

# Algoritmi e Strutture Dati

## Lezione 18

3 novembre 2025

Code con priorità

## Code con priorità

- Collezioni di dati da cui gli elementi vengono prelevati secondo un *criterio di priorità*
- Ogni elemento ha una *chiave*:  
Chiave più bassa indica priorità maggiore

## Code con priorità: operazioni

- **findMin()**

Restituisce l'elemento minimo della coda (senza rimuoverlo)

- **deleteMin()**

Restituisce l'elemento minimo della coda e lo rimuove

- **insert(elemento e, chiave k)**

Inserisce nella coda un elemento *e* con associata una chiave (priorità) *k*

- **delete(elemento e)**

Cancella l'elemento *e*

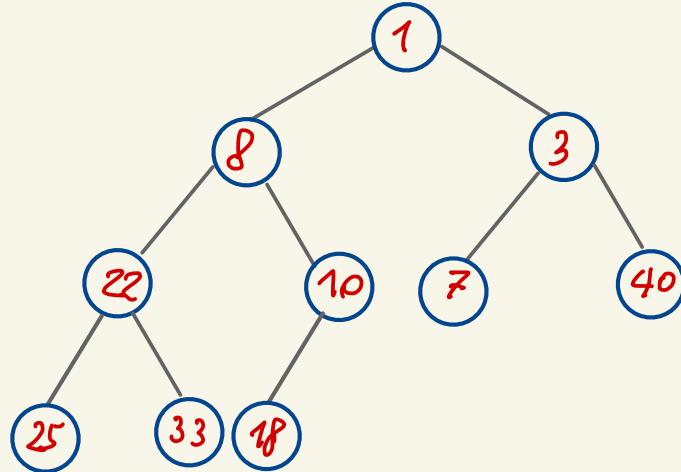
- **changeKey(elemento e, chiave d)**

Modifica la priorità dell'elemento *e*, assegnando come nuovo valore *d*

# Code con priorità

Implementazione mediante *MinHeap*

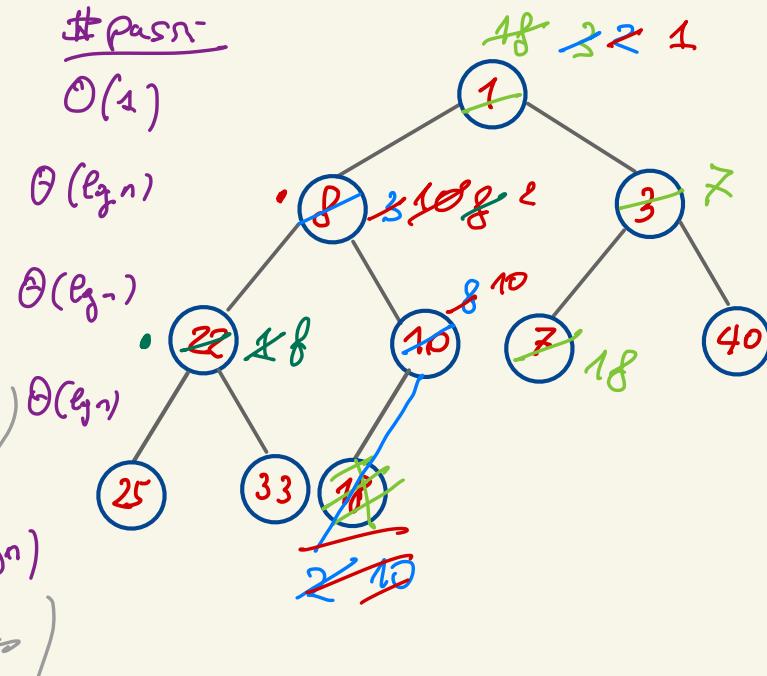
- `findMin()`
- `deleteMin()`
- `insert(elemento e, chiave k)`
- `delete(elemento e)`
- `changeKey(elemento e, chiave d)`



