

# Arbori parțiali de cost minim – Aplicație



# Aplicații – Clustering

Gruparea unor obiecte în  $k$  clase cât mai *bine separate*  
( $k$  dat)

- obiecte din clase diferite *să fie cât mai diferite*

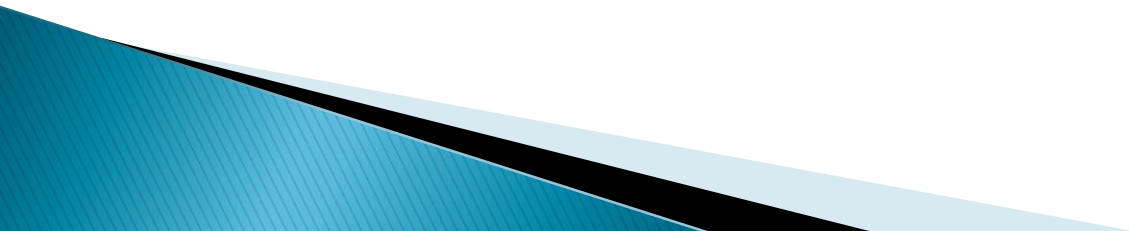
# Aplicații – Clustering

Gruparea unor obiecte în  $k$  clase cât mai *bine separate* ( $k$  dat)

- obiecte din clase diferite *să fie cât mai diferite*

Data – o distanță care măsoară gradul de asemănare între obiecte

Exemplu?



# Aplicații – Clustering

Gruparea unor obiecte în  $k$  clase cât mai *bine separate* ( $k$  dat)

- obiecte din clase diferite *să fie cât mai diferite*

**Data** – o distanță care măsoară gradul de asemănare între obiecte

**Exemplu** – distanța de editare

# Distanțe de editare

**Distanțe de editare** – numărul minim de operații (inserări, modificări, ștergeri etc) de caractere necesar pentru transforma prima secvență în cea de a doua

**Distanța de editare Levenshtein** – sunt permise operații de inserare, modificare și ștergere

**Exemplu:** Distanța de la **care** la **antet** este 4

care  $\xrightarrow{\text{stergem c}}$  are  $\xrightarrow[\text{r} \leftrightarrow \text{n}]{\text{modificăm}}$  ane  $\xrightarrow{\text{inseram t}}$  ante  $\xrightarrow{\text{inseram t}}$  antet

# Aplicații – Clustering

**Exemplu:**  $k=3$ , mulțime de cuvinte:

sinonim, ana, apa, care, martian, este, case, partial, arbore, minim

3 clase



Cum evaluăm cât de “bună” este partiționarea?

# Aplicații – Clustering

**Exemplu:**  $k=3$ , mulțime de cuvinte:

sinonim, ana, apa, care, martian, este, case, partial, arbore, minim

3 clase



**Sunt necesare (se dau):**

- Criteriu de “asemănare” între 2 obiecte  $\Rightarrow$  o distanță
- Măsură a gradului de separare a claselor

# Aplicații – Clustering

## Cadru formal

Se dau:

- ▶ O mulțime de **n obiecte**  $S = \{o_1, \dots, o_n\}$ 
  - cuvinte, imagini, fișiere, specii de animale etc
- ▶ O funcție de **distanță**  $d : S \times S \rightarrow \mathbb{R}_+$ 
  - $d(o_i, o_j) = \text{gradul de asemănare între } o_i \text{ și } o_j$
- ▶  $k$  – un număr natural
  - $k = \text{numărul de clase}$



# Aplicații – Clustering

## Definiții

- ▶ Un **k-clustering** a lui  $S$  = o **partiționare** a lui  $S$  în  $k$  submulțimi nevide (numite **clase** sau **clustere**)

$$\mathcal{C} = (C_1, \dots, C_k)$$

# Aplicații – Clustering

## Definiții

- ▶ Un **k-clustering** a lui  $S$  = o **partiționare** a lui  $S$  în  $k$  submulțimi nevide (numite **clase** sau **clustere**)

$$\mathcal{C} = (C_1, \dots, C_k)$$

- ▶ **Gradul de separare** a lui  $\mathcal{C}$

= distanța minimă dintre două obiecte aflate în clase diferite

= distanța minimă dintre două clase ale lui  $\mathcal{C}$

$\text{sep}(\mathcal{C}) = \min\{d(o, o') \mid o, o' \in S, o \text{ și } o' \text{ sunt în clase diferite ale lui } \mathcal{C}\}$

$$= \min\{d(C_i, C_j) \mid i \neq j \in \{1, \dots, k\}\}$$

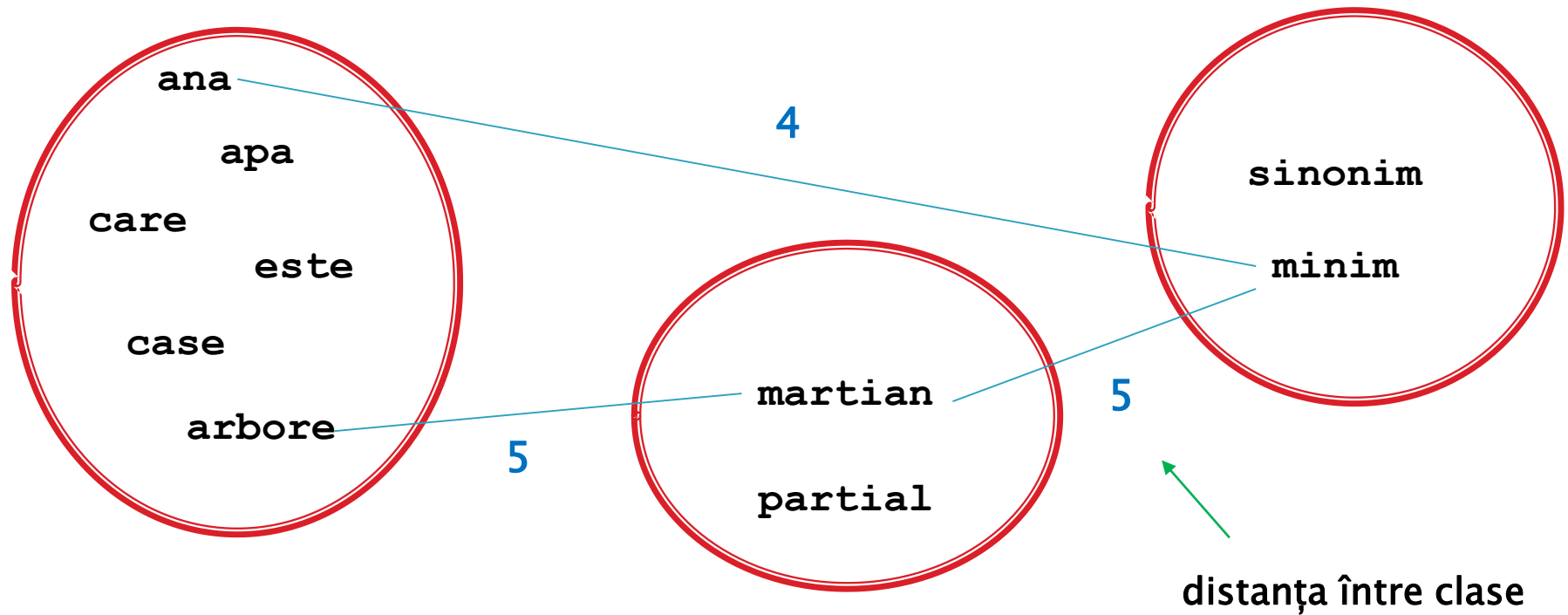
# Aplicații – Clustering

- ▶ obiecte= cuvinte
- ▶  $d$  = distanța de editare     $d(\text{ana}, \text{care}) = 3$ : ana → cana → cara → care
- ▶  $k = 3$

care  
este      martian  
ana      apa      sinonim  
minim      partial  
arbore      case

# Aplicații – Clustering

- ▶ obiecte= cuvinte,
- ▶  $d$  = distanța de editare
- ▶  $k = 3$



3-clustering cu gradul de separare = 4

# Aplicații – Clustering

## Problemă Clustering:

Date  $S$ ,  $d$  și  $k$ , să se determine un  $k$ -clustering cu grad de separare maxim

# Aplicații – Clustering



Idee

este

martian

care

ana

apa

sinonim

minim

partial

case

arbore

# Aplicații – Clustering

## Idee

- Inițial fiecare obiect (cuvânt) formează o clasă
- La un pas determinăm **cele mai asemănătoare (apropiate) două obiecte** aflate în clase diferite (cu distanța cea mai mică între ele) și unim clasele lor

# Aplicații – Clustering

## Idee

- Inițial fiecare obiect (cuvânt) formează o clasă
- La un pas determinăm **cele mai asemănătoare (apropiate) două obiecte** aflate în clase diferite (cu distanța cea mai mică între ele) și unim clasele lor
- Repetăm până obținem  $k$  clase  $\Rightarrow n - k$  pași



# Aplicații – Clustering

## Cuvinte – distanța de editare

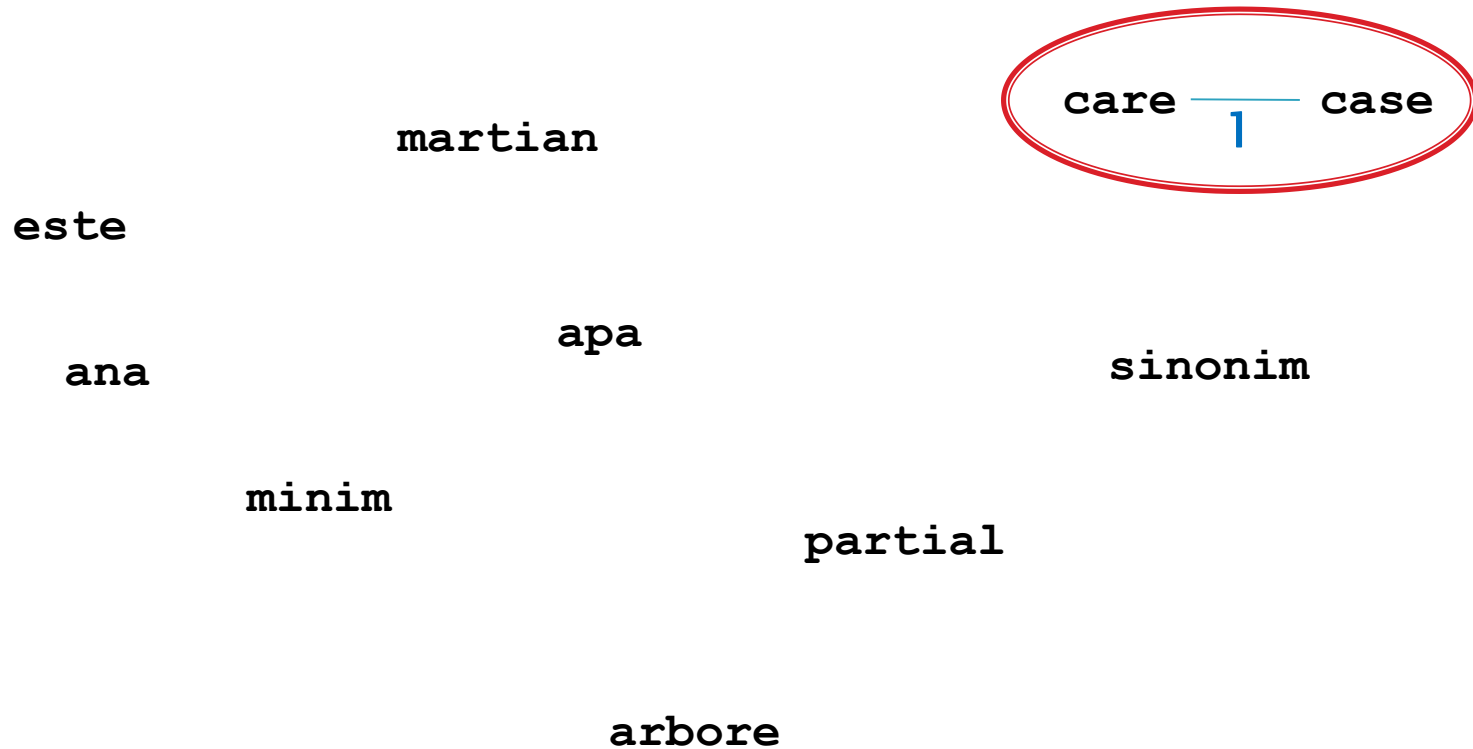


A scatter plot showing the relative positions of 11 words based on their edit distance. The words are distributed across the plot as follows:

- care**: Top right
- martian**: Top center
- este**: Middle left
- ana**: Lower middle left
- apa**: Center
- sinonim**: Middle right
- minim**: Lower middle left, below 'ana'
- partial**: Lower middle right
- arbore**: Bottom center
- case**: Bottom right

