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*

Dată – 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*

Dată – 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 stergere

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

care $\xrightarrow{\text{stergem c}}$ are $\xrightarrow[\text{r <-> n}]{\text{modificăm}}$ 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 ⇒ o distanță
- Măsură a gradului de separare a claselor

Aplicații – Clustering

Cadru formal

Se dau:

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

$$\mathscr{C} = (C_1, \ldots, C_k)$$

Aplicații – Clustering

Definiții

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

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

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

  = distanța minimă dintre două obiecte aflate în clase diferite

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

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

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

Aplicații – Clustering

- obiecte= cuvinte
- d = distanța de editare d(ana, care) = 3: ana → cana → cara → care
- k = 3

care

este martian

sinonim

apa

ana

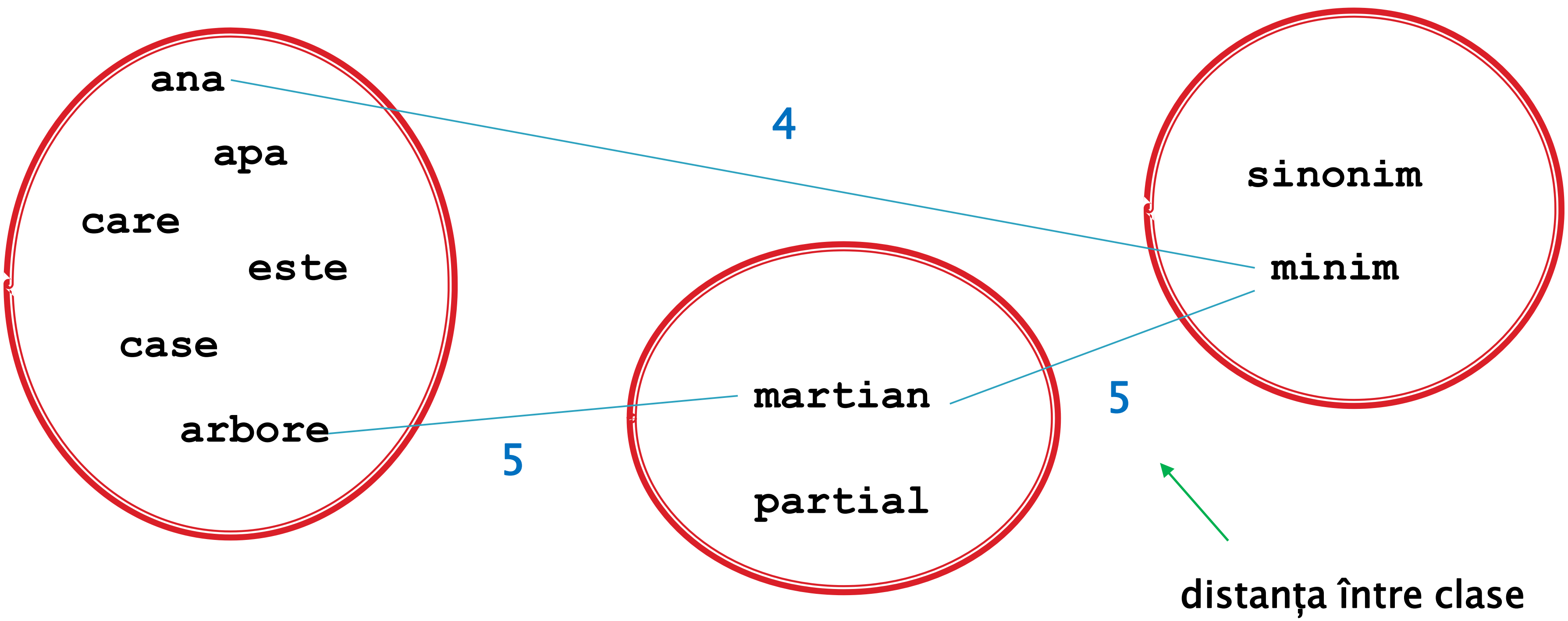
minim partial

case

arbore

Aplicații – Clustering

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



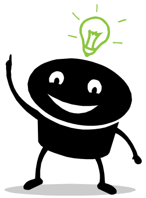
3-clustering cu gradul de separare = 4

Aplicații - Clustering

Problemă Clustering:

