Algoritmi e Strutture Dati

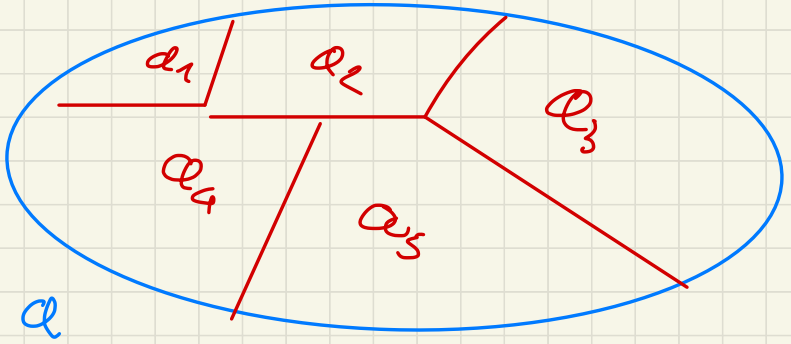
Lezione 19

11 novembre 2022

Rappresentazione di partizioni

Insieme A

PARTIZIONE



$A_1, A_2, \dots, A_k \subseteq A$

$A_i \neq \emptyset \quad i = 1 \dots k$

$A_i \cap A_j = \emptyset \quad i \neq j$

$A_1 \cup A_2 \cup \dots \cup A_k = A$

- S: collezione di elementi tra loro distinti esempio: interi da 1 a n

- ORGANIZZAZIONE INIZIALE:

  n insiemi disgiunti: $\{1\}$ $\{2\}$ ... $\{n\}$

- OGNI INSIEME HA UN NOME

  nome iniziale $\{i\} \longrightarrow i$

Problema UNION-FIND

Rappresentare una collezione di insiemi disgiunti, con le operazioni:

UNION(A,B) unisce gli insiemi A e B in un unico insieme di nome A

FIND(x) restituisce il nome dell'insieme che contiene l'elemento x

MAKESET(x) crea un nuovo insieme {x} di nome x (x nuovo elemento)

UNION-FIND: esempio

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

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

{2, 3} {5, 6}

$S = \{1, 2, 3, 4, 5, 6\}$

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

FIND(2) → 2

UNION(3, 2)

UNION(5, 6)

FIND(2) → 3

{2, 3, 5, 6}

{7}

UNION(5, 3)

FIND(2) → 5

MAKESET(7)

$S = \{1, 2, 3, 4, 5, 6, 7\}$

$\{\{1\}, \{2, 3, 5, 6\}, \{4\}, \{7\}\}$

VARI TIPI DI SOLUZIONE

- soluzioni elementari
- soluzioni evolute

In tutte le soluzioni presentate:

- ogni insieme è rappresentato da un albero con radice
  - NODI → elementi dell'insieme
  - RADICE → nome dell'insieme

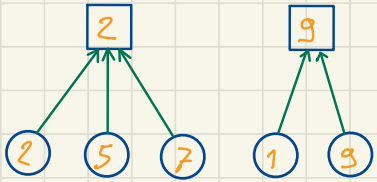
puntatori verso l'alto

- partizione: Foreste di alberi

ALGORITMI "QuickFind"

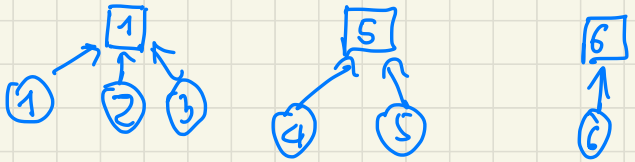
- ALBERI DI ALTEZZA 1
- Elementi dell'insieme ⟷ FOGLIE
- Nome dell'insieme ⟷ RADICE

Esempio



{2, 5 7} {1, 9}

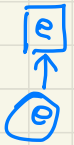
Esempio {1, 2, 3} {4, 5} {6}



"QuickFind": operazioni

Tempo

MAKESET (elemento e)



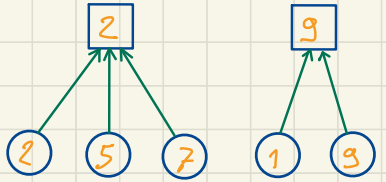
$T(n) = O(1)$

MAKESET (7)

