$$

$$\geq \frac{1}{2} \cdot \left(\frac{1}{3}\right)^{\frac{n}{2}}$$

Algoritmul lui SCHÖNING pentru 3-SAT

- **Probabilitatea de succes?**

- **O primă analiză:**

$$p \geq \frac{1}{2}\left(\frac{1}{3}\right)^{\frac{n}{2}}$$

După k repetări, probabilitatea de eșec va fi

$$\left(1-p\right)^{k} \leq e^{-pk}$$

Algoritmul lui SCHÖNING pentru 3-SAT

- **Probabilitatea de succes?**

- **O primă analiză:**

$$p \geq \frac{1}{2}\left(\frac{1}{3}\right)^{\frac{n}{2}}$$

După k repetări, probabilitatea de eșec va fi

$$(1-p)^k \leq e^{-pk}$$

Pentru $k = \frac{\ln n}{p} \cdot r$ probabilitatea de eșec va fi ≤ $\frac{1}{n^r}$

$$p \geq \frac{1}{2}\left(\frac{1}{3}\right)^{\frac{n}{2}} \Rightarrow k \leq 2r\left(\sqrt{3}\right)^n \ln(n)$$

$$O(\left(\sqrt{3}\right)^n \ln(n))$$

Algoritmul lui SCHÖNING pentru 3-SAT

Probabilitatea de succes?

- **Analiza 2:**
  - Fie A_k evenimentul: după pasul 1 (de inițializare) x și v diferă **pe exact k poziții**:

    dist(x,v) = k

$$\Pr[A_k] = ?$$

Algoritmul lui SCHÖNING pentru 3-SAT

Probabilitatea de succes?

- **Analiza 2:**
  - Fie A_k evenimentul: după pasul 1 (de inițializare) x și v diferă **pe exact k poziții**:

    dist(x,v) = k

  - $$\Pr[A_k] = \binom{n}{k} \cdot \left(\frac{1}{2}\right)^k \cdot \left(\frac{1}{2}\right)^{n-k} = \binom{n}{k} \cdot \left(\frac{1}{2}\right)^n$$

Algoritmul lui SCHÖNING pentru 3-SAT

Probabilitatea de succes?

- **Analiza 2:**

Probabilitatea de succes

$$p \geq \sum_{k=0}^{n} \Pr[A_k] \cdot \Pr[succes \mid A_k]$$

Algoritmul lui SCHÖNING pentru 3-SAT

Probabilitatea de succes?

- **Analiza 2:**

Probabilitatea de succes

$$p \geq \sum_{k=0}^{n} \Pr[A_k] \cdot \Pr[succes \mid A_k] =$$

$$= \sum_{k=0}^{n} \binom{n}{k} \cdot \left(\frac{1}{2}\right)^n \left(\frac{1}{3}\right)^k$$

Algoritmul lui SCHÖNING pentru 3-SAT

Probabilitatea de succes?

- **Analiza 2:**

Probabilitatea de succes

$$p \geq \sum_{k=0}^{n} \Pr[A_k] \cdot \Pr[succes \mid A_k] =$$

$$= \sum_{k=0}^{n} \binom{n}{k} \cdot \left(\frac{1}{2}\right)^n \left(\frac{1}{3}\right)^k = \left(\frac{1}{2}\right)^n \sum_{k=0}^{n} \binom{n}{k} \cdot \left(\frac{1}{3}\right)^k =$$

$$= \left(\frac{1}{2}\right)^n \cdot \left(1 + \frac{1}{3}\right)^n = \left(\frac{2}{3}\right)^n$$

Algoritmul lui SCHÖNING pentru 3-SAT

- **Probabilitatea de succes?**

- **Analiza 2:**

$$p \ge \left(\frac{2}{3}\right)^n$$

După k repetări, probabilitatea de eșec va fi

$$(1-p)^k \le e^{-pk}$$

Pentru $k = \frac{\ln n}{p} \cdot r$ probabilitatea de eșec va fi $\le \frac{1}{n^r}$

$$p \ge \left(\frac{2}{3}\right)^n \Rightarrow k \le r\left(\frac{3}{2}\right)^n \ln(n)$$

$$O(\left(\frac{3}{2}\right)^n \ln(n)) \Rightarrow O((1,5)^n \ln(n))$$

Algoritmul lui SCHÖNING pentru 3-SAT

Probabilitatea de succes?

- **Varianta îmbunătățită – repet de 3n ori**

Analiza

http://www.comp.nus.edu.sg/~rahul/allfiles/cs6234-16-random-3sat.pdf

https://www.cs.cmu.edu/~15210/recitations/Randomized3Sat.pdf

U. Schöning: **A probabilistic algorithm for k-SAT and constraint satisfaction problems**, Proc. 40th Annual Symp. Foundations of Computer Science, IEEE Computer Society (1999), 410 - 414

Algoritmi Las Vegas

Algoritmi Las Vegas

- **Nu furnizează totdeauna un rezultat**, dar dacă furnizează un rezultat atunci acesta este **corect**

- **Probabilitatea ca algoritmul să se termine creşte pe măsură ce timpul disponibil creşte**

```
repeat
   if LV()
        stop
until false
```

Algoritmi Las Vegas

Se dau n texte (n foarte mare) cu următoarele proprietăţi:

- **există un unic text t_0 care apare de cel puţin 10% ori;**
- **celelalte texte sunt distincte.**

Se cere determinarea textului t_0.

Algoritmi Las Vegas - Text

Algoritm probabilist

Idee

- Generăm aleatoriu doi indici i și j și testăm dacă

  $i \neq j$ și $t_i = t_j$

  până când testul se încheie cu succes

Algoritmi Las Vegas - Text

```
repeat
   if LVText()
      stop
until false


LVText()
    i ← random(1..n); j ← random(1..n);
     if i≠j and t_i=t_j
         write t_i; return true
    else
        return false
```

Algoritmi Las Vegas - Text

Probabilitatea p de succes la un pas

(**`LVText`**() returnează true):

= probabilitatea p ca $t_i = t_j = t_0$, $j \neq i$

p = ?

Algoritmi Las Vegas - Text

Probabilitatea p de succes (**`LVText`**() returnează true):

= probabilitatea p ca $t_i = t_j = t_0$, $j \neq i$

- probabilitatea ca $t_i = t_0$

- Dacă $t_i = t_0$, probabilitatea ca $t_j = t_0$, $j \neq i$

Algoritmi Las Vegas - Text

Probabilitatea p de succes (`LVText()` returnează true):

= probabilitatea p ca $t_i = t_j = t_0$, $j \neq i$

- probabilitatea ca $t_i = t_0 \Rightarrow \frac{\frac{1}{10}n}{n} = 1/10$
- probabilitatea ca $t_j = t_0$, $j \neq i \Rightarrow \frac{\frac{1}{10}n-1}{n}$ =1/10-1/n

(dacă $t_i = t_0$)

Algoritmi Las Vegas - Text

Probabilitatea p de succes (**`LVText`**() returnează true):

= probabilitatea p ca $t_i = t_j = t_0$, $j \neq i$

- probabilitatea ca $t_i = t_0 \Rightarrow \frac{\frac{1}{10}n}{n} = 1/10$
- probabilitatea ca $t_j = t_0$, $j \neq i \Rightarrow \frac{\frac{1}{10}n - 1}{n}$ =1/10-1/n