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

Aplicații – Clustering



Idee

care

este martian

sinonim

apa

ana

minim partial

case

arbore

Aplicații – Clustering

Idee

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

Aplicații – Clustering

Idee

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

Aplicații – Clustering

Cuvinte – distanța de editare

care

martian

este

apa

ana

sinonim

minim

partial

arbore

case

K = 3 clustere

Aplicații – Clustering

care 1 case

martian

este

apa

sinonim

ana

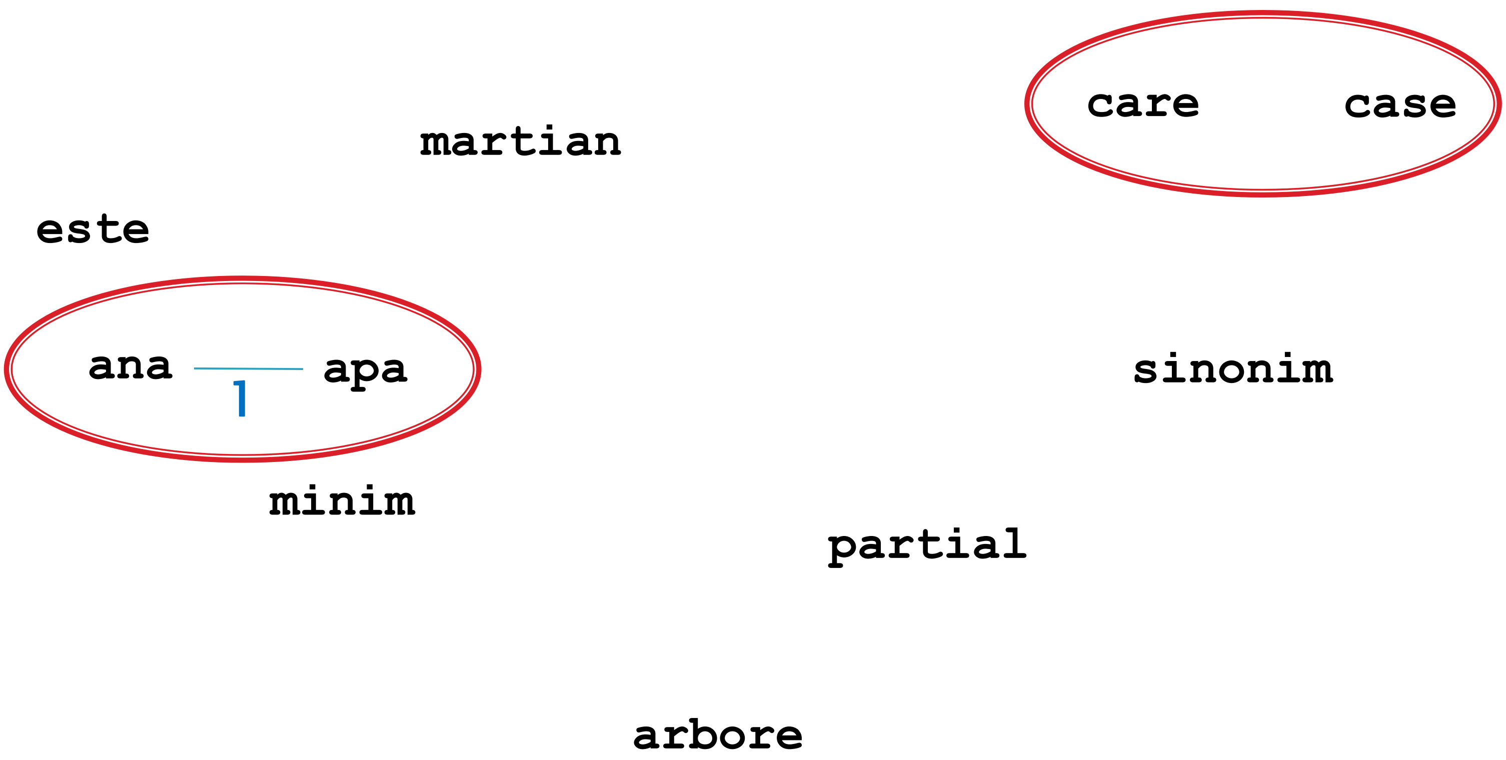
minim

partial

arbore

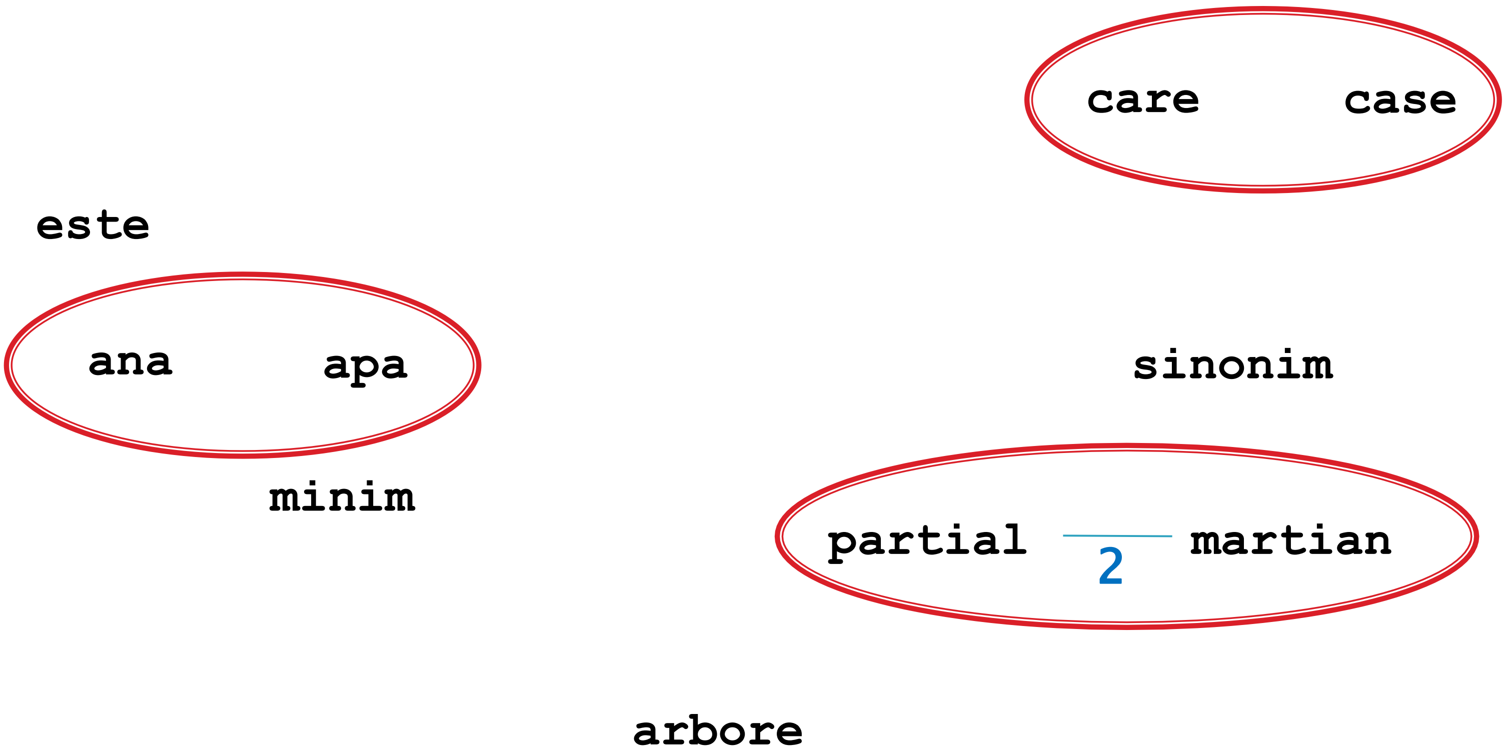
K = 3 clustere

Aplicații – Clustering



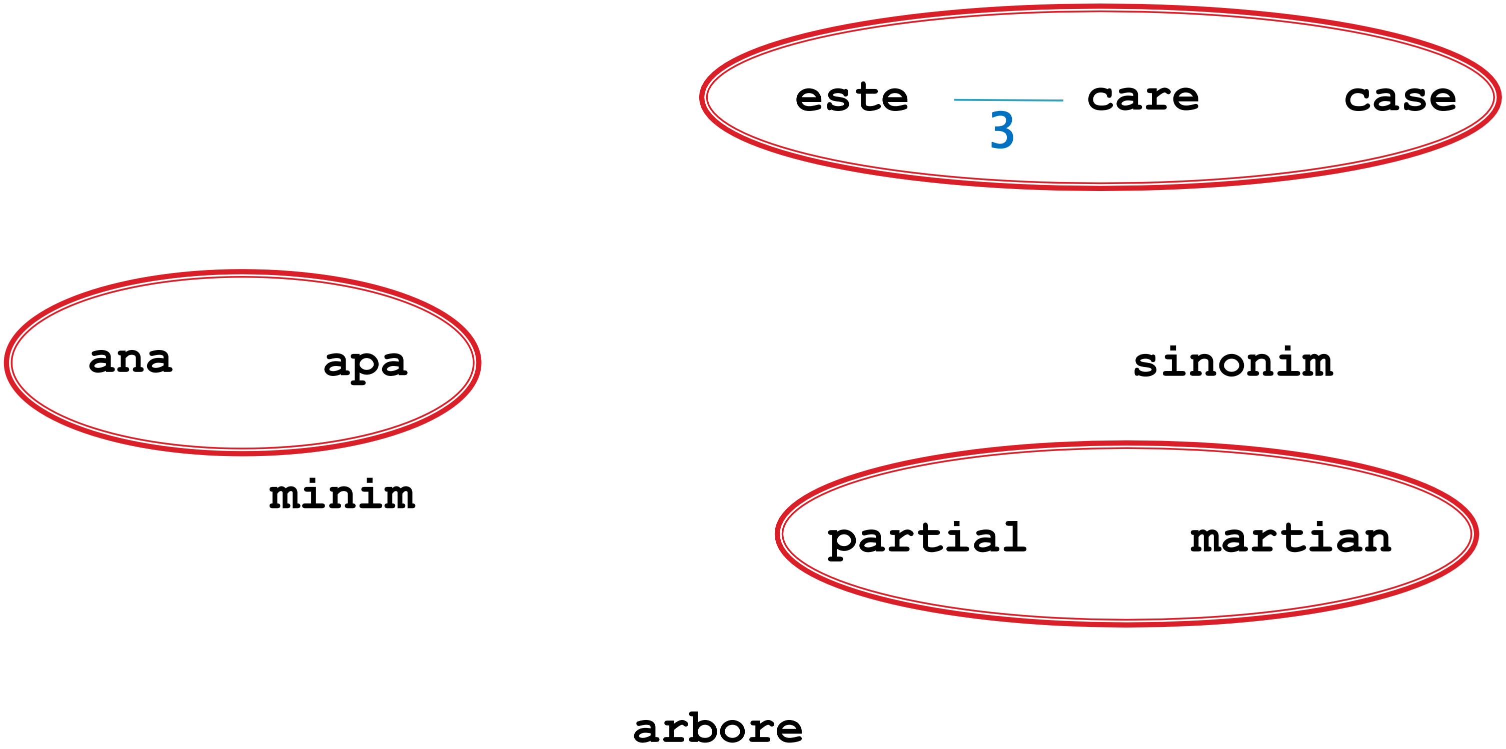
K = 3 clustere

Aplicații – Clustering



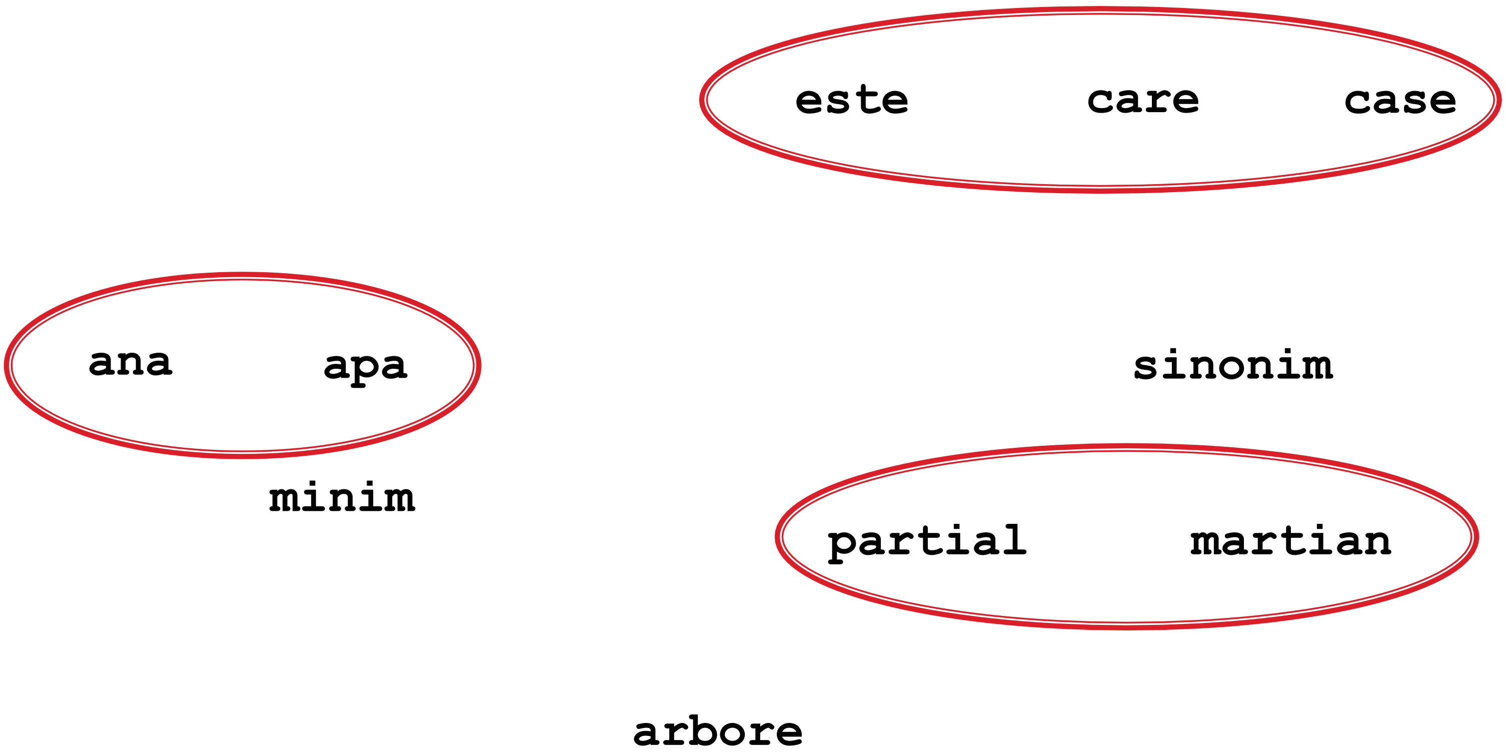
K = 3 clustere

Aplicații – Clustering



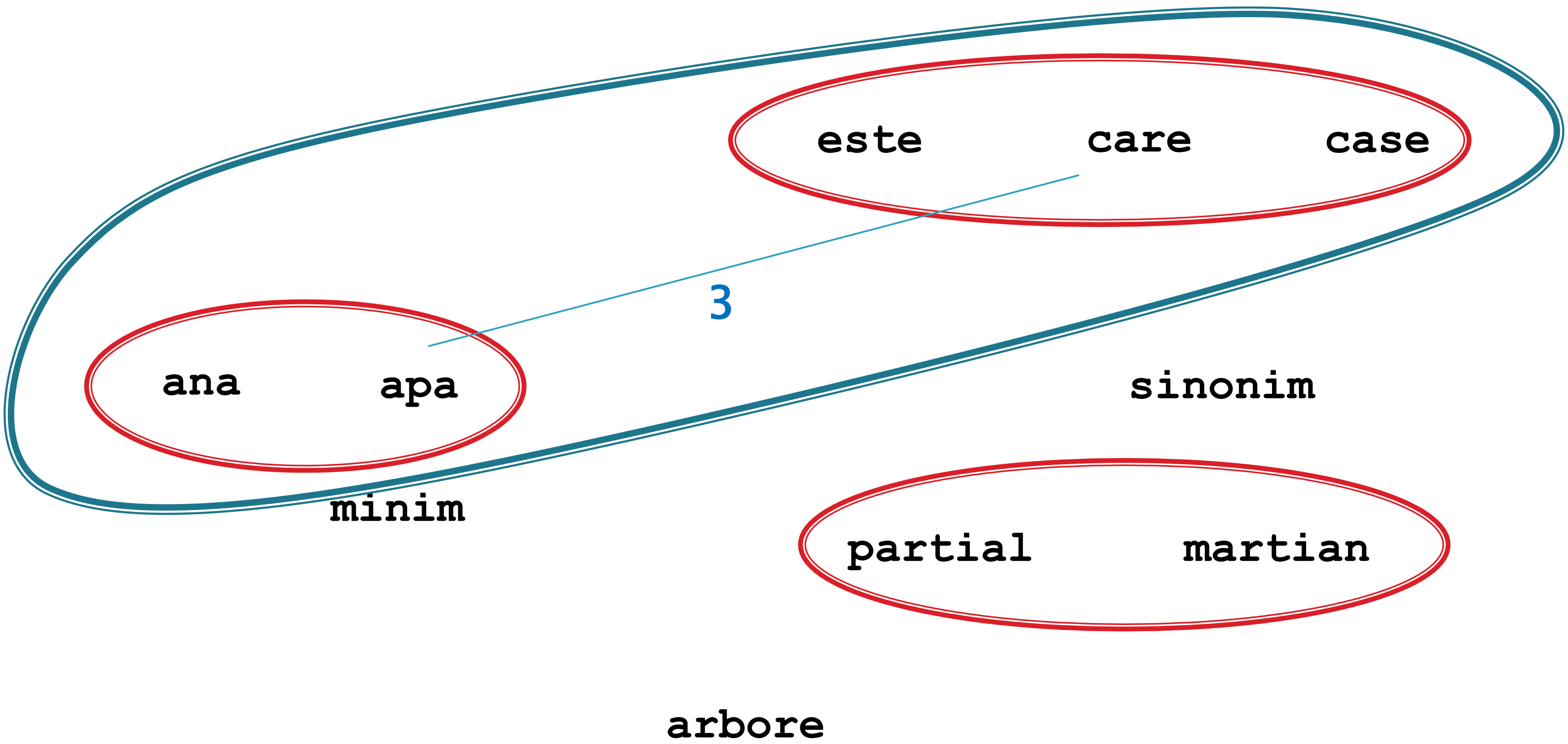
K = 3 clustere

Aplicații – Clustering



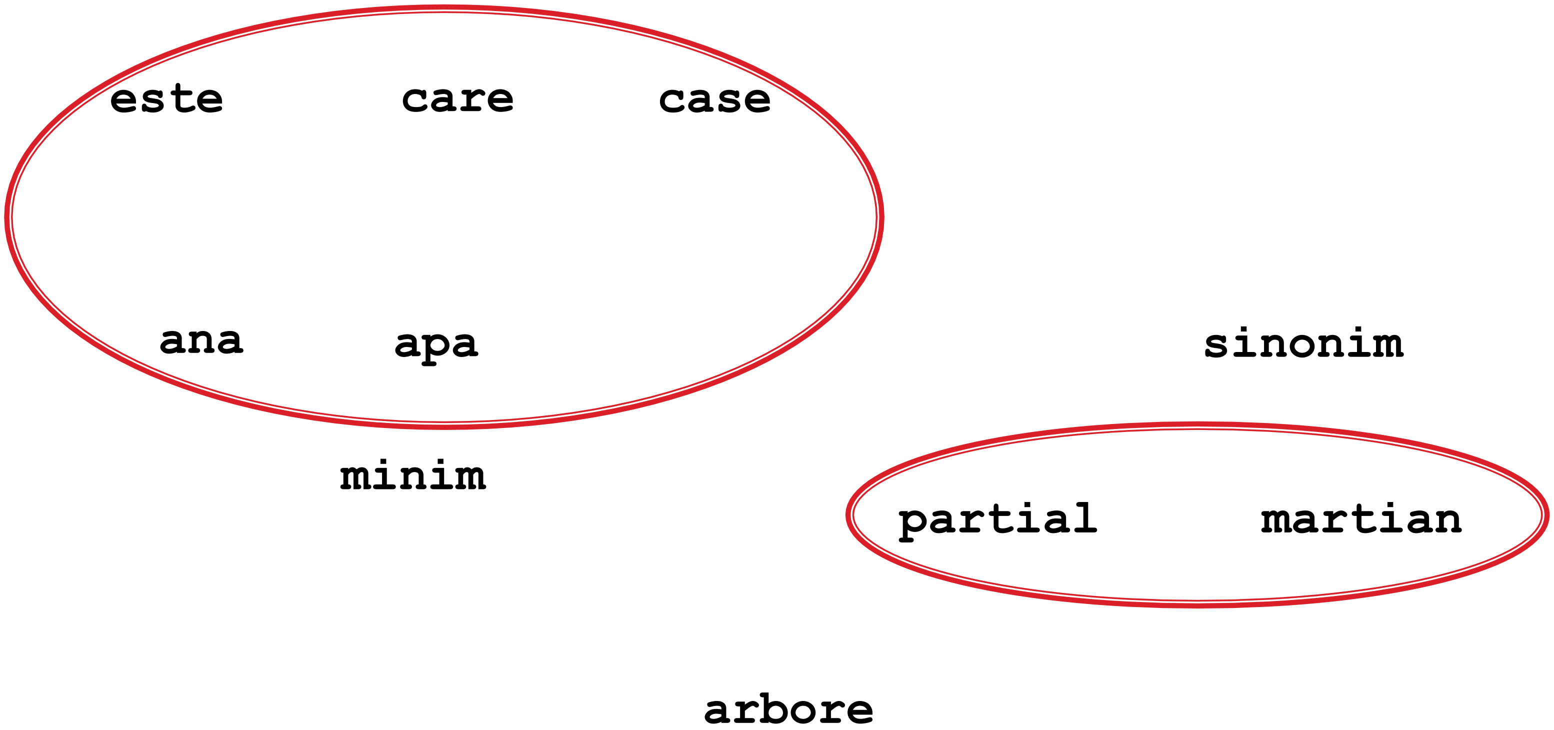
K = 3 clustere

Aplicații – Clustering



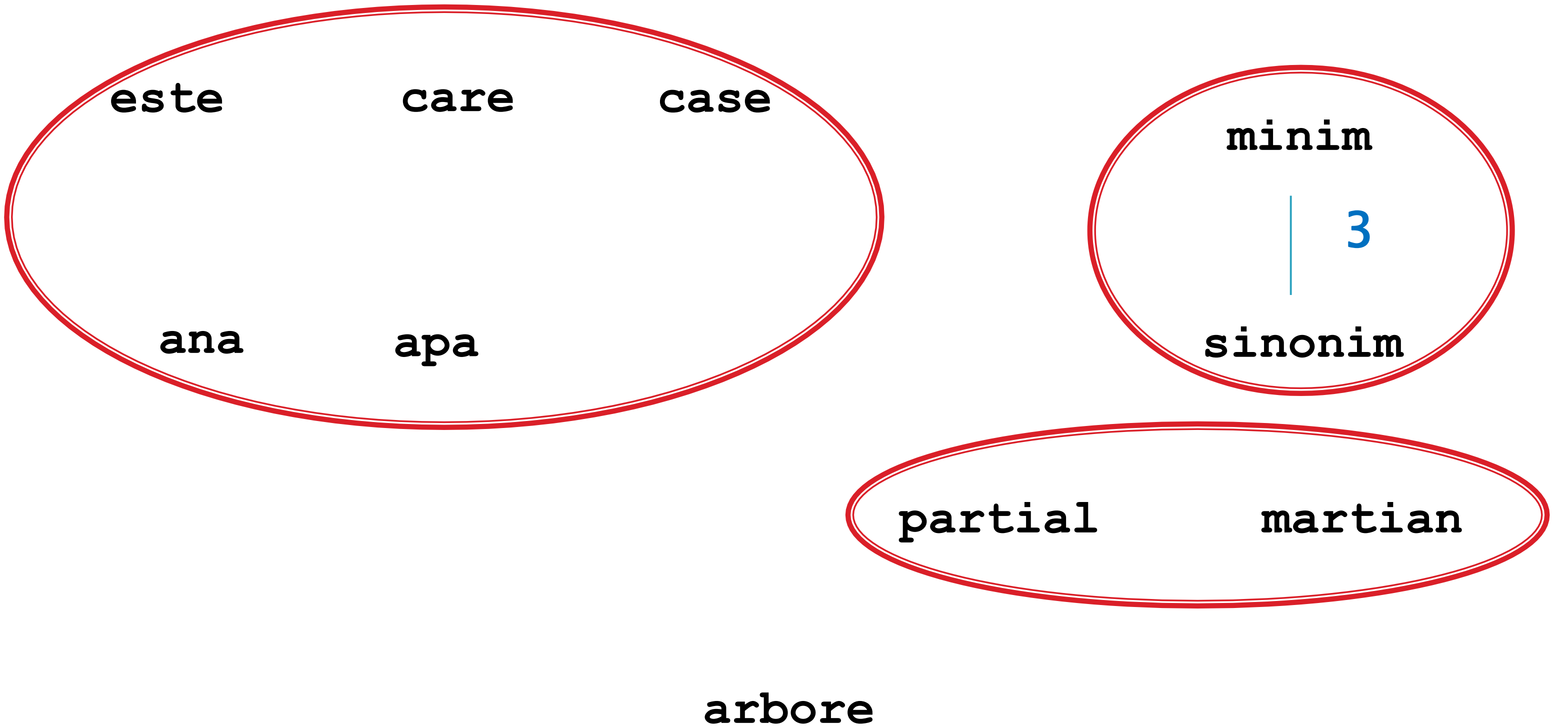
K = 3 clustere

Aplicații – Clustering



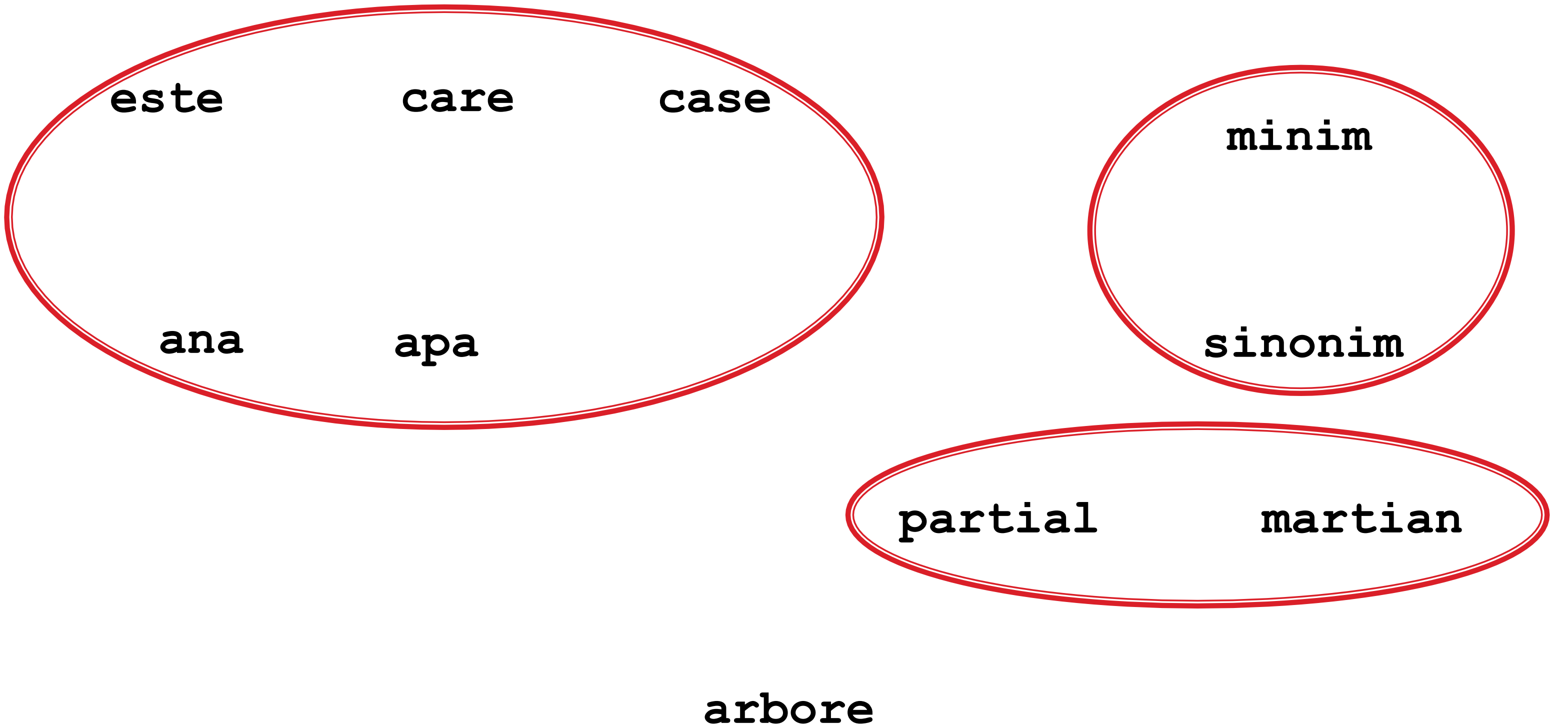
K = 3 clustere

Aplicații – Clustering



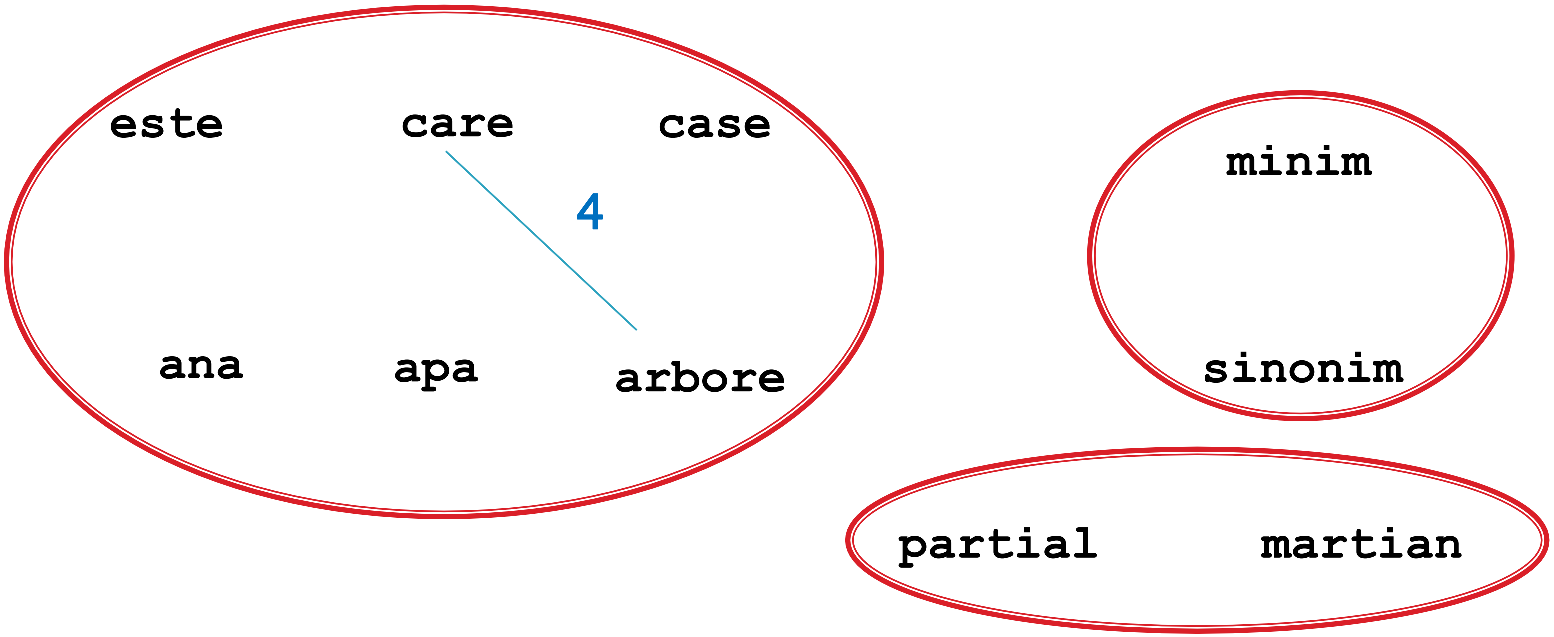
K = 3 clustere

Aplicații – Clustering



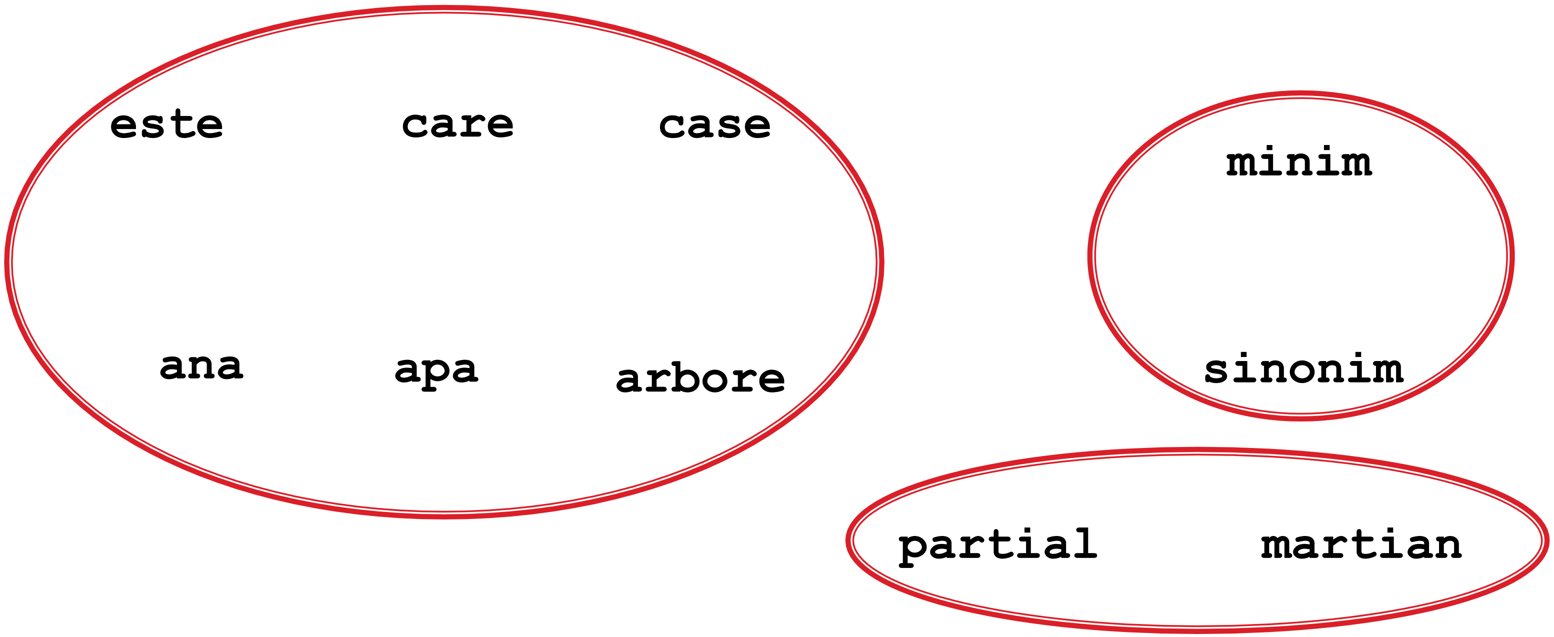
K = 3 clustere

Aplicații – Clustering



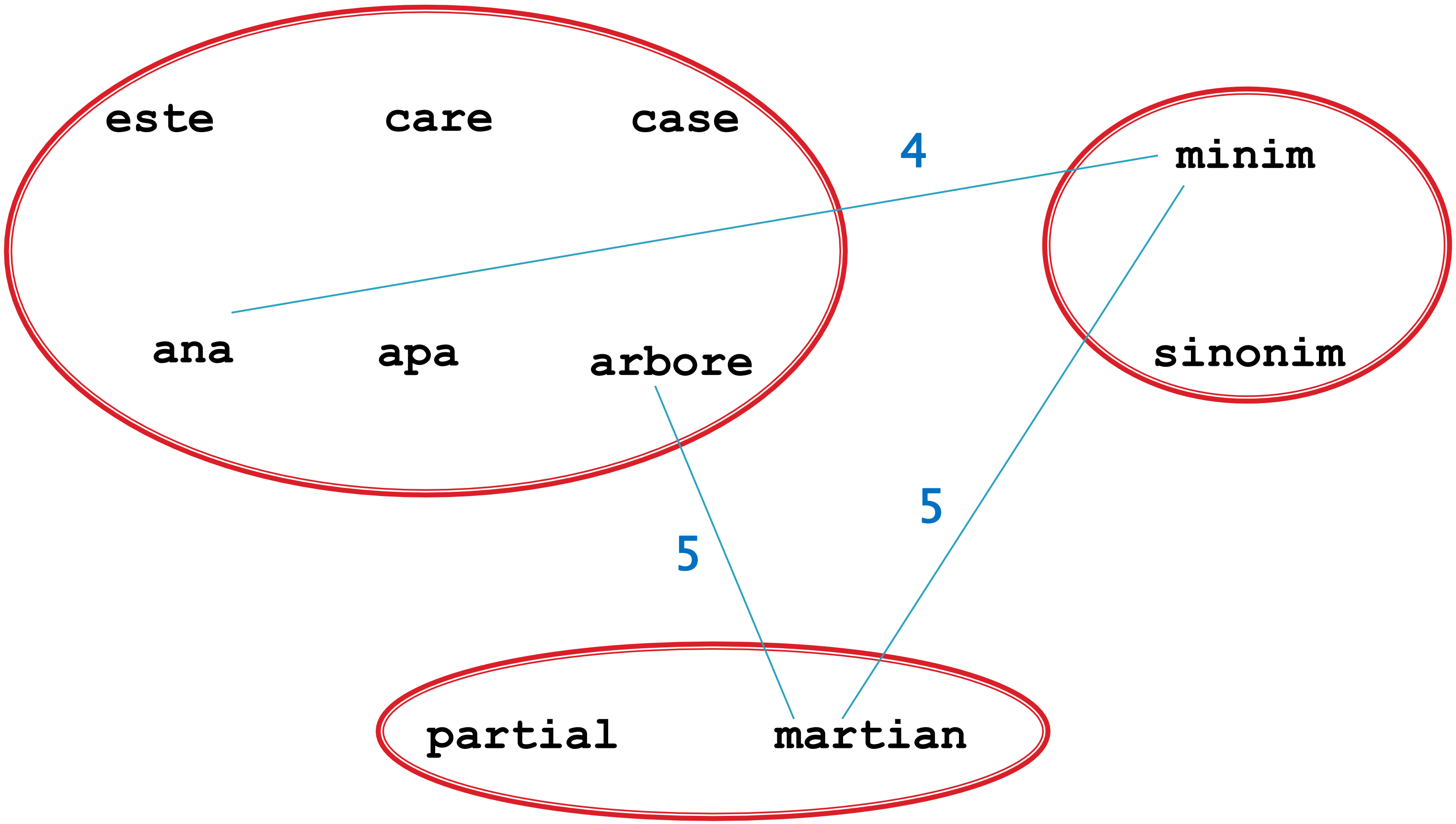
K = 3 clustere

Aplicații – Clustering



Soluția cu k= 3 clustere

Aplicații – Clustering



Grad de separare =4

Aplicații – Clustering

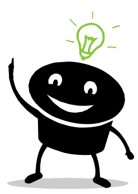
Pseudocod:

- 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

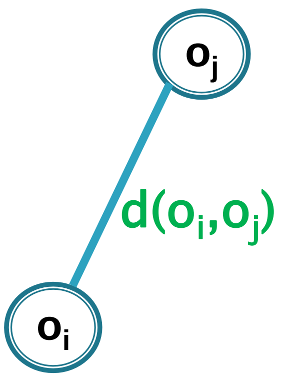
Pseudocod:

- 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)

⇒ n – k pași din algoritmul lui Kruskal



Aplicații - Clustering

Pseudocod:	**Pseudocod - modelare cu graf complet G:**
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 returneză cele k clase obținute	$V = \{o_1, \dots, o_n\}$, $w(o_i o_j) = d(o_i, o_j)$

Aplicații - Clustering

<table>
<tr>
<td>

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

returneză cele k clase obținute

</td>
<td>

Pseudocod - modelare cu graf complet G:

$V = \{o_1, \ldots, o_n\}$, $w(o_io_j) = d(o_i,o_j)$

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

</td>
</tr>
</table>

Aplicații – Clustering

Pseudocod:	Pseudocod – modelare cu graf complet G:
	$V = \{o_1, \ldots, o_n\}$, $w(o_i o_j) = d(o_i, o_j)$
Inițial fiecare obiect (cuvânt) formează o clasă	Inițial fiecare vârf formează o componentă conexă (clasă): **$T' = (V, \varnothing)$**
pentru i = 1, n-k	pentru i = 1, n-k
• alege două obiecte o_r, o_t din clase diferite cu $d(o_r, o_t)$ minimă	• 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 clasa lui o_r și clasa lui o_t	• reunește componenta lui u și componenta lui v: **$E(T') = E(T') \cup \{uv\}$**
returnează cele k clase obținute	

Aplicații – Clustering

Pseudocod:	Pseudocod – modelare cu graf complet G:
	$V = \{o_1, \ldots, o_n\}$, $w(o_io_j) = d(o_i, o_j)$
Inițial fiecare obiect (cuvânt) formează o clasă	Inițial fiecare vârf formează o componentă conexă (clasă): **$T' = (V, \emptyset)$**
pentru i = 1, n−k	pentru i = 1, n−k
• alege două obiecte o_r, o_t din clase diferite cu $d(o_r, o_t)$ minimă	• 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 clasa lui o_r și clasa lui o_t	• reunește componenta lui u și componenta lui v: **$E(T') = E(T') \cup \{uv\}$**
returnează cele k clase obținute	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

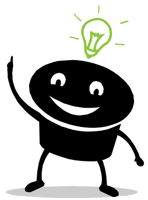
- **Observație**. Algoritmul este echivalent cu următorul, mai general
  - **determină un apcm** T al grafului complet G
  - consideră mulțimea $\{e_{n-k+1}, \dots, e_{n-1}\}$ formată cu k-1 muchii **cu cele mai mari ponderi în** T
  - fie pădurea $\mathbf{T'=T-\{e_{n-k+1}, \dots, e_{n-1}\}}$
  - definește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

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

care $\xrightarrow[\text{stergem c}]{}$ are $\xrightarrow[\text{modificăm r <-> n}]{}$ 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:

$\mathbf{x}_1\mathbf{x}_2\ldots\mathbf{x}_n$

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

Evidențiem ultima operație

Distanțe de editare

$x_1x_2 \ldots x_n \Rightarrow y_1y_2 \ldots y_m$

Evidențiem ultima operație:

- $x_n = y_m$ – problema se reduce la a transforma $x_1 \ldots x_{n-1}$ în $y_1 \ldots y_{m-1}$

  $(x_1 \ldots x_{n-1} \Rightarrow y_1 \ldots y_{m-1})$ **+ pastrăm** x_n

-

-

-

Distanțe de editare

$x_1x_2...x_n$ => $y_1y_2...y_m$

Evidențiem ultima operație:

- $x_n = y_m$ – problema se reduce la a transforma $x_1...x_{n-1}$ în $y_1...y_{m-1}$

  ($x_1...x_{n-1}$ => $y_1...y_{m-1}$) + pastrăm x_n

- x_n a fost șters – problema se reduce la a transforma $x_1...x_{n-1}$ în $y_1...y_m$ (după care se șterge x_n)

  ($x_1...x_{n-1}$ => $y_1...y_m$) + ștergem x_n

-

-

Distanțe de editare

$x_1x_2 \ldots x_n$ => $y_1y_2 \ldots y_m$

Evidențiem ultima operație:

- $x_n = y_m$ – problema se reduce la a transforma $x_1 \ldots x_{n-1}$ în $y_1 \ldots y_{m-1}$

  ($x_1 \ldots x_{n-1}$ => $y_1 \ldots y_{m-1}$) + pastrăm x_n

- x_n a fost șters – problema se reduce la a transforma $x_1 \ldots x_{n-1}$ în $y_1 \ldots y_m$ (după care se șterge x_n)

  ($x_1 \ldots x_{n-1}$ => $y_1 \ldots y_m$) + ștergem x_n

- x_n a fost modificat în y_m – problema se reduce la a transforma $x_1 \ldots x_{n-1}$ în $y_1 \ldots y_{m-1}$ (după care se modifică x_n în y_m)

  ($x_1 \ldots x_{n-1}$ => $y_1 \ldots y_{m-1}$) + modificăm $x_n \leftrightarrow y_m$

-

Distanțe de editare

$x_1 x_2 \ldots x_n \Rightarrow y_1 y_2 \ldots y_m$

Evidențiem ultima operație:

- $x_n = y_m$ – problema se reduce la a transforma $x_1 \ldots x_{n-1}$ în $y_1 \ldots y_{m-1}$

  $(x_1 \ldots x_{n-1} \Rightarrow y_1 \ldots y_{m-1})$ **+ pastrăm** x_n

- x_n a fost șters – problema se reduce la a transforma $x_1 \ldots x_{n-1}$ în $y_1 \ldots y_m$ (după care se șterge x_n)

  $(x_1 \ldots x_{n-1} \Rightarrow y_1 \ldots y_m)$ **+ ștergem** x_n

- x_n a fost modificat în y_m – problema se reduce la a transforma $x_1 \ldots x_{n-1}$ în $y_1 \ldots y_{m-1}$ (după care se modifică x_n în y_m)

  $(x_1 \ldots x_{n-1} \Rightarrow y_1 \ldots y_{m-1})$ **+ modificăm** $x_n \leftrightarrow y_m$

- a fost inserat y_m – problema se reduce la a transforma $x_1 \ldots x_n$ în $y_1 \ldots y_{m-1}$ (după care se inserează y_m)

  $(x_1 \ldots x_n \Rightarrow y_1 \ldots y_{m-1})$ **+ inserăm** y_m

Distanțe de editare

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

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

Subprobleme:

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

Distanțe de editare

Subprobleme:

`c[i][j]` = număul 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:

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

Distanțe de editare

Subprobleme:

c[i][j] = număul minim de operații de inserare, ștergere, modificare pentru a transforma $\mathbf{x_1 \ldots x_i}$ în $\mathbf{y_1 \ldots y_j}$

Relații de recurență – corespund cazurilor:

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

$$c[i][j] = \begin{cases} c[i-1][j-1], \text{ dacă } \mathrm{x_i} = \mathrm{y_j} \\ 1 + \min\{\underbrace{c[i-1][j]}_{\textbf{ștergem } \mathbf{x_i}}, \underbrace{c[i-1][j-1]}_{\mathbf{x_i} \leftrightarrow \mathbf{y_j}}, \underbrace{c[i][j-1]}_{\textbf{inserăm } \mathbf{y_j}}\}, \text{ altfel} \end{cases}$$

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)**:
  - $x_1 \ldots x_{i-1}$ **=> secvență vidă** prin i ștergeri succesive
  - **secvență vidă =>** $y_1 \ldots y_j$ prin j inserări succesive

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

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

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

Ordine de calcul a matricei: **i = 0,…,n; j = 0,…,m**

Distanțe de editare

- **Exemplu** `care => antet`

C:

	0	1 a	2 n	3 t	4 e	5 t
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 = care => s2 = antet`

C:

	0	1 a	2 n	3 t	4 e	5 t
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 =>
```

Distanțe de editare

- **Exemplu** `s1 = care => s2 = antet`

c:

	0	1 a	2 n	3 t	4 e	5 t
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 =>
        c[i][j] = 1 + min(c[i-1][j], c[i][j-1], c[i-1][j-1])
                = 1 + 0 = 1
```

Distanțe de editare

- **Exemplu** `care => antet`

C:

	0	1 a	2 n	3 t	4 e	5 t
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`

C:

	0	1 a	2 n	3 t	4 e	5 t
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`

C:

	0	1 a	2 n	3 t	4 e	5 t
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`

C:

	0	1 a	2 n	3 t	4 e	5 t
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 =>
      c[i][j] = 1 + min(c[i-1][j], c[i][j-1], c[i-1][j-1])
              = 1 + 1 = 2
```

Distanțe de editare

- **Exemplu** `care => antet`

c:

	0	1 a	2 n	3 t	4 e	5 t
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`

C:

	0	1 a	2 n	3 t	4 e	5 t
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`

c:

	0	1 a	2 n	3 t	4 e	5 t
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`

c:

		0	1 a	2 n	3 t	4 e	5 t
	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`

c:

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

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

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

Distanțe de editare

- **Exemplu** `care => antet`

c:

	0	1 a	2 n	3 t	4 e	5 t
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] != s2[5] =>

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

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

Distanțe de editare

- **Exemplu** `care => antet`

C:

		0	1 a	2 n	3 t	4 e	5 t
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] => c[4][4] = c[3][3] si nu s-a facut nicio operatie`

Distanțe de editare

- **Exemplu** `care => antet`

C:

	0	1 a	2 n	3 t	4 e	5 t
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

pastram e

inserăm t

```
s1[3] != s2[3] =>
c[3][3] = 1 + min(c[2][3],c[3][2], c[2][2])
        = 1+ c[3][2] = 1 + c[2][2]
!soluția nu este unică, alegem una: c[3][3] = 1+ c[3][2]
⇒ (3,3) s-a obtinut din (3,2) print inserarea caracterului
        s2[3]=t
```

Distanțe de editare

- **Exemplu** `care => antet`

		0	1 a	2 n	3 t	4 e	5 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, pastram e

inserăm t

```
s1[3] != s2[2] =>
c[3][2] = 1 + min(c[2][2],c[3][1], c[2][1])
        = 1+ c[2][1]
=>(3,2) s-a obtinut din (2,1)  prin modificarea
          s1[3]=r ↔ s2[2]=n
```

Distanțe de editare

- **Exemplu** `care => antet`

C:

	0	1 a	2 n	3 t	4 e	5 t
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 ↔ n

inserăm t, pastram e

inserăm t

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

Distanțe de editare

- **Exemplu** `care => antet`

C:

	0	1 a	2 n	3 t	4 e	5 t
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, pastram e

inserăm t

`i=1, j=0:`

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

Distanțe de editare

- **Exemplu** `care => antet`

c:

	0	1 a	2 n	3 t	4 e	5 t
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 obţinut de algoritm are grad de separare maxim

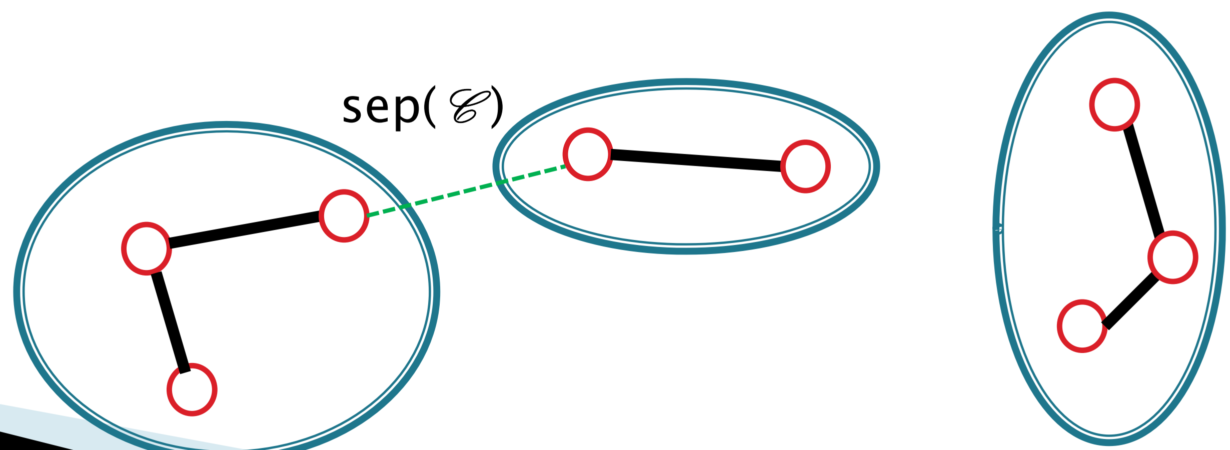
Jon Kleinberg, Éva Tardos, **Algorithm Design**, Addison-Wesley 2005 **Secțiunea 4.7**

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

Aplicații - Clustering

Demonstrație

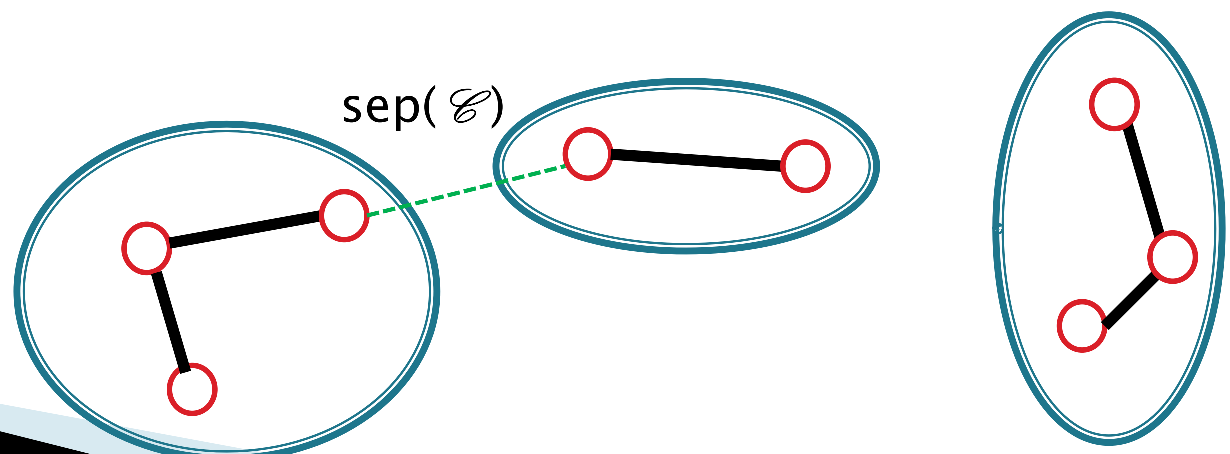
- La finalul algoritmului
  - $E(T') = \{e_1, \ldots, e_{n-k}\}$, cu $w(e_1) \leq \cdots \leq w(e_{n-k})$
  - **T '** este o pădure cu k componente conexe, vârfurile componentelor determinând clasele lui $\mathscr{C}$.



Aplicații – Clustering

Demonstrație

- La finalul algoritmului
  - $E(T') = \{e_1, \dots, e_{n-k}\}$, cu $w(e_1) \le \dots \le w(e_{n-k})$
  - **T '** este o pădure cu k componente conexe, vârfurile componentelor determinând clasele lui $\mathscr{C}$.
- sep($\mathscr{C}$) = min{w(e)| e = uv ∈ E(G) ce unește două componente conexe din T’}



Aplicații - Clustering

Demonstrație

- Atunci

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

Aplicații - Clustering

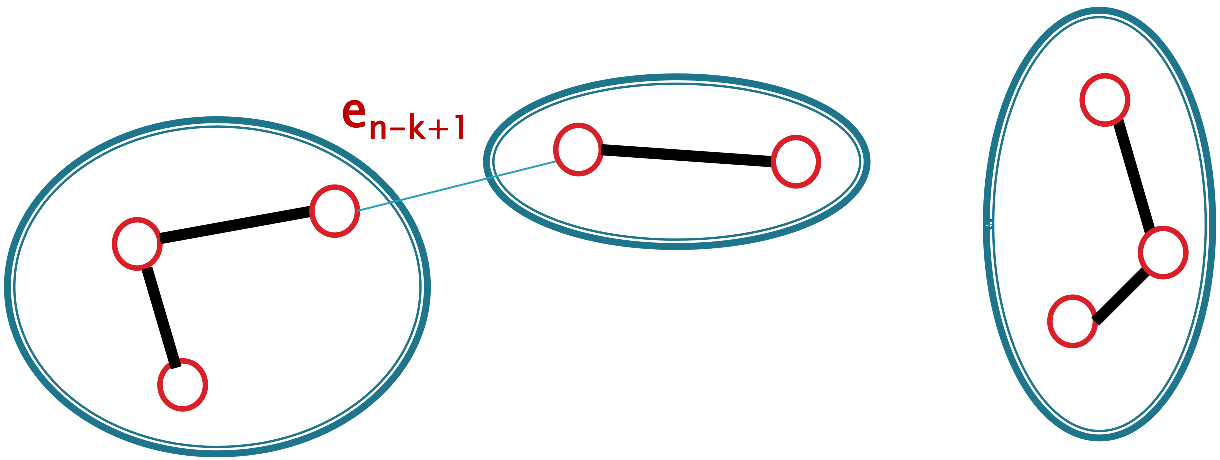
Demonstrație

- Atunci

$\text{sep}(\mathscr{C}) = w(e_{n-k+1})$, 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 $\mathscr{C}'$ cu

$$sep(\mathscr{C}') > sep(\mathscr{C})$$

Aplicații - Clustering

Demonstrație

- PRA că există un alt k-clustering $\mathscr{C}'$ cu

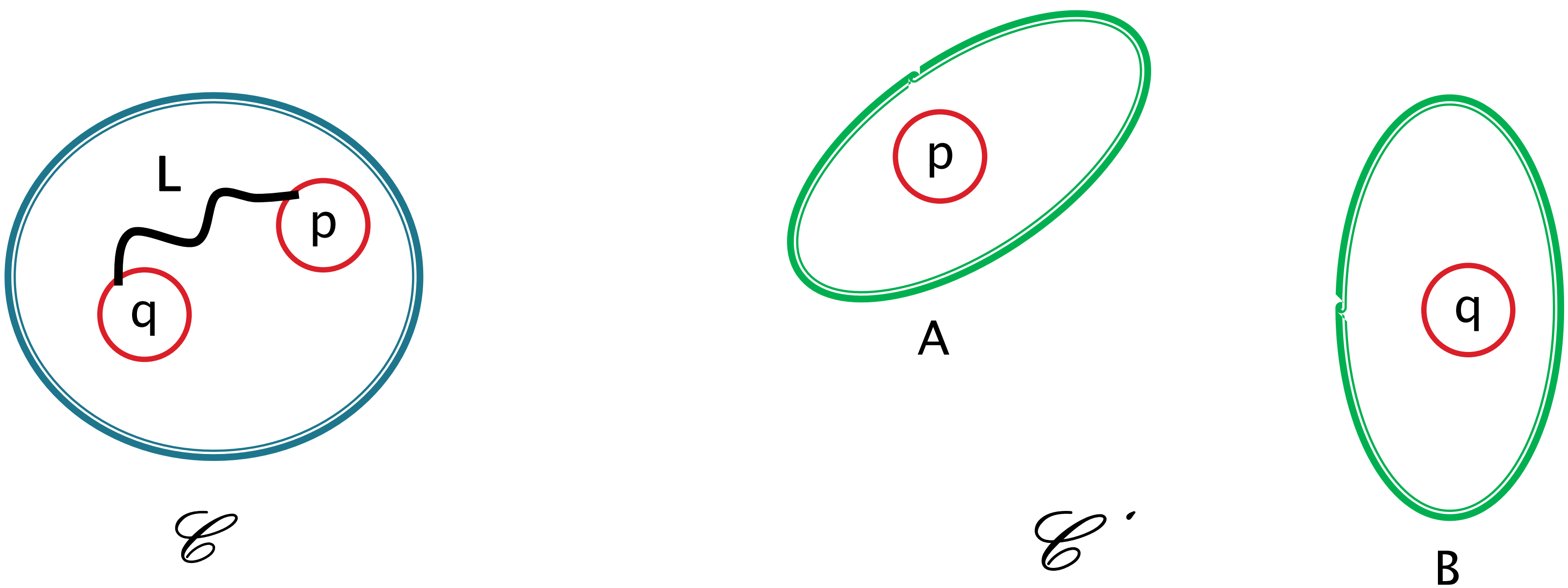
$$\mathrm{sep}(\mathscr{C}') > \mathrm{sep}(\mathscr{C})$$

- Atunci există două obiecte p și q care sunt
  - în aceeași clasă în $\mathscr{C}$,
  - în două clase diferite în $\mathscr{C}'$ - notate A, B

Aplicații - Clustering

Demonstrație

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



Aplicații – Clustering

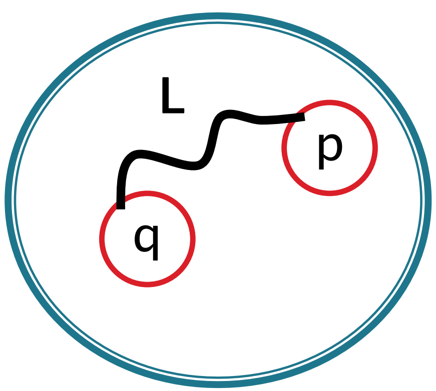
Demonstrație

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

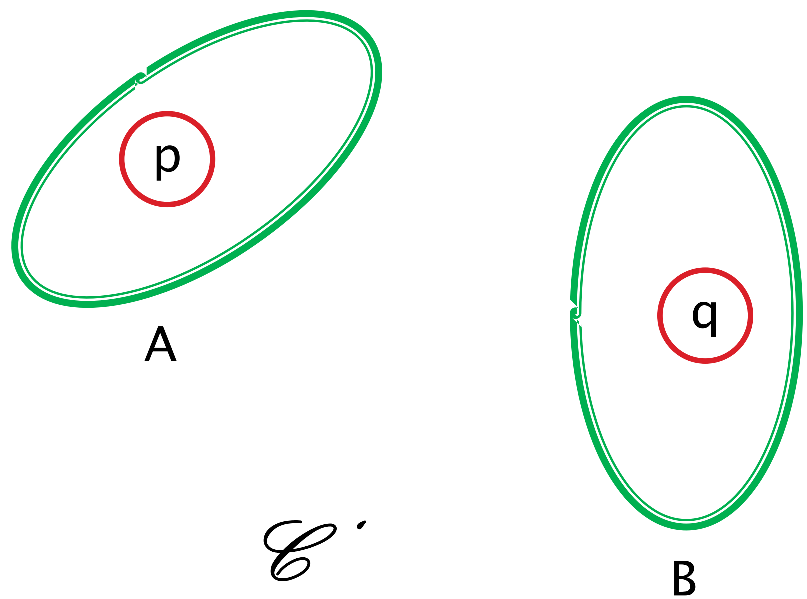
  $\Rightarrow$ în aceeași componentă a lui T’

  $\Rightarrow$ există L un lanț de la p la q **în T’**

  (cu muchii din mulțimea $= \{e_1, \dots, e_{n-k}\}$)



$\mathscr{C}$

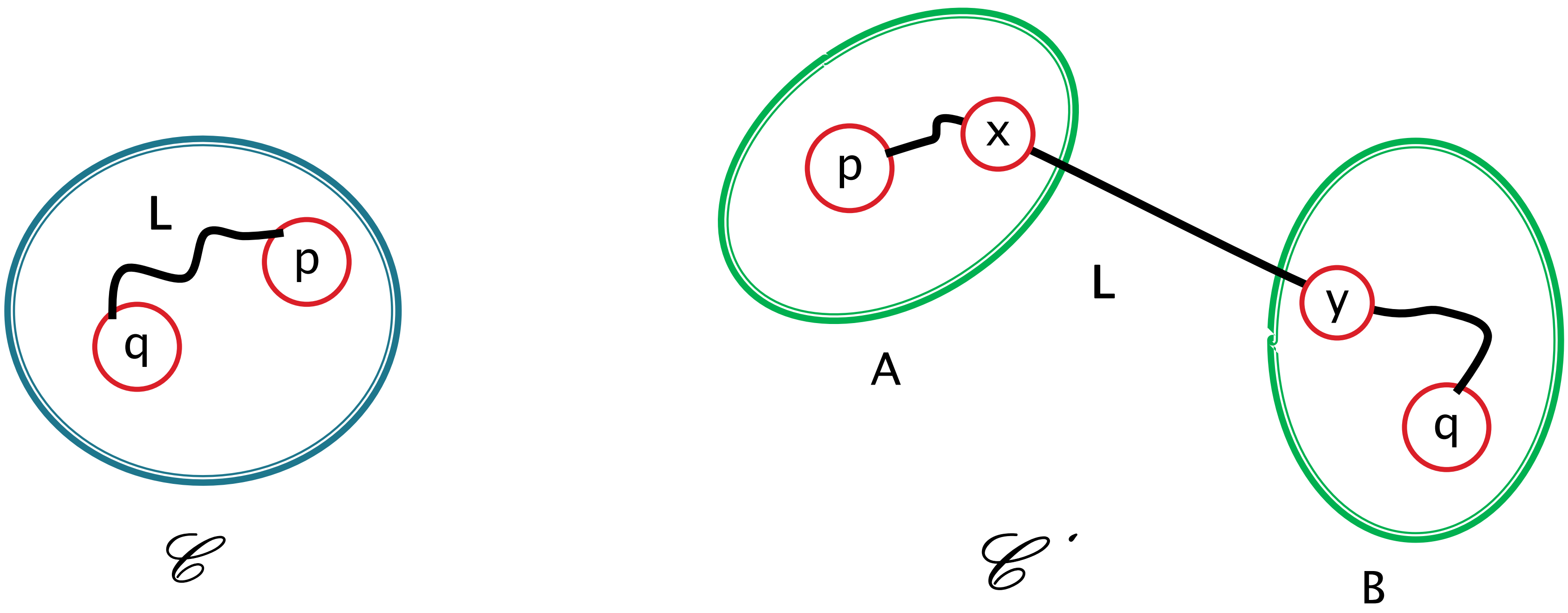


$\mathscr{C}'$

Aplicații - Clustering

Demonstrație

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

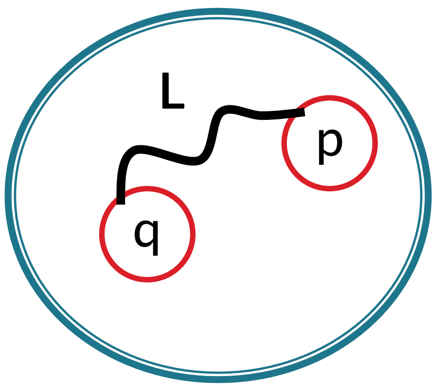


Aplicații - Clustering

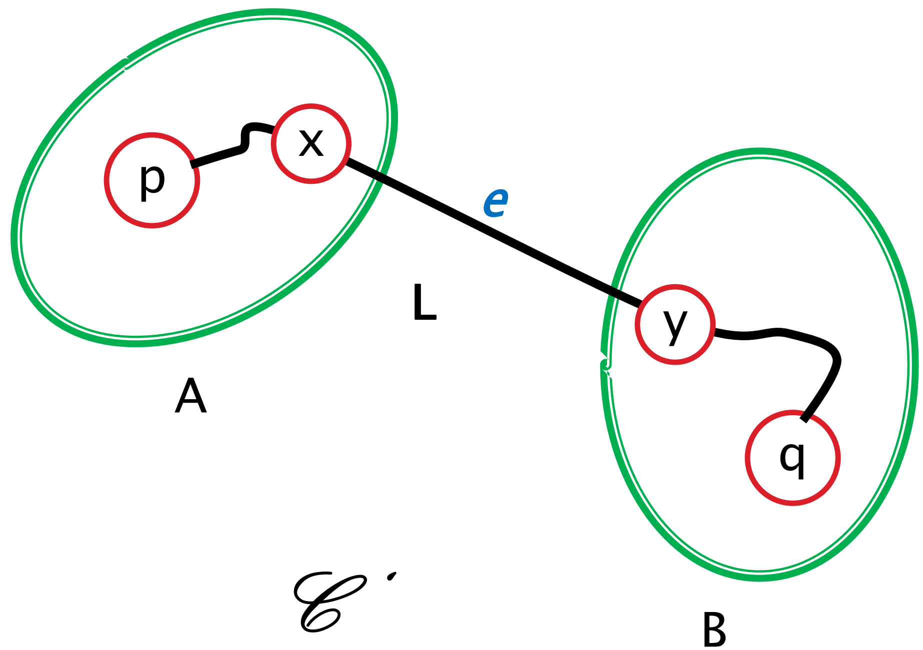
Demonstrație

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

  $\Rightarrow$ există în L o muchie e=xy cu o extremitate în A și cealaltă în B



$\mathscr{C}$



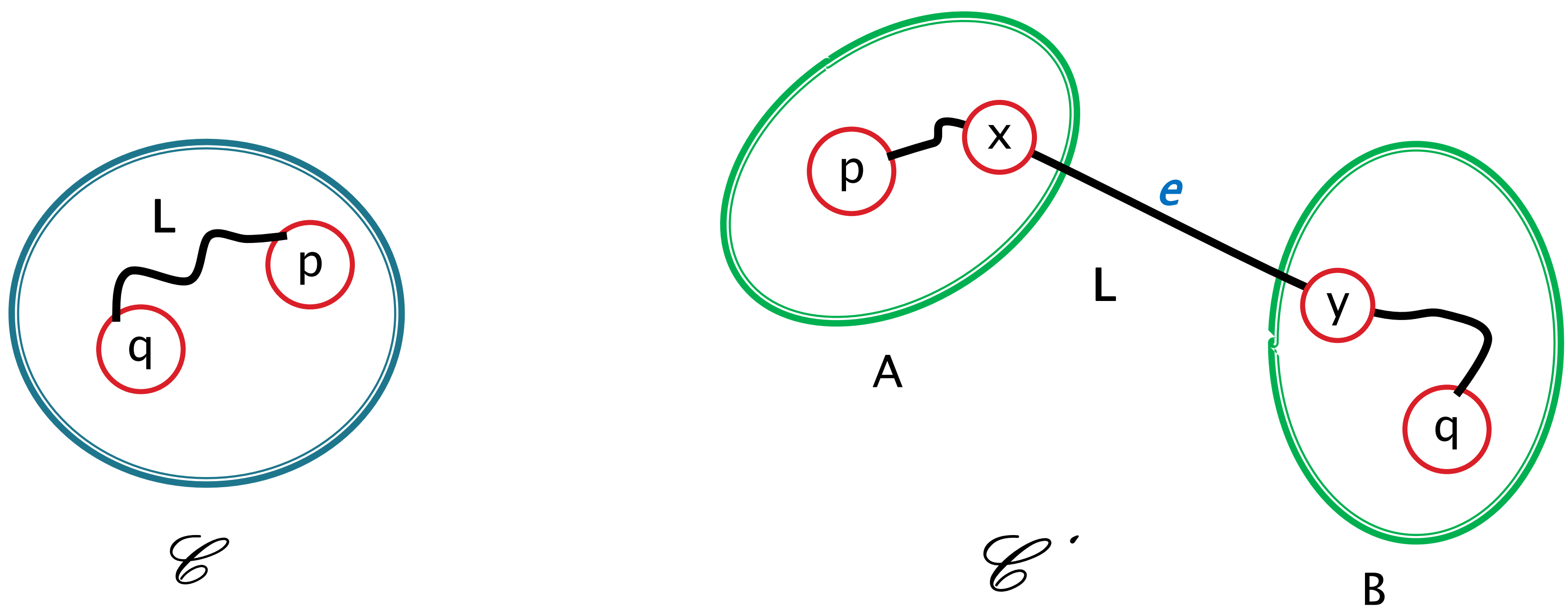
$\mathscr{C}'$

Aplicații - Clustering

Demonstrație

- Avem

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



Aplicații - Clustering

Demonstrație

- Avem

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

$e \in E(T')$