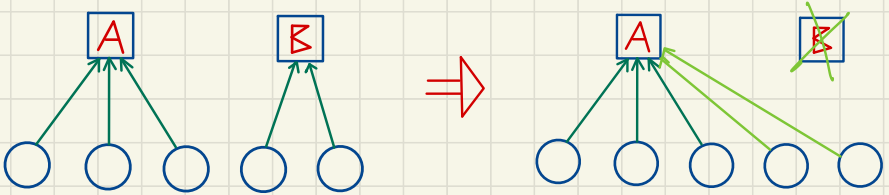
FIND (elemento e) → nome

$T(n) = O(1)$

FIND (5)

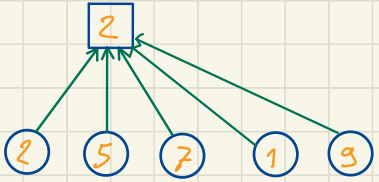


UNION (nome A, nome B)



UNION (2, 9)

$T(n) = O(n)$



Spazio $S(n) = O(n)$

"QuickFind" con bilanciamento

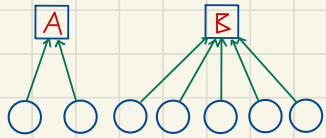
UNION (nome A, nome B)

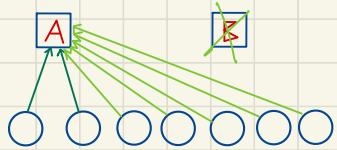
Quando #B > #A

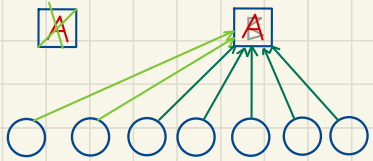
anziché collegare gli elementi di B sotto A

conviene

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







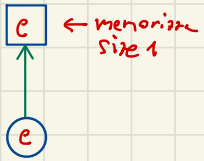
Al max spostiamo $\frac{n}{2}$ puntatori

$T(n) \approx O(n)$

"QuickFind" con bilanciamento: operazioni

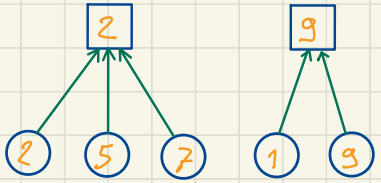
Nella radice si memorizza il numero di elementi dell'insieme (SIZE)

MAKESET (elemento e)



$T(n) = O(1)$

FIND (elemento e) → nome

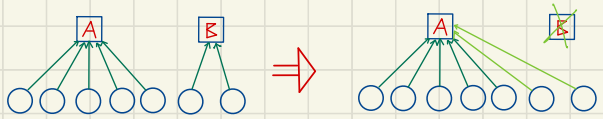


$T(n) = O(1)$

UNION (nome A, nome B)

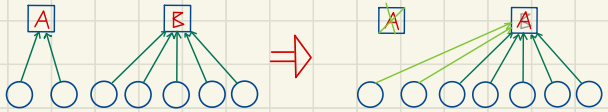
IF size(A) ≥ size(B) THEN

- sposta i puntatori dalle foglie di B ad A
- rimuovi la radice di B



ELSE

- sposta i puntatori dalle foglie di A a B
- rinomina la radice di B come A
- rimuovi la vecchia radice di A



size(A) ← size(A) + size(B)

"QuickFind" con bilanciamento

Si può dimostrare che effettuando una sequenza di

- n MAKESET
- $O(n)$ UNION e FIND

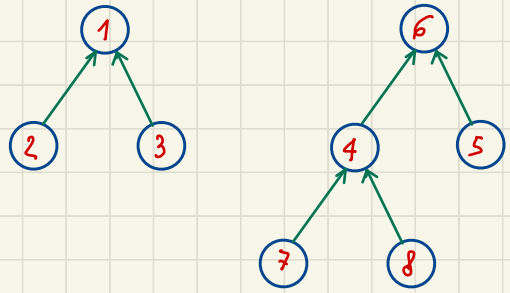
il tempo totale è $O(n \lg n)$

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

ALGORITMI "QuickUnion"

- ALBERI DI VARIE ALTEZZE
- Elementi dell'insieme ⟷ NODI
- Elemento che dà il nome all'insieme ⟷ RADICE

Esempio



Spazio $O(n)$

"QuickUnion": operazioni

MAKESET (elemento e)