# Code con priorità

Implementazione mediante *MinHeap*

- **findMin()** controllo radice  $\Theta(1)$
  
  - ■ **deleteMin()** ~~delete Min()~~ (*risistema*)
  - ■ **insert(elemento e, chiave k)** ~~insert(..., 2)~~ (*risistema del Basso*)
  - ■ **delete(elemento e)** ~~delete (elem nolo.)~~ (*risistema risistem del Basso*)
  - ■ **changeKey(elemento e, chiave d)** ~~changeKey (elem nolo., 1)~~ (*risistem - risistem del Basso*)  $\Theta(\lg n)$
- $\Theta(\lg n)$  conosciuto
- E- posizione dell'elemento da cancellare o modificare



# Analisi Ammortizzata

Quanto costa incrementare un contatore a  $n$  bit?

costo =  $n^{\circ}$  su  $n$  bit da ispezionare

$$\begin{array}{r} 0010111 \\ \text{---} \\ 0011000 \end{array}$$

$$\begin{array}{r} 111111 \\ \text{---} \\ 000000 \end{array}$$

ALGORITMO incrementaContatore (Array  $v[0..n-1]$  di bit)

$i \leftarrow 0$

WHILE  $i < n$  AND  $v[i] = 1$  DO

$v[i] \leftarrow 0$   
 $i \leftarrow i + 1$

IF  $i < n$  THEN  $v[i] \leftarrow 1$

- costo costante  $\Theta(n)$

- costo di  $K$  operazioni di incremento  
sullo stesso contatore  $\Theta(Kn)$

## Esempio

# bit  
modificati:

0 000000

il bit 0 cambia ogni volta

1 000001

1

" " 1 cambia ogni 2 volte

2 000010

2

" " 2 " " 2 volte

3 000011

1

!

4 000100

3

il bit i cambia ogni 2 volte

5 000101

1

6 000110

2

7 000111

1

8 001000

4

:

$$\frac{T(n, k)}{n} = k + \left\lfloor \frac{k}{2^1} \right\rfloor + \left\lfloor \frac{k}{2^2} \right\rfloor + \dots + \left\lfloor \frac{k}{2^{n-1}} \right\rfloor$$

$$\sum_{i=0}^{\infty} q^i = \frac{1}{1-q}$$

0 < q < 1

n° di operazioni  
 per incrementare  
 k volte un  
 carattere q  
 n bit, che  
 parli da 0

$$= \sum_{i=0}^{n-1} \left\lfloor \frac{k}{2^i} \right\rfloor \leq \sum_{i=0}^{n-1} \frac{k}{2^i} = k \sum_{i=0}^{n-1} \frac{1}{2^i}$$

$$\leq k \sum_{i=0}^{\infty} \frac{1}{2^i} = k \sum_{i=0}^{\infty} \left(\frac{1}{2}\right)^i = k \frac{1}{1 - \frac{1}{2}} = 2k$$

$$\overbrace{T_{\alpha, n}(k)}^{\text{costo}} = \frac{T(n, k)}{k} = \frac{2k}{k} = 2$$

costo  
 di moltiplicazione

## Rappresentazione di partizioni

union / find

Insieme  $\alpha$

PARTIZIONE di  $\alpha$ :

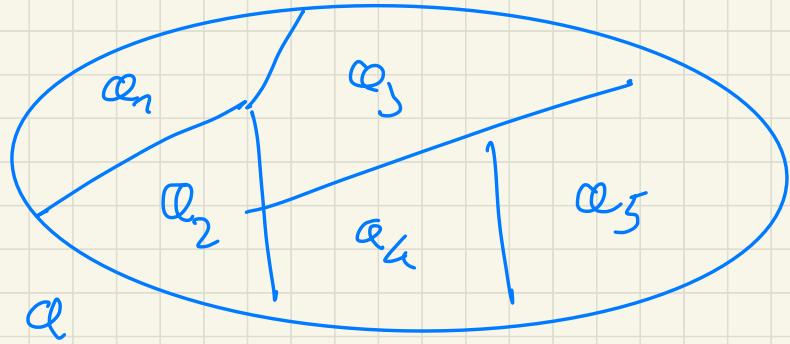
Famiglia di sottoinsiemi

$$\alpha_1, \alpha_2, \dots, \alpha_k \subseteq \alpha$$

t.c.