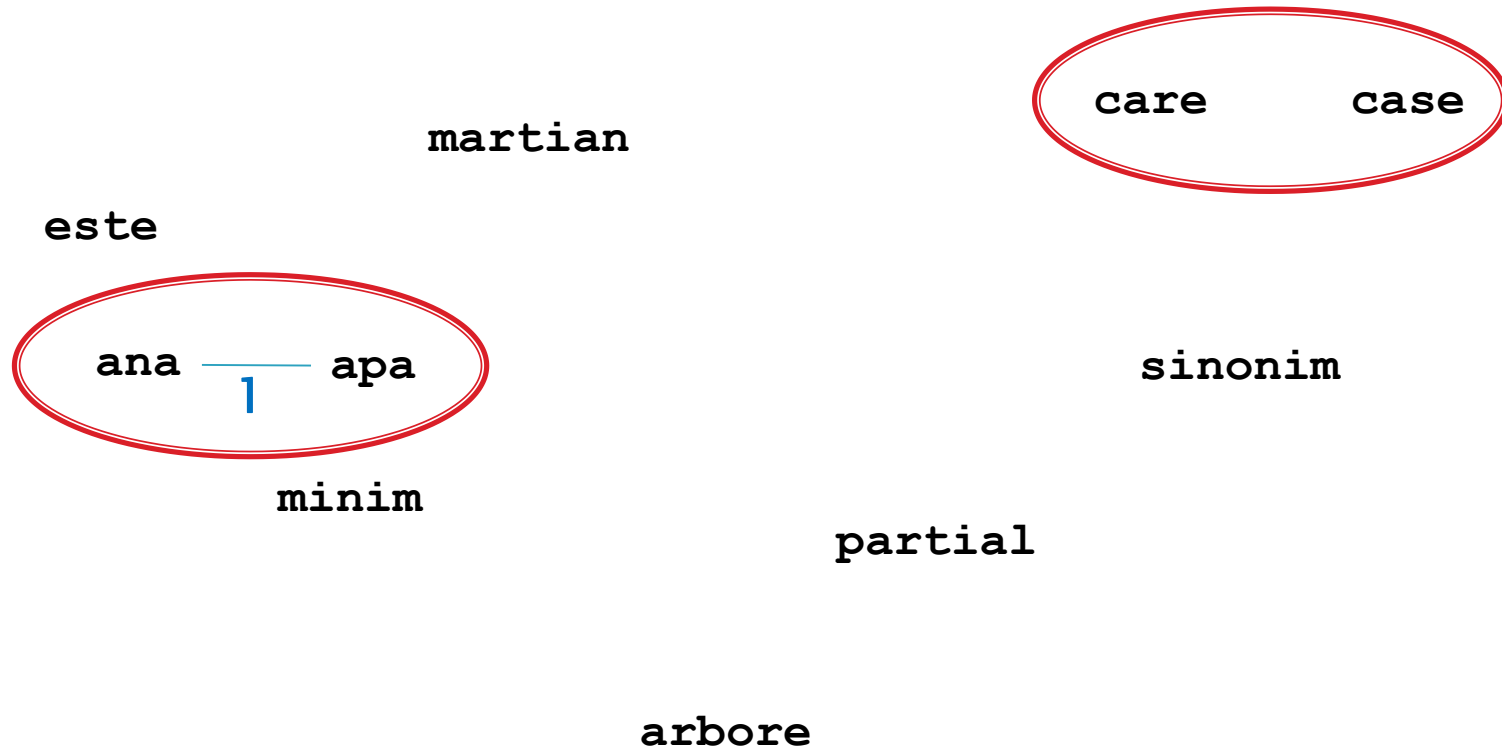
**K = 3 clustere**

# Aplicații – Clustering



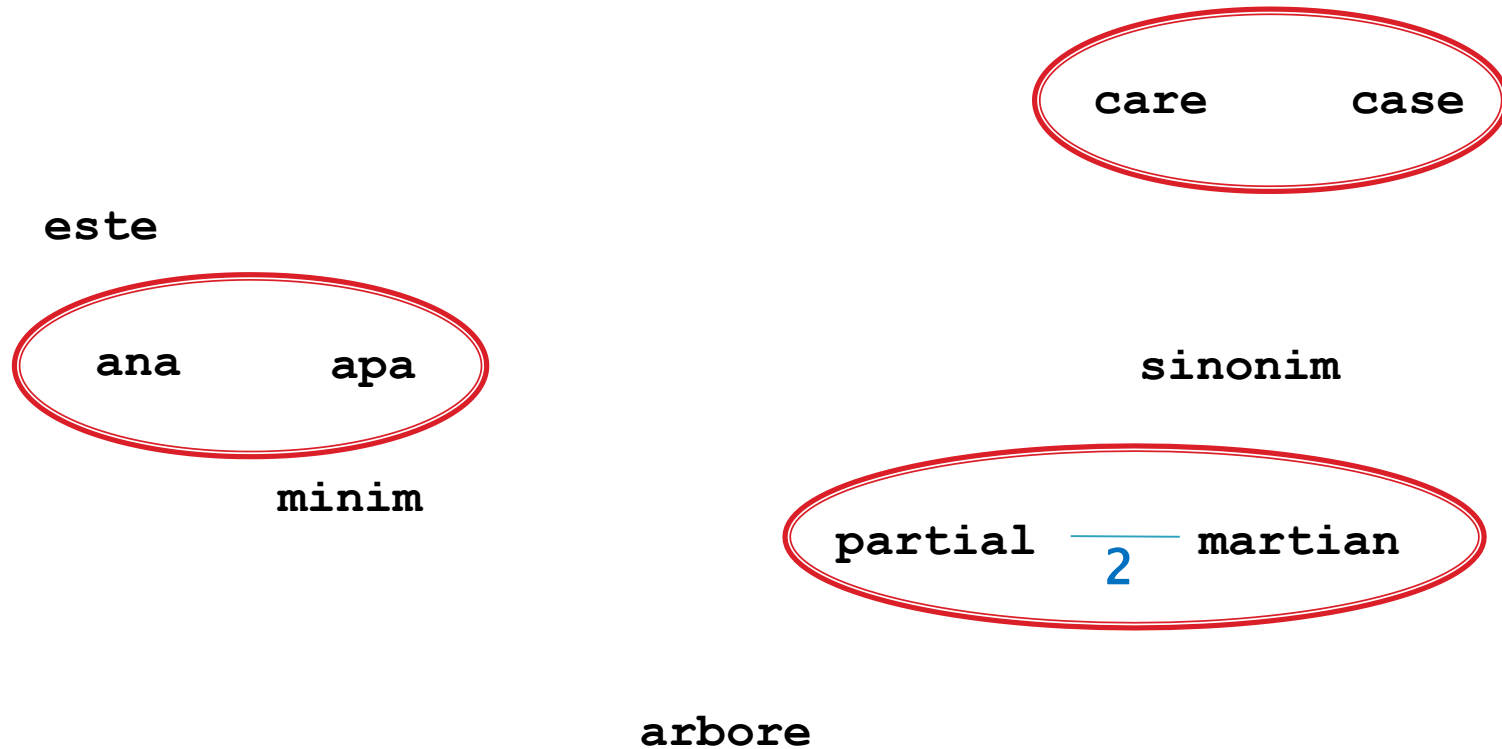
**K = 3 clustere**

# Aplicații – Clustering



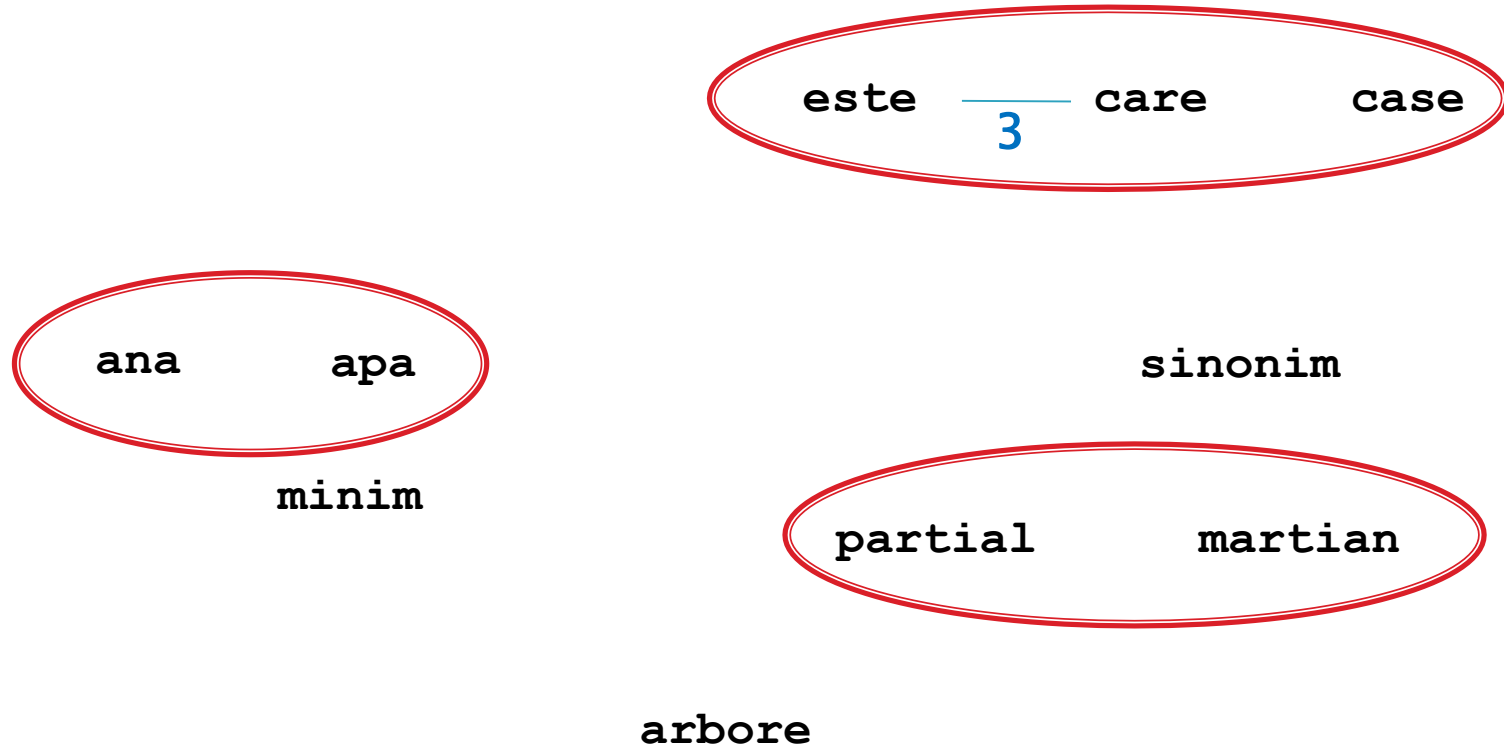
**K = 3 clustere**

# Aplicații – Clustering



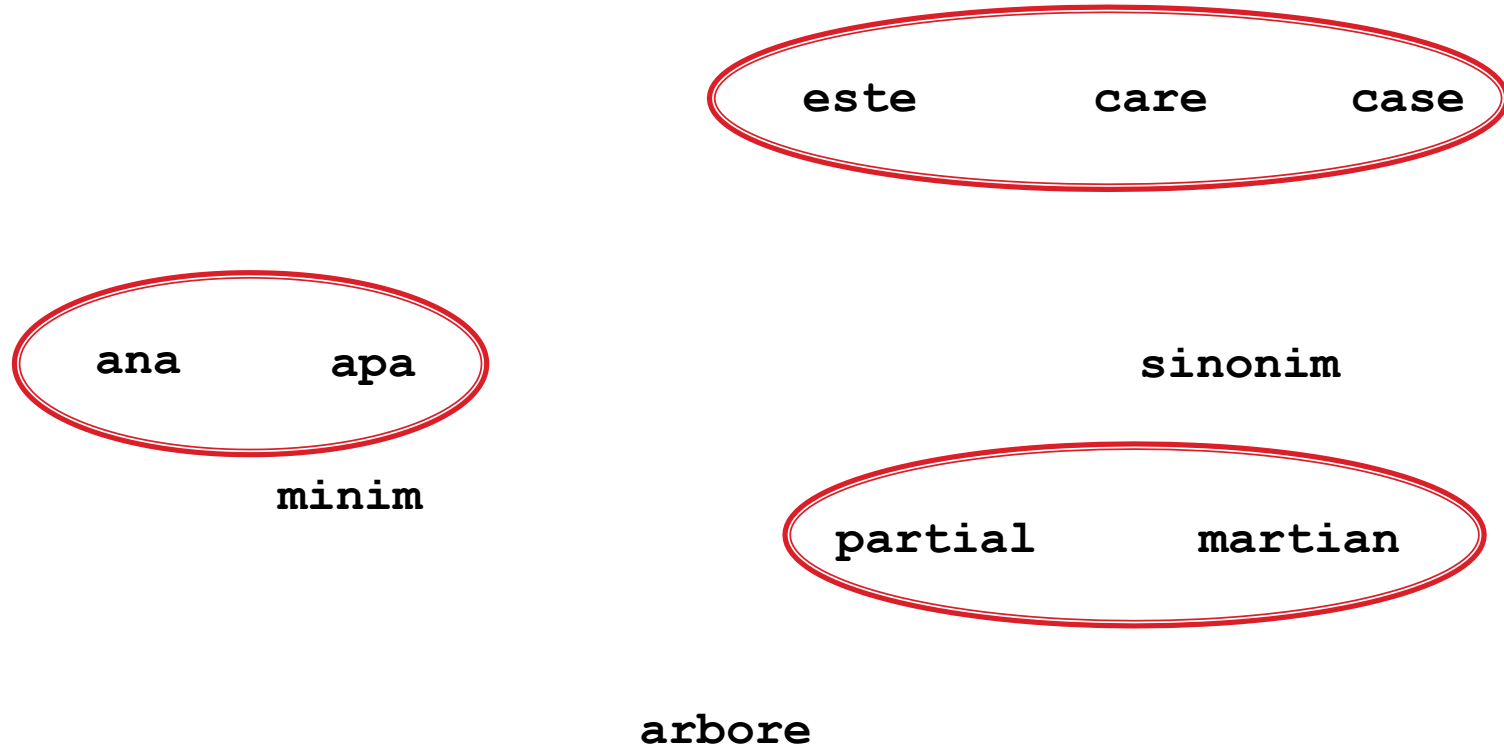
**K = 3 clustere**

# Aplicații – Clustering



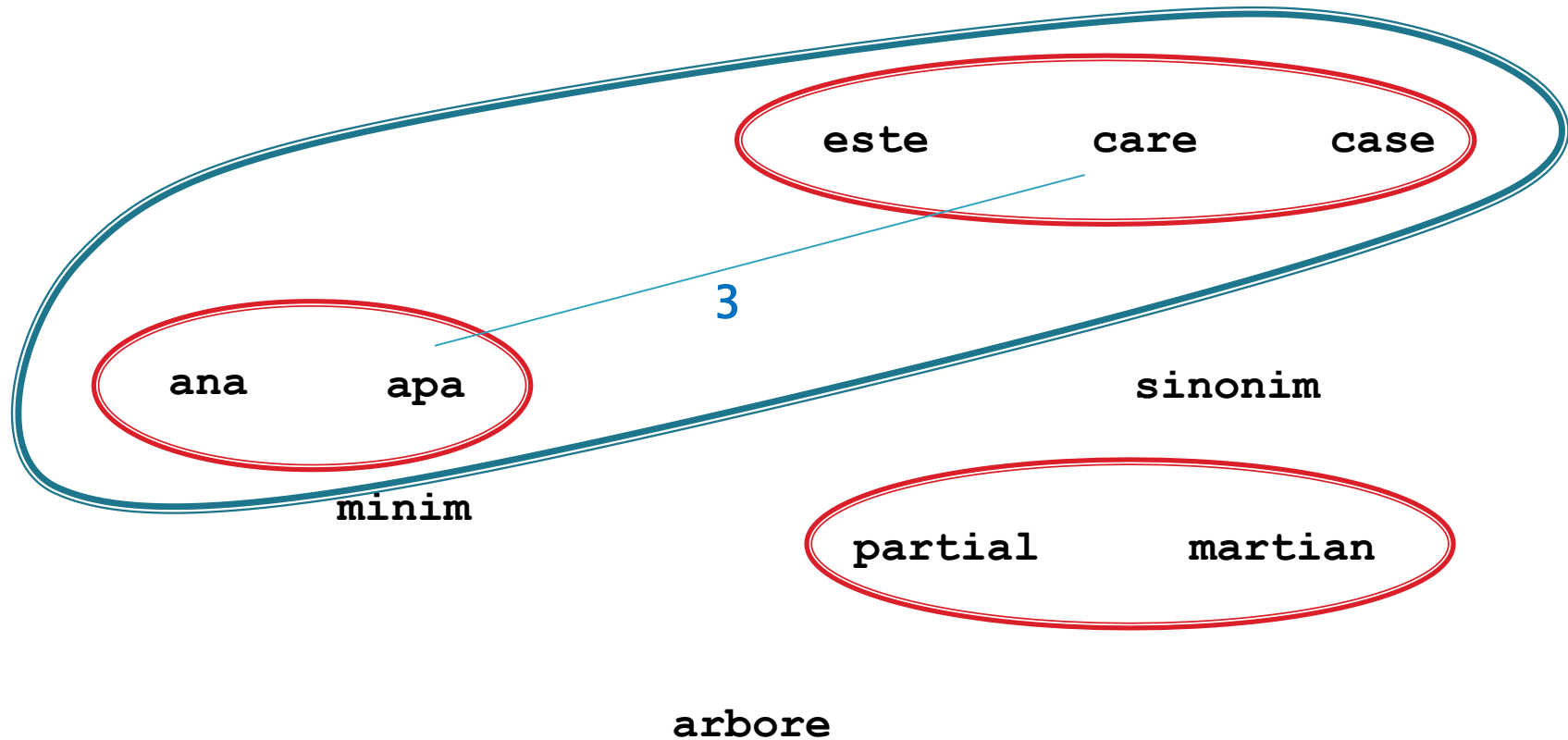
**K = 3** clustere

# Aplicații – Clustering



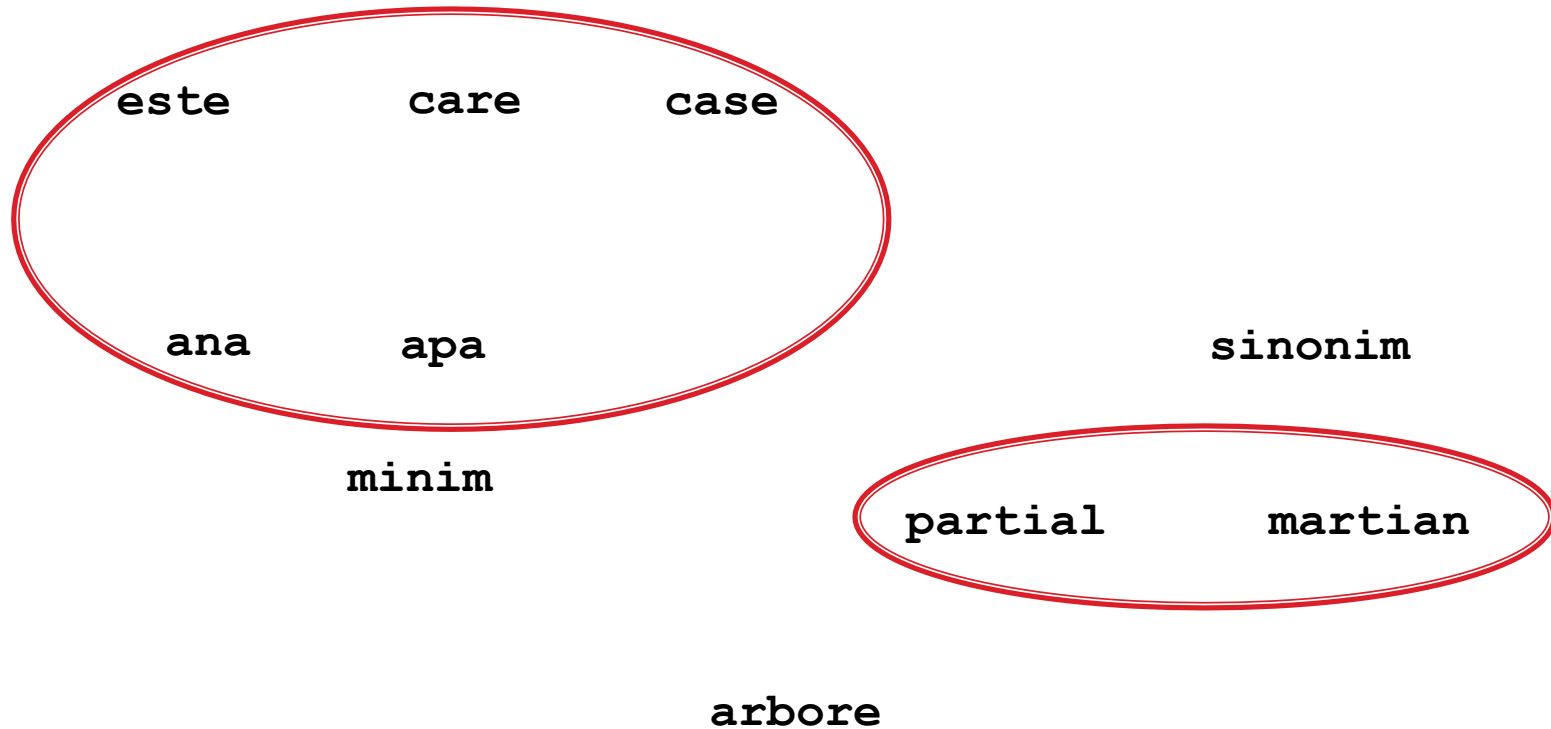