(dacă $t_i = t_0$)

$$p = \frac{1}{10}\left(\frac{1}{10} - \frac{1}{n}\right) \geq \frac{9}{1000} \text{ pentru } n \geq 100$$

Algoritmi Las Vegas - Text

Probabilitatea p de succes (**`LVText`**() returnează true):

= probabilitatea p ca $t_i = t_j = t_0$, $j \neq i$

- probabilitatea ca $t_i = t_0 \Rightarrow \dfrac{\frac{1}{10}n}{n} = 1/10$
- probabilitatea ca $t_j = t_0$, $j \neq i \Rightarrow \dfrac{\frac{1}{10}n - 1}{n}$ =1/10-1/n

(dacă $t_i = t_0$)

$p = \frac{1}{10}\left(\frac{1}{10} - \frac{1}{n}\right) \geq \frac{9}{1000}$ pentru $n \geq 100$

Teoretic sunt suficiente $t = 1/p \approx$ **112 încercări**

(apeluri ale **`LVText`**())

Algoritmi Las Vegas

➢ **Analiza timpului estimat pentru răspuns cu succes**

```
repeat
   if LV()
        stop
until false
```

- s = timpul mediu a unei rulări cu succes a **`LV()`**
- f = timpul mediu a unei rulări cu eșec a **`LV()`**
- p = probabilitatea de succes
- t = **timpului estimat pentru răspuns cu succes** (repetând funcția **`LV()`**)

  = ?

Algoritmi Las Vegas

➢ **Analiza timpului estimat pentru răspuns cu succes**

```
repeat
   if LV()
        stop
until false
```

- s = timpul mediu a unei rulări cu succes a **`LV()`**
- f = timpul mediu a unei rulări cu eșec a **`LV()`**
- p = probabilitatea de succes
- t = **timpului estimat pentru răspuns cu succes** (repetând funcția **`LV()`**)

$$\mathbf{t = p \cdot s + (1-p) \cdot (f+t) \;\Rightarrow\; t = s + f \cdot (1-p)/p}$$

Algoritmi Las Vegas

➢ **Analiza timpului estimat pentru răspuns cu succes**

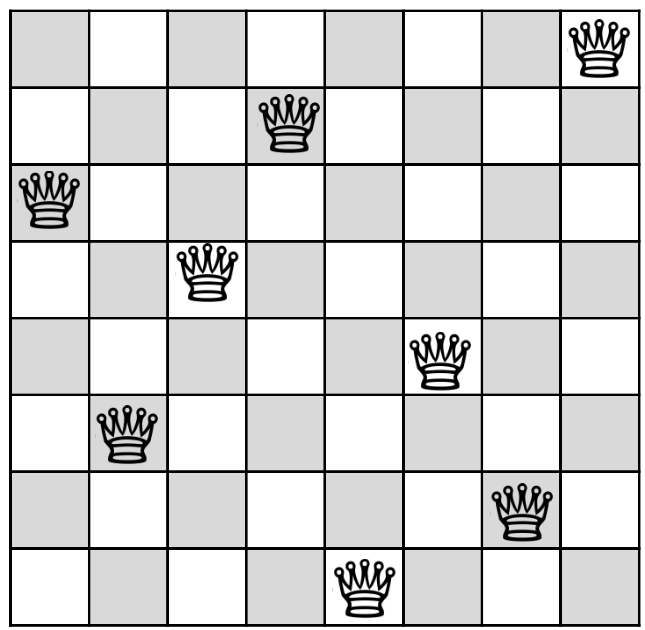
- Pentru s = f (cum este cazul **`LVText()`**) obținem

$$\mathbf{t = s \cdot 1/p}$$

Algoritmi Las Vegas

Problema celor n dame

Se consideră un caroiaj n×n. Prin analogie cu o tablă de şah (n=8), se doreşte plasarea a n dame pe pătrăţelele caroiajului, astfel încât să nu existe două dame una în bătaia celeilalte



Algoritmi Las Vegas – Problema damelor

- **Reprezentarea soluţiei**

$\mathbf{x = \{x_1, x_2, \ldots, x_n\}}$, unde

$\mathbf{x_k}$ = coloana pe care este plasată dama de pe linia $\mathbf{k}$

$\mathbf{x_k \in \{1, 2, \ldots, n\}}$

Algoritmi Las Vegas – Problema damelor

- **Backtracking**

  `n = 8 –` explorate 114 vârfuri (din 2057)

  din arborele asociat spațiului de căutare

  până când este găsită prima soluție

Algoritmi Las Vegas – Problema damelor

Algoritm probabilist

Idee

- pornim de la prima linie

repetă

- plasăm o damă aleatoriu pe linia curentă
- dacă dama nu atacă nicio damă deja plasată se trece la linia următoare

  altfel

până când se ajunge la linia n+1

Algoritmi Las Vegas – Problema damelor

Algoritm probabilist

Idee

- pornim de la prima linie

repetă

- plasăm o damă aleatoriu pe linia curentă
- dacă dama nu atacă nicio damă deja plasată se trece la linia următoare

altfel **eșec** (return false) **reluăm întreg algoritmul, nu facem doar un pas înapoi ca la Backtracking** (linia curentă devine prima linie)

până când se ajunge la linia n+1

Algoritmi Las Vegas – Problema damelor

Cum gestionăm diagonalele și coloanele pe care s-au plasat deja dame?

Algoritmi Las Vegas – Problema damelor

Vectorii:

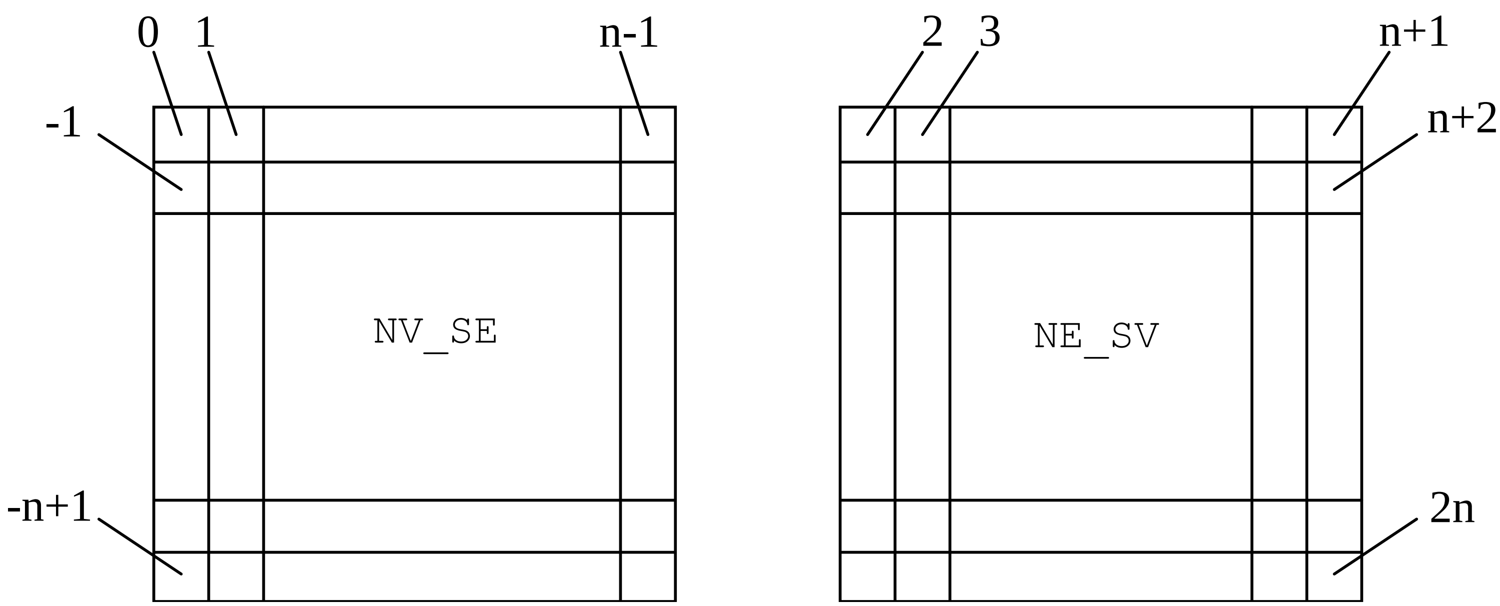
- Diagonale paralele cu diagonala principală (**j - i = constant**)

