

# Projecte

## LP 2018-19



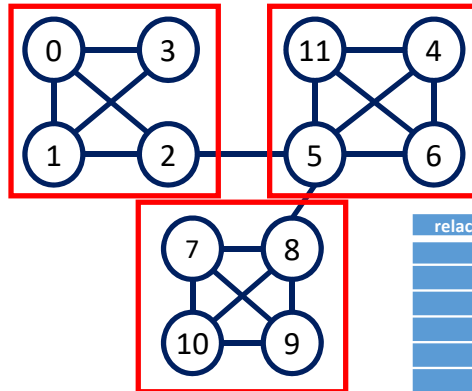
## Context

- Tenim implementada una matriu dispersa per representar la matriu d'ajacència d'un graf que representa una xarxa social.
- Ara volem crear el graf i calcular les comunitats que apareixen a ell.
- Hem escollit algorisme presentat a: **Aaron Clauset, MEJ Newman and Cristopher Moore. Finding community structure in a very large networks. Phys. Rev. E 70, 066111. December 2004. DOI: <https://doi.org/10.1103/PhysRevE.70.066111>**
- **Idea:** anem agrupant els nodes d'un graf (que representen persones) per construir comunitats de persones relacionades entre sí més que amb la resta.



# Idea General: Graf de la xarxa

Graf representa les relacions entre usuaris.



Matriu Adjacència

relacions	0	1	2	3	4	5	6	7	8	9	10	11
0		X	X	X								
1	X			X								
2	X	X				X						
3	X	X										
4						X	X					X
5					X		X		X			X
6					X	X						X
7								X	X	X		
8						X		X	X	X	X	
9								X	X		X	
10								X	X	X		
11					X	X	X					

Veiem 3 comunitats clares:

- 0,1,2,3
- 4,5,6,11
- 7,8,9,10

Matriu Adjacència  
Representada  
com a CSR

IniFiles	Cols	Valors
0	1	1
3	2	1
6	3	1
9	0	1
11	2	1
14	3	1
18	0	1
21	1	1
24	5	1
29	0	1
32	1	1
35	5	1
6	1	1
11	1	1
4	1	1
6	1	1
8	1	1
11	1	1
4	1	1
5	1	1
11	1	1
8	1	1
9	1	1
10	1	1
5	1	1
7	1	1
8	1	1
9	1	1
10	1	1
7	1	1
8	1	1
9	1	1
10	1	1
4	1	1
5	1	1
6	1	1

Matriu Adjacència  
Representada  
com a Coordenades

fila	col	valor
0	1	1
0	2	1
0	3	1
1	0	1
1	2	1
1	3	1
2	0	1
2	1	1
2	5	1
3	0	1
3	1	1
4	5	1
4	6	1
4	11	1
5	4	1
5	6	1
5	8	1
5	11	1
6	4	1
6	5	1
6	11	1
7	8	1
7	9	1
7	10	1
8	5	1
8	7	1
8	8	1
8	9	1
8	10	1
9	7	1
9	8	1
9	10	1
10	7	1
10	8	1
10	9	1
11	4	1
11	5	1
11	6	1

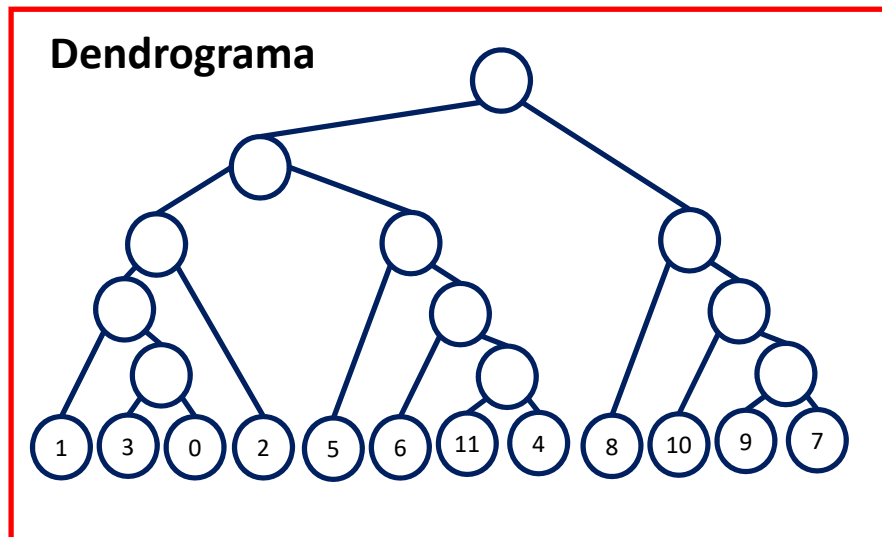
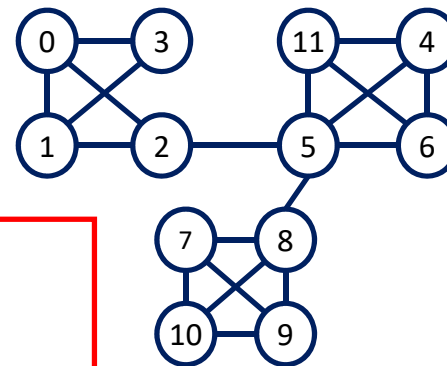


## Idea General: Dendrograma

Si comencem tenint una comunitat per cada node de la xarxa i a cada pas ajuntem dues comunitats fins a tenir una sola veiem que el resultat és un dendrograma.

Comunitats

1. 0,1,2,3
2. 4,5,6,11
3. 7,8,9,10



## Idea General: Algorisme

Com decidim si unim dues comunitats?

**Idea:** Mesurem si la construcció d'un grup de comunitats a un graf és bona. La idea intuïtiva diu que una separació bona en comunitats donarà grups de nodes que estan molt relacionats entre ells i poc relacionats amb la resta.

**Problema:** Però això donaria com a partició millor una que tingués tots els nodes a una mateixa comunitat.

**Solució:**

- Concepte de **Modularitat (Q)**: Per evitar-ho es defineix el concepte de modularitat que té en compte les relacions dels nodes però també les relacions dels nodes si haguessin estat creats de manera aleatòria. Així si les connexions són similars a les que tindríem si haguessin estat creades aleatòriament la Q donaria 0, i sinó un nombre real superior. Els valors de Q estaran entre -1 i 1.
- Concepte **d'increment de modularitat (deltaQ)**: Es defineix per poder fer els càlculs de forma eficient. Així calculem quin seria l'increment de la modularitat Q si dues comunitats veïnes s'unissin. Així fem aquest càlcul per totes les comunitats veïnes i escollim la unió que maximitzi aquest increment.



# Algorisme Càlcul Comunitats

## ALGORISME DE CÀLCUL COMUNITATS

### 1. Inicialitzar Dades:

**M2** = nombre d'arestes del graf \* 2.

**k[j]** = grau de cada node

**a[j]** =  $k[j]/m2$  grau normalitzat pel nombre d'arestes \* 2 :

**deltaQ[i,j]** =  $(1/m2) - ( (k[i]*k[j]) / (m2*m2) )$  increment modularitat si unim nodes (i,j)

**maxDeltaQ[i]** el màxim **deltaQ** indicant nodes (i,j)

**IndexComs** “llista” de comunitats actives. Inicialment serà tots els nodes del graf.

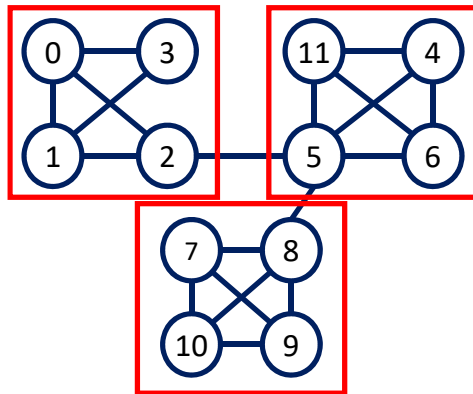
**Heap hTotal** amb tots els valors de **maxDeltaQ**.

vector **vDendrogrames** : Un arbre binari (amb un sol node) per cada node del graf.