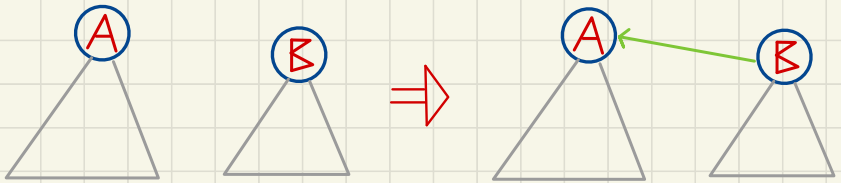
$T(n) = O(1)$

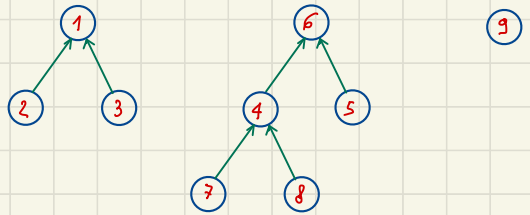
e

MAKESET (7)

7

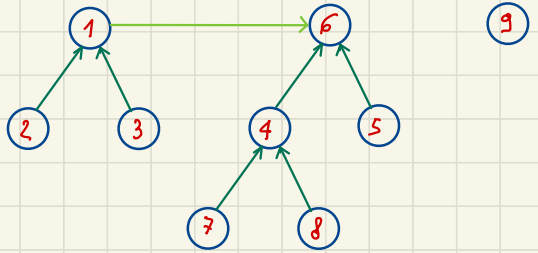
UNION (nome A, nome B)





UNION (6, 1)

$T(n) = O(1)$



NOTA: albero "peggiore"

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

UNION(n-1, n)

UNION(n-2, n-1)

⋮

UNION(2, 3)

UNION(1, 2)

Albero di altezza n-1

"QuickUnion": operazioni

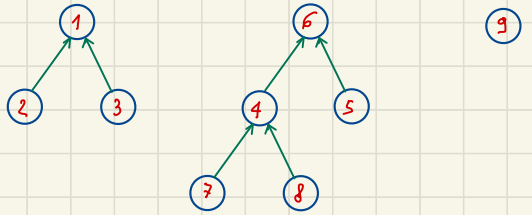
FIND (elemento e) → nome

↓

passi ≤ altezza albero

$T(n) = O(n)$

FIND (8)

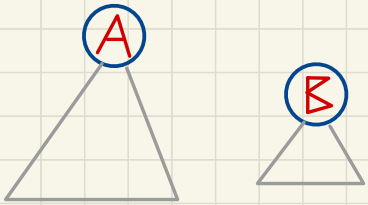


"QuickUnion" con UNION bilanciata

UNION by rank

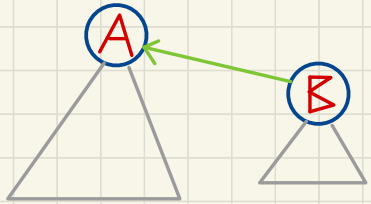
UNION: attacchiamo la radice dell'albero più basso sotto quella dell'albero più alto

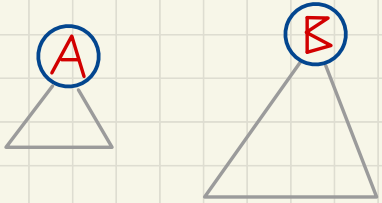
"rank" = altezza



rank(A) ≥ rank(B)

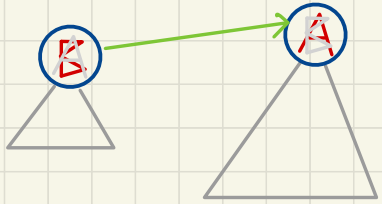
UNION (A,B)





rank(B) > rank(A)

UNION (A,B)



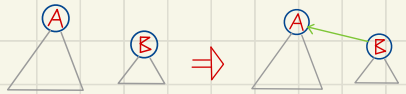
Scambio radici

UNION by rank

Nella radice si memorizza l'altezza dell'albero (rank)

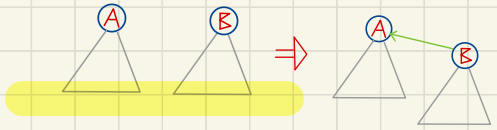
UNION (nome A, nome B)