  NV_SE[-n+1..n-1]

- Diagonale paralele cu diagonala secundară (**j + i = constant**)

  NE_SV[2..2n]

- Coloane **C[1..n]**



Algoritmi Las Vegas – Problema damelor

```
repeat
    if LVDame() then
        stop
until false
```

LVDame()

```
• iniţializăm C, NV_SE, NE_SV cu true
• k ← 1
repeat

until k=n+1
write(x)
return true
```

```
LVDame()
    • inițializăm C, NV_SE, NE_SV cu true
    • k ← 1
  repeat
      • formăm un vector a cu acele poziții i∈{1,...,n}
        cu  C[i] = NV_SE[i-k] = NE_SV[i+k] = true
       și notăm na lungimea vectorului obținut









    until k=n+1
    write(x)
    return true
```

```
LVDame()
      • iniţializăm C, NV_SE, NE_SV cu true
      • k ← 1
   repeat
        • formăm un vector a cu acele poziţii i∈{1,...,n}
          cu  C[i] = NV_SE[i-k] = NE_SV[i+k] = true
         şi notăm na lungimea vectorului obţinut
        • if na>0 then
        aleg aleator i∈{1,...,na};  poz ← a_i
           x_k ← poz ;{plasam aleator dama pe o pozitie corecta}




     until k=n+1
     write(x)
     return true
```

```
LVDame()
    • iniţializăm C, NV_SE, NE_SV cu true
    • k ← 1
  repeat
      • formăm un vector a cu acele poziţii i∈{1,...,n}
        cu  C[i] = NV_SE[i-k] = NE_SV[i+k] = true
       şi notăm na lungimea vectorului obţinut
      • if na>0 then
      aleg aleator i∈{1,...,na};  poz ← a_i
         x_k ← poz ;
      NV_SE[poz-k] ← false; NE_SV[poz+k] ← false;
         C[poz] ← false;
      k ← k+1
      else

    until k=n+1
    write(x)
    return true
```

```
LVDame()
    • iniţializăm C, NV_SE, NE_SV cu true
    • k ← 1
  repeat
      • formăm un vector a cu acele poziţii i∈{1,...,n}
        cu  C[i] = NV_SE[i-k] = NE_SV[i+k] = true
       şi notăm na lungimea vectorului obţinut
      • if na>0 then
      aleg aleator i∈{1,...,na};  poz ← a_i
          x_k ← poz ;
      NV_SE[poz-k] ← false; NE_SV[poz+k] ← false;
          C[poz] ← false;
      k ← k+1
      else
      return false
    until k=n+1
    write(x)
    return true
```

Algoritmi Las Vegas – Problema damelor

- **Analiza probabilitate succes**
  - p = probabilitatea de succes
  - s = numărul mediu de vârfuri explorate la un succes
  - f = numărul mediu de vârfuri explorate la un eșec
  - t = numărul de vârfuri explorate până la încheierea cu succes (repetând funcția LVDame())

    $t = s + f \cdot (1-p)/p$

Algoritmi Las Vegas – Problema damelor

- **Experimente $n = 8$**
  - $p \approx 0.1293$
  - $f \approx 6.971$
  - $s = 9$
  - $\mathbf{t = s + f \cdot (1-p)/p} \approx 56$

Algoritmi Las Vegas – Problema damelor

- **Experimente $n = 8$**
  - $p \approx 0.1293$
  - $f \approx 6.971$
  - $s = 9$
  - $\mathbf{t = s + f \cdot (1 - p)/p} \approx 56$

- **Backtracking -114**

- **Soluții mixte – k dame plasate aleatoriu, apoi backtracking**

Algoritmi numerici

Algoritmi numerici

- Urmăresc determinarea aproximativă a unei valori
- **Cu cât timpul alocat executării algoritmului este mai mare, precizia rezultatului se îmbunătăţeşte**

Algoritmi numerici

- **Aproximarea** lui π
- **Aproximarea** $\int_a^b f(x)\,dx$

$$f:[a,b]\to[c,d]$$

Aproximarea lui π

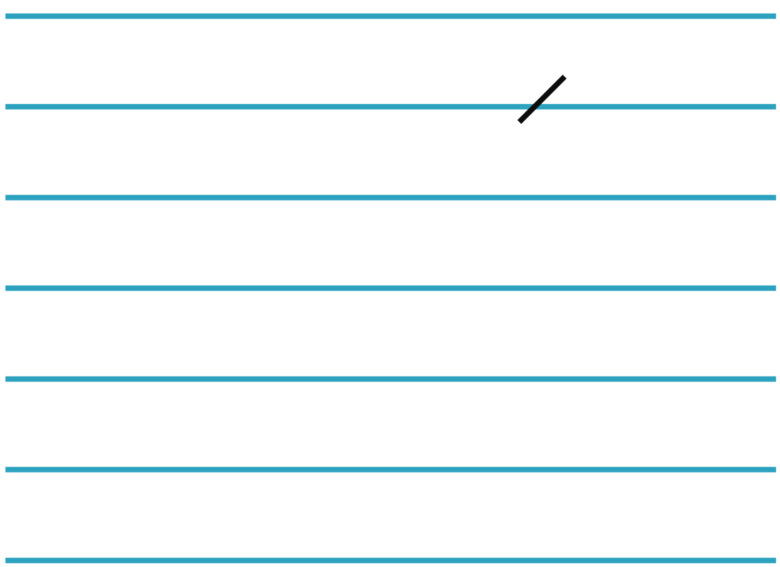
1. Acul lui Buffon

Se consideră o mulţime de linii paralele astfel încât oricare două linii vecine sunt la distanţă de o unitate.