**K = 3 clustere**

# Aplicații – Clustering



**K = 3 clustere**

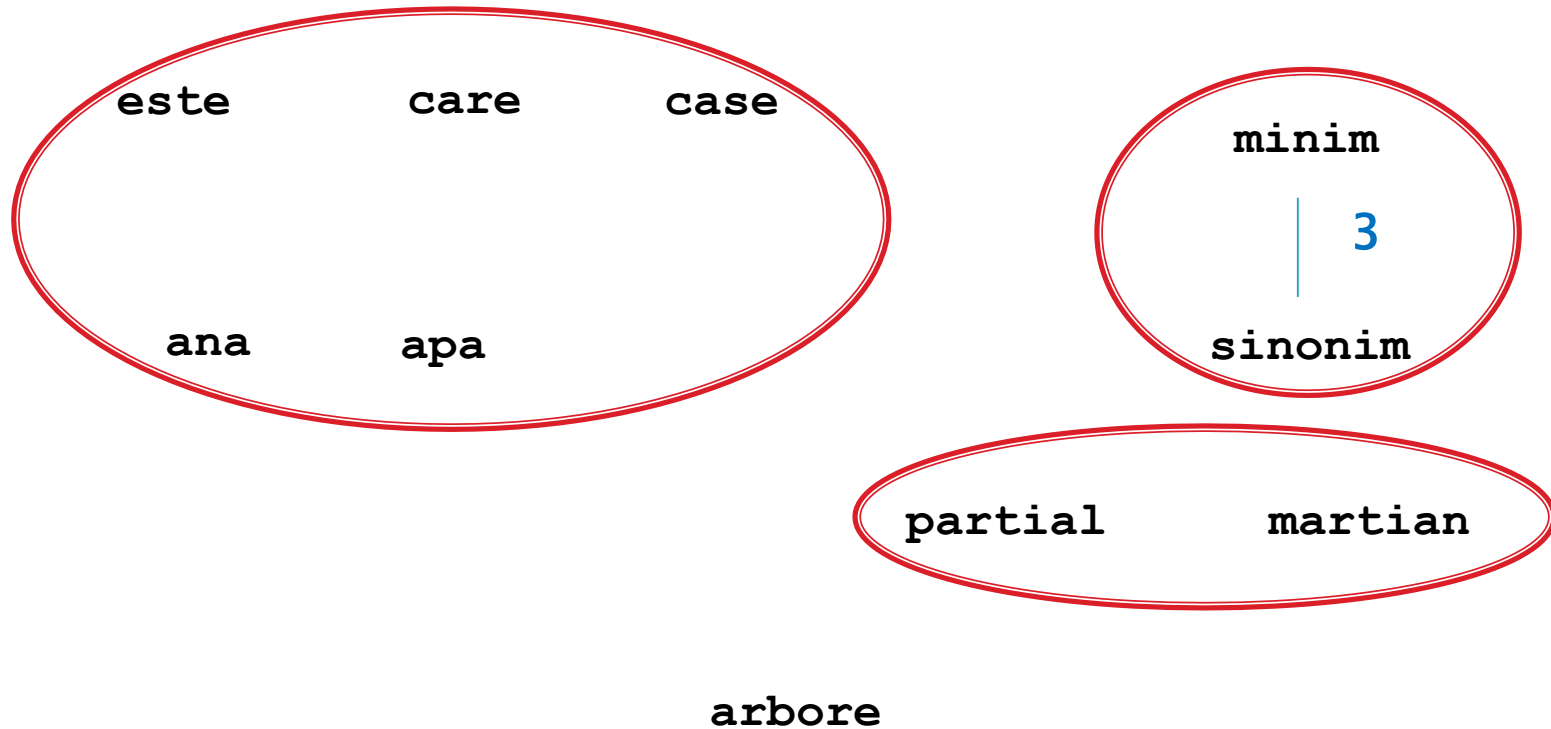
# Aplicații – Clustering



**K = 3 clustere**

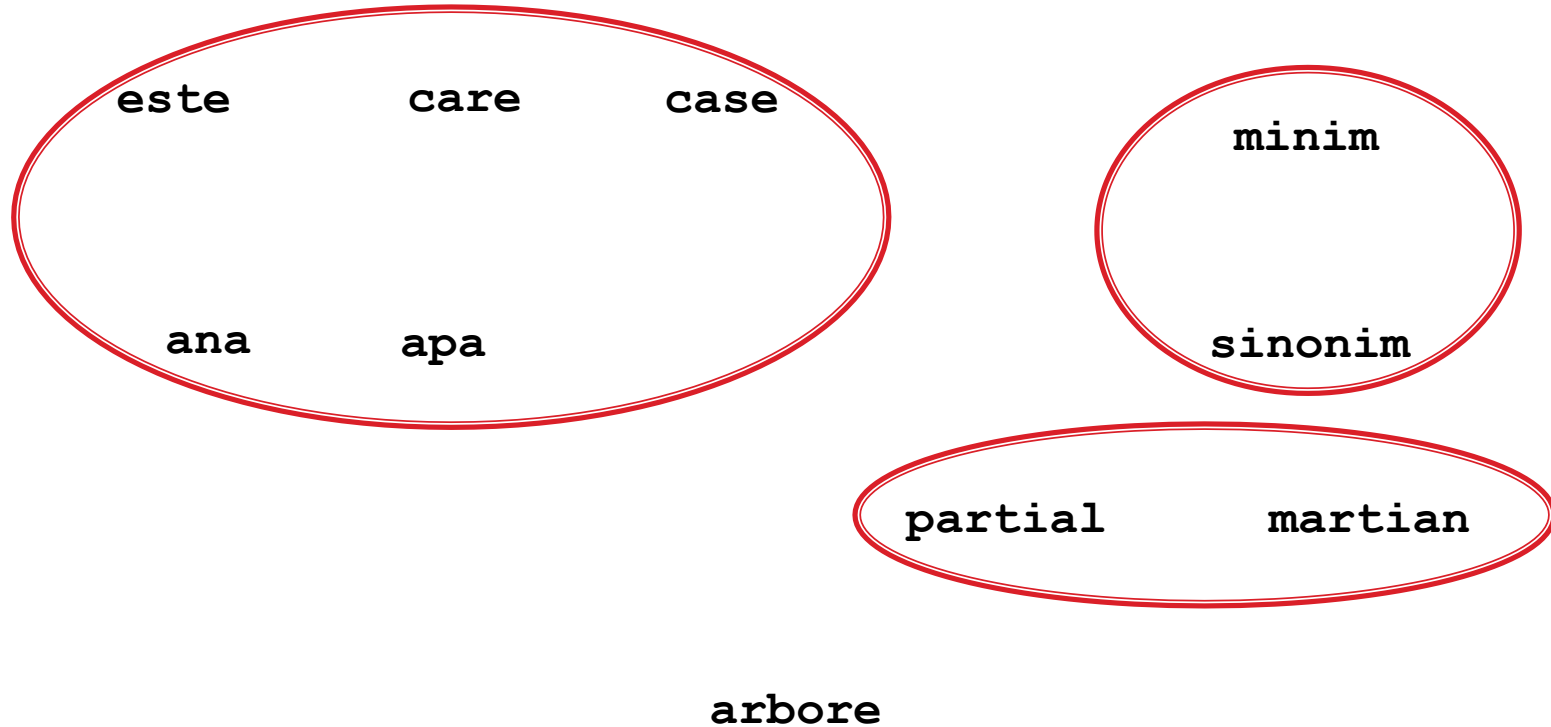


# Aplicații – Clustering



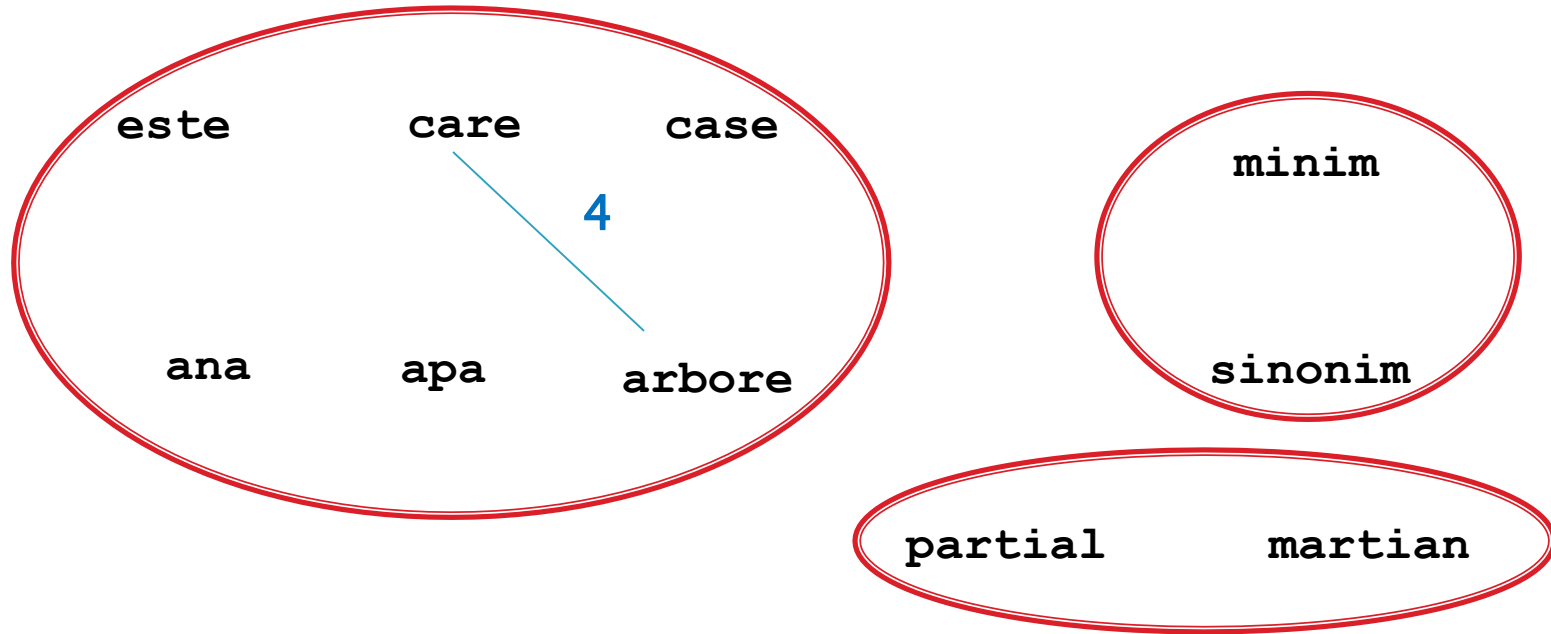
**K = 3 clustere**

# Aplicații – Clustering



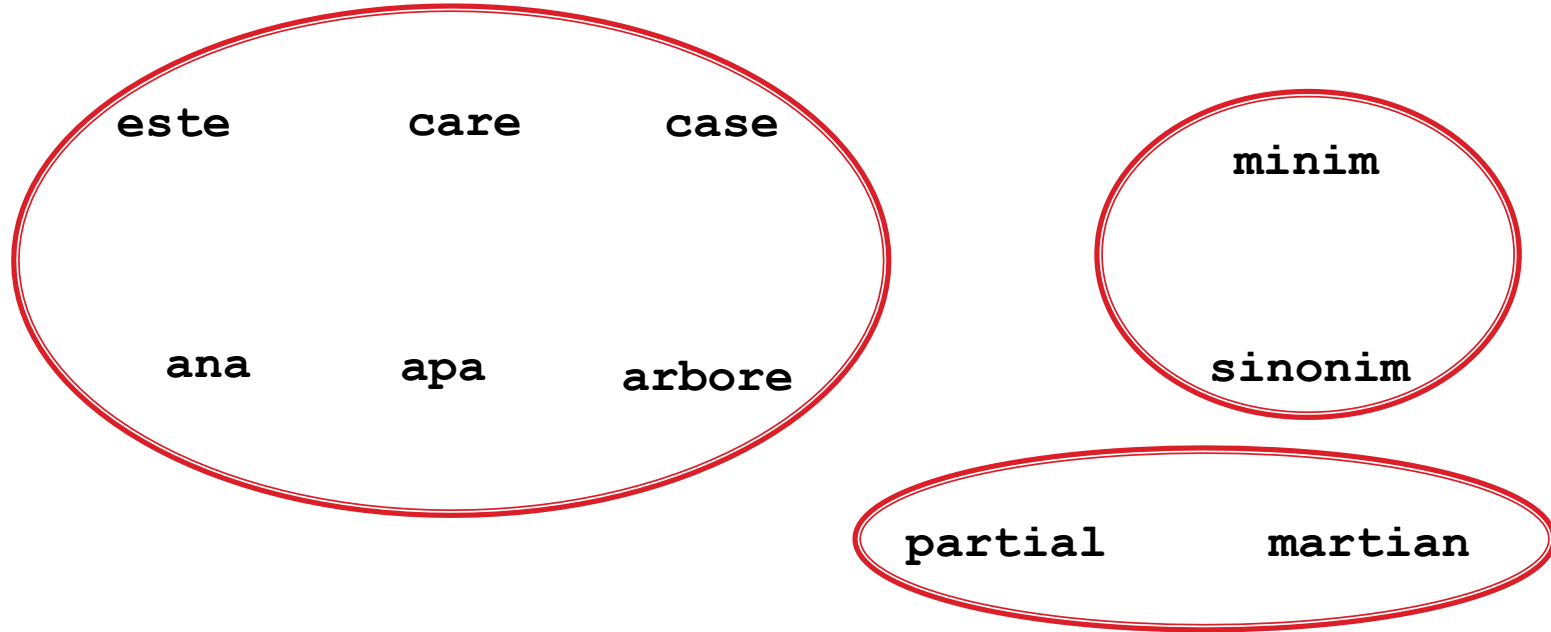
**K = 3 clustere**

# Aplicații – Clustering



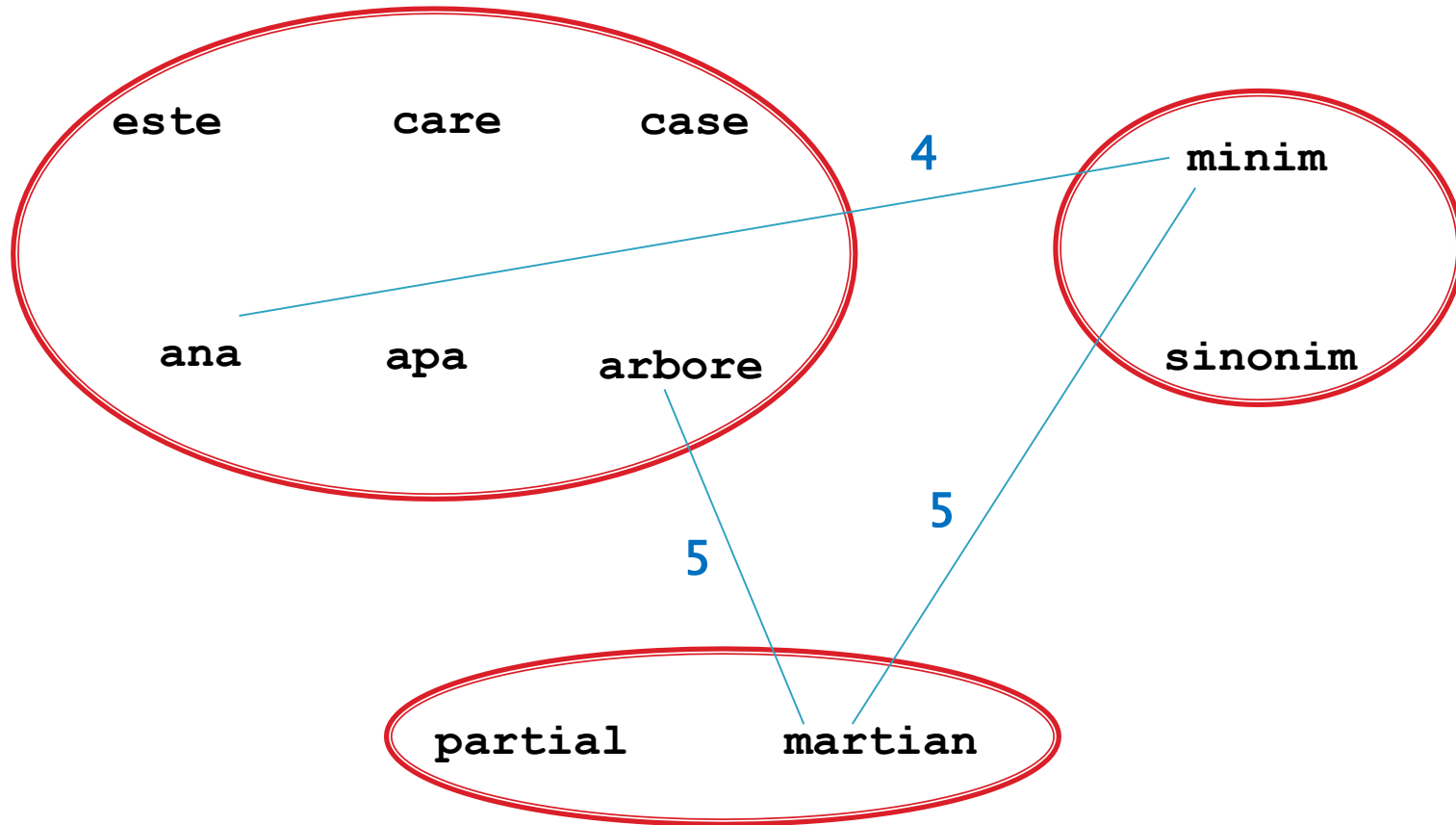
**K = 3** clustere

# Aplicații – Clustering



**Soluția cu  $k=3$  clustere**

# Aplicații – Clustering



**Grad de separare =4**

# Aplicații – Clustering

## Pseudocod:

- Inițial fiecare obiect (cuvânt) formează o clasă
- pentru  $i = 1, n-k$ 
  - alege două obiecte  $o_r, o_t$  din clase diferite cu  $d(o_r, o_t)$  minimă
  - reunește (clasa lui  $o_r$ , clasa lui  $o_t$ )
- afișează cele  $k$  clase obținute

