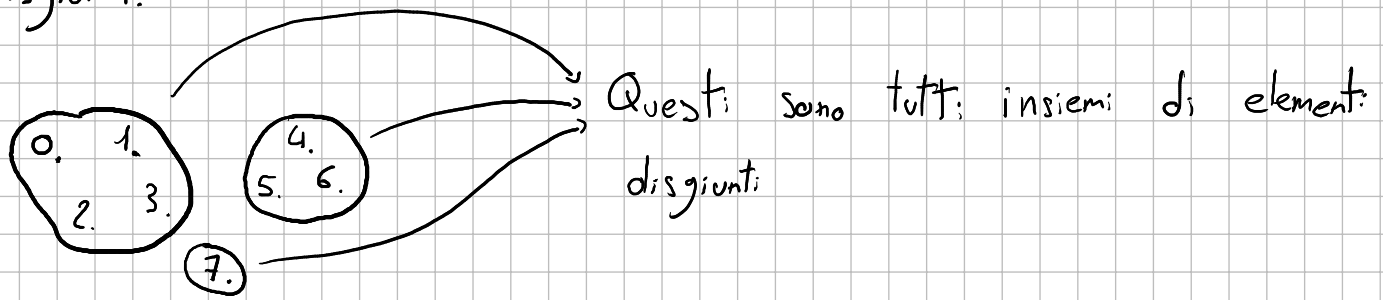


Union Find Algorithms

Per prima cosa dobbiamo capire cos'è un insieme di elementi disgiunti.



Gli algoritmi di Union Find servono a gestire gli insiemi di elementi disgiunti.

Le Union Find offrono 3 operazioni:

- $\text{Create}(n)$: Crea n insiemi.
- $\text{Union}(S_1, S_2)$: Unisce l'insieme S_2 nell'insieme S_1 , quest'operazione si può fare solo tra rappresentanti di insiemi.
- $\text{Find}(x)$: Restituisce l'insieme dell'elemento x .

Esistono 3 tipi di implementazioni della Union Find:

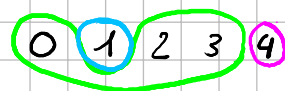
- Quick Find
- Quick Union
- Union by Rank

Ora le discutiamo. Prima però introduciamo il concetto di rappresentante, ovvero l'elemento che rappresenta un insieme.

È l'array dei rappresentanti: è un array di dimensione n , dove in posizione di un elemento c'è il rappresentante del suo insieme.

- Quick Find:

Prendiamo come esempio il seguente insieme



Array Rappresentanti:

2	1	2	2	4
0	1	2	3	4

// è stato scelto 2 come rappresentante

→ Come puoi notare $r[2] = r[3]$, quindi fanno parte dello stesso insieme

- Union (1,4)



Array Rappresentanti:

2	1	2	2	4
0	1	2	3	4

→ Si cambiano tutte le celle dell'insieme viola con il rappresentante dell'insieme azzurro, in questo caso 1.

2	1	2	2	1
0	1	2	3	4

- Union (1,2)



Array Rappresentanti:

2	1	2	2	4
0	1	2	3	4

1	1	1	1	4
0	1	2	3	4

- Find (3) 

Array Rappresentanti:

2	1	2	2	4
0	1	2	3	4

→ prendo il contenuto di $r[3]$, in questo caso 2.

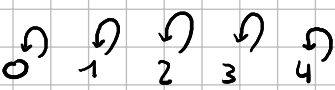
In questo modo so di che insieme fa parte 3.

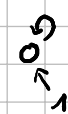
Costi quick find:


- Union: $O(h)$

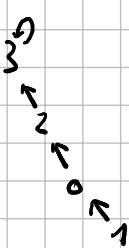
- Find: $O(1)$

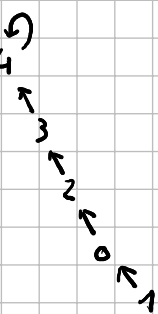
Quick Union:

- Create (5): 

Union (0,1): 

Union (2,0): 

Union (3,2): 

Union (4,3): 

Con gli array dei rappresentanti abbiamo questa situazione:

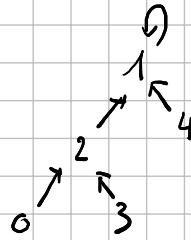
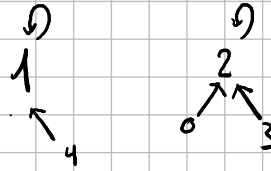
Dato:

Array Rapp

2	1	2	2	1
0	1	2	3	4

Union (1,2):

2	1	1	2	1
0	1	2	3	4



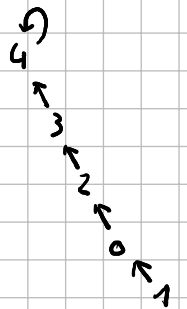
→ Come puoi notare abbiamo cambiato il contenuto di una sola cella.