Aproximarea lui π

1. Acul lui Buffon

Se consideră o mulţime de linii paralele astfel încât oricare două linii vecine sunt la distanţă de o unitate. Un ac de lungime o jumătate de unitate este aruncat aleator şi se numără de câte ori a intersectat vreo linie



Aproximarea lui π

1. Acul lui Buffon

- Probabilitatea ca acul să intersecteze o linie este $1/\pi$

Aproximarea lui π

1. Acul lui Buffon

- **Probabilitatea ca acul să intersecteze o linie este $1/\pi$**
- După un număr "suficient de mare" de încercări, raportul între:

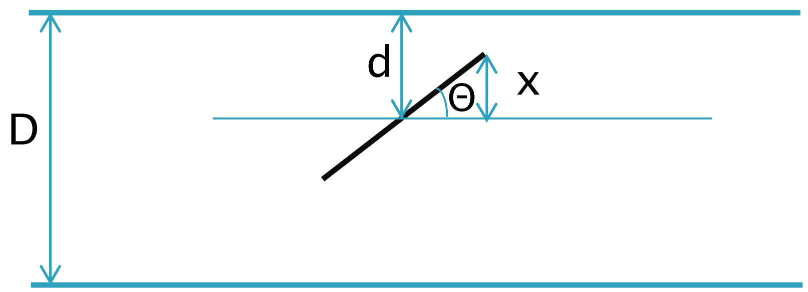
  - **numărul total de încercări**
  - **numărul cazurilor în care acul a intersectat vreo linie**

va fi "suficient de aproape" de π.

Aproximarea lui π

1. Acul lui Buffon

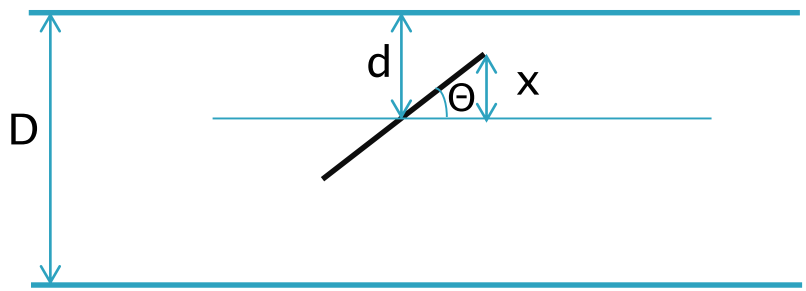
- **Justificare:**
  - L – lungimea acului (exp L=1/2)
  - D – distanța dintre drepte (L<D, exp D=1)
  - Acul – unic identificat de perechea (Θ,d), unde



Aproximarea lui π

1. Acul lui Buffon

- **Justificare:**
  - L – lungimea acului (exp L=1/2)
  - D – distanța dintre drepte (L<D, exp D=1)
  - Acul – unic identificat de perechea (Θ,d), unde
    - d = distanța de la centrul acului la cea mai apropiată dreaptă (linie) din mulțime
    - Θ = unghiul format de ac cu direcția dreptelor paralele

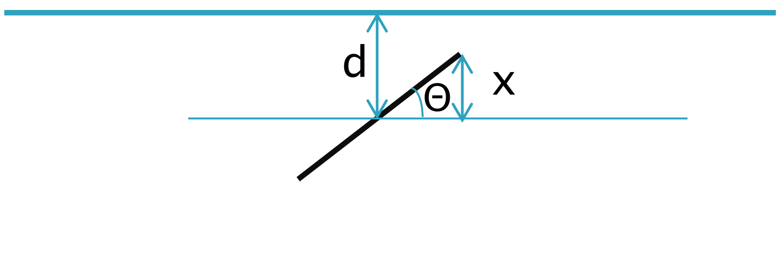


Aproximarea lui π

1. Acul lui Buffon

- **Justificare:**
  - Acul – unic identificat de perechea (Θ,d), unde
    - d $\in$[0, D/2]
    - Θ $\in$[0, π]
  - **“Aruncare ac” $\Leftrightarrow$ generare pereche (Θ,d)** / (sin(Θ),d)
  - Acul intersectează dreapta cea mai apropiată $\Leftrightarrow$

    d ≤ x=L/2 sin(Θ)



Aproximarea lui π

1. Acul lui Buffon

- **Justificare:**
  - Poziții **posibile ac**:
    - $T = \{(\Theta, d) \mid d \in [0, D/2], \Theta \in [0, \pi]\}$

Aproximarea lui π

1. Acul lui Buffon

- **Justificare:**
  - Poziții **posibile ac**:
    - $T = \{(\Theta,d) \mid d \in [0, D/2], \Theta \in [0, \pi] \}$
  - Poziții **ac – care intersectează dreaptă**:
    - $F = \{(\Theta,d) \mid \Theta \in [0, \pi], 0 \leq d \leq L/2 \sin(\Theta) \}$

Aproximarea lui π

1. Acul lui Buffon

- **Justificare:**
  - Poziții **posibile ac**:
    - $T = \{(\Theta,d) \mid d \in [0, D/2], \Theta \in [0, \pi]\}$
  - Poziții **ac – care intersectează dreaptă**:
    - $F = \{(\Theta,d) \mid \Theta \in [0, \pi], 0 \le d \le L/2 \sin(\Theta)\}$
  - Probabilitatea ca acul să intersecteze dreapta:

$$\frac{arie(F)}{arie(T)} =$$

Aproximarea lui π

1. Acul lui Buffon

- **Justificare:**
  - Poziții **posibile ac**:
    - $T = \{(\Theta, d) \mid d \in [0, D/2], \Theta \in [0, \pi]\}$
  - Poziții **ac – care intersectează dreaptă**:
    - $F = \{(\Theta, d) \mid \Theta \in [0, \pi], 0 \le d \le L/2 \sin(\Theta)\}$
  - Probabilitatea ca acul să intersecteze dreapta:

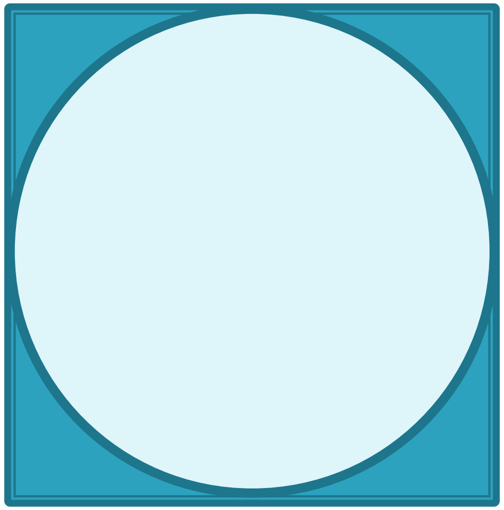
$$\frac{arie(F)}{arie(T)} = \frac{\left|\int_0^{\pi} \frac{L}{2} \sin(\Theta)\, d\Theta\right|}{\pi D/2} = \frac{2L/2}{\pi D/2} = \frac{2L}{\pi D}$$

Pentru L=1/2 și D=1 probabilitatea este $\frac{1}{\pi}$

Aproximarea lui π

2. Se aruncă repetat cu o săgeată într-un panou pătrat, cu ținta un cerc înscris în pătrat.

Se presupune că săgeata nimereşte totdeauna panoul.



Aproximarea lui π

2. Se aruncă repetat cu o săgeată într-un panou pătrat, cu ținta un cerc înscris în pătrat.

Se presupune că săgeata nimereşte totdeauna panoul.