IF rank(A) > rank(B) THEN
- rendi la radice di B figlia di quella di A



ELSE IF rank(A) < rank(B) THEN
- rendi la radice di A figlia di quella di B
- scambia i contenuti dei nodi A e B

ELSE
- rendi la radice di B figlia di quella di A
- rank(A) ← rank(A) + 1



$T(n) = O(1)$

"QuickUnion" con UNION bilanciata: operazioni

MAKESET (elemento e)

occorre memorizzare nella radice che il rank è 0

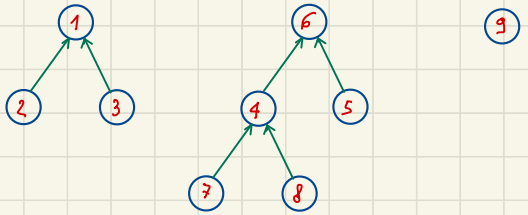
MAKESET (7)

$T(n) \Rightarrow O(1)$

FIND (elemento e) → nome

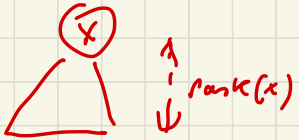
- occorre risalire nell'albero
- il tempo dipende dall'altezza dell'albero

FIND (8)



passi = O (altezza dell'albero)

Lemma Ogni albero "QuickUnion" costruito effettuando operazioni di UNION bilanciate contiene almeno $2^{rank(x)}$ nodi, x radice



$$\#\text{nodi}(x) \geq 2^{rank(x)}$$

Dim k = # operazioni UNION utilizzate per ottenere l'albero con radice x

Induzione su k

$\boxed{k=0}$ o UNION

(x) ALBERO
$rank(x) = 0$
#nodi = 1

$\boxed{k>0}$ ultima UNION:

UNION(A, B)

→ alberi ottenuti con meno di k union

ip. ind. $\#\text{nodi}(A) \geq 2^{rank(A)}$

$\#\text{nodi}(B) \geq 2^{rank(B)}$

Se $rank(A) > rank(B)$

$rank(x) = rank(A)$

$\#nodi'(x) \geq \#nodi'(A) + \#nodi'(B)$

$\geq 2^{rank(A)} + 2^{rank(B)}$

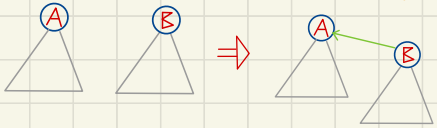
$\geq 2^{rank(A)} = 2^{rank(x)}$

Se $rank(A) < rank(B)$

$rank(x) = rank(B)$

$\longrightarrow$ analogo

Se $rank(A) = rank(B)$



$rank(x) = rank(A) + 1$

$\#nodi'(x) = \#nodi'(A) + \#nodi'(B) \geq$

$2^{rank(A)} + 2^{rank(B)} = \leftarrow rank(A) = rank(B)$

$= 2 \cdot 2^{rank(A)}$

$= 2^{rank(A)+1} = 2^{rank(x)}$

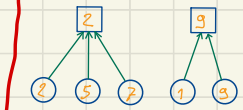
Lemma Ogni albero "QuickUnion" costruito effettuando operazioni di UNION bilanciate contiene almeno $2^{rank(x)}$ nodi, x radice

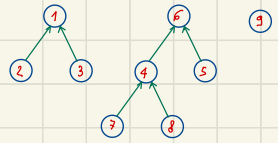
Corollario n nodi $\rightarrow$ altezza $\leq \lg_2 n$

$\Rightarrow$ FIND in tempo $O(\lg n)$

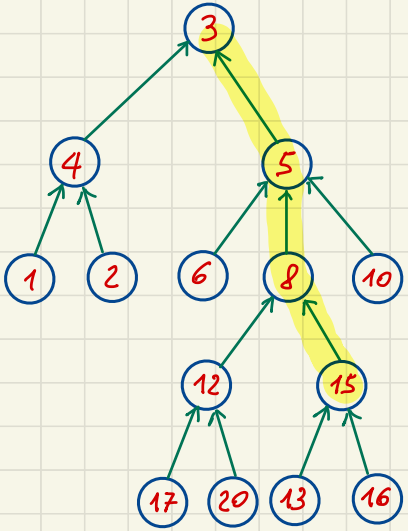
UNION-FIND riepilogo

	MAKESET	UNION	FIND
QuickFind	$O(1)$	$O(n)$	$O(1)$
QuickFind bilan.	$O(1)$	$O(\log n)$ amm.	$O(1)$
QuickUnion	$O(1)$	$O(1)$	$O(n)$
QuickUnion bilan.	$O(1)$	$O(1)$	$O(\log n)$

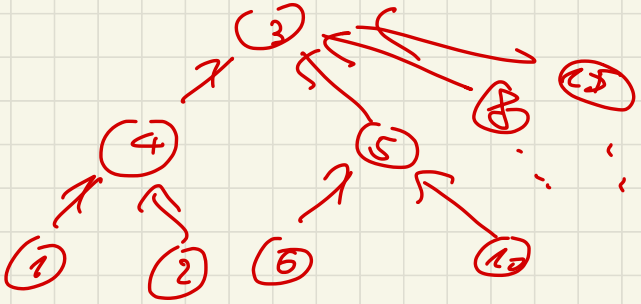




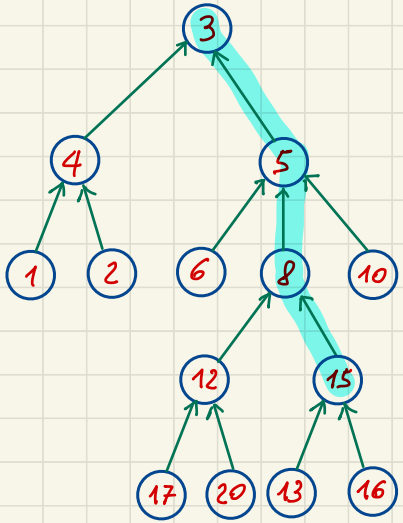
"QuickUnion" bilanciata con compressione di cammino



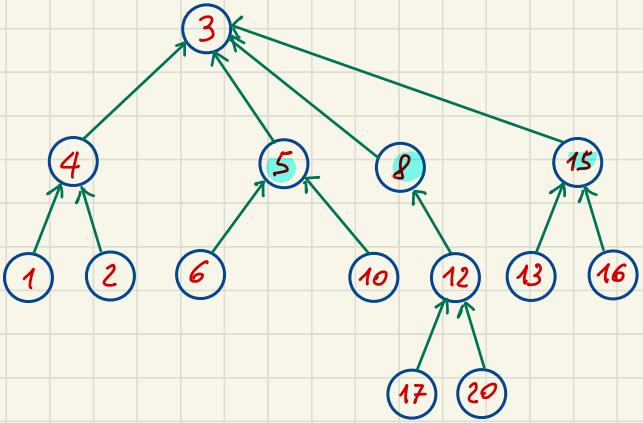
FIND(15)

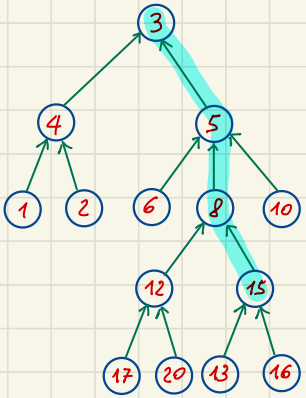


"QuickUnion" bilanciata con compressione di cammino

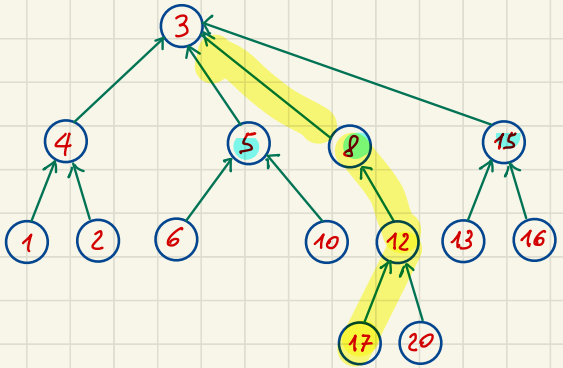


FIND(15)

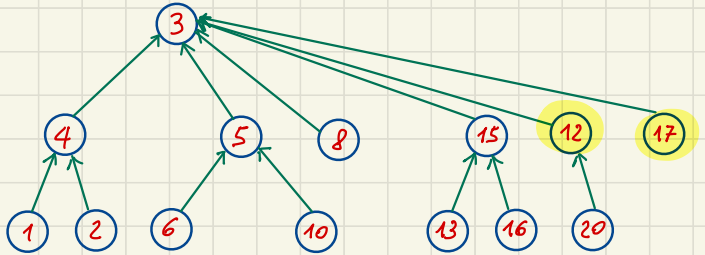




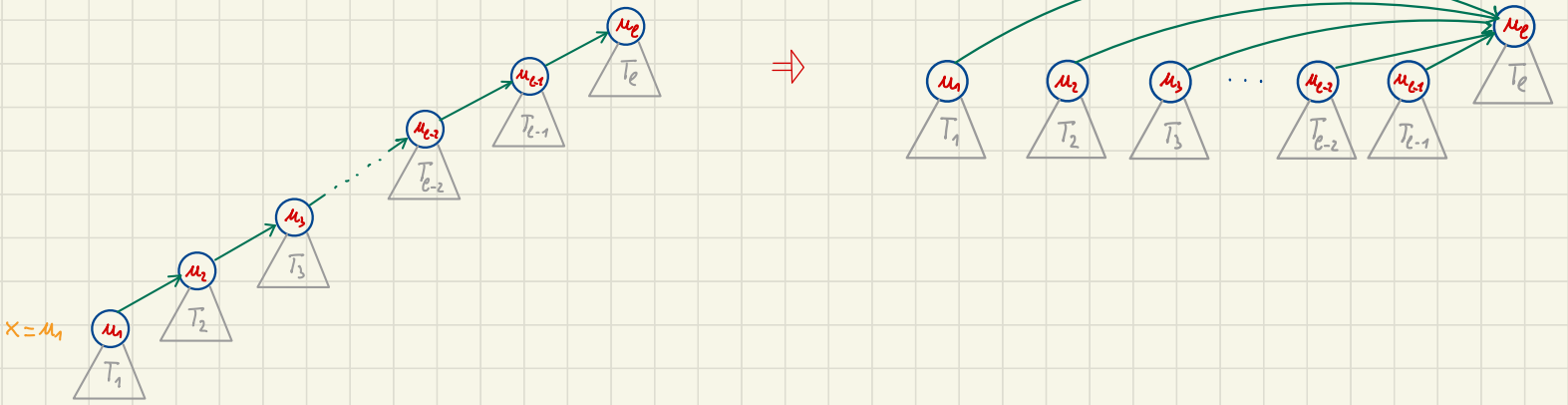
FIND(15)



FIND(17)



FIND(x) Compressione di cammino



LOGARITMO ITERATO

$$\lg^{(1)} n = \lg_2 n$$

$$\lg^{(2)} n = \lg_2 (\lg^{(1)} n)$$

$$\vdots$$

$$\lg^{(i)} n = \lg_2 (\lg^{(i-1)} n) = \underbrace{\lg_2 \lg_2 \dots \lg_2}_{i \text{ volte}} n$$

$$\lg^* n = \{ \min i \mid \lg^{(i)} n \leq 1 \}$$

quante volte devo applicare il logaritmo, partendo da n, per ottenere un numero ≤ 1

"Quick Union" bilanciata con compressione di cammino

Si può dimostrare che effettuando una sequenza di

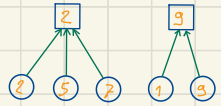
- n MAKESET
- $O(n)$ UNION e FIND

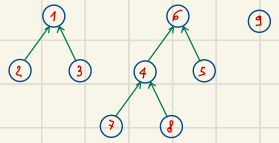
il tempo totale è $O(n \log^* n)$

$\Rightarrow$ costo "ammortizzato" di FIND è $O(\log^* n)$

UNION-FIND riepilogo

	MAKESET	UNION	FIND
QuickFind	$O(1)$	$O(n)$	$O(1)$
QuickFind bilan.	$O(1)$	$O(\lg n)$ amm	$O(1)$
QuickUnion	$O(1)$	$O(1)$	$O(n)$
QuickUnion bilan.	$O(1)$	$O(1)$	$O(\lg n)$
QuickUnion bilan. con compressione cammino	$O(1)$	$O(1)$	$O(\lg^* n)$ amm





va all'∞ al crescere di n

nella "pratica" costante ≤ 6