**Q=0**. Modularitat total



# Algorisme Fusió Comunitats



Calculem:

**M2 = nombre d'arestes del graf \* 2 = 38.**

**k[j] = grau de cada node**

**a[j] = k[j]/m2 grau normalitzat pel nombre d'arestes \* 2 :**

	k
0	3
1	3
2	3
3	2
4	3
5	5
6	3
7	3
8	4
9	3
10	3
11	3

	A
0	3/38
1	3/38
2	3/38
3	2/38
4	3/38
5	5/38
6	3/38
7	3/38
8	4/38
9	3/38
10	3/38
11	3/38



# Algorisme Càlcul Comunitats

## ALGORISME DE CÀLCUL COMUNITATS

### 1. Inicialitzar Dades:

**M2** = nombre d'arestes del graf \* 2.

**k[j]** = grau de cada node

**a[j]** =  $k[j]/m2$  grau normalitzat pel nombre d'arestes \* 2 :

**deltaQ[i,j]** =  $(1/m2) - ( (k[i]*k[j]) / (m2*m2) )$  increment modularitat si unim nodes (i,j)

**maxDeltaQ[i]** el màxim **deltaQ** indicant nodes (i,j)

**IndexComs** “llista” de comunitats actives. Inicialment serà tots els nodes del graf.

**Heap hTotal** amb tots els valors de **maxDeltaQ**.

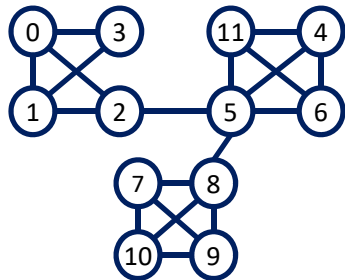
vector **vDendrogrames** : Un arbre binari (amb un sol node) per cada node del graf.

**Q=0**. Modularitat total





# Algorisme Fusió Comunitats

	deltaQ	deltaQ	deltaQ	deltaQ		deltaQ		k
0	1: 1/38-(3*3)/38^2	2:1/38-(3*3)/38^2	3:1/38-(3*2)/38^2				0	3
1	0:1/38-(3*3)/38^2	2:1/38-(3*3)/38^2	3:1/38-(3*2)/38^2				1	3
2	0:1/38-(3*3)/38^2	1:1/38-(3*3)/38^2	5:1/38-(3*5)/38^2				2	3
3	0:1/38-(2*3)/38^2	1:1/38-(2*3)/38^2					3	2
4	5:1/38-(3*3)/38^2	6:1/38-(3*3)/38^2	11:1/38-(3*3)/38^2				4	3
5	2:1/38-(5*3)/38^2	4:1/38-(5*3)/38^2	6:1/38-(5*3)/38^2	8:1/38-(5*4)/38^2	11:1/38-(3*3)/38^2		5	5
6	4:1/38-(3*3)/38^2	5:1/38-(3*5)/38^2	11:1/38-(3*3)/38^2				6	3
7	8:1/38-(3*4)/38^2	9:1/38-(3*3)/38^2	10:1/38-(3*3)/38^2				7	3
8	5:1/38-(4*5)/38^2	7:1/38-(4*3)/38^2	9:1/38-(4*3)/38^2	10:1/38-(3*3)/38^2			8	4
9	7:1/38-(3*3)/38^2	8:1/38-(3*4)/38^2	10:1/38-(3*3)/38^2				9	3
10	7:1/38-(3*3)/38^2	8:1/38-(3*4)/38^2	9:1/38-(3*3)/38^2				10	3
11	4:1/38-(3*3)/38^2	5:1/38-(3*5)/38^2	6:1/38-(3*3)/38^2				11	3

Calculam  $\text{deltaQ}[i,j] = (1/m^2) - ((k[i]*k[j]) / (m^2*m^2))$



# Algorisme Càlcul Comunitats

## ALGORISME DE CÀLCUL COMUNITATS

...

### 2. Cos algorisme

**Mentre** tinguem més d'una comunitat i no haguem acabat

**7.1** Escollim i esborrem el màxim al heap **htotal**.

**7.2** Si el màxim fa que la **Q** augmenti de valor

**7.2.1 Fusionem** les dues comunitats (veure FUSIO COMUNITATS)

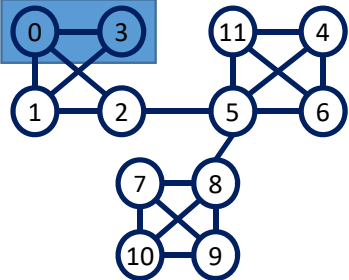
**7.3 Sinó hem acabat.**

**3.** Retornem una llista de **Dendrogrames** que recull els dendrogrames de **vDendrogrames** de les comunitats actives.



# Algorisme Fusió Comunitats

La deltaQ de 3,0 és la més gran → Fusionem

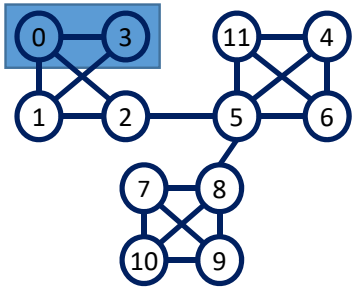
	deltaQ	deltaQ	deltaQ	deltaQ	deltaQ			k
0	1: 1/38-(3*3)/38^2	2:1/38-(3*3)/38^2	3:1/38-(3*2)/38^2			0	3	
1	0:1/38-(3*3)/38^2	2:1/38-(3*3)/38^2	3:1/38-(3*2)/38^2			1	3	
2	0:1/38-(3*3)/38^2	1:1/38-(3*3)/38^2	5:1/38-(3*5)/38^2			2	3	
3	0:1/38-(2*3)/38^2	1:1/38-(2*3)/38^2				3	2	
4	5:1/38-(3*3)/38^2	6:1/38-(3*3)/38^2	11:1/38-(3*3)/38^2			4	3	
5	2:1/38-(5*3)/38^2	4:1/38-(5*3)/38^2	6:1/38-(5*3)/38^2	8:1/38-(5*4)/38^2	11:1/38-(5*3)/38^2	5	5	
6	4:1/38-(3*3)/38^2	5:1/38-(3*5)/38^2	11:1/38-(3*3)/38^2			6	3	
7	8:1/38-(3*4)/38^2	9:1/38-(3*3)/38^2	10:1/38-(3*3)/38^2			7	3	
8	5:1/38-(4*5)/38^2	7:1/38-(4*3)/38^2	9:1/38-(4*3)/38^2	10:1/38-(4*3)/38^2		8	4	
9	7:1/38-(3*3)/38^2	8:1/38-(3*4)/38^2	10:1/38-(3*3)/38^2			9	3	
10	7:1/38-(3*3)/38^2	8:1/38-(3*4)/38^2	9:1/38-(3*3)/38^2			10	3	
11	4:1/38-(3*3)/38^2	5:1/38-(3*5)/38^2	6:1/38-(3*3)/38^2			11	3	



# Algorisme Fusió Comunitats

La deltaQ de 3,0 és la més gran → Fusionem

Mirem els veïns de 0:

	deltaQ	deltaQ	deltaQ	deltaQ	deltaQ		k
0	1: <b>RECALCULEM</b>	2:1/38-(3*3)/38^2	3:1/38-(3*2)/38^2			0	3
1	0: <b>RECALCULEM</b>	2:1/38-(3*3)/38^2	3: <b>ELIMINEM</b>			1	3
2	0:1/38-(3*3)/38^2	1:1/38-(3*3)/38^2	5:1/38-(3*5)/38^2			2	3
3	0:1/38-(2*3)/38^2	1:1/38-(2*3)/38^2				3	2
4	5:1/38-(3*3)/38^2	6:1/38-(3*3)/38^2	11:1/38-(3*3)/38^2			4	3
5	2:1/38-(5*3)/38^2	4:1/38-(5*3)/38^2	6:1/38-(5*3)/38^2	8:1/38-(5*4)/38^2	11:1/38-(5*3)/38^2	5	5
6	4:1/38-(3*3)/38^2	5:1/38-(3*5)/38^2	11:1/38-(3*3)/38^2			6	3
7	8:1/38-(3*4)/38^2	9:1/38-(3*3)/38^2	10:1/38-(3*3)/38^2			7	3
8	5:1/38-(4*5)/38^2	7:1/38-(4*3)/38^2	9:1/38-(4*3)/38^2	10:1/38-(4*3)/38^2		8	4
9	7:1/38-(3*3)/38^2	8:1/38-(3*4)/38^2	10:1/38-(3*3)/38^2			9	3
10	7:1/38-(3*3)/38^2	8:1/38-(3*4)/38^2	9:1/38-(3*3)/38^2			10	3
11	4:1/38-(3*3)/38^2	5:1/38-(3*5)/38^2	6:1/38-(3*3)/38^2	deltaQ(2,1): deltaQ(2,1)		11	3

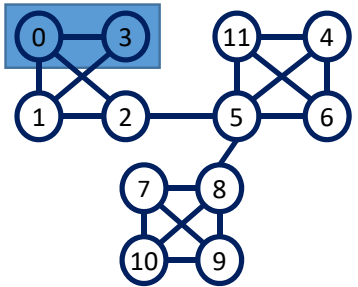
deltaQ(0,1)+ deltaQ(3,1).