Atunci raportul dintre:

- numărul cazurilor în care săgeata nimereşte în cercul înscris în pătrat

- numărul total de încercări

tinde la

$$\frac{arie\ cerc}{arie\ patrat} = \frac{\pi r^2}{(2r)^2} = \frac{\pi}{4}$$

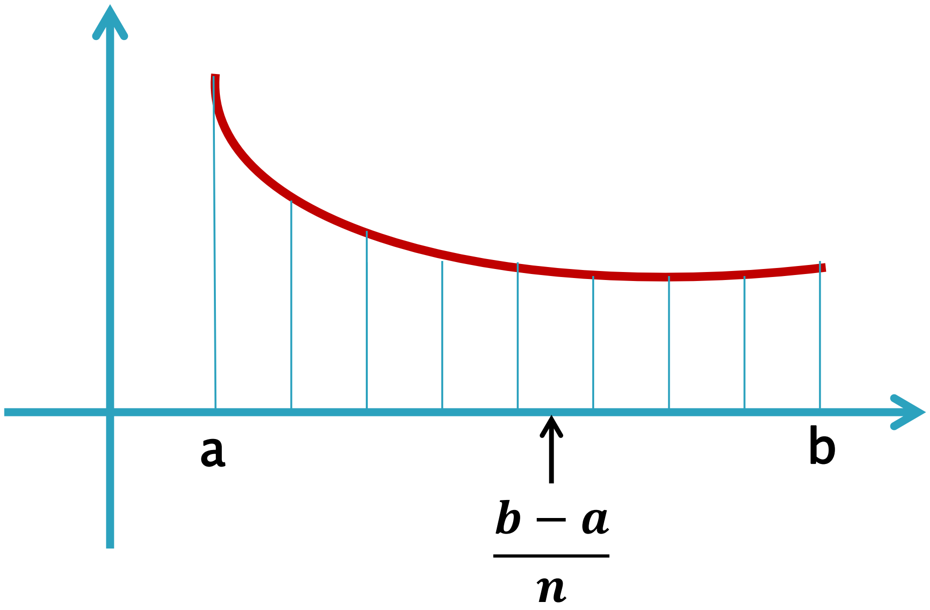
Aproximarea integralei

$$\int_a^b f(x)\,dx, \quad f:[a,b]\rightarrow[c,d]$$

Aproximarea integralei

$$\int_a^b f(x)\,dx, \qquad f:[a,b]\to[c,d]$$

n încercări

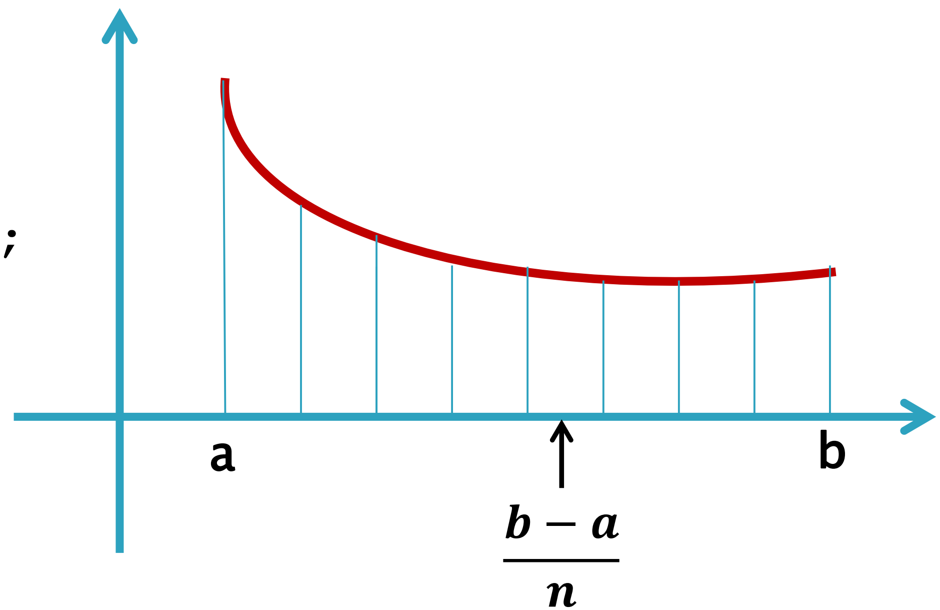


Aproximarea integralei

$$\int_a^b f(x)\,dx, \quad f:[a,b]\to[c,d]$$

```
s ← 0
for i=1,n
    x ← random([a,b]);
    s ← s+f(x)

s ← s·(b-a)/n
write(s)
```



Algoritmi de aproximare

Algoritmi euristici Greedy

Algoritmi de aproximare

- Soluții suboptimale – timp polinomial
- **Cât de aproape este de soluția optimă? (garantat)**
  - **Algoritm de α-aproximare**

    Pentru orice instanță I a problemei, dacă OPT(I) este soluția optimă și SOL(I) este soluția găsită de algoritm, avem

  - pentru problemă de **minimizare**:

    - **$OPT(I) \leq SOL(I) \leq \alpha \cdot OPT(I) \quad (\alpha > 1)$**

Algoritmi de aproximare

- Soluții suboptimale – timp polinomial
- **Cât de aproape este de soluția optimă? (garantat)**
  - **Algoritm de α–aproximare**

    Pentru orice instanță I a problemei, dacă OPT(I) este soluția optimă și SOL(I) este soluția găsită de algoritm, avem

  - pentru problemă de **minimizare**:
    - **$OPT(I) \leq SOL(I) \leq \alpha \cdot OPT(I) \quad (\alpha > 1)$**
  - pentru problemă de **maximizare**:
    - **$\alpha \cdot OPT(I) \leq SOL(I) \leq OPT(I) \quad (\alpha < 1)$**

Algoritmi de aproximare

- **Problema rucsacului – varianta discretă (0-1 Knapsack)**

  $g_i \le G, \; g_1 + ... + g_n > G$

  - **Varianta continuă (fracționară)** – algoritm Greedy corect

    Amintim – obiectele ordonate descrescător după câștigul la unitatea de greutate:

    $$\frac{c_1}{g_1} \ge \frac{c_2}{g_2} \ge ... \ge \frac{c_n}{g_n}$$

```
Gr ← G { Gr reprezintă greutatea disponibilă }
for i=1,n
      if gi ≤ Gr then xi←1; Gr←Gr-gi
      else xi ← Gr/gi;
            k ← i-1 {ultimul obiect incarcat nefractionat}
           for j = i+1,n
                  xj ← 0
           stop
```

$$\text{OPT\_FRACT} = \sum_{j=1}^{n} x_j c_j$$

Algoritmi de aproximare

- **Problema rucsacului – varianta discretă (0-1 Knapsack)**

  $g_i \le G, \; g_1 + ... + g_n > G$

  - **Varianta continuă** – algoritm Greedy corect

  Observații

    - **k este cel mai mare indice cu proprietatea** $g_1 + \ldots + g_k \le G$
    - $$x_{k+1} = \frac{G - (g_1 + ... + g_k)}{g_{k+1}}$$
    - $\Rightarrow$ **soluția optimă pentru problema continuă (fracționară) este**

$$\mathrm{OPT\_FRACT} = \sum_{j=1}^{k} c_j + \frac{G - (g_1 + ... + g_k)}{g_{k+1}} c_{k+1}$$

Algoritmi de aproximare

- **Problema rucsacului – varianta discretă** (0-1 Knapsack)
  - **Pentru varianta discretă** – o idee similară (ne oprim dacă $g_i > G_r$, nu mai fracționăm obiectul) – nu furnizează soluția optimă

Greedy_Aprox_1

```
      Gr ← G { Gr reprezintă greutatea disponibilă }
      for i=1,n
         if gi ≤ Gr then xi←1; Gr←Gr-gi
         else k ← i-1 {ultimul obiect incarcat}
              for j=i+1,n  xj ← 0
              stop
```

$$SOL_1 = \sum_{j=1}^{k} c_j = \sum_{j=1}^{k} x_j c_j = \sum_{j=1}^{n} x_j c_j$$

Cât de mare poate fi raportul între SOL_1 și soluția optima OPT?