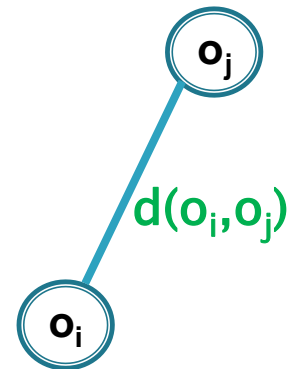
# Aplicații – Clustering

## Pseudocod:

- Inițial fiecare obiect (cuvânt) formează o clasă
- pentru  $i = 1, n-k$ 
  - alege două obiecte  $o_r, o_t$  din clase diferite cu  $d(o_r, o_t)$  minimă
  - reunește (clasa lui  $o_r$ , clasa lui  $o_t$ )
- afișează cele  $k$  clase obținute



**Modelare cu graf ponderat (complet)**  
 **$\Rightarrow n - k$  pași din algoritmul lui Kruskal**



# Aplicații – Clustering

## Pseudocod:

Inițial fiecare obiect (cuvânt)  
formează o clasă

pentru  $i = 1, n-k$

- alege două obiecte  $o_r, o_t$  din clase diferite cu  $d(o_r, o_t)$  minimă
- reunește clasa lui  $o_r$  și clasa lui  $o_t$

returnează cele  $k$  clase obținute

## Pseudocod – modelare cu graf complet $G$ :

$$V = \{o_1, \dots, o_n\}, \quad w(o_i o_j) = d(o_i, o_j)$$



# Aplicații – Clustering

## Pseudocod:

Inițial fiecare obiect (cuvânt)  
formează o clasă

pentru  $i = 1, n-k$

- alege două obiecte  $o_r, o_t$  din clase diferite cu  $d(o_r, o_t)$  minimă
- reunește clasa lui  $o_r$  și clasa lui  $o_t$

returnează cele  $k$  clase obținute

## Pseudocod – modelare cu graf complet $G$ :

$$V = \{o_1, \dots, o_n\}, \quad w(o_i o_j) = d(o_i, o_j)$$

Inițial fiecare vârf formează o componentă conexă (clasă):  $T' = (V, \emptyset)$

# Aplicații – Clustering

## Pseudocod:

Inițial fiecare obiect (cuvânt)  
formează o clasă

pentru  $i = 1, n-k$

- alege două obiecte  $o_r, o_t$  din clase diferite cu  $d(o_r, o_t)$  minimă
- reunește clasa lui  $o_r$  și clasa lui  $o_t$

returnează cele  $k$  clase obținute

## Pseudocod – modelare cu graf complet $G$ :

$$V = \{o_1, \dots, o_n\}, \quad w(o_i o_j) = d(o_i, o_j)$$

Inițial fiecare vârf formează o componentă conexă (clasă):  $T' = (V, \emptyset)$

pentru  $i = 1, n-k$

- alege o muchie  $e_i = uv$  de cost minim din  $G$  astfel încât  $u$  și  $v$  sunt în componente conexe diferite ale lui  $T'$
- reunește componenta lui  $u$  și componenta lui  $v$ :  $E(T') = E(T') \cup \{uv\}$

# Aplicații – Clustering

## Pseudocod:

Inițial fiecare obiect (cuvânt)  
formează o clasă

pentru  $i = 1, n-k$

- alege două obiecte  $o_r, o_t$  din clase diferite cu  $d(o_r, o_t)$  minimă
- reunește clasa lui  $o_r$  și clasa lui  $o_t$

returnează cele  $k$  clase obținute

## Pseudocod – modelare cu graf complet $G$ :

$V = \{o_1, \dots, o_n\}, \quad w(o_i o_j) = d(o_i, o_j)$

Inițial fiecare vârf formează o componentă conexă (clasă):  $T' = (V, \emptyset)$

pentru  $i = 1, n-k$

- alege o muchie  $e_i = uv$  de cost minim din  $G$  astfel încât  $u$  și  $v$  sunt în componente conexe diferite ale lui  $T'$
- reunește componenta lui  $u$  și componenta lui  $v$ :  $E(T') = E(T') \cup \{uv\}$

returnează cele  $k$  mulțimi formate cu vârfurile celor  $k$  componente conexe ale lui  $T'$

# Aplicații – Clustering

- ▶ Observație. Algoritmul este echivalent cu următorul, mai general
  - **determină un apcm**  $T$  al grafului complet  $G$
  - 
  - 
  -

# Aplicații – Clustering

- ▶ **Observație.** Algoritmul este echivalent cu următorul, **mai general**
  - **determină un apcm**  $T$  al grafului complet  $G$
  - consideră mulțimea  $\{e_{n-k+1}, \dots, e_{n-1}\}$  formată cu  $k-1$  muchii cu cele mai mari ponderi în  $T$
  - fie pădurea  $T' = T - \{e_{n-k+1}, \dots, e_{n-1}\}$
  -

# Aplicații – Clustering

- ▶ **Observație.** Algoritmul este echivalent cu următorul, mai general
  - **determină un apcm**  $T$  al grafului complet  $G$
  - consideră mulțimea  $\{e_{n-k+1}, \dots, e_{n-1}\}$  formată cu  $k-1$  muchii **cu cele mai mari ponderi** în  $T$
  - fie pădurea  **$T' = T - \{e_{n-k+1}, \dots, e_{n-1}\}$**
  - definește clasele  $k$ -clustering-ului  $\mathcal{C}$  ca fiind mulțimile vârfurilor celor  $k$  componente conexe ale pădurii astfel obținute

# Aplicații – Clustering

**Cum calculăm distanța de editare?**



# Aplicații – Clustering

**Cum calculăm distanța de editare?**



Programare dinamică



# Distanțe de editare

**Distanțe de editare** – numărul minim de operații (inserări, modificări, ștergeri etc) de caractere necesar pentru transforma prima secvență în cea de a doua

**Distanța de editare Levenshtein** – sunt permise operații de inserare, modificare și ștergere

**Exemplu:** Distanța de la **care** la **antet** este 4

care  $\xrightarrow{\text{stergem c}}$  are  $\xrightarrow[\text{r} \leftrightarrow \text{n}]{\text{modificăm}}$  ane  $\xrightarrow{\text{inseram t}}$  ante  $\xrightarrow{\text{inseram t}}$  antet

# Distanțe de editare

- ▶ La fiecare nepotrivire a unui caracter cu cel din destinație avem 3 operații posibile
- ▶ Dacă analizăm pe rând fiecare variantă => backtracking  
=> ineficient
- ▶ **Soluție: Programare dinamică**

# Distanțe de editare

## ▶ Principiu de optimalitate:

Considerăm o transformare cu număr minim de operații:

$$\mathbf{x}_1\mathbf{x}_2\cdots\mathbf{x}_n$$
$$\mathbf{y}_1\mathbf{y}_2\cdots\mathbf{y}_m$$

Evidențiem ultima operație

# Distanțe de editare

$$\mathbf{x}_1\mathbf{x}_2\ldots\mathbf{x}_n \Rightarrow \mathbf{y}_1\mathbf{y}_2\ldots\mathbf{y}_m$$

Evidențiem ultima operație:

- ▶  $\mathbf{x}_n = \mathbf{y}_m$  – problema se reduce la a transforma  $\mathbf{x}_1\ldots\mathbf{x}_{n-1}$  în  $\mathbf{y}_1\ldots\mathbf{y}_{m-1}$   
 $(\mathbf{x}_1\ldots\mathbf{x}_{n-1} \Rightarrow \mathbf{y}_1\ldots\mathbf{y}_{m-1}) + \text{pastrăm } \mathbf{x}_n$

▶

▶

▶

# Distanțe de editare

$$\mathbf{x}_1\mathbf{x}_2\ldots\mathbf{x}_n \Rightarrow \mathbf{y}_1\mathbf{y}_2\ldots\mathbf{y}_m$$

Evidențiem ultima operație:

▶  $\mathbf{x}_n = \mathbf{y}_m$  – problema se reduce la a transforma  $\mathbf{x}_1\ldots\mathbf{x}_{n-1}$  în  $\mathbf{y}_1\ldots\mathbf{y}_{m-1}$   
 $(\mathbf{x}_1\ldots\mathbf{x}_{n-1} \Rightarrow \mathbf{y}_1\ldots\mathbf{y}_{m-1}) + \text{pastrăm } \mathbf{x}_n$

▶  $\mathbf{x}_n$  a fost șters – problema se reduce la a transforma  $\mathbf{x}_1\ldots\mathbf{x}_{n-1}$  în  $\mathbf{y}_1\ldots\mathbf{y}_m$  (după care se șterge  $\mathbf{x}_n$ )  
 $(\mathbf{x}_1\ldots\mathbf{x}_{n-1} \Rightarrow \mathbf{y}_1\ldots\mathbf{y}_m) + \text{ștergem } \mathbf{x}_n$

▶

▶

# Distanțe de editare

$$\mathbf{x}_1\mathbf{x}_2\ldots\mathbf{x}_n \Rightarrow \mathbf{y}_1\mathbf{y}_2\ldots\mathbf{y}_m$$

Evidențiem ultima operație:

- ▶  $\mathbf{x}_n = \mathbf{y}_m$  – problema se reduce la a transforma  $\mathbf{x}_1\ldots\mathbf{x}_{n-1}$  în  $\mathbf{y}_1\ldots\mathbf{y}_{m-1}$   
 $(\mathbf{x}_1\ldots\mathbf{x}_{n-1} \Rightarrow \mathbf{y}_1\ldots\mathbf{y}_{m-1}) + \text{pastrăm } \mathbf{x}_n$
- ▶  $\mathbf{x}_n$  a fost șters – problema se reduce la a transforma  $\mathbf{x}_1\ldots\mathbf{x}_{n-1}$  în  $\mathbf{y}_1\ldots\mathbf{y}_m$  (după care se șterge  $\mathbf{x}_n$ )  
 $(\mathbf{x}_1\ldots\mathbf{x}_{n-1} \Rightarrow \mathbf{y}_1\ldots\mathbf{y}_m) + \text{ștergem } \mathbf{x}_n$
- ▶  $\mathbf{x}_n$  a fost modificat în  $\mathbf{y}_m$  – problema se reduce la a transforma  $\mathbf{x}_1\ldots\mathbf{x}_{n-1}$  în  $\mathbf{y}_1\ldots\mathbf{y}_{m-1}$  (după care se modifică  $\mathbf{x}_n$  în  $\mathbf{y}_m$ )  
 $(\mathbf{x}_1\ldots\mathbf{x}_{n-1} \Rightarrow \mathbf{y}_1\ldots\mathbf{y}_{m-1}) + \text{modificăm } \mathbf{x}_n \leftrightarrow \mathbf{y}_m$

▶

# Distanțe de editare

$$\mathbf{x}_1\mathbf{x}_2\ldots\mathbf{x}_n \Rightarrow \mathbf{y}_1\mathbf{y}_2\ldots\mathbf{y}_m$$

Evidențiem ultima operație:

- ▶  $\mathbf{x}_n = \mathbf{y}_m$  – problema se reduce la a transforma  $\mathbf{x}_1\ldots\mathbf{x}_{n-1}$  în  $\mathbf{y}_1\ldots\mathbf{y}_{m-1}$   
 $(\mathbf{x}_1\ldots\mathbf{x}_{n-1} \Rightarrow \mathbf{y}_1\ldots\mathbf{y}_{m-1}) + \text{pastrăm } \mathbf{x}_n$
- ▶  $\mathbf{x}_n$  a fost șters – problema se reduce la a transforma  $\mathbf{x}_1\ldots\mathbf{x}_{n-1}$  în  $\mathbf{y}_1\ldots\mathbf{y}_m$  (după care se șterge  $\mathbf{x}_n$ )  
 $(\mathbf{x}_1\ldots\mathbf{x}_{n-1} \Rightarrow \mathbf{y}_1\ldots\mathbf{y}_m) + \text{ștergem } \mathbf{x}_n$
- ▶  $\mathbf{x}_n$  a fost modificat în  $\mathbf{y}_m$  – problema se reduce la a transforma  $\mathbf{x}_1\ldots\mathbf{x}_{n-1}$  în  $\mathbf{y}_1\ldots\mathbf{y}_{m-1}$  (după care se modifică  $\mathbf{x}_n$  în  $\mathbf{y}_m$ )  
 $(\mathbf{x}_1\ldots\mathbf{x}_{n-1} \Rightarrow \mathbf{y}_1\ldots\mathbf{y}_{m-1}) + \text{modificăm } \mathbf{x}_n \leftrightarrow \mathbf{y}_m$
- ▶ a fost inserat  $\mathbf{y}_m$  – problema se reduce la a transforma  $\mathbf{x}_1\ldots\mathbf{x}_n$  în  $\mathbf{y}_1\ldots\mathbf{y}_{m-1}$  (după care se inserează  $\mathbf{y}_m$ )  
 $(\mathbf{x}_1\ldots\mathbf{x}_n \Rightarrow \mathbf{y}_1\ldots\mathbf{y}_{m-1}) + \text{inserăm } \mathbf{y}_m$

# Distanțe de editare

Avem deci 4 cazuri, și problema se reduce la a transforma un prefix  $\mathbf{x}_1 \dots \mathbf{x}_i$  al primului cuvânt într-un prefix  $\mathbf{y}_1 \dots \mathbf{y}_j$  al celui de al doilea cuvânt (subprobleme PD)

- ▶  $\mathbf{x}_i = \mathbf{y}_j$ :  $(\mathbf{x}_1 \dots \mathbf{x}_{i-1} \Rightarrow \mathbf{y}_1 \dots \mathbf{y}_{j-1}) + \text{pastrăm } \mathbf{x}_i$
- ▶  $(\mathbf{x}_1 \dots \mathbf{x}_{i-1} \Rightarrow \mathbf{y}_1 \dots \mathbf{y}_j) + \text{ștergem } \mathbf{x}_i$
- ▶  $(\mathbf{x}_1 \dots \mathbf{x}_{i-1} \Rightarrow \mathbf{y}_1 \dots \mathbf{y}_{j-1}) + \text{modificăm } \mathbf{x}_i \leftrightarrow \mathbf{y}_j$
- ▶  $(\mathbf{x}_1 \dots \mathbf{x}_i \Rightarrow \mathbf{y}_1 \dots \mathbf{y}_{j-1}) + \text{inserăm } \mathbf{y}_j$

## Subprobleme:

$c[i][j]$  = numărul minim de operații de inserare, ștergere, modificare pentru a transforma  $\mathbf{x}_1 \dots \mathbf{x}_i$  în  $\mathbf{y}_1 \dots \mathbf{y}_j$



# Distanțe de editare

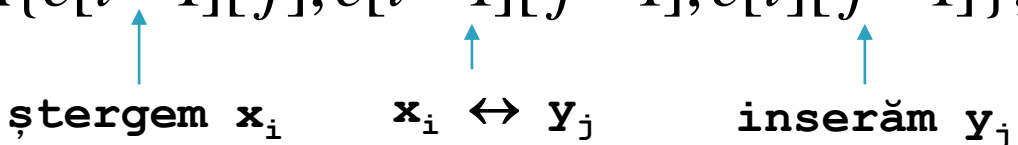
## Subprobleme:

$c[i][j]$  = numărul minim de operații de inserare, ștergere, modificare pentru a transforma  $x_1 \dots x_i$  în  $y_1 \dots y_j$

**Relații de recurență** – corespund cazurilor:

- ▶  $x_i = y_j$ :  $(x_1 \dots x_{i-1} \Rightarrow y_1 \dots y_{j-1}) + \text{pastrăm } x_i$
- ▶  $(x_1 \dots x_{i-1} \Rightarrow y_1 \dots y_j) + \text{ștergem } x_i$
- ▶  $(x_1 \dots x_{i-1} \Rightarrow y_1 \dots y_{j-1}) + \text{modificăm } x_i \leftrightarrow y_j$
- ▶  $(x_1 \dots x_i \Rightarrow y_1 \dots y_{j-1}) + \text{inserăm } y_j$

$$c[i][j] = \begin{cases} c[i-1][j-1], & \text{dacă } x_i = y_j \\ 1 + \min\{c[i-1][j], c[i-1][j-1], c[i][j-1]\}, & \text{altfel} \end{cases}$$

  
ștergem  $x_i$        $x_i \leftrightarrow y_j$       inserăm  $y_j$

# Distanțe de editare

**Soluția**  $c[n][m]$

**Ce valori din  $c$  știm direct:**

- ▶  $c[0][0] = 0$  (ambele cuvinte sunt vide)
- ▶ pentru  $i=0$  sau  $j=0$  (unul dintre cuvinte este vid):
  - $x_1 \dots x_{i-1} \Rightarrow$  secvență vidă prin  $i$  ștergeri succesive
  - secvență vidă  $\Rightarrow y_1 \dots y_j$  prin  $j$  inserări succesive

$$c[0][0] = 0$$

$$c[i][0] = 1 + c[i-1][0] = i, \text{ pentru } i = 1, \dots, n$$

$$c[0][j] = 1 + c[0][j-1] = j, \text{ pentru } j = 1, \dots, m$$

**Ordine de calcul a matricei:**  $i = 0, \dots, n; j = 0, \dots, m$

# Distanțe de editare

► Exemplu      care => antet

		0	1	2	3	4	5
			a	n	t	e	t
c:	0	0	1	2	3	4	5
	1 c	1					
	2 a	2					
	3 r	3					
	4 e	4					

$i = 0: c[0][j] = j$

$j = 0: c[i][0] = i$

# Distanțe de editare

► Exemplu  $s1 = \text{care} \Rightarrow s2 = \text{antet}$

		0	1	2	3	4	5
			a	n	t	e	t
c:	0	0	1	2	3	4	5
	1 c	1					
	2 a	2					
	3 r	3					
	4 e	4					

$i = 1, j = 1:$

$s1[i] = c, s2[j] = a$  - sunt diferite  $\Rightarrow$

# Distanțe de editare

► Exemplu  $s1 = \text{care} \Rightarrow s2 = \text{antet}$

		0	1	2	3	4	5
			a	n	t	e	t
c:	0	0	1	2	3	4	5
	1 c	1	1				
	2 a	2					
	3 r	3					
	4 e	4					

$i = 1, j = 1:$

$s1[i] = c, s2[j] = a$  - sunt diferite  $\Rightarrow$

$$\begin{aligned} c[i][j] &= 1 + \min(c[i-1][j], c[i][j-1], c[i-1][j-1]) \\ &= 1 + 0 = 1 \end{aligned}$$

# Distanțe de editare

## ► Exemplu      care => antet

		0	1	2	3	4	5
			a	n	t	e	t
c:	0	0	1	2	3	4	5
	1 c	1	1	2	3	4	5
	2 a	2					
	3 r	3					
	4 e	4					

← c nu aparține cuvântului anet

$$c[i][j] = 1 + \min(c[i-1][j], c[i][j-1], c[i-1][j-1])$$

# Distanțe de editare

## ► Exemplu      care => antet

		0	1	2	3	4	5
			a	n	t	e	t
c:	0	0	1	2	3	4	5
	1 c	1	1	2	3	4	5
	2 a	2	1				
	3 r	3					
	4 e	4					

← c nu aparține cuvântului anet

$i = 2, j = 1$ :


$s1[i] = a, s2[j] = a$  - sunt egale =>

$c[i][j] = c[i-1][j-1] = 1 = 1$

# Distanțe de editare

► Exemplu      care => antet

		0	1	2	3	4	5
			a	n	t	e	t
c:	0	0	1	2	3	4	5
	1 c	1	1	2	3	4	5
	2 a	2	1	2	3	4	5
	3 r	3					
	4 e	4					



$$c[i][j] = 1 + \min(c[i-1][j], c[i][j-1], c[i-1][j-1])$$



# Distanțe de editare

► Exemplu      care => antet

		0	1	2	3	4	5
			a	n	t	e	t
c:	0	0	1	2	3	4	5
	1 c	1	1	2	3	4	5
	2 a	2	1	2	3	4	5
	3 r	3	2				
	4 e	4					

$i = 3, j = 1$ :

$s1[i] = r, s2[j] = a$  - sunt diferite =>

$$\begin{aligned}c[i][j] &= 1 + \min(c[i-1][j], c[i][j-1], c[i-1][j-1]) \\ &= 1 + 1 = 2\end{aligned}$$

# Distanțe de editare

► Exemplu      care => antet

		0	1	2	3	4	5
			a	n	t	e	t
c:	0	0	1	2	3	4	5
	1 c	1	1	2	3	4	5
	2 a	2	1	2	3	4	5
	3 r	3	2	2	3	4	5
	4 e	4					

← r nu aparține cuvântului anet

$$c[i][j] = 1 + \min(c[i-1][j], c[i][j-1], c[i-1][j-1])$$

# Distanțe de editare

► Exemplu      care => antet

		0	1	2	3	4	5
			a	n	t	e	t
c:	0	0	1	2	3	4	5
	1 c	1	1	2	3	4	5
	2 a	2	1	2	3	4	5
	3 r	3	2	2	3	4	5
	4 e	4	3	3	3	3	

$i = 4, j = 4$ :

$s1[i] = e, s2[j] = e$  - sunt egale =>

$c[i][j] = c[i-1][j-1] = 3$

# Distanțe de editare

► Exemplu      care => antet

		0	1	2	3	4	5
			a	n	t	e	t
c:	0	0	1	2	3	4	5
	1 c	1	1	2	3	4	5
	2 a	2	1	2	3	4	5
	3 r	3	2	2	3	4	5
	4 e	4	3	3	3	3	4

Soluția:  $c[4][5] = 4$

# Distanțe de editare

► Exemplu      care => antet

		0	1	2	3	4	5
			a	n	t	e	t
c:	0	0	1	2	3	4	5
	1 c	1	1	2	3	4	5
	2 a	2	1	2	3	4	5
	3 r	3	2	2	3	4	5
	4 e	4	3	3	3	3	4

Soluția:  $c[4][5] = 4$

Cum determinăm operațiile?

# Distanțe de editare


► Exemplu      care => antet

		0	1	2	3	4	5
			a	n	t	e	t
c:	0	0	1	2	3	4	5
	1 c	1	1	2	3	4	5
	2 a	2	1	2	3	4	5
	3 r	3	2	2	3	4	5
	4 e	4	3	3	3	3	4

Soluția:  $c[4][5] = 4$

Cum determinăm operațiile – mergând succesiv înapoi de la (4,5) în celula pentru care s-a obținut egalitatea în relația de recurență:

$$c[i][j] = \begin{cases} c[i-1][j-1], & \text{dacă } x_i = y_j \\ 1 + \min\{c[i-1][j], c[i-1][j-1], c[i][j-1]\}, & \text{altfel} \end{cases}$$



ștergem  $x_i$        $x_i \leftrightarrow y_j$       inserăm  $y_j$

# Distanțe de editare

► Exemplu      care => antet

		0	1	2	3	4	5
			a	n	t	e	t
c:	0	0	1	2	3	4	5
	1 c	1	1	2	3	4	5
	2 a	2	1	2	3	4	5
	3 r	3	2	2	3	4	5
	4 e	4	3	3	3	3	4

Soluția:  $c[4][5] = 4$

$s1[4] \neq s2[5] \Rightarrow$

$c[4][5] = 1 + \min(c[4][4], c[3][5], c[3][4])$

$= 1 + c[4][4] \Rightarrow (4,5)$  s-a obtinut din  $(4,4)$  prin  
inserarea caracterului  $s2[5]=t$

# Distanțe de editare

## ► Exemplu      care => antet

		0	1	2	3	4	5
			a	n	t	e	t
c:	0	0	1	2	3	4	5
	1 c	1	1	2	3	4	5
	2 a	2	1	2	3	4	5
	3 r	3	2	2	3	4	5
	4 e	4	3	3	3	3	4

inserăm t

$s1[4] = s2[4] \Rightarrow c[4][4] = c[3][3]$  si nu s-a facut nicio  
operatie



# Distanțe de editare

## ► Exemplu      care => antet

		0	1	2	3	4	5
			a	n	t	e	t
c:	0	0	1	2	3	4	5
	1 c	1	1	2	3	4	5
	2 a	2	1	2	3	4	5
	3 r	3	2	2	3	4	5
	4 e	4	3	3	3	3	4

pastrăm e  
inserăm t

$s1[3] \neq s2[3] \Rightarrow$

$$\begin{aligned} c[3][3] &= 1 + \min(c[2][3], c[3][2], c[2][2]) \\ &= 1 + c[3][2] = 1 + c[2][2] \end{aligned}$$

!soluția nu este unică, alegem una:  $c[3][3] = 1 + c[3][2]$

$\Rightarrow (3,3)$  s-a obținut din  $(3,2)$  print inserarea caracterului  
 $s2[3]=t$

# Distanțe de editare

## ► Exemplu      care => antet

		0	1	2	3	4	5
			a	n	t	e	t
c:	0	0	1	2	3	4	5
	1 c	1	1	2	3	4	5
	2 a	2	1	2	3	4	5
	3 r	3	2	2	3	4	5
	4 e	4	3	3	3	3	4

inserăm t, pastrăm e

inserăm t

$s1[3] \neq s2[2] \Rightarrow$

$$c[3][2] = 1 + \min(c[2][2], c[3][1], c[2][1])$$
$$= 1 + c[2][1]$$

$\Rightarrow (3,2)$  s-a obtinut din  $(2,1)$  prin modificarea

$s1[3]=r \leftrightarrow s2[2]=n$

# Distanțe de editare

## ► Exemplu      care => antet

		0	1	2	3	4	5
			a	n	t	e	t
c:	0	0	1	2	3	4	5
	1 c	1	1	2	3	4	5
	2 a	2	1	2	3	4	5
	3 r	3	2	2	3	4	5
	4 e	4	3	3	3	3	4

modificăm  $r \leftrightarrow n$

inserăm t, pastrăm e

inserăm t

$s1[2] = s2[1] \Rightarrow c[2][1] = c[1][0]$  si nu s-a facut nicio  
operatie (păstrăm a)

# Distanțe de editare

## ► Exemplu      care => antet

		0	1	2	3	4	5
			a	n	t	e	t
c:	0	0	1	2	3	4	5
	1 c	1	1	2	3	4	5
	2 a	2	1	2	3	4	5
	3 r	3	2	2	3	4	5
	4 e	4	3	3	3	3	4

păstrăm a

modificăm r ↔ n

inserăm t, păstrăm e

inserăm t

i=1, j=0:

Deoarece j=0,  $c[1][0]=1+c[0][0]$  corespunzător unei ștergeri - a caracterului  $s1[i] = c$

# Distanțe de editare

## ► Exemplu      care => antet

		0	1	2	3	4	5
			a	n	t	e	t
c:	0	0	1	2	3	4	5
	1 c	1	1	2	3	4	5
	2 a	2	1	2	3	4	5
	3 r	3	2	2	3	4	5
	4 e	4	3	3	3	3	4

ștergem c

păstrăm a

modificăm r ↔ n

inserăm t, pastram e

inserăm t

# Aplicații – Clustering

## Corectitudine

- k-clusteringul obținut de algoritm are grad de separare maxim

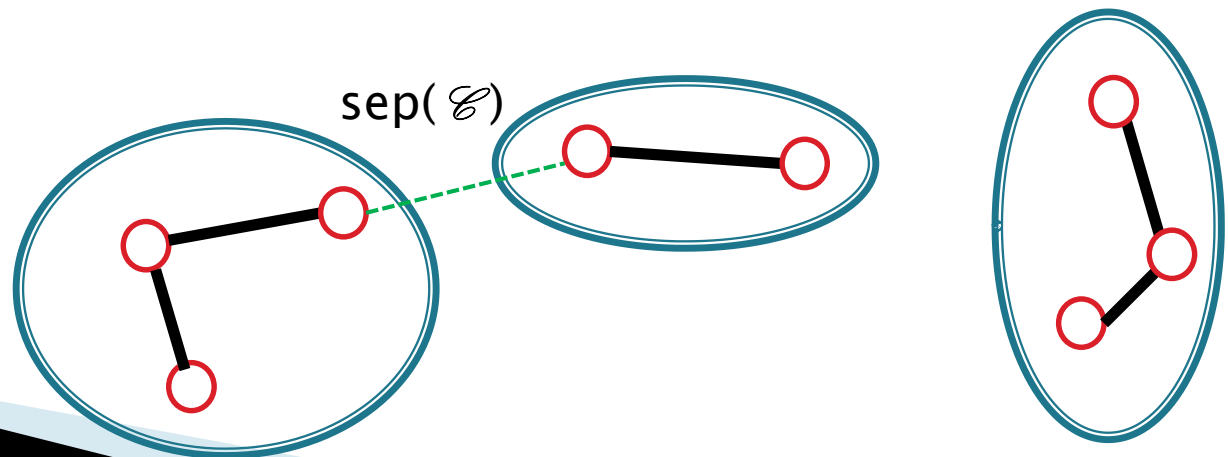
Jon Kleinberg, Éva Tardos, **Algorithm Design**, Addison–Wesley  
2005 **Secțiunea 4.7**

<http://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/04GreedyAlgorithmsII-2x2.pdf>

# Aplicații – Clustering

## Demonstrație

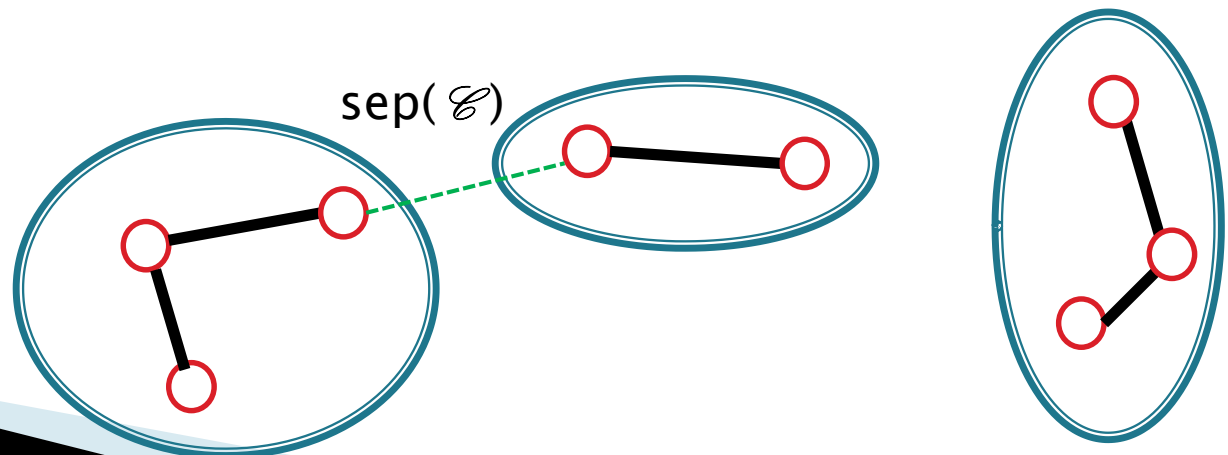
- ▶ La finalul algoritmului
  - $E(T') = \{e_1, \dots, e_{n-k}\}$ , cu  $w(e_1) \leq \dots \leq w(e_{n-k})$
  - $T'$  este o pădure cu  $k$  componente conexe, vârfurile componentelor determinând clasele lui  $\mathcal{C}$ .



# Aplicații – Clustering

## Demonstrație

- ▶ La finalul algoritmului
  - $E(T') = \{e_1, \dots, e_{n-k}\}$ , cu  $w(e_1) \leq \dots \leq w(e_{n-k})$
  - $T'$  este o pădure cu  $k$  componente conexe, vârfurile componentelor determinând clasele lui  $\mathcal{C}$ .
- ▶  $\text{sep}(\mathcal{C}) = \min\{w(e) \mid e = uv \in E(G) \text{ ce unește două componente conexe din } T'\}$





# Aplicații – Clustering

## Demonstrație

- ▶ Atunci

$$\text{sep}(\mathcal{C}) = ?$$

# Aplicații – Clustering

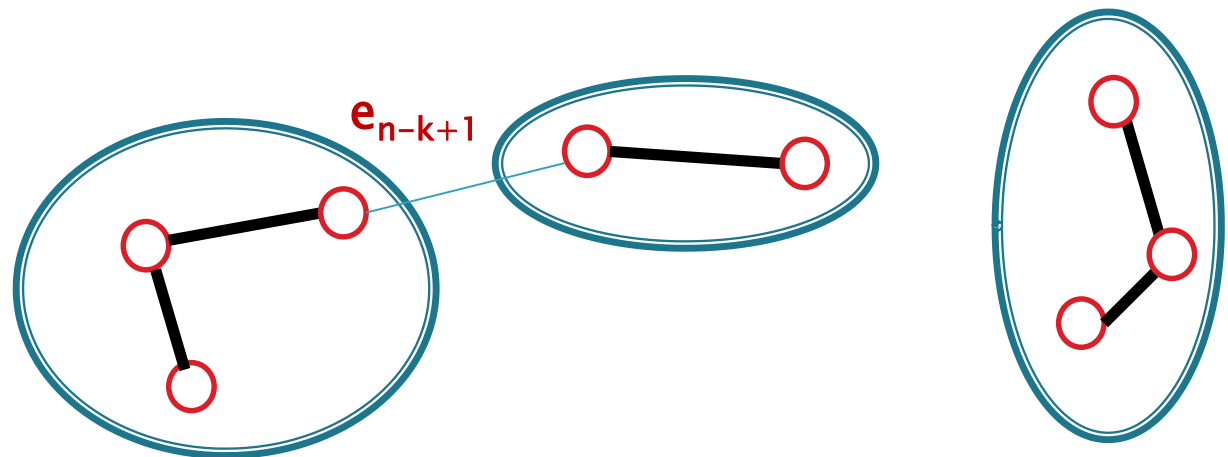
## Demonstrație

► Atunci

$$\text{sep}(\mathcal{C}) = w(e_{n-k+1}), \text{ unde}$$

$e_{n-k+1}$  = muchia de cost minim care unește  
două componente conexe din  $T'$

= **următoarea muchie care ar fi fost  
selectată de algoritm dacă ar fi continuat cu  $i = n-k+1$**



# Aplicații – Clustering

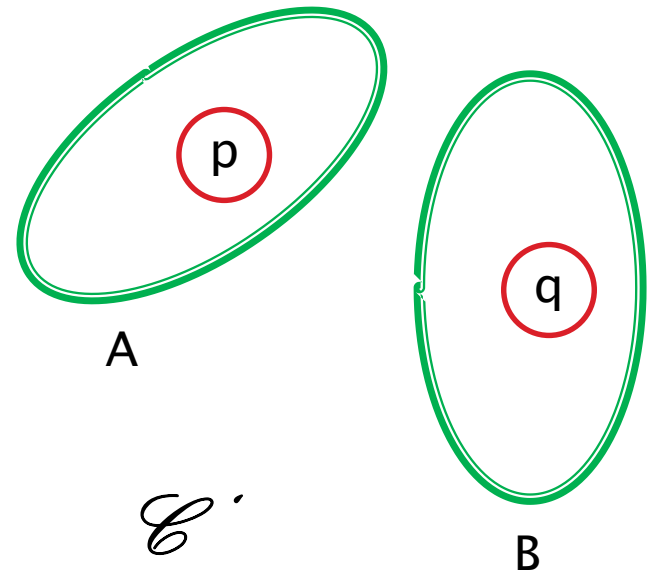
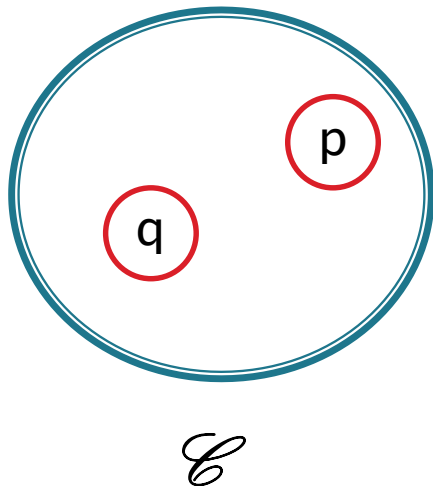
## Demonstrație

- ▶ **PRA** că există un alt  $k$ -clustering  $\mathcal{C}'$  cu  
$$\text{sep}(\mathcal{C}') > \text{sep}(\mathcal{C})$$

# Aplicații – Clustering

## Demonstrație

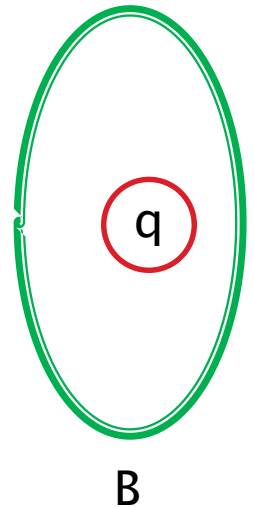
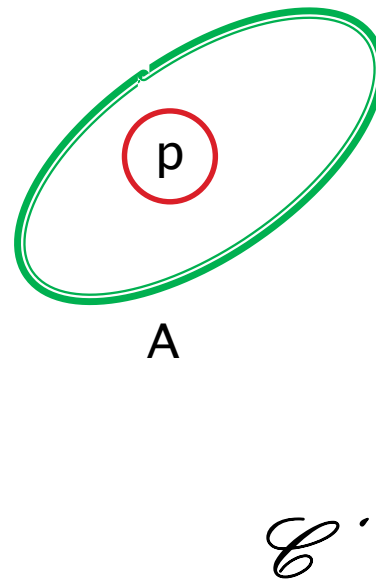
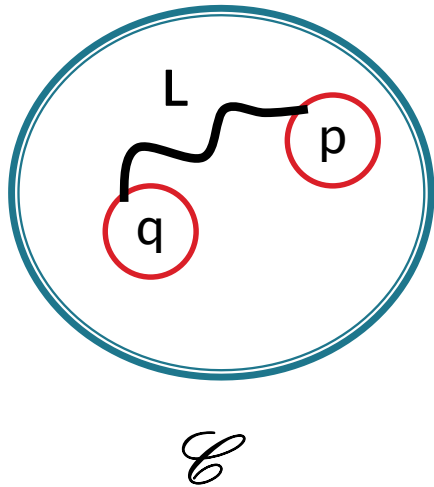
- ▶ PRA că există un alt  $k$ -clustering  $\mathcal{C}'$  cu  $\text{sep}(\mathcal{C}') > \text{sep}(\mathcal{C})$
- ▶ Atunci există două obiecte  $p$  și  $q$  care sunt
  - în aceeași clasă în  $\mathcal{C}$ ,
  - în două clase diferite în  $\mathcal{C}'$  – notate  $A, B$



# Aplicații – Clustering

## Demonstrație

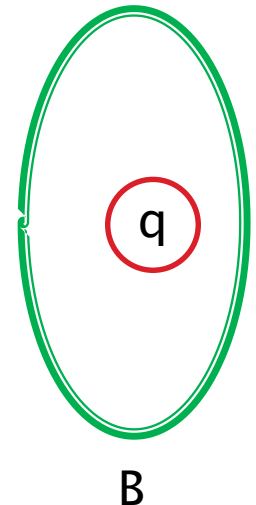
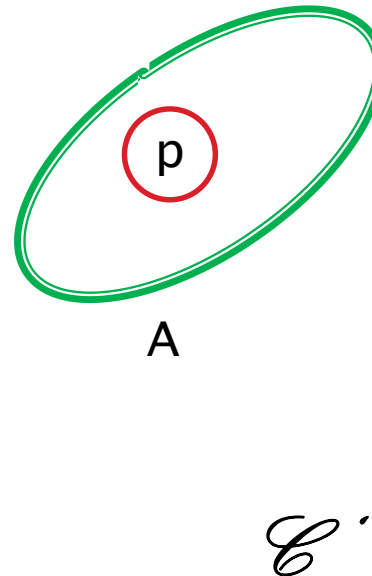
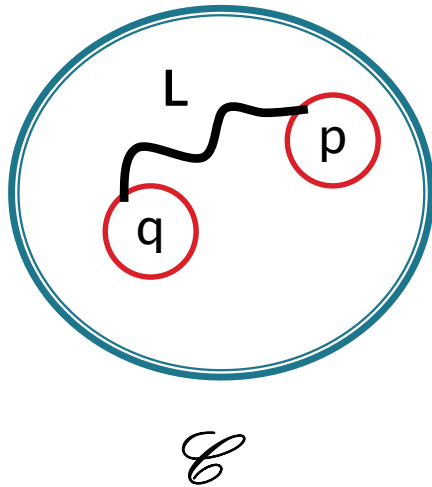
- ▶  $p$  și  $q$  sunt în aceeași clasă în  $\mathcal{C}$



# Aplicații – Clustering

## Demonstrație

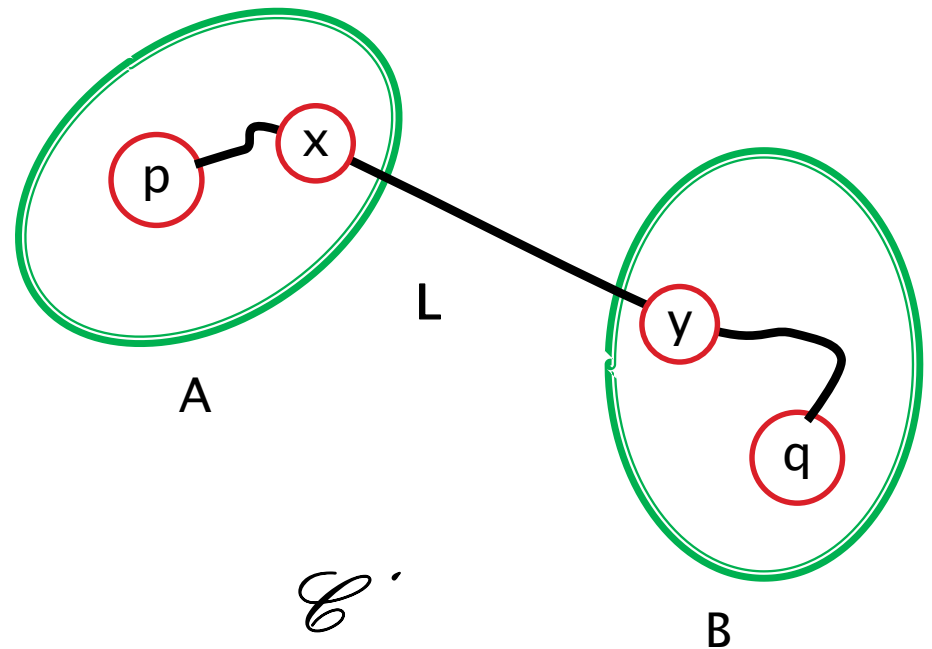
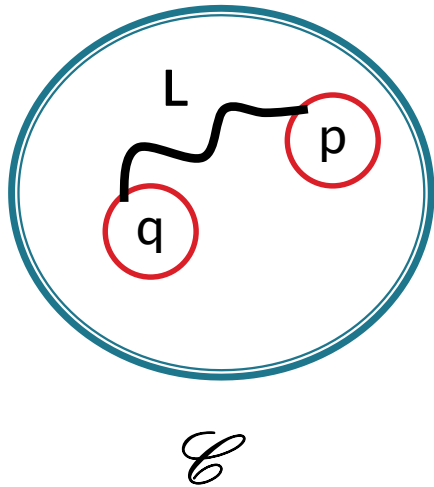
- ▶  $p$  și  $q$  sunt în aceeași clasă în  $\mathcal{C}$ 
  - $\Rightarrow$  în aceeași componentă a lui  $T'$
  - $\Rightarrow$  există  $L$  un lanț de la  $p$  la  $q$  **în  $T'$**   
(cu muchii din mulțimea  $= \{e_1, \dots, e_{n-k}\}$ )



# Aplicații – Clustering

## Demonstrație

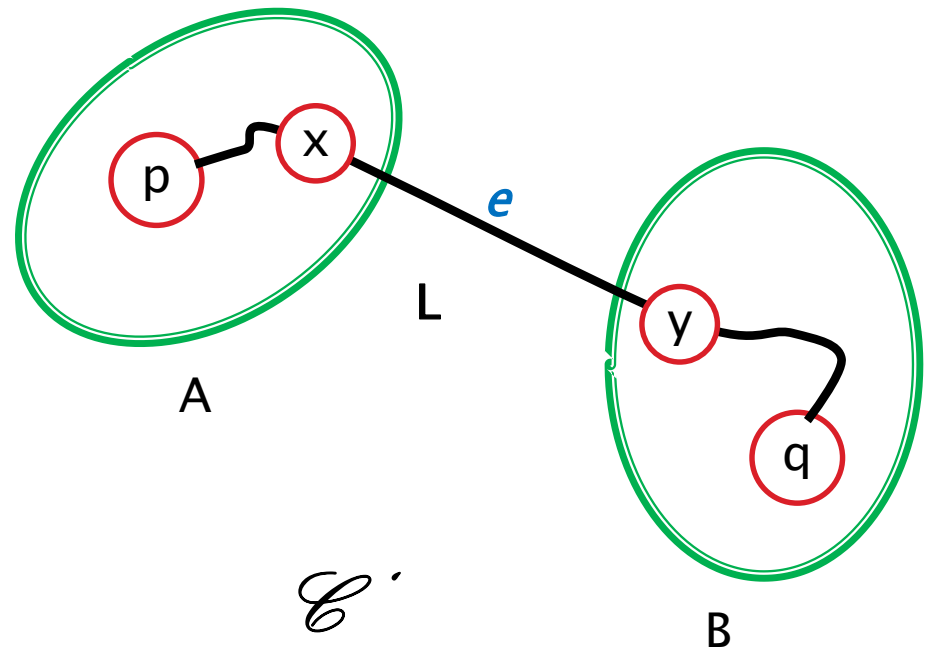
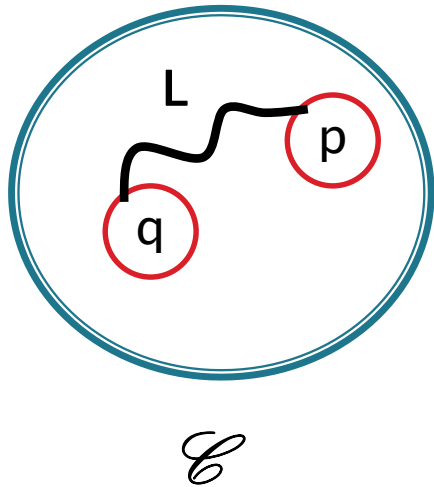
- ▶  $p$  și  $q$  sunt în clase diferite în  $\mathcal{C}'$  ( $p \in A$ ,  $q \in B$ )



# Aplicații – Clustering

## Demonstrație

- ▶  $p$  și  $q$  sunt în clase diferite în  $\mathcal{C}'$  ( $p \in A$ ,  $q \in B$ )  
 $\Rightarrow$  există în  $L$  o muchie  $e=xy$  cu o extremitate în  $A$  și cealaltă în  $B$



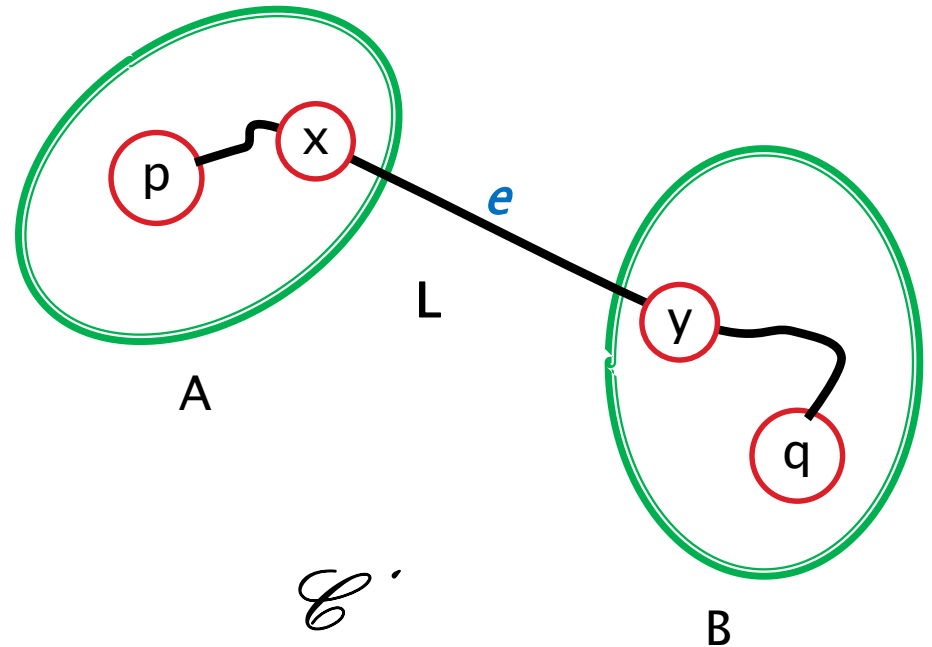
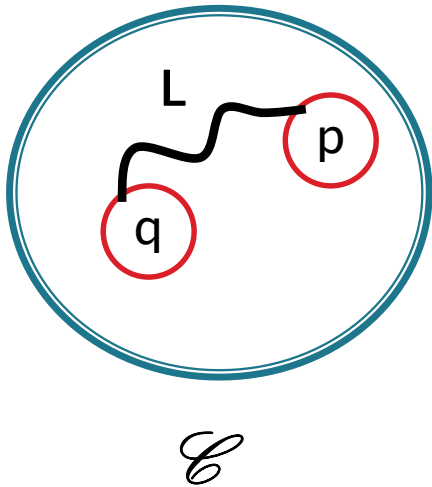


# Aplicații – Clustering

## Demonstrație

- ▶ Avem

$$\text{sep}(\mathcal{C}') \leq w(e)$$



# Aplicații – Clustering

## Demonstrație

► Avem

$$\text{sep}(\mathcal{C}') \leq w(e) \leq w(e_{n-k}) \leq w(e_{n-k+1}) = \text{sep}(\mathcal{C}) \quad \text{Contradicție}$$

$\uparrow$   
 $e \in E(T')$