# Algorisme Fusió Comunitats

La deltaQ de 3,0 és la més gran → Fusionem

Mirem els veïns de 0:

	deltaQ	deltaQ	deltaQ	deltaQ	deltaQ		k
0	1: RECALCULEM	2: RECALCULEM	3:1/38-(3*2)/38^2			0	3
1	0: RECALCULEM	2:1/38-(3*3)/38^2	3: ELIMINEM			1	3
2	0: RECALCULEM	1:1/38-(3*3)/38^2	5:1/38-(3*5)/38^2			2	3
3	0:1/38-(2*3)/38^2	1:1/38-(2*3)/38^2				3	2
4	5:1/38-(3*3)/38^2	6:1/38-(3*3)/38^2	11:1/38-(3*3)/38^2			4	3
5	2:1/38-(5*3)/38^2	4:1/38-(5*3)/38^2	6:1/38-(5*3)/38^2	8:1/38-(5*4)/38^2	11:1/38-(5*3)/38^2	5	5
6	4:1/38-(3*3)/38^2	5:1/38-(3*5)/38^2	11:1/38-(3*3)/38^2			6	3
7	8:1/38-(3*4)/38^2	9:1/38-(3*3)/38^2	10:1/38-(3*3)/38^2			7	3
8	5:1/38-(4*5)/38^2	7:1/38-(4*3)/38^2	9:1/38-(4*3)/38^2	10:1/38-(4*3)/38^2		8	4
9	7:1/38-(3*3)/38^2	8:1/38-(3*4)/38^2	10:1/38-(3*3)/38^2			9	3
10	7:1/38-(3*3)/38^2	8:1/38-(3*4)/38^2	9:1/38-(3*3)/38^2			10	3
11	4:1/38-(3*3)/38^2	5:1/38-(3*5)/38^2	6:1/38-(3*3)/38^2			11	3

$$\text{deltaQ}(2,0) - 2 A[3] \cdot A[2].$$

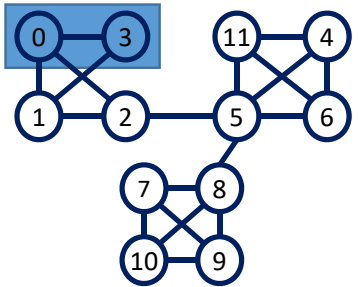
$$\text{deltaQ}(0,2) - 2 A[3] \cdot A[2].$$



# Algorisme Fusió Comunitats

La deltaQ de 3,0 és la més gran → Fusionem

Mirem els veïns de 0:

	deltaQ	deltaQ	deltaQ	deltaQ	deltaQ		k
0	1: RECALCULEM	2: RECALCULEM	3:1/38-(3*2)/38^2			0	3
1	0: RECALCULEM	2:1/38-(3*3)/38^2	3: ELIMINEM			1	3
2	0: RECALCULEM	1:1/38-(3*3)/38^2	5:1/38-(3*5)/38^2			2	3
3	0:1/38-(2*3)/38^2	1:1/38-(2*3)/38^2				3	2
4	5:1/38-(3*3)/38^2	6:1/38-(3*3)/38^2	11:1/38-(3*3)/38^2			4	3
5	2:1/38-(5*3)/38^2	4:1/38-(5*3)/38^2	6:1/38-(5*3)/38^2	8:1/38-(5*4)/38^2	11:1/38-(5*3)/38^2	5	5
6	4:1/38-(3*3)/38^2	5:1/38-(3*5)/38^2	11:1/38-(3*3)/38^2			6	3
7	8:1/38-(3*4)/38^2	9:1/38-(3*3)/38^2	10:1/38-(3*3)/38^2			7	3
8	5:1/38-(4*5)/38^2	7:1/38-(4*3)/38^2	9:1/38-(4*3)/38^2	10:1/38-(4*3)/38^2		8	4
9	7:1/38-(3*3)/38^2	8:1/38-(3*4)/38^2	10:1/38-(3*3)/38^2			9	3
10	7:1/38-(3*3)/38^2	8:1/38-(3*4)/38^2	9:1/38-(3*3)/38^2			10	3
11	4:1/38-(3*3)/38^2	5:1/38-(3*5)/38^2	6:1/38-(3*3)/38^2			11	3

$$\text{deltaQ}(2,0) - 2 A[3] \cdot A[2].$$

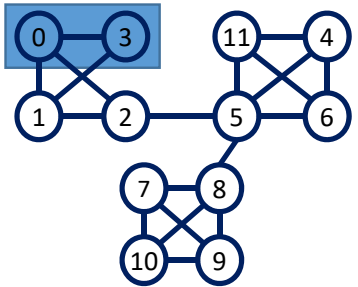
$$\text{deltaQ}(0,2) - 2 A[3] \cdot A[2].$$



# Algorisme Fusió Comunitats

La deltaQ de 3,0 és la més gran → Fusionem

Mirem els veïns de 0:

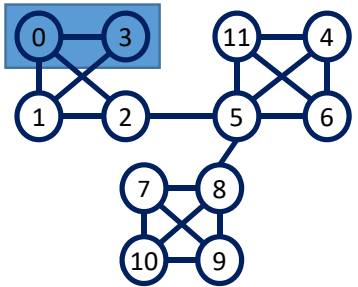
	deltaQ	deltaQ	deltaQ	deltaQ	deltaQ		k
0	1: RECALCULEM	2: RECALCULEM	3: ELIMINEM			0	3
1	0: RECALCULEM	2: 1/38-(3*3)/38^2	3: ELIMINEM			1	3
2	0: RECALCULEM	1: 1/38-(3*3)/38^2	5: 1/38-(3*5)/38^2			2	3
3	ELIMINEM					3	2
4	5: 1/38-(3*3)/38^2	6: 1/38-(3*3)/38^2	11: 1/38-(3*3)/38^2			4	3
5	2: 1/38-(5*3)/38^2	4: 1/38-(5*3)/38^2	6: 1/38-(5*3)/38^2	8: 1/38-(5*4)/38^2	11: 1/38-(5*3)/38^2	5	5
6	4: 1/38-(3*3)/38^2	5: 1/38-(3*5)/38^2	11: 1/38-(3*3)/38^2			6	3
7	8: 1/38-(3*4)/38^2	9: 1/38-(3*3)/38^2	10: 1/38-(3*3)/38^2			7	3
8	5: 1/38-(4*5)/38^2	7: 1/38-(4*3)/38^2	9: 1/38-(4*3)/38^2	10: 1/38-(4*3)/38^2		8	4
9	7: 1/38-(3*3)/38^2	8: 1/38-(3*4)/38^2	10: 1/38-(3*3)/38^2			9	3
10	7: 1/38-(3*3)/38^2	8: 1/38-(3*4)/38^2	9: 1/38-(3*3)/38^2			10	3
11	4: 1/38-(3*3)/38^2	5: 1/38-(3*5)/38^2	6: 1/38-(3*3)/38^2			11	3



# Algorisme Fusió Comunitats

La deltaQ de 3,0 és la més gran → Fusionem

Mirem els veïns de 0:

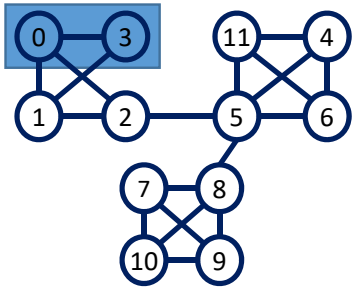
	deltaQ	deltaQ	deltaQ	deltaQ	deltaQ		k
0	1: RECALCULEM	2: RECALCULEM	3: ELIMINEM			0	3
1	0: RECALCULEM	2: 1/38-(3*3)/38^2	3: ELIMINEM			1	3
2	0: RECALCULEM	1: 1/38-(3*3)/38^2	5: 1/38-(3*5)/38^2			2	3
3	ELIMINEM					3	2
4	5: 1/38-(3*3)/38^2	6: 1/38-(3*3)/38^2	11: 1/38-(3*3)/38^2			4	3
5	2: 1/38-(5*3)/38^2	4: 1/38-(5*3)/38^2	6: 1/38-(5*3)/38^2	8: 1/38-(5*4)/38^2	11: 1/38-(5*3)/38^2	5	5
6	4: 1/38-(3*3)/38^2	5: 1/38-(3*5)/38^2	11: 1/38-(3*3)/38^2			6	3
7	8: 1/38-(3*4)/38^2	9: 1/38-(3*3)/38^2	10: 1/38-(3*3)/38^2			7	3
8	5: 1/38-(4*5)/38^2	7: 1/38-(4*3)/38^2	9: 1/38-(4*3)/38^2	10: 1/38-(4*3)/38^2		8	4
9	7: 1/38-(3*3)/38^2	8: 1/38-(3*4)/38^2	10: 1/38-(3*3)/38^2			9	3
10	7: 1/38-(3*3)/38^2	8: 1/38-(3*4)/38^2	9: 1/38-(3*3)/38^2			10	3
11	4: 1/38-(3*3)/38^2	5: 1/38-(3*5)/38^2	6: 1/38-(3*3)/38^2			11	3





# Algorisme Fusió Comunitats

Amb la nova configuració es torna a buscar la deltaQ i,j més gran → Fusionem i , j

	deltaQ	deltaQ	deltaQ	deltaQ	deltaQ		k
0	1: RECALCULEM	2: RECALCULEM	3: ELIMINEM			0	3
1	0: RECALCULEM	2:1/38-(3*3)/38^2	3: ELIMINEM			1	3
2	0: RECALCULEM	1:1/38-(3*3)/38^2	5:1/38-(3*5)/38^2			2	3
3	ELIMINEM					3	2
4	5:1/38-(3*3)/38^2	6:1/38-(3*3)/38^2	11:1/38-(3*3)/38^2			4	3
5	2:1/38-(5*3)/38^2	4:1/38-(5*3)/38^2	6:1/38-(5*3)/38^2	8:1/38-(5*4)/38^2	11:1/38-(5*3)/38^2	5	5
6	4:1/38-(3*3)/38^2	5:1/38-(3*5)/38^2	11:1/38-(3*3)/38^2			6	3
7	8:1/38-(3*4)/38^2	9:1/38-(3*3)/38^2	10:1/38-(3*3)/38^2			7	3
8	5:1/38-(4*5)/38^2	7:1/38-(4*3)/38^2	9:1/38-(4*3)/38^2	10:1/38-(4*3)/38^2		8	4
9	7:1/38-(3*3)/38^2	8:1/38-(3*4)/38^2	10:1/38-(3*3)/38^2			9	3
10	7:1/38-(3*3)/38^2	8:1/38-(3*4)/38^2	9:1/38-(3*3)/38^2			10	3
11	4:1/38-(3*3)/38^2	5:1/38-(3*5)/38^2	6:1/38-(3*3)/38^2			11	3



# Algorisme Càlcul Comunitats

## ALGORISME DE CÀLCUL COMUNITATS

### 1. Inicialitzar Dades:

**M2** = nombre d'arestes del graf \* 2.

**k[j]** = grau de cada node

**a[j]** =  $k[j]/m2$  grau normalitzat pel nombre d'arestes \* 2 :

**deltaQ[i,j]** =  $(1/m2) - ( (k[i]*k[j]) / (m2*m2) )$  increment modularitat si unim nodes (i,j)

**maxDeltaQ[i]** el màxim **deltaQ** indicant nodes (i,j)

**IndexComs** “llista” de comunitats actives. Inicialment serà tots els nodes del graf.

**Heap hTotal** amb tots els valors de **maxDeltaQ**.

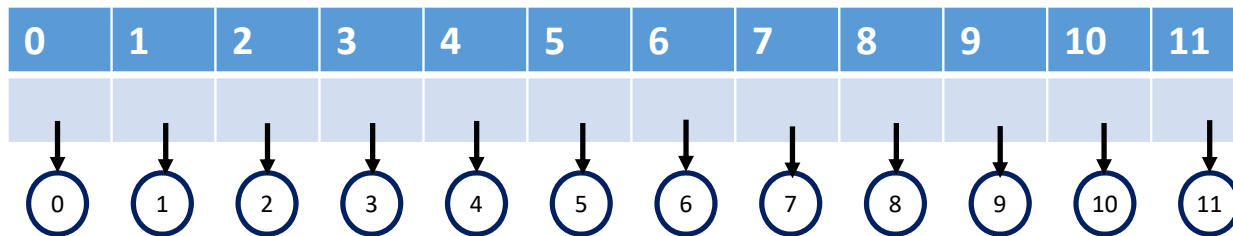
vector **vDendrogrames** : Un arbre binari (amb un sol node) per cada node del graf.

**Q=0**. Modularitat total

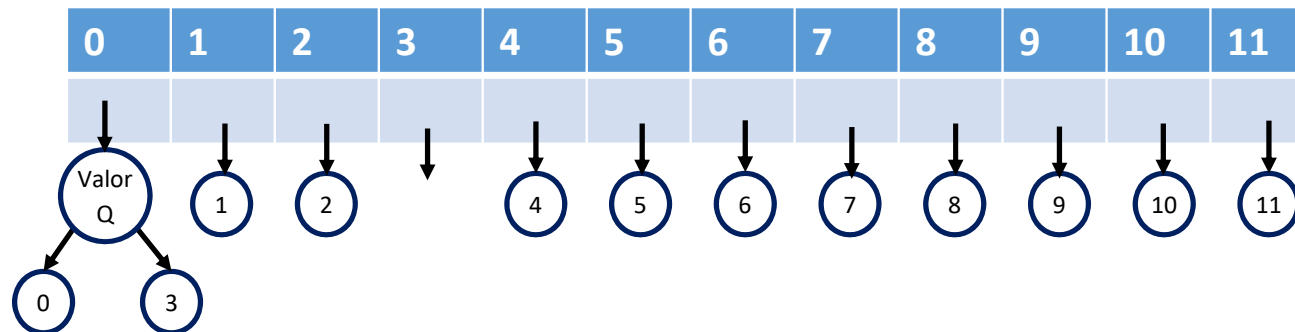


# Estructura de Dades

m\_vDendrogram



Si unim 3,0



# Algorisme Fusió Comunitats

**FUSIÓ COMUNITATS (i,j):** Eliminem comunitat i. Mantenim comunitat j com a fusió de (i,j).

## 1. Idea general:

- **Comunitat j resultant:** veïna de **veïns de i** i de **veïns de j**: Modifiquem o afegim els valors de  $\text{deltaQ}[j,x]$  per tots els x veïns de i i de j
- Veïns de i o de j ara ho seran només de j. Modifiquem o afegim els valors de  $\text{deltaQ}[x,j]$ , i eliminem els valors de  $\text{deltaQ}[x,i]$ .

### A. 1. Pels x veïns de i i j:

- Modifiquem  $\text{deltaQ}(x,j) = \text{deltaQ}(x,i) + \text{deltaQ}(x,j)$ .
- Esborrem  $\text{deltaQ}[x,i]$ .
- Modifiquem  $\text{deltaQ}(j,x) = \text{deltaQ}(i,x) + \text{deltaQ}(j,x)$ .

### A.2. Pels x veïns només de i:

- Afegim  $\text{deltaQ}(x,j) = \text{deltaQ}(x,i) - 2 A[j] * A[x]$ .
- Esborrem  $\text{deltaQ}(x,i)$ .
- Afegim  $\text{deltaQ}(j,x) = \text{deltaQ}(i,x) - 2 A[j] * A[x]$ .

### A.3. Pels x veïns només de j:

- Modifiquem  $\text{deltaQ}(x,j) = \text{deltaQ}(x,j) - 2 A[i] * A[x]$
- Modifiquem  $\text{deltaQ}(j,x) = \text{deltaQ}(j,x) - 2 A[i] * A[x]$ .

**Nota:** al fer totes aquestes modificacions, si cal, modificarem el  $\text{maxDeltaQ}(j,x)$  per tots els x veïns tractats a més del heap hTotal.



## Algorisme Fusió Comunitats

**FUSIÓ COMUNITATS (i,j):** Eliminem comunitat i. Mantenim comunitat j com a fusió de (i,j).

...

2. Eliminem la **comunitat i** de la “llista” de comunitats actives **IndexComs**.

3. Fusionem els arbres de les posicions **i** i **j** del vector **vDendrogrames** posant-los a la posició **j** del vector.

4. Recalculem la  $A[j]=A[j]+A[i]$



## Algorisme Fusió Comunitats

**FUSIÓ COMUNITATS (i,j):** Eliminem comunitat i. Mantenim comunitat j com a fusió de (i,j).

...

2. Eliminem la **comunitat i** de la “llista” de comunitats actives **IndexComs**.

3. Fusionem els arbres de les posicions **i** i **j** del vector **vDendrogrames** posant-los a la posició **j** del vector.

4. Recalculem la  $A[j]=A[j]+A[i]$

**Observació:** Veiem que el vector **k** ja no el necessitem i per això un cop calculada la **A** es podria fer un **clear()** de la **k** per tal de no tenir ocupada més memòria de la necessària.



# Classe Graph

```
class Graph
{public:
    Graph() {};
    Graph(string nomFitxerRels):mAdjacencia(nomFitxerRels){ };
    ~Graph() {};
    friend std::ostream& operator<<(std::ostream& os, const Graph& g)
        { os << g._mAdjacencia; return os; };
    void calculaComunitats(list<Tree<double>*>& listDendrogram);
    void clear() { m_mAdjacencia.clear(); }

private:
    MatriuSparse m_Adjacencia;
};
```

Volem afegir el mètode calculaComunitats: que definirà un element de tipus Comunitats i cridarà al mètode de la classe comunitats del mateix nom.

Alliberem tota la memòria del graf, esborrant tota la seva estructura.



## Classe Comunitat

```
class Comunitat
{public:    ...

private:
    vector<map<pair<int, int>, double>> m_deltaQ;
    vector<pair<int, int>> m_indexComs;
    vector<pair<int, double>> m_maxDeltaQFil;
    int m_primComdeltaQ;
    vector<Tree<double>*> m_vDendrograms;
    vector<int> m_k;
    vector<double> m_A;
    Heap m_hTotal;
    double m_Q;
    int m_M2;
    MatriuSparse* m_pMAdj;
};
```

Guardem un apuntador a una matriu per poder crear comunitats fora d'un graf i poder executar el joc de proves.





## Classe Comunitat: Atributs

- **m\_M2**: Nombre arestes graf \* 2
- **m\_Q**: Modularitat d'un agrupament en clústers donada. Inicialment val 0.
- **m\_k**: Graus dels nodes.
- **m\_A**: Proporcions d'arestes que arriben a una comunitat. Inicialment per un node **j** serà  $k[j]/m2$ .
- **m\_deltaQ**: vector que per cada posició **j** **conté la informació de l'increment de modularitat** que es produiria si uníssim **j** amb qualsevol de les seves veïnes. Aquesta informació s'ha de moficar cada cop que s'uneixi la comunitat **j** a una altre o cada cop que una veïna a **j** s'uneixi a una altra. Per això necessitem cercar a partir d'una parella de veïns donada **(j,x)**. Per això la guardarem a **un arbre binari balancejat (map de la stl)** a on tindrem un **pair<int,int>** amb la info dels índexs de la parella relacionada i el **valor double** del seu increment de modularitat. Els valors estaran ordenats segons **<i,j>**.
- **m\_maxDeltaQ**: vector que contindrà els valors màxims de cada fila de **m\_deltaQ**. Amb info dels nodes relacionats: **<pair<i,j>,deltaQ>**.
- **m\_hTotal**: Heap de màxims per mantenir la informació dels màxims de deltaQ de cada fila per tenir a dalt de tot el màxim, i poder-lo escollir a cada pas. Implementada a la Classe Heap. Té unes particularitats que ens permeten modificar-lo sense haver de reconstruir-lo a cada pas. Mireu la descripció de la classe més endavant.



## Classe Comunitat: Atributs

- **m\_primComdeltaQ**: Indica quina és la primera comunitat activa. Inicialitzat a 0.
  - **m\_indexComs**: Vector parelles de sencers. Simula una llista de índexs de comunitats actives, per eficiència utilitzem vector amb dos sencers i el sencer m\_primCom explicat a la línia superior. Inicialment tindrem una llista de tots els nodes del graf i aquest vector valdrà:
    - **indexComs[i].first=i-1 ;**
    - **indexComs[i].second=i+1.**Si unim i,j j queda activa i la i desapareix. La comunitat anterior a la i (**indexComs[indexComs[i].second]**) haurà d'estar encadenada amb la comunitat següent a la i (**indexComs[indexComs[i].first]**). Per això:
    - **indexComs[indexComs[i].first].second = indexComs[i].second**
    - **indexComs[indexComs[i].second].first = indexComs[i].first.**L'índex primCom s'actualitzarà només si la comunitat eliminada és la primera.
- Així podem recórrer només les posicions corresponents a comunitats actives del vector **deltaQ** seguint aquests índexs a partir de **m\_primComdeltaQ**.
- **m\_vDendrogrames**: vector apuntadors a Tree. Manté dendrograma que anem creant amb les unions de comunitats. Per cada posició j del vector tindrem l'arbre que representa la comunitat que s'està creant en aquella posició **vDendrogrames[j]= adreça del Tree**. Idea: unir arbres. Inicialment cada arbre serà un sol node representant al node del graf.



## Classe Comunitat: mètodes

```
class Comunitat
{public:Comunitat(MatriuSparse* pMAdj);
    ~Comunitat();
    void calculaM2() { m_M2 = m_pMAdj->getNValues(); };
    void calculaK() { m_pMAdj->calculaGrau(m_k); };
    void calculaA();
    void creaDeltaQHeap();
    void creaIndexComs();
    void InicialitzaDendrograms()
        { m_pMAdj->calculaDendrograms(m_vDendrograms); }
    void calculaComunitats(list<Tree<double>*>& listDendrogram);
    void fusiona(int com1, int com2);
    void modificaVei(int com1, int com2, int vei, int cas);
private:...
};
```



## Classe Comunitat: mètodes

```
class Comunitat
{public:...
    int getM2() { return m_M2; }
    vector<int> getK() { return m_k; }
    vector<double> getA() { return m_A; }
    vector<map<pair<int, int>, double>> getdeltaQ() {return m_deltaQ;}
    Heap gethTotal() {return m_hTotal;}
    vector<pair<int, int>> getIndexComs() { return m_indexComs; }
    void clear();

private:...
};
```



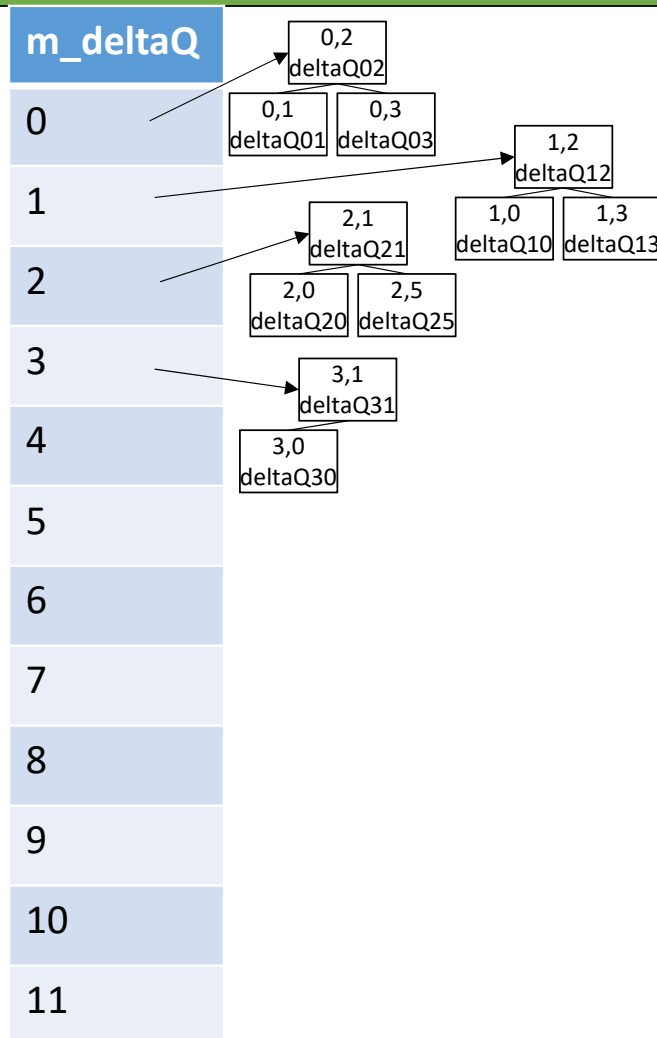
## main

```
int main()
{...
    MatriuSparse mXarxaCom(nomFitxer);
    Comunitat cXarxaCom(&mXarxaCom);
    cXarxaCom.calculaM2();
    int m2XarxaCom = cXarxaCom.getM2();
    cXarxaCom.calculaK();
    vector<int> vKXarxaCom = cXarxaCom.getK();
    cXarxaCom.calculaA();
    vector<double> vAXarxaCom = cXarxaCom.getA();
    cXarxaCom.creaDeltaQHeap();
    vector<map<pair<int, int>, double>> vDeltaQXaxaCom = cXarxaCom.getdeltaQ();
    Heap hXarxaCom = cXarxaCom.gethTotal();
    cXarxaCom.creaIndexComs();
    vector<pair<int, int>> vindexComsXarxaCom = cXarxaCom.getIndexComs();
    Graph gXarxaCom(nomFitxer);
    list<Tree<double>*> listDendrogramsXarxaCom;
    gXarxaCom.calculaComunitats(listDendrogramsXarxaCom);
```