- per  $i = 1, \dots, k$ ,  $\alpha_i \neq \emptyset$
- per  $i, j = 1, \dots, k$ , se  $i \neq j$  allora  $\alpha_i \cap \alpha_j = \emptyset$
- $\bigcup_{i=1}^k \alpha_i = \alpha$

$\alpha_1, \alpha_2, \dots, \alpha_k$  si chiamano PARTI



• S: collezione di elementi fra loro distinti esempio: interi da 1 a n

• ORGANIZZAZIONE INIZIALE:

n insiemi disgiunti:

$\{1\} \{2\} \dots \{n\}$

• OGNI INSIEME HA UN NOME

nome iniziale  $\{i\} \rightarrow i$

## Problema UNION-FIND

Rappresentare una collezione di insiemi disgiunti,  
con le operazioni:

- $\text{UNION}(A, B)$

unisce gli insiemi  $A$  e  $B$

in unico insieme di nome  $A$

- $\text{FIND}(x)$

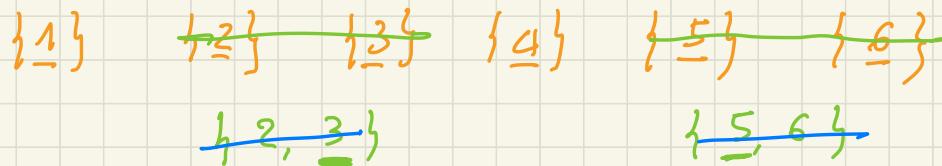
restituisce il nome dell'insieme  
che contiene l'elemento  $x$

- $\text{MAKESET}(x)$

crea un nuovo insieme  $\{x\}$  di nome  $x$   
con  $x$  nuovo elemento

## UNION-FIND : esempio

MAKESET(1), MAKESET(2), ..., MAKESET(6)



{2, 3, 5, 6}

$S = \{1, 2, \dots, 6\}$

Partizioni

{1} {2} ... {6}

Find(2)  $\rightarrow$  2

UNION(3,2)

UNION(5,6)

Find(2)  $\rightarrow$  3

UNION(5,3)

Find(2)  $\rightarrow$  5

Partizioni { {1}, {4} } {2, 3, 5, 6 }

## VARI TIPI DI SOLUZIONE

Soluzioni elementari

Soluzioni evolute

In tutte le soluzioni presentate:

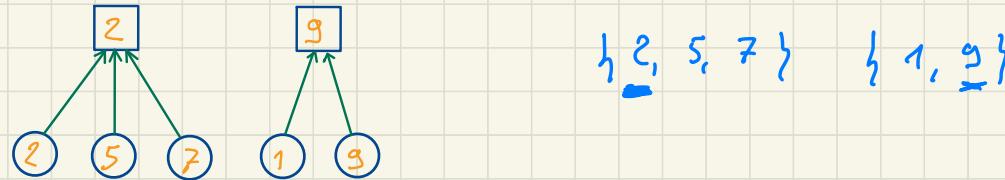
- Ogni insieme (parte) è rappresentato da un albero con radice
  - ✓ NODI → elementi dell'insieme
  - ✓ RADICE → nome dell'insieme
- partizione: foreste gli alberi

ponfiorci verso l'alto

# ALGORITMI "QuickFind"

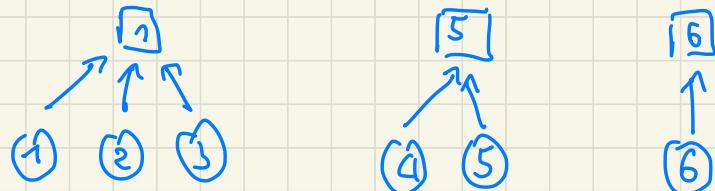
- ALBERI DI ALTEZZA 1
- Elementi dell'insieme  $\leftrightarrow$  FOGLIE
- Nome dell'insieme  $\leftrightarrow$  RADICE

Esempio



Esempio

$\{ \underline{1}, 2, 3 \}$      $\{ 4, \underline{5} \}$      $\{ \underline{6} \}$



"QuickFind": operazioni:

MAKESET (elemento e)



Fermo

MAKESET (?)