Cost: Quick Union;

- Union: $O(1)$

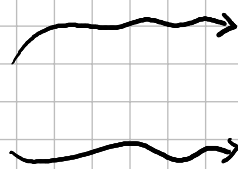
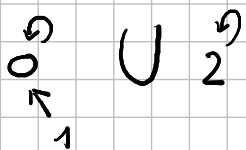
- Find: $O(n) \rightarrow$ perché dobbiamo scorrere una lista.

Es: Find (1)



Come scegliere il rappresentante in modo ottimizzato (anticipazione del Union by Rank)

Esempio:



Abbiamo scelto male il rappresentante

Abbiamo scelto bene il rappresentante

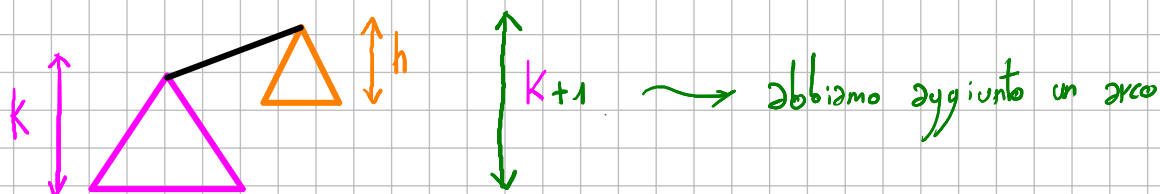
Questo ci serve per ridurre il costo dell'operazione Find (della Quick Union)

Come scegliere bene il rappresentante?

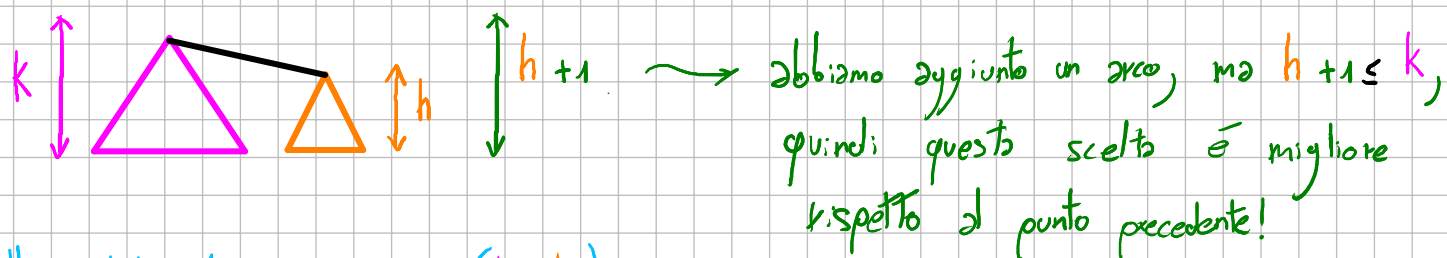
3 Casi:

- Albero piú grande appeso sull'albero piú piccolo
- Albero piú piccolo appeso sull'albero piú grande
- Union con alberi della stessa profondità

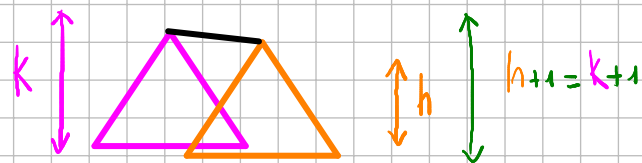
• Piú grande appeso al piú piccolo ($k > h$)



• Piú piccolo appeso al piú grande ($k > h$)



• Alberi della stessa profondità ($k = h$)

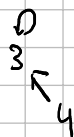
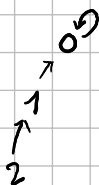


Union by Rank

Introduciamo Array dei rank, ovvero la profondità dell'albero (e quindi dell'insieme)

Array Rank:

0	0	1	3	3
0	1	2	3	4



Array Rank (ci interessa solo il Rank dei rappresentanti degli insiemi)

2	x	x	1	x
0	1	2	3	4

// Qualunque valore sia x non ci riguarda.

