Tema 7

MF-Sets

Objetius

- Desenvolupament d'estructures de dades eficients per a agrupar n elements diferents en una col·lecció de k conjunts disjunts S ={S1, S2,…, Sk } amb dos tipus d'operacions:
  - Unió de dos conjunts disjunts
  - Cerca per a saber a quina conjunt pertany un element

Bibliografia

Michael T. Goodrich and Roberto Tamassia. “*Data Structures & Algorithms in Java*” (4th edition), John Wiley & Sons, 2005

(apartat 6 del capítol 11)

Continguts

1. Introducció
2. Representació de MF-Sets
3. Millores en la eficiència
   - Combinació per rang
   - Compressió de camis
4. Implementació

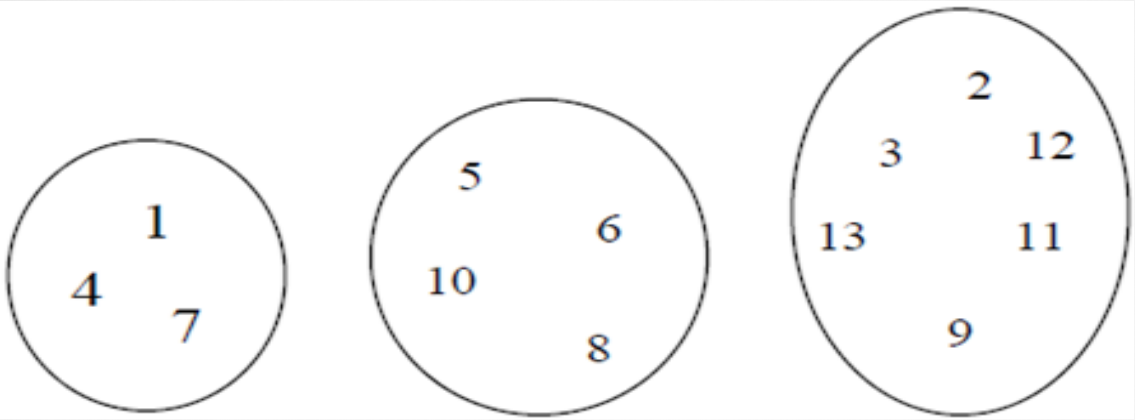
1. Introducció *Relacions d'equivalència*

- Una relació R definida en un conjunt C és un subconjunt del producte cartesià CxC de manera que aRb denota que (a, b) ∈ R
- R és una relació d'equivalència si compleix les següents propietats:
  - Reflexiva: aRa per a tot a ∈ C.
  - Simètrica: aRb si i solament si bRa, per a tot a, b ∈ C.
  - Transitiva: aRb i bRc implica aRc, per a tot a, b, c ∈ C.
- Un conjunt d'elements es pot particionar en classes d'equivalència a partir de la definició d'una relació d'equivalència

1. Introducció *MF-Sets*

- Els MF-Sets (Merge-Find Set) són unes estructures eficients per a determinar les possibles particions d'un conjunt
  - Els elements estan organitzats en subconjunts disjunts
  - El nombre d'elements és fix (no s'afigen ni s'esborren)
- Les seues operacions característiques són:
  - Combina (Merge): unió de dos conjunts disjunts
  - Troba (Find): donat un element ha de determinar al quin conjunt pertany

1. Introducció *Aplicacions dels MF-Sets*



Cada subconjunt es pot identificar per un dels seus membres

- Aplicacions:
  - Obtenció de l'arbre de recobriment de mínim pes en un graf no dirigit (Kruskal)
  - Components connexes d'un graf no dirigit
  - Equivalència entre autòmats finits