Algoritmi de aproximare

- **Problema rucsacului – varianta discretă** (0–1 Knapsack)

```
Greedy_Aprox_2
        Gr ← G { Gr reprezintă greutatea disponibilă }
        for i=1,n
            if gi ≤ Gr then xi←1; Gr←Gr-gi
            else k ← i-1 {ultimul obiect incarcat}
                  for j=i+1,n  xj ← 0
                  stop
```

$$SOL = \max\left(\sum_{j=1}^{k} c_j,\ c_{k+1}\right)$$

Cât de mare poate fi raportul între SOL și soluția optima OPT?

Algoritmi de aproximare

- **Problema rucsacului – varianta discretă** (0-1 Knapsack)

`Greedy_Aprox_2`

Cât de mare poate fi raportul între SOL și soluția optima OPT pentru o instanță I?

Avem

$$OPT\_FRACT \geq OPT$$

Algoritmi de aproximare

- **Problema rucsacului – varianta discretă (0-1 Knapsack)**

`Greedy_Aprox_2`

Cât de mare poate fi raportul între SOL și soluția optima OPT pentru o instanță I?

Avem

$$OPT\_FRACT \geq OPT$$

O soluție pentru varianta discretă este soluție și pentru varianta continuă

Algoritmi de aproximare

- **Problema rucsacului – varianta discretă** (0-1 Knapsack)

`Greedy_Aprox_2`

Cât de mare poate fi raportul între SOL și soluția optima OPT pentru o instanță I?

Avem

$$SOL = \max\left(\sum_{j=1}^{k} c_j,\ c_{k+1}\right)$$

$$2 \cdot SOL \geq \sum_{j=1}^{k} c_j + c_{k+1} \geq$$

$$\geq$$

Algoritmi de aproximare

- **Problema rucsacului – varianta discretă (0-1 Knapsack)**

`Greedy_Aprox_2`

Cât de mare poate fi raportul între SOL și soluția optima OPT pentru o instanță I?

Avem

$$SOL = \max\left(\sum_{j=1}^{k} c_j,\ c_{k+1}\right)$$

$$2 \cdot SOL \geq \sum_{j=1}^{k} c_j + c_{k+1} \geq$$

$$\geq \sum_{j=1}^{k} c_j + \frac{G-(g_1+\ldots+g_k)}{g_{k+1}} c_{k+1} = \text{OPT\_FRACT} \geq \text{OPT}$$

Algoritmi de aproximare

- **Problema rucsacului – varianta discretă** (0-1 Knapsack)

`Greedy_Aprox_2`

Cât de mare poate fi raportul între SOL și soluția optima OPT pentru o instanță I?

Rezultă

$$\frac{1}{2}OPT \leq SOL \leq OPT$$

$$(\alpha = \frac{1}{2})$$

Algoritmi de aproximare

- **Problema rucsacului – varianta discretă (0-1 Knapsack)**

`Greedy_Aprox_2`

Exemplu în care factorul se apropie de 2:

```
G = 100
n = 4 obiecte
g:  51  50  50  51
c:  52  50  50  50
```

SOL = Câștigul **`Greedy_Aprox_2`** = **52**

OPT = Câștigul optim = **100**

Algoritmi de aproximare

- **Exemplu – Problema rucsacului – varianta discretă (0-1 Knapsack)**
  - https://www.coursera.org/ – Algorithms, Stanford University

Algoritmi de aproximare

- **O problemă de planificare echilibrată**

m resurse $R_1,\ldots, R_m$

n activități – cu duratele $t_1,\ldots,t_n$

Se cere o distribuire echilibrată a celor n activități pe cele m resurse, mai exact astfel încât timpul total de funcționare a unei resurse să fie cât mai mic (sa se minimizeze timpul maxim de funcționare al unei resurse).

Exemplu:

m=3

n=5: 2, 4, 3, 1, 3

O soluție: {2, 3}, {4}, {3, 1}

- Jon Kleinberg and Éva Tardos – Algorithm Design

Algoritmi de aproximare

O problemă de planificare echilibrată

- Pentru o repartizare a activităților pe resurse notăm
  - **A(i)** = mulțimea de activități asociate resursei i
  - **T_i** = durata de lucru a resursei R_i pentru a executa activitățile din A(i)

$$T_i = \sum_{a \in A(i)} t_a$$

- Trebuie **minimizat**

$$T = \max\{T_i \mid i = 1, \ldots, m\}$$

- NP-dificilă

Algoritmi de aproximare

O problemă de planificare echilibrată

- Notam cu **OPT** – soluția optima

  = valoarea minimă care se poate obține pentru $\max_{k=1,n} (T_k)$

- Pentru a putea decide cât de aproape este soluția dată de un algoritm de OPT – utile **limite inferioare pentru OPT**

Algoritmi de aproximare

O problemă de planificare echilibrată

- Notam cu **OPT** – soluția optima

= valoarea minimă care se poate obține pentru $\max_{k=1,n} (T_k)$

- Pentru a putea decide cât de aproape este soluția dată de un algoritm de OPT – utile **limite inferioare pentru OPT**

$$(1) \quad OPT \geq \max\{t_a \mid a = 1, \ldots, n\}$$

$$(2) \quad OPT \geq \frac{1}{m} \sum_{a=1}^{n} t_a$$

Algoritmi de aproximare

O problemă de planificare echilibrată

- Un prim algoritm Greedy
  - Alocăm activitatea curentă resursei cu durata de lucru T_i cea mai mică

Exemplu: m=3

n=5: 2, 4, 3, 1, 3

2,	4,	3,	1,	3
R_1	R_2	R_3	R_1	R_1
$T_1=2$	$T_2=4$	$T_3=3$	$T_1=3$	$T_1=6$

$\Longrightarrow$

R_1: {2,1,3}

R_2: {4}

R_3: {3}

T= 6

Algoritmi de aproximare

O problemă de planificare echilibrată

- Un prim algoritm Greedy
  - Alocăm activitatea curentă resursei cu durata de lucru T_i cea mai mică

```
pentru i = 1,m
    T_i = 0, A(i) = Ø

pentru a = 1,n
    fie R_i resursa cu durata T_i minima
                  (pentru care se atinge min_k T_k la acest pas)
    A(i) = A(i) ∪ {a}
    T_i = T_i + t_a

SOL = max_i T_i
```