# Classe Comunitat: Estructura de Dades

	m_A
0	3/38
1	3/38
2	3/38
3	2/38
4	3/38
5	5/38
6	3/38
7	3/38
8	4/38
9	3/38
10	3/38
11	3/38



m_maxDeltaQ
0
1
2
3
4
5
6
7
8
9
10
11

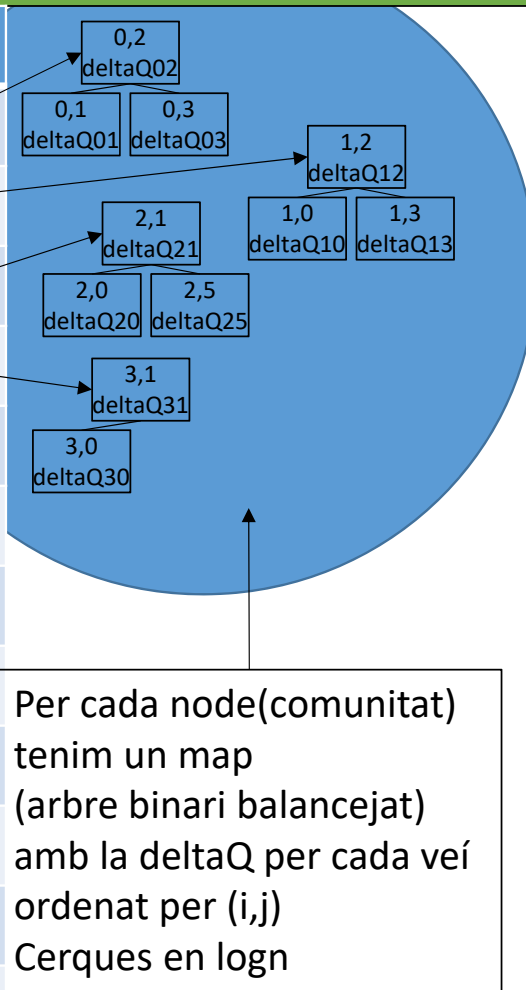
m_hTotal
0
1
2
3
4
5
6
7
8
9
10
11



# Estructura de Dades

	m_A
0	3/38
1	3/38
2	3/38
3	2/38
4	3/38
5	5/38
6	3/38
7	3/38
8	4/38
9	3/38
10	3/38
11	3/38

m_deltaQ
0
1
2
3
4
5
6
7
8
9
10
11



m_maxdeltaQ
0
1
2
3
4
5
6
7
8
9
10
11

m_hTotal
0
1
2
3
4
5
6
7
8
9
10
11



# Estructura de Dades

	m_A
0	3/38
1	3/38
2	3/38
3	2/38
4	3/38
5	5/38
6	3/38
7	3/38
8	4/38
9	3/38
10	3/38
11	3/38

m_deltaQ		m_maxdeltaQ	m_hTotal
0	0,2 deltaQ02	0	0 deltaQ30 3,0
1	0,1 deltaQ01   0,3 deltaQ03	1 3,deltaQ03	1 deltaQ03 0,3
2	2,1 deltaQ21   1,2 deltaQ12	2 2,deltaQ12	2 deltaQ20 2,0
3	2,0 deltaQ20   2,5 deltaQ25   1,0 deltaQ10   1,3 deltaQ13	3 0,deltaQ20	3
4	3,1 deltaQ31	4 0,deltaQ30	4
5	3,0 deltaQ30	5	5
6		6	6
7		7	7
8		8	8
9		9	9
10		10	10
11		11	11

Per cada comunitat de deltaQ  
Escollim el màxim i el  
posem a MaxDeltaQ  
Així no hem de recalculer tot:  
Si modifiquem deltaQ  
mirem si cal o no  
modificar MaxDeltaQ.  
Si no modifiquem MaxDeltaQ  
Tampoc modifiquem hTotal





# Estructura de Dades

	m_A
0	3/38
1	3/38
2	3/38
3	2/38
4	3/38
5	5/38
6	3/38
7	3/38
8	4/38
9	3/38
10	3/38
11	3/38

m_deltaQ	
0	0,2 deltaQ02
1	0,1 deltaQ01
2	0,3 deltaQ03
3	1,2 deltaQ12
4	2,1 deltaQ21
5	1,0 deltaQ10
6	2,0 deltaQ20
7	2,5 deltaQ25
8	3,1 deltaQ31
9	1,3 deltaQ13
10	3,0 deltaQ30
11	

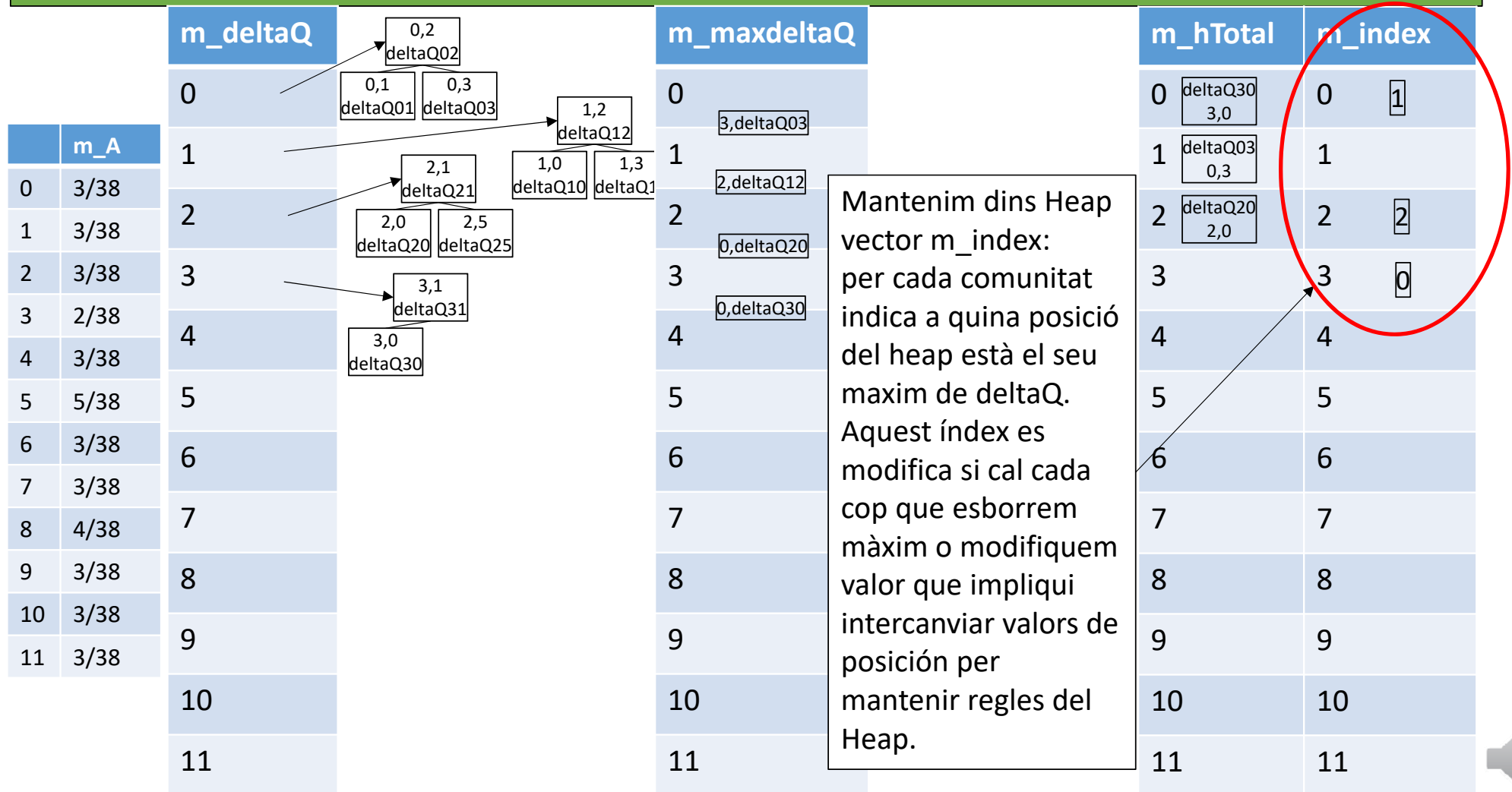
m_maxdeltaQ	
0	3,deltaQ03
1	2,deltaQ12
2	0,deltaQ20
3	0,deltaQ30
4	
5	
6	
7	
8	
9	
10	
11	

m_hTotal	
0	deltaQ30 3,0
1	deltaQ03 0,3
2	deltaQ20 2,0
3	
4	
5	
6	
7	
8	
9	
10	
11	

Afegim tots els valors  
De MaxDeltaQ el primer cop  
Després només eliminem  
el màxim ( $O(1)$ ).  
I modifiquem els que calgui.  
Però valors ordenats per  
deltaQ  
Com podem cercar un valor  
Segons la fila i modificar-lo?



# Estructura de Dades



# Estructura de Dades

Següent	m_indexComs		m_deltaQ	m_max deltaQ		m_hTotal	m_index	
	1	-1	0	0	3,deltaQ03	0	0	1
	2	0	1	1	2,deltaQ12	1	1	
	3	1	2	2	0,deltaQ20	2	2	2
	4	2		3	0,deltaQ30	3	3	0
	5	3		4		4	4	
	6	4		5		5	5	
	7	5		6		6	6	
	8	6		7		7	7	
	9	7		8		8	8	
	10	8		9		9	9	
	11	9		10		10	10	
	12	10		11		11	11	

m_A	
0	3/38
1	3/38
2	3/38
3	2/38
4	3/38
5	5/38
6	3/38
7	3/38
8	4/38
9	3/38
10	3/38
11	3/38

m_index	
0	1
1	
2	2
3	0
4	
5	
6	
7	
8	
9	
10	
11	

Si esborrem comunitat 3



# Estructura de Dades

m_indexComs		m_deltaQ		m_max deltaQ		m_hTotal		m_index	
1	-1	0	0,2 deltaQ02	0	3,deltaQ03	0	deltaQ30 3,0	0	1
2	0	1	0,1 deltaQ01	1	2,deltaQ12	1	deltaQ03 0,3	1	
4	1	2	2,1 deltaQ21	2	0,deltaQ20	2	deltaQ20 2,0	2	2
4	2	3	1,0 deltaQ10	3	0,deltaQ30	3		3	0
5	2	4	2,5 deltaQ25	4		4		4	
6	4	5	3,1 deltaQ31	5		5		5	
7	5	6	3,0 deltaQ30	6		6		6	
8	6	7		7		7		7	
9	7	8		8		8		8	
10	8	9		9		9		9	
11	9	10		10		10		10	
12	10	11		11		11		11	

m_A	
0	3/38
1	3/38
2	3/38
3	2/38
4	3/38
5	5/38
6	3/38
7	3/38
8	4/38
9	3/38
10	3/38
11	3/38



## Classe Matriu

Implementa els mètodes:

- `int getNValues() const` : ens retorna el nombre d'arestes que hi ha al graf.
- `void calculaGrau(vector<int>& graus) const`: calcula el grau de cada node i el retorna a un vector. Serà el nostre `m_k` inicial.
- `void creaMaps(vector<map<pair<int, int>, double>>& vMaps, vector<pair<int, int>>& indexComs) const`: Per cada node ens retorna un map amb un parell d'enters que representen els indexos de les comunitats (en aquest cas encara nodes) i un 0 com a `deltaQ`. És important que ho faci la matriu perquè és qui te la informació de les relacions entre nodes.
- `void calculaDendrograms(vector<Tree<double>*>& vDendrograms) const`: Genera els primers dendrogrames, és a dir un vector amb tantes posicions com nodes té el graf i un arbre a cada una que conté un sol node amb la informació de l'índex del node corresponent.
- `void clear()`: Allibera tota la memòria utilitzada per la matriu. Farà clear de tots els vectors o altres estructures que utilitzi.



# Classe Comunitats

## Implementa mètodes:

- void calculaM2(): Calcula  $m\_M2$ , el nombre d'arestes\*2 de la Matriu. Per fer-ho cridarem al mètode int getNValues() de la classe Matriu.

```
void calculaM2() { m_M2 = m_pMAdj->getNValues(); }
```

- void calculaK(); Calcula el vector  $m\_k$  dels graus dels nodes. Per fer-ho cridarem al mètode void calculaGrau(vector<int>& graus) const de la classe Matriu.

```
void calculaK() { m_pMAdj->calculaGrau(m_k); };
```

- void calculaA(): Calcula el vector  $m\_A$ , el grau normalitzat pel nombre d'arestes \* 2 :  $a[j] = k[j]/m2$



# Classe Comunitats

## Implementa mètodes:

- void creaDeltaQHeap(): Crea el vector m\_deltaQ. Per fer-ho necessitarà cridar al mètode creaMaps de la matriu d'adjacència, que generarà tots els elements (i,j) dels maps de cada fila. Un cop tingui això omplirà els valors de cada element del map calculant deltaQ. A mesura que es faci això anirem omplint el vector m\_maxDeltaQ amb els màxims corresponents a cada fila i afegint-los al heap m\_hTotal amb els seus valors inicials.

- Crea el vector m\_deltaQ:

- Crida mètode: void creaMaps(vector<map<pair<int, int>, double>>& vMaps, vector<pair<int, int>>& indexComs) const. que generarà tots els elements (i,j) dels maps de cada fila.
- Calcula valor de cada deltaQ d'un node i als seus veïns j.

$$\text{ValorDeltaQij} = (1/m^2) - (k[i]*k[j]) / (m^2*m^2)$$

- Inserim valor al map deltaQ[i] creant un element pair(pair(i,j),ValorDeltaQ)
- Mentre genero aquests valors per un node i, guardo valors del màxim per aquell node i al final el guardo a **maxDeltaQ[i]**, el màxim deltaQ calculat indicant entre quins nodes (i,j) s'ha produït aquest màxim.



# Classe Comunitats

## Implementa mètodes:

- void creaIndexComs(): Inicialitzem el vector de comunitats actives **IndexComs**, que tindrà una relació de totes les comunitats actives en un moment donat. Inicialment serà tots els nodes del graf.
  - **indexComs[indexComs[i].first].second = indexComs[i].second**
  - **indexComs[indexComs[i].second].first = indexComs[i].first.**
  - **primCom=0;**
- void InicialitzaDendrograms(){ m\_pMAdj->calculaDendrograms(m\_vDendrograms); }: Inicialitzarà el vector de dendrograms creant un arbre per cada node del graf. El vector tindrà tants elements com nodes te el graf original.





# Algorisme Fusió Comunitats: Classe Comunitat

```
void calculaComunitats(list<Tree<double>*>& listDendrogram);
```

Calcularà les comunitats i les retornarà a una llista d'apuntadors a arbres que seran els dendrogrames.

## 1. Inicialitzar Dades:

1. Calcula **m\_M2**
2. Calcula **m\_k**
3. Calcula **m\_A**
4. Calcula **m\_deltaQ**, **maxDeltaQ**, **m\_hTotal**
5. Inicialitza **m\_indexComs**
6. Crea **m\_vDendrograms**
7. Inicialitza **m\_Q = 0**.