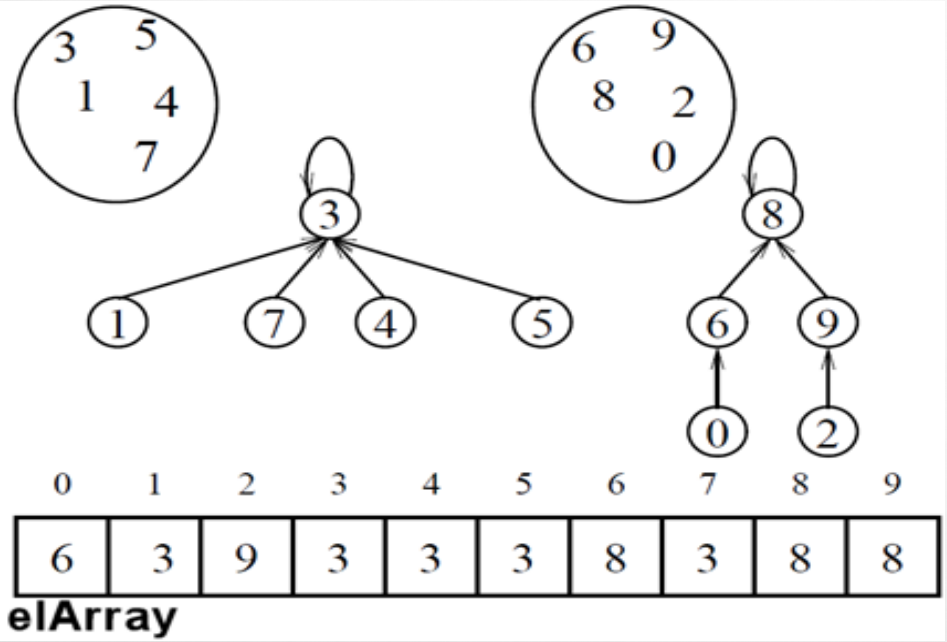
2. Representació dels *MF-Sets* *Operacions*

```
public interface MFSet
{
  /** Torna l'identificador del conjunt
   *  al que pertany l'element x
   */
  int find(int x);

  /** Uneix els conjunts als que pertanyen
   *  x i y
   */
  void merge(int x, int y);
}
```

2. Representació dels *MF-Sets* *Representació en bosc*

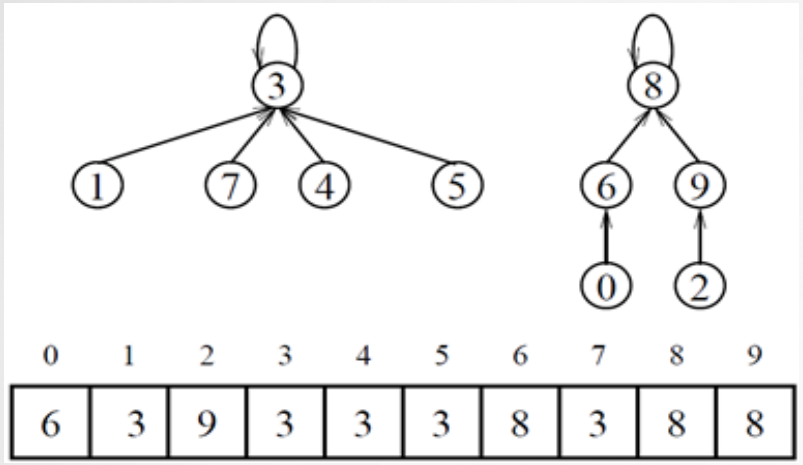
- Cada subconjunt es guarda com un arbre:
  - Els nodes de l'arbre són els elements del subconjunt
  - En cada node guardem una referència al pare
  - L'element arrel de l'arbre s'usa per a representar el subconjunt



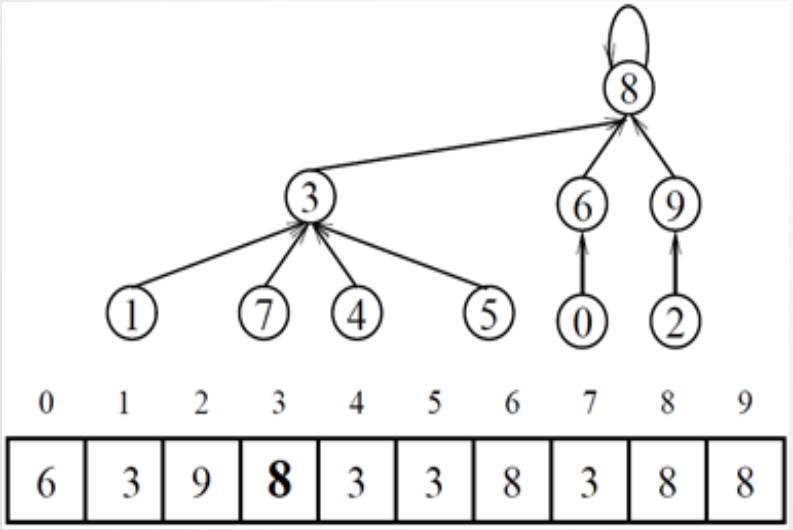
- *elArray*[*i*] és el pare de l'element *i*
- Si *elArray*[*i*]=*i*, *i* és la arrel d'un arbre