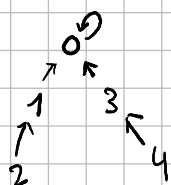
Union (0,3):

Rank(0): 2
Rank(3): 1

Scelgo nuovo rappresentante prendendo quello con Rank maggiore.

Così abbiamo l'ottimizzazione
del punto 2 visto prima.

Quindi otteniamo



Array Rapor:

0	0	1	0	3
0	1	2	3	4

Array Rank:

2	x	x	x	x
0	1	2	3	4

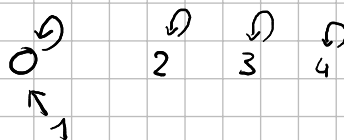
Ulteriori esempi:

Array Rapor:

0	0	2	3	4
0	1	2	3	4

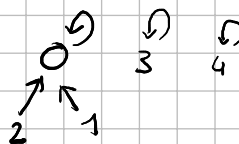
Array Rank:

1	x	0	0	0
0	1	2	3	4



Union (2,0)

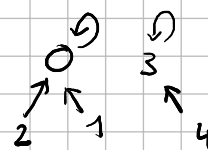
Rank (2): 0
Rank (0): 1 ✓



Union (3,4):

Rank (3): 0 ✓
Rank (4): 0

// Indifferente,
potrei anche scegliere
4, terzo punto
2220xro.



Array Rapor:

0	0	0	3	4
0	1	2	3	4

Array Rank:

1	x	x	0	0
0	1	2	3	4



Array Rapor:

0	0	0	3	3
0	1	2	3	4

Array Rank:

1	x	x	1	x
0	1	2	3	4

Dimostrare che la massima profondità è $O(\log n)$, e di conseguenza anche la Find (Rank)

$$\text{se } |A| = n \Rightarrow \left[\text{allora il Rank}(A) \leq \log n \right] \Leftrightarrow 2^{\text{Rank}(A)} \leq n$$

perché percorre tutta la
profondità dell'albero

Dimostrazione per Induzione Strutturale // guardo la struttura dell'insieme

- Base: $|A| = 1$ rank(A) = 0 $2^0 \leq 1$ ✓

- Induzione:

- Passo induttivo: $A = B \cup C$

- Ipotesi induttiva: $2^{\text{Rank}(B)} \leq |B|$ e $2^{\text{Rank}(C)} \leq |C|$

- Tesi: vero la tesi per A

Quanto vale $|A|$?

$|A| = |B| + |C|$, perché sono insiemi disgiunti.

Quindi per l'ipotesi induttiva abbiamo:

$$|A| = |B| + |C| \geq 2^{\text{Rank}(B)} + 2^{\text{Rank}(C)}$$

Noi abbiamo 3 possibili casi:

stessa cosa di dire $\text{Rank}(A)$

$$1) \text{Rank}(B) < \text{Rank}(C) \rightsquigarrow \text{Rank}(B \cup C) = \text{Rank}(C)$$

$$2) \text{Rank}(C) < \text{Rank}(B) \rightsquigarrow \text{Rank}(B \cup C) = \text{Rank}(B)$$

// prendo quelli
con Rank
maggiore

$$3) \text{Rank}(B) = \text{Rank}(C) \rightsquigarrow r, \text{ ovvero } \text{Rank}(B) = \text{Rank}(C)$$

$$1) \geq 2^{\text{Rank}(C)} = 2^{\text{Rank}(A)}$$

$$2) \geq 2^{\text{Rank}(B)} = 2^{\text{Rank}(A)}$$

$$3) 2^r + 2^r = 2^{r+1} = 2^{\text{Rank}(A)}$$

→ punto 3 azzurro

Abbiamo dimostrato la tesi, in questo modo ora tramite la Union by Rank
ho:

- Union: $O(1)$

- Find: $O(\log n)$

Kruskal ottimizzato mediante Union by Rank

KruskalOptimized {

$A = \emptyset$

$\forall e \text{ minHeap.add}(e)$

while !minHeap.isEmpty

$e = \text{minHeap.remove}$

if (Find(u) == find(v))

 scaricare e

else

$$A = A \cup \{e\}$$

Union (find(u), find(v))

Costo totale: $O(m \log n)$