Exemplu: $m=3$

$n=5$: 2, 4, 3, 1, 3

2,	4,	3,	1,	3
R_1	R_2	R_3	R_1	R_1
$T_1=2$	$T_2=4$	$T_3=3$	$T_1=3$	$T_1=6$

$\Rightarrow$

R_1: {2,1,3}

R_2: {4}

R_3: {3}

$T= 6$

Algoritmi de aproximare

O problemă de planificare echilibrată

- **Un prim algoritm Greedy** – arătam SOL $\leq$ 2OPT
  - Fie R_i resursa pentru care se atinge $\mathbf{SOL = max_k T_k}$
  - Fie a ultima activitate planificată pentru R_i

Algoritmi de aproximare

O problemă de planificare echilibrată

- **Un prim algoritm Greedy** – arătam SOL $\leq$ 2OPT
  - Fie R_i resursa pentru care se atinge **`SOL = max`**$_k$**`T`**$_k$
  - Fie a ultima activitate planificată pentru R_i

    – înainte să fie asignata activitatea a resursei R_i, aceasta avea timpul minim (conform alegerii algoritmului)

$$\Rightarrow T_k \geq T_i - t_a, \forall i \neq k$$

Algoritmi de aproximare

O problemă de planificare echilibrată

- **Un prim algoritm Greedy** – arătam SOL $\leq$ 2OPT
  - Fie R_i resursa pentru care se atinge **SOL = max$_k$T$_k$**
  - Fie a ultima activitate planificată pentru R_i

    – înainte să fie asignata activitatea a resursei R_i, aceasta avea timpul minim (conform alegerii algoritmului)

$$\Rightarrow T_k \geq T_i - t_a, \forall i \neq k$$

$$\Rightarrow \sum_{k=1}^{m} T_k \geq m(T_i - t_a)$$

Algoritmi de aproximare

O problemă de planificare echilibrată

- **Un prim algoritm Greedy** – arătam SOL ≤ 2OPT
  - Fie R_i resursa pentru care se atinge `SOL = max`$_k$`T`$_k$
  - Fie a ultima activitate planificată pentru R_i

– înainte să fie asignata activitatea a resursei R_i, aceasta avea timpul minim (conform alegerii algoritmului)

$$\Rightarrow T_k \geq T_i - t_a, \forall i \neq k$$

$$\Rightarrow \sum_{k=1}^{m} T_k \geq m(T_i - t_a)$$

$$\Rightarrow T_i - t_a \leq \frac{1}{m}\sum_{k=1}^{m} T_k = \frac{1}{m}\sum_{j=1}^{n} t_j \leq OPT$$

Algoritmi de aproximare

O problemă de planificare echilibrată

- **Un prim algoritm Greedy** – arătam SOL ≤ 2OPT
  - Fie R_i resursa pentru care se atinge **`SOL = max`**$_k$**`T`**$_k$
  - Fie a ultima activitate planificată pentru R_i

– înainte să fie asignata activitatea a resursei R_i, aceasta avea timpul minim (conform alegerii algoritmului)

$$\Rightarrow T_k \geq T_i - t_a, \forall i \neq k$$

$$\Rightarrow \sum_{k=1}^{m} T_k \geq m(T_i - t_a)$$

$$\Rightarrow T_i - t_a \leq \frac{1}{m}\sum_{k=1}^{m} T_k = \frac{1}{m}\sum_{j=1}^{n} t_j \leq OPT$$

$$\Rightarrow \quad SOL = T_i = (T_i - t_a) + t_a \leq OPT + OPT \leq 2 \cdot OPT$$

Algoritmi de aproximare

O problemă de planificare echilibrată

- **Un prim algoritm Greedy**
  - algoritm de 2-aproximare

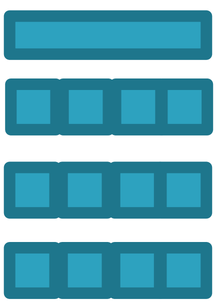
Algoritmi de aproximare

O problemă de planificare echilibrată

- **Un prim algoritm Greedy**
  - **Exemplu** – factor aproape de 2
    - m
    - n = m(m–1)+1

    durate: t = (1, 1, 1, 1, 1, 1, m)

SOL = m – 1 + m = 2m–1



OPT = m

Algoritmi de aproximare

O problemă de planificare echilibrată

- **Un prim algoritm Greedy**
  - **Exemplu** – factor aproape de 2
    - m
    - n = m(m−1)+1

   durate: t = (1, 1, 1, 1, 1, 1, m)

SOL = m – 1 + m = 2m−1 **OPT = m**

Considerăm activitățile în ordine descrescătoare după durată

Algoritmi de aproximare

O problemă de planificare echilibrată

- Al doilea algoritm Greedy

Ordonăm activitățile descrescător după t și le renotăm astfel încât $t_1 \geq ... \geq t_n$

```
pentru i = 1,m
    T_i = 0, A(i) = Ø
pentru a = 1,n
    fie R_i resursa cu durata T_i minima
                (pentru care se atinge min_k T_k la acest pas)
    A(i) = A(i) ∪ {a}
    T_i = T_i + t_a
SOL = max_i T_i
```

Algoritmi de aproximare

O problemă de planificare echilibrată

- **Al doilea algoritm Greedy :** SOL $\leq$ 3/2 OPT

Algoritmi de aproximare

O problemă de planificare echilibrată

- **Al doilea algoritm Greedy :** SOL $\leq$ 3/2 OPT
  - Dacă $m \geq n \Rightarrow$ SOL = OPT
  - **Presupunem $m < n$**

Algoritmi de aproximare

O problemă de planificare echilibrată

- **Al doilea algoritm Greedy :** SOL $\leq$ 3/2 OPT
  - Dacă $m \geq n \Rightarrow$ SOL = OPT
  - **Presupunem m < n**

    $\Rightarrow$ există o resursă care are alocate cel puțin două joburi din primele m+1 și $t_1 \geq \ldots \geq t_{m+1}$

    $\Rightarrow$ OPT $\geq 2t_{m+1}$

Algoritmi de aproximare

O problemă de planificare echilibrată

- **Al doilea algoritm Greedy :** SOL $\leq$ 3/2 OPT
  - Dacă $m \geq n \Rightarrow$ SOL = OPT
  - **Presupunem m < n**

    $\Rightarrow$ există o resursă care are alocate cel puțin două joburi din primele m+1 și $t_1 \geq \ldots \geq t_{m+1}$

    $\Rightarrow$ OPT $\geq 2t_{m+1}$

    - Fie R_i resursa pentru care se atinge **SOL = $\max_k T_k$**

      – dacă R_i are asociată o singură resursă $\Rightarrow$ SOL = OPT

Algoritmi de aproximare

O problemă de planificare echilibrată

- **Al doilea algoritm Greedy :** SOL $\leq$ 3/2 OPT
  - Dacă $m \geq n \Rightarrow$ SOL = OPT
  - **Presupunem $m < n$**

    $\Rightarrow$ există o resursă care are alocate cel puțin două joburi din primele $m+1$ și $t_1 \geq \ldots \geq t_{m+1}$

    $\Rightarrow$ OPT $\geq 2t_{m+1}$

    - Fie R_i resursa pentru care se atinge $\mathbf{SOL = max_k T_k}$
      - dacă R_i are asociată o singură resursă $\Rightarrow$ SOL = OPT
      - altfel, fie a ultima activitate planificată pentru R_i

Algoritmi de aproximare

O problemă de planificare echilibrată

- **Al doilea algoritm Greedy :** SOL $\leq$ 3/2 OPT
  - Dacă $m \geq n \Rightarrow$ SOL = OPT
  - **Presupunem m < n**

    $\Rightarrow$ există o resursă care are alocate cel puțin două joburi din primele m+1 și $t_1 \geq \ldots \geq t_{m+1}$

    $\Rightarrow$ OPT $\geq 2t_{m+1}$

    - Fie R_i resursa pentru care se atinge **`SOL = max`**$_k$**`T`**$_k$

      – dacă R_i are asociată o singură resursă $\Rightarrow$ SOL = OPT

      – altfel, fie a ultima activitate planificată pentru R_i

$$t_a \leq t_{m+1} \leq \frac{1}{2} \cdot OPT$$

Algoritmi de aproximare

O problemă de planificare echilibrată

- **Al doilea algoritm Greedy :** SOL $\leq$ 3/2 OPT
  - Dacă m $\geq$ n $\Rightarrow$ SOL = OPT
  - **Presupunem m < n**

    $\Rightarrow$ există o resursă care are alocate cel puțin două joburi din primele m+1 și $t_1 \geq \ldots \geq t_{m+1}$

    $\Rightarrow$ OPT $\geq 2t_{m+1}$

    - Fie R_i resursa pentru care se atinge **SOL = $\max_k T_k$**

      – dacă R_i are asociată o singură resursă $\Rightarrow$ SOL = OPT

      – altfel, fie a ultima activitate planificată pentru R_i

$$t_a \leq t_{m+1} \leq \frac{1}{2} \cdot OPT$$

$$\Rightarrow \quad SOL = T_i = (T_i - t_a) + t_a \leq OPT + \frac{1}{2} OPT \leq \frac{3}{2} \cdot OPT$$

Principiul lui Dirichlet

Principiul lui Dirichlet

Dacă $m>k\times n$ obiecte sunt plasate în n căsuţe,

atunci va exista o căsuţă ce va conţine mai mult de k obiecte.

Principiul lui Dirichlet

Se dă vectorul $a = (a_1, \ldots, a_n)$. Să se determine doi indicii $i < j$ astfel încât

$$a_i + \ldots + a_j$$

este multiplu de n

Principiul lui Dirichlet

Considerăm sumele parţiale (**n sume**)

$$s_k = a_1 + \ldots + a_k, \quad k=1,\ldots,n$$

Principiul lui Dirichlet

Considerăm sumele parţiale (**n sume**)

$s_k = a_1 + \ldots + a_k, \quad k=1,\ldots,n$

- Clasele de resturi modulo n:

$\widehat{s_k} \in \{\hat{0}, \hat{1}, \ldots, \widehat{n-1}\}$

Principiul lui Dirichlet

Considerăm sumele parţiale (**n sume**)

$s_k = a_1 + \dots + a_k, \quad k=1,\dots,n$

- Clasele de resturi modulo n:

$$\widehat{s_k} \in \{\hat{0}, \hat{1}, \dots, \widehat{n-1}\}$$

- Avem cazurile:
  - $\widehat{s_k} = \hat{0}$
  - $\widehat{s_k} = \widehat{s_l} \in \{\hat{1}, \dots, \widehat{n-1}\}$, $k < l$

Principiul lui Dirichlet

Considerăm sumele parţiale (**n sume**)

$$s_k = a_1 + \dots + a_k, \quad k=1,\dots,n$$

- Clasele de resturi modulo n:

$$\widehat{s_k} \in \{\hat{0}, \hat{1}, \dots, \widehat{n-1}\}$$

- Avem cazurile:
  - $\widehat{s_k} = \hat{0} \Rightarrow$ $n \mid a_1 + \dots + a_k$
  - $\widehat{s_k} = \widehat{s_l} \in \{\hat{1}, \dots, \widehat{n-1}\}$, $k < l$

Principiul lui Dirichlet

Considerăm sumele parţiale (**n sume**)

$s_k = a_1 + \dots + a_k, \quad k=1,\dots,n$

- Clasele de resturi modulo n:

$$\widehat{s_k} \in \{\hat{0}, \hat{1}, \dots, \widehat{n-1}\}$$

- Avem cazurile:
  - $\widehat{s_k} = \hat{0} \Rightarrow n \mid a_1 + \dots + a_k$
  - $\widehat{s_k} = \widehat{s_l} \in \{\hat{1}, \dots, \widehat{n-1}\}, \; k < l$

    $\Rightarrow \widehat{s_l - s_k} = \hat{0}$

    $\Rightarrow n \mid a_{k+1} + \dots + a_l$

Principiul lui Dirichlet

Se consideră n numere **naturale nenule** a căror sumă este mai mică decât 2n. Să se determine o submulțime de sumă n

Temă – v. prima problemă

Principiul lui Dirichlet

Dat un număr natural n, să se determine un multiplu m al său (nenul) în a cărui scriere în baza 10 apar doar cifrele 0 și 1

Principiul lui Dirichlet

$$s_k = \underbrace{11\ldots1}_{k\,ori},\quad k = 1,\ldots,n$$

Principiul lui Dirichlet

Dat un număr natural **impar n**, să se determine un multiplu m al său în a cărui scriere în baza **2** apare doar cifra 1

Principiul lui Dirichlet

$$s_k = 2^k - 1, \;\; k = 1, \ldots, n$$

Principiul lui Dirichlet

$$s_k = 2^k - 1, \;\; k = 1, \dots, n$$

$$n \,\middle|\, s_l - s_k = 2^k(2^{l-k} - 1) \Rightarrow n \,\middle|\, 2^{l-k} - 1$$

Principiul lui Dirichlet

Se dau $(m-1)(n-1)+1$ numere naturale oarecare.

Să se arate că printre ele există

- m care se divid unul pe altul

sau

- n care nu se divid între ele.

Principiul lui Dirichlet

- Considerăm un caroiaj cu m-1 linii şi n-1 coloane.
- Presupunem că există şi linia imaginară cu numărul 0, pe care este plasat numărul 1.
- Ordonăm crescător numerele date şi apoi le plasăm pe rând în caroiaj;
  - fiecare număr va fi plasat pe linia i cu i maxim având proprietatea că numărul considerat se divide cu un număr aflat pe linia i-1.

Principiul lui Dirichlet

- Exemplu: m=4 şi n=5

```
3, 5, 9, 12, 14, 15, 24, 33, x,...
```

3	5	14	
9	12	15	33
24			

x=72

x=35

Principiul lui Dirichlet

- Principiul lui Dirichlet $\Rightarrow$ cel mai târziu după plasarea ultimului număr vom ieşi din "cutie": aici caroiajul $(m-1)\times(n-1)$
  - pe coloana n => n numere care nu se divid între ele.
  - pe linia m => m numere care se divid între ele.