2. Representación de *MF-Sets* *Operacions*

Exemple



find(0) = 8
find(9) = 8
find(4) = 3
find(3) = 3



merge(3, 8)

3. Millores en la eficiència *Introducció*

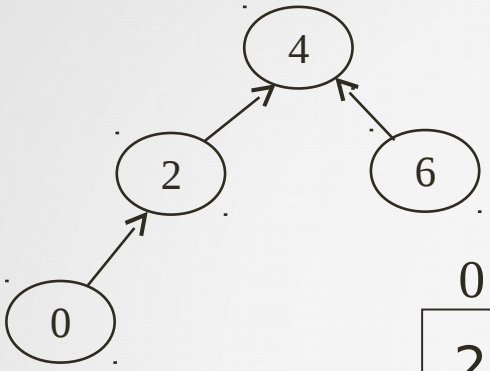
- El mètode find pot tenir un cost lineal en funció del nombre de nodes si els **arbres** estan **desequilibrats**
- Les millores en el cost es basen a reduir l'altura dels arbres:
  - Combinar per rang
  - Compressió de camins
- Amb aquestes millores el cost amortitzat de les operacions és pràcticament constant

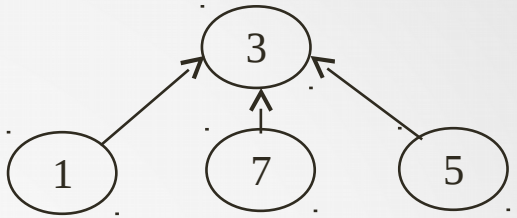
3. Millores en la eficiència *Combinació per altura*

- En l'operació merge fem que l'arrel de l'arbre de menor altura apunte a l'arrel del de major altura
  - Si les altures dels dos arbres a unir són diferents, l'altura de l'arbre resultant serà la de l'arbre de major altura
  - Si tots dos arbres tenen la mateixa altura, l'altura de l'arbre resultant serà una unitat major que la dels arbres a unir
- Per a açò és necessari guardar l'altura de cada arbre
  - Aquest valor es pot mantenir en el propi vector, en el node associat a l'arrel de cada arbre, però amb signe negatiu
    - Llavors, si elArray[i]<0, i és l'arrel d'un arbre i, a més
    - elArray[i] – 1 serà l'altura d'aquest arbre

3. Millores en la eficiència *Combinació per altura*

Exemple:

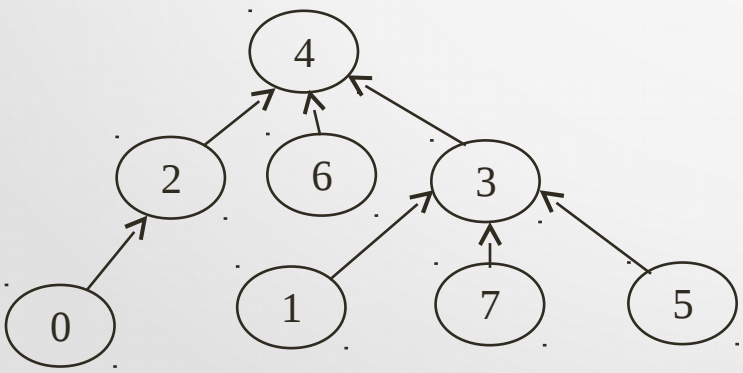




0	1	2	3	4	5	6	7
2	3	4	-2	-3	3	4	3

El 3 és l'arrel d'un arbre d'altura 1

El 4 és l'arrel d'un arbre d'altura 2



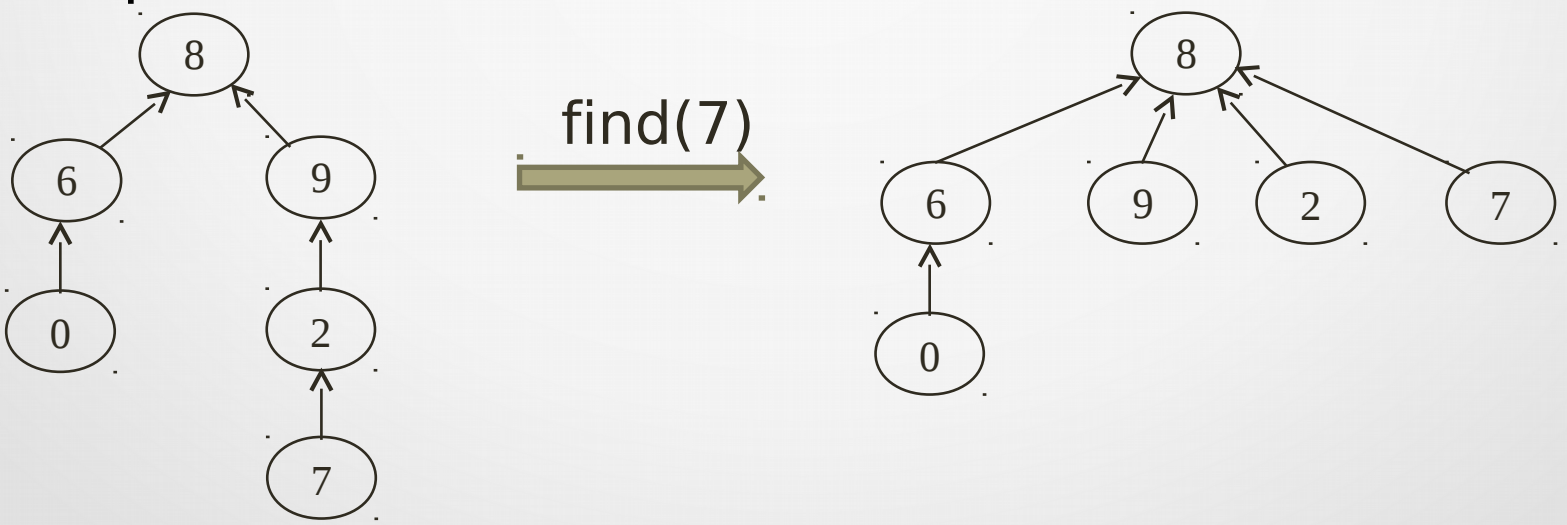
- En unir el dos arbres, pengem el de menor altura (el 3) del més alt (el 4)

0	1	2	3	4	5	6	7
2	3	4	4	-3	3	4	3

3. Millores en la eficiència *Compressió de camins*

- En les operacions de cerca (find) fem que cada node pel qual passem apunte directament a l'arrel de l'arbre.
- La compressió de camins no és totalment compatible amb la combinació per altura. Per açò s'usa la Unió per Rang que es basa en una estimació de l'altura. L'algorisme Merge és el mateix, però el valor que queda emmagatzemat com a altura no té per què ser exactament l'altura, és una estimació (que és cota superior de l'altura)

- Exemple:



4. Implementació *Atributs i constructor*

```
public class ForestMFSet implements MFSet {
  protected int talla;                    // Nº de elementos
  protected int elArray[];

  // Crea un MFSet de talla n. Al principi es creen n
  // arbres distints d'un sol element (altura 0)
  public ForestMFSet(int n) {
    talla = n;
    elArray = new int[talla];
    for (int i = 0; i < talla; i++)
      elArray[i] = -1;                    // Altura 0
  }
```

4. Implementació *Cerca d'elements*

```
/** Torna l'identificador del conjunt al que pertany
  * l'element x, a mes d'enllacar totos els elements
  * del cami visitat directament amb l'arrel */
  public int find(int x) {
      if (elArray[x]<0)
          return x;
      else
          return elArray[x]=find(elArray[x]);
  }
```

4. Implementació *Unió per Rang*

```
/** Uneix els conjunts als que pertanyen x i y */
public void merge(int x, int y) {
  int raizX = find(x);
  int raizY = find(y);
  if (raizX != raizY) {  // Unim les particions
     if (elArray[raizX]==elArray[raizY]) {// Altures iguals
      elArray[raizX] = raizY;              // Pengem x de y
      elArray[raizY]--;      // Incrementem la altura de y
     } else if (elArray[raizX] < elArray[raizY])
             elArray[raizY] = raizX;  // Pengem y de x
            else
             elArray[raizX] = raizY;  // Pengem x de y
  }
}
```