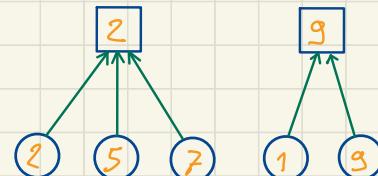
$$T(n) = \Theta(1)$$



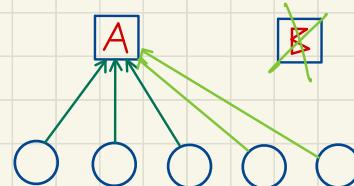
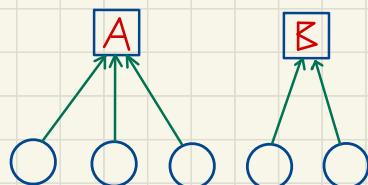
FIND (elemento e) → nome

$$T(n) = \Theta(1)$$

FIND (5)



UNION (nome A, nome B)



$$T(n) = \Theta(n)$$



$$\text{Spazio} = \Theta(n)$$

Caso peggior:  
B contiene n-1 nodi

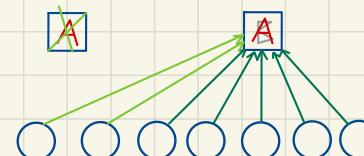
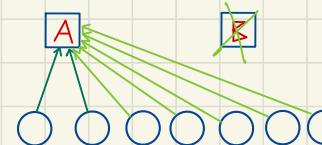
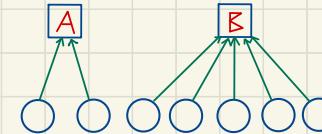
# "QuickFind" con bilanciamento

UNION (nome A, nome B)

Quando  $\#B > \#A$

andré collegare gli  
elementi di B  
sotto A

conviene collegare gli  
- elementi di A sotto B  
- rinominare la radice



Se  $\#B > \#A \rightarrow$

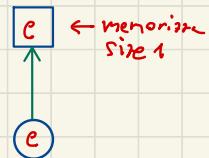
Caso peggiore  $\frac{n}{2}$  nodi in A e  $\frac{n}{2}$  nodi in B

$$\mathcal{T}(n) = O(n)$$

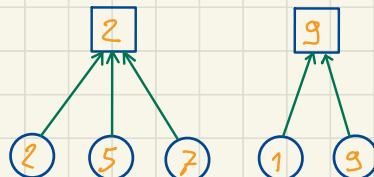
"QuickFind" con bilanciamento: operazioni:

Nella radice si memorizza il numero di elementi dell'insieme (SIZE)

MAKESET (elemento e)



FIND (elemento e) → nome



**UNION ( nome A, nome B )**

IF  $\text{size}(A) \geq \text{size}(B)$  THEN

| Sposta i puntatori delle foglie  
di B sotto  
| rinomini la radice di B

ELSE

| Sposta i puntatori delle foglie  
di A a B  
| rinomini la radice di B come A  
| rimuovi la vecchia radice A

$\text{size}(A) \leftarrow \text{size}(A) + \text{size}(B)$



"QuickFind" con bilanciamento

Si può dimostrare che effettuando una sequenza di:

- n MAKESET

-  $O(n)$  UNION e FIND

il tempo totale è  $O(n \lg n)$

$\Rightarrow$  costo "ammortizzato" UNION  $O(\lg n)$

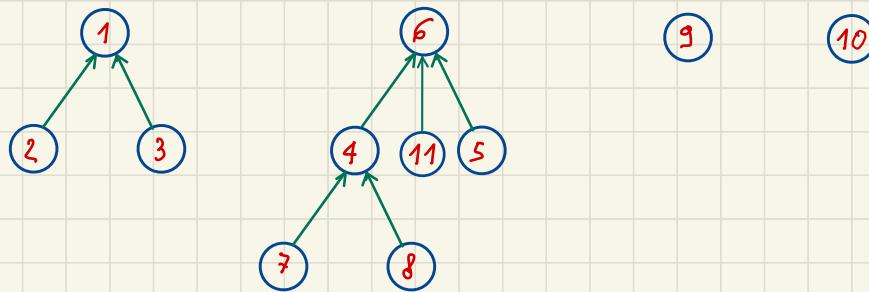
# ALGORITMI "QuickUnion"

## • ALBERI DI VARIE ALTEZZE

• Elementi dell'insieme  $\leftrightarrow$  NODI

• Elemento che dà il nome all'insieme  $\leftrightarrow$  RADICE

Esempio



{ 1, 2, 3 }

{ 7, 8, 4, 11, 5, 6 }

{ 9 }

{ 10 }

"QuickUnion": operation:

Tempo.