# Algorisme Fusió Comunitats: Classe Comunitat

```
void calculaComunitats(list<Tree<double>*>& listDendrogram);
```

## 2. Cos algorisme

**Mentre** tinguem més d'una comunitat i no haguem acabat

**7.1** Escollim i esborrem el màxim al heap `m_htotal`.

**7.2** Si el màxim fa que la **Q** augmenti de valor

**7.2.1 Fusionem** les dues comunitats (veure FUSIO COMUNITATS)

**7.3 Sinó hem acabat.**

**3.** Construïm per retornar com a paràmetre per referència la llista de dendrogrames `listDendrogram`. Posem a la llista tots els elements del vector de dendrogrames `m_vDendrograms` que representen comunitats actives. Per saber quines son les comunitats actives utilitzem el vector `m_indexComs`.



# Algorisme Fusió Comunitats: Classe Heap

```
class Heap {
public:...
private:
    vector<ElemHeap> m_data;
    //Guardem indexs del veí inicial per cada un dels valors que tenim guardats
    vector<int> m_index;
    int m_maxEls; //indica nombre total de nodes: array va de 0 a m_maxEls-1
    int m_act; //indica posicio ultima ocupada: inicialment -1
    int pare(int pos) const { return ((pos - 1) / 2); }
    int fillEsq(int pos) const { return ((2 * pos) + 1); }
    int fillDret(int pos) const { return ((2 * pos) + 2); }
    void intercanvia(int pos1, int pos2);
    std::ostream& printRec(std::ostream& out, int pos, int n) const;
    void ascendir(int pos);
    void descendir(int pos);
};
```


```
class ElemHeap
{public:
private:
    double m_Val;
    pair<int, int> m_Pos;
};
```



# Algorisme Fusió Comunitats: Classe Heap

```
class Heap {  
public: Heap() { m_act = -1; m_maxEls = 0; };  
    Heap(int maxEls);  
    Heap(const Heap& h);  
    ~Heap() {};  
    Heap& operator=(const Heap& h);  
    ElemHeap& max() { return m_data[0]; }  
    int size() { return m_act+1; }  
    friend std::ostream& operator<<(std::ostream& out, const Heap& h);  
    void insert(const ElemHeap& el);  
    void resize(int mida);  
    void delMax();  
    void delElem(int pos);  
    void modifElem(const ElemHeap& nouVal);  
    void clear();  
    bool operator==(const Heap& h);  
private:  
    ...  
};
```

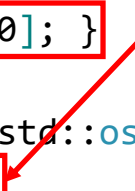
h.insert(ElemHeap({ max,{fila,colMax} }));



# Algorisme Fusió Comunitats: Classe Heap

```
class Heap {  
public: Heap() { m_act = -1; m_maxEls = 0; };  
    Heap(int maxEls);  
    Heap(const Heap& h);  
    ~Heap() {};  
    Heap& operator=(const Heap& h);  
    ElemHeap& max() { return m_data[0]; }  
    int size() { return m_act+1; }  
    friend std::ostream& operator<<(std::ostream& out, const Heap& h);  
    void insert(const ElemHeap& el);  
    void resize(int mida);  
    void delMax();  
    void delElem(int pos);  
    void modifElem(const ElemHeap& nouVal);  
    void clear();  
    bool operator==(const Heap& h);  
private:  
    ...  
};
```

h.insert(ElemHeap({ max,{fila,colMax} }));



## Algorisme Fusió Comunitats: Classe Heap

Guarda valors ElemHeap i manté un vector amb les posicions al vector `m_data` dels valors  $\langle i, j \rangle$  deltaQ per cada  $i$  que tenim. A on la  $i$  és el valor de la comunitat:

- `void resize(int mida)`: Fer un resize de mida per poder inicialitzar la mida del vector que guardarà les dades. Inicialment necessitem que sigui de la mida del nombre de nodes que té el graf.
- `void insert(const ElemHeap& e1)`: Permet inserir un element de tipus ElemHeap seguint les propietats de heap de màxims pel seu valor de deltaQ.
- `ElemHeap& max()`: retorna el màxim del heap.
- `void delMax()`: Elimina el màxim recomposant el Heap perquè compleixi les propietats de Heap de màxims.
- `void modifElem(const ElemHeap& nouVal)`: Modifica el valor  $j$  i deltaQ d'un element  $(i, j, \text{deltaQ})$ . La idea és que al heap tinguem una entrada per cada comunitat  $i$ , i sabem en tot moment a quina posició del vector es troba gràcies al vector `m_index`. Per això podem modificar els valors associats a una comunitat  $i$ . És a dir quina és la deltaQ màxima que pot generar en un moment donat  $i$  amb quina altre comunitat la genera. Aquesta modificació es fa mantenint les propietats de heap de màxims.



## Algorisme Fusió Comunitats: Classe ElemHeap

`class ElemHeap` Guarda els índexs (i,j) de comunitats i el seu valor deltaQ operadors comparació  
{`public`: nomes sobre el valor deltaQ. No podem cercar per valor (i,j) directament.

```
ElemHeap() { m_Val = 0; m_Pos = { 0, 0 }; }  
ElemHeap(double val, pair<int, int> pos) { m_Val = val; m_Pos = pos; };  
double getVal() const { return m_Val; }  
pair<int,int> getPos() const { return m_Pos; }  
bool operator<(const ElemHeap& e) { return (m_Val < e.m_Val); }  
bool operator<=(const ElemHeap& e) { return (m_Val <= e.m_Val); }  
bool operator>(const ElemHeap& e) { return (m_Val > e.m_Val); }  
bool operator>=(const ElemHeap& e) { return (m_Val >= e.m_Val); }  
bool operator==(const ElemHeap& e) { return ( m_Val == e.m_Val); }  
~ElemHeap() {};  
ElemHeap& operator=(const ElemHeap& e);  
friend std::ostream& operator<<(std::ostream& out, const ElemHeap& elHeap);  
private:  
double m_Val;  
pair<int, int> m_Pos;  
};
```



# Algorisme Fusió Comunitats: Classe Tree

```
template <class T>
class Tree {
public:
    Tree();
    Tree(const T& data);
    Tree(const Tree<T>& t);
    Tree(string nomFitxer);
    ~Tree();
    bool isLeave() const { return ((m_left == NULL) && (m_right == NULL)); }
    bool isEmpty() const { return (m_data == NULL); }
    Tree<T>* getRight(){return m_right;}
    Tree<T>* getLeft() { return m_left; }
    void setRight(Tree<T>* tR);
    void setLeft(Tree<T>* tL);
    T& getData() { return (*m_data); }
    friend std::ostream& operator<<(std::ostream& out, const Tree<T>& t);

private:
    Tree<T>* m_left;
    Tree<T>* m_right;
    Tree<T>* m_father;
    T* m_data;
    void TreeRec(ifstream& fitxerNodes, int h, Tree<T>* father);
    std::ostream& coutArbreRec(int n, std::ostream& out) const;
};
```





## Algorisme Fusió Comunitats: Classe Tree

Permet crear un arbre i assignar-li els seus fills esquerre i dret sense crear nous arbres. El que volem es poder crear el dendrograma sense haver de reconstruir cada cop els arbres. El que farem és crear els nodes intermedis però aprofitarem els ja creats per anar-los agrupant en arbres més grans. Per això hem definit els mètodes setLeft i setRight que no reserven nova memòria:

```
template<class T>
void Tree<T>::setLeft(Tree<T>* tL)
{
    m_left = tL;
    if (m_left != NULL)
    {
        //Fem que this sigui el pare de left
        m_left->m_father = this;
    }
}

template<class T>
void Tree<T>::setRight(Tree<T>* tR)
{
    m_right = tR;
    if (m_right != NULL)
    {
        //Fem que this sigui el pare de right
        m_right->m_father = this;
    }
}
```



# Lliurament a Caronte

**Fitxers necessiteu per poder implementar la pràctica:**

## [1. FitxersPerLliuramentFinal](#)

**Nomes heu de lliurar:**

1. Comunitat.cpp
2. Comunitat.h
3. MatriuSparse.cpp
4. MatriuSparse.h
5. Els fitxers associats a aquestes classes si és necessari (en principi no caldrien més).