MAKESET (elemento e)

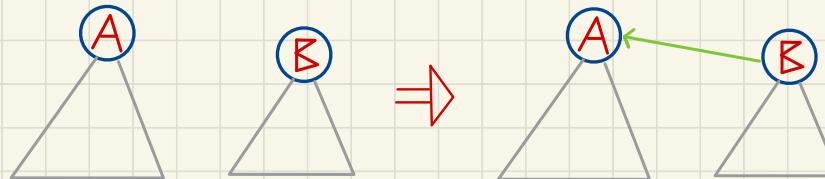
(e)

MAKESET (?)

(?)

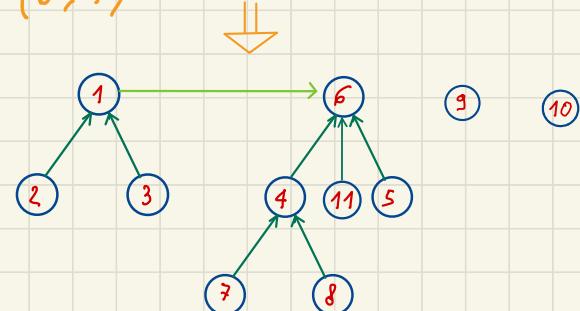
$T(n) \approx \Theta(1)$

UNION (nome A, nome B)



$T(n) = \Theta(1)$

UNION (6, 1)



NOTA: albero "peggiore"

MAKESET(1), MAKESET(2), ..., MAKESET(n)



UNION(n-1, n)

UNION(n-2, n-1)

UNION(n-3, n-2)

;

UNION(2, 1)

ALBERO di altezza n-1

"QuickUnion": operazioni:

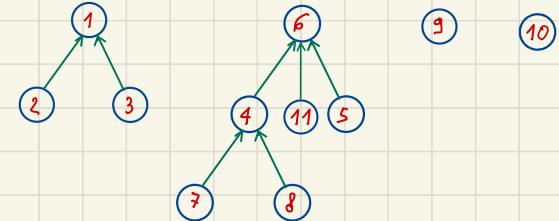
FIND (elemento e) → nome

risalire da e

fini alla radice

# possi: albero delle radici

FIND(8)

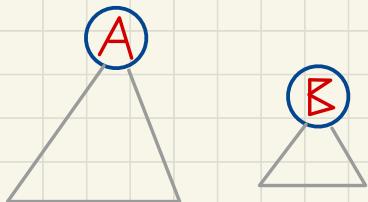


$$T(n) = O(n)$$

# "QuickUnion" con UNION bilanciata

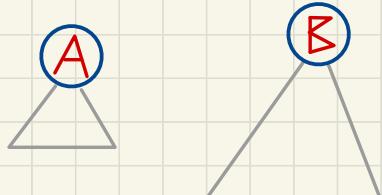
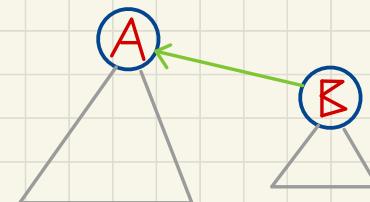
UNION: attacchiamo la radice dell'albero più basso  
sotto quella dell'albero più alto

"rank" = altezza



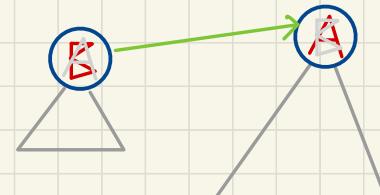
$$\text{rank}(A) \geq \text{rank}(B)$$

UNION(A,B)  
⇒



$$\text{rank}(B) > \text{rank}(A)$$

UNION(A,B)  
⇒



Scambio  
radici: