POLITECNICO
DI TORINO

Dipartimento
di Automatica e Informatica

L'Analisi di complessità

Paolo Camurati

Analisi di complessità

Definizione:

- previsione delle risorse (memoria, tempo) richieste dall'algoritmo per la sua esecuzione
  - empirica
  - analitica

Caratteristiche:

- indipendente dalla macchina. Ipotesi: modello monoprocessore sequenziale (architettura tradizionale)
- indipendente dai dati di ingresso di una particolare istanza del problema

Esempio:

- problema P: ordinare dati interi
- istanza I: i dati sono 45 10 6 7 99
- dimensione dell’istanza |I|: numero di bit per codificare I, in questo caso 5 x la dimensione dell’intero o più semplicemente 5

- Dipendente dalla dimensione n del problema. Esempi:
  - numero di cifre degli operandi per moltiplicazione di interi
  - dimensione del file da ordinare
  - numero di caratteri di una stringa di testo
  - numero di dati da ordinare per algoritmi di ordinamento
- Output:
  - S(n): occupazione di memoria
  - T(n): tempo di esecuzione.

Classificazione degli algoritmi

- 1: costante
- log n: logaritmico
- n: lineare
- n log n: *linearitmico*
- n^2: quadratico
- n^3: cubico
- 2^n: esponenziale

Analisi asintotica di caso peggiore

Scopo:

- Stima di un limite superiore a T(n) di un algoritmo su n dati nel caso peggiore

Asintotica: $n \rightarrow \infty$:

- per n piccolo, la complessità è irrilevante.

Perché il caso peggiore?

- Conservatività della stima
- Caso peggiore molto frequente
- Caso medio:
  - o coincide con il caso peggiore
  - o non è definibile a meno di ipotesi complesse sui dati.

Importanza dell’analisi della complessità

Vantaggi di una complessità inferiore:

- compensa l’(in)efficienza dell’hardware.

Esempio:

- Algoritmo 1:
  - $T(n) = 2n^2$
  - macchina 1: 10^8 istruzioni/secondo
- Algoritmo 2:
  - $T(n) = 50n \lg_2 n$
  - macchina 2: 10^6 istruzioni/secondo

Se $n = 1M = 10^6$:

- Algoritmo 1: $2 \cdot (10^6)^2 / 10^8 = 2 \cdot 10^4 = 20000$ s = 333,33 minuti
- Algoritmo 2: $50 \cdot 10^6 \lg_2 10^6 / 10^6 = 50 \cdot 6 \cdot \lg_2 10 = 1000$ s = 16,67 minuti

Un algoritmo inefficiente «spreca» rapidamente l'incremento di prestazioni dell'hardware!

Esempi

Discrete Fourier Transform:

- decomposizione di una forma d'onda di N campioni in componenti periodiche
- applicazioni: DVD, JPEG, astrofisica, ….
- algoritmo rozzo: quadratico (N^2)
- FFT (Fast Fourier Transform): N log N

Simulazione di N corpi:

- simula l'interazione gravitazionale tra N corpi
- algoritmo rozzo: quadratico (N^2)
- Algoritmo di Barnes-Hut: N log N

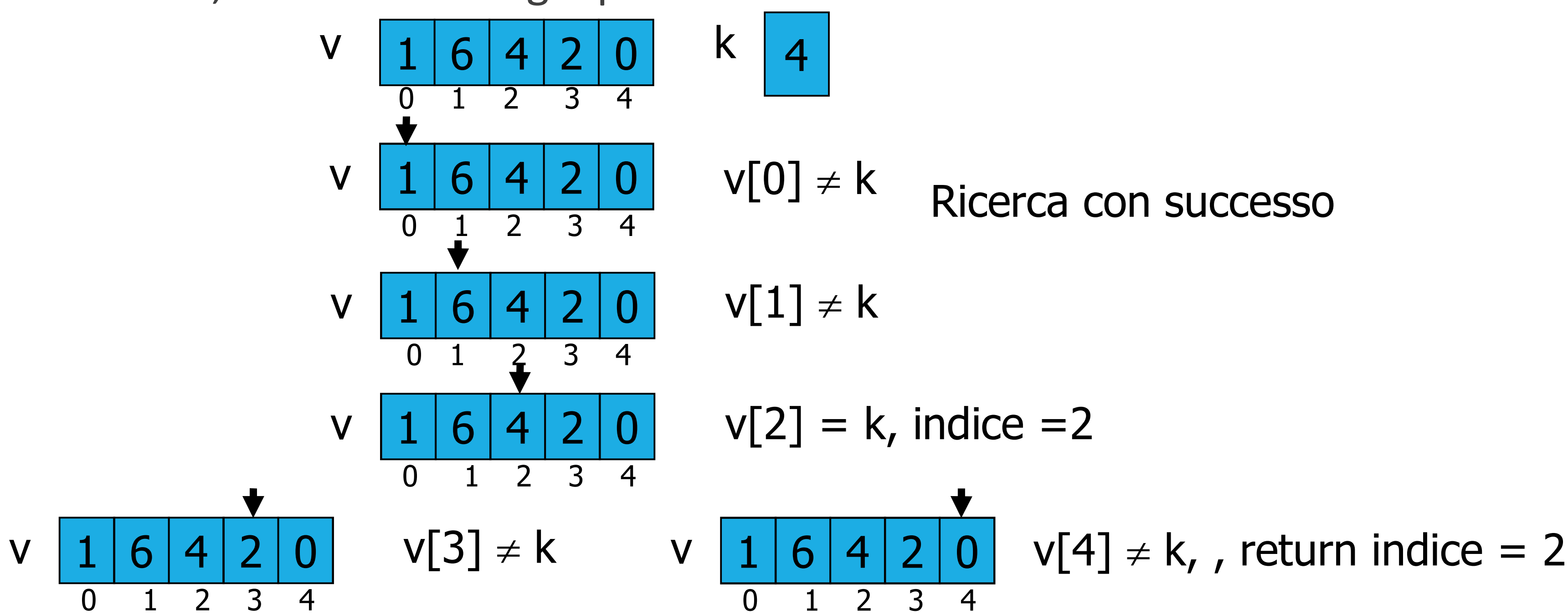
Algoritmi di ricerca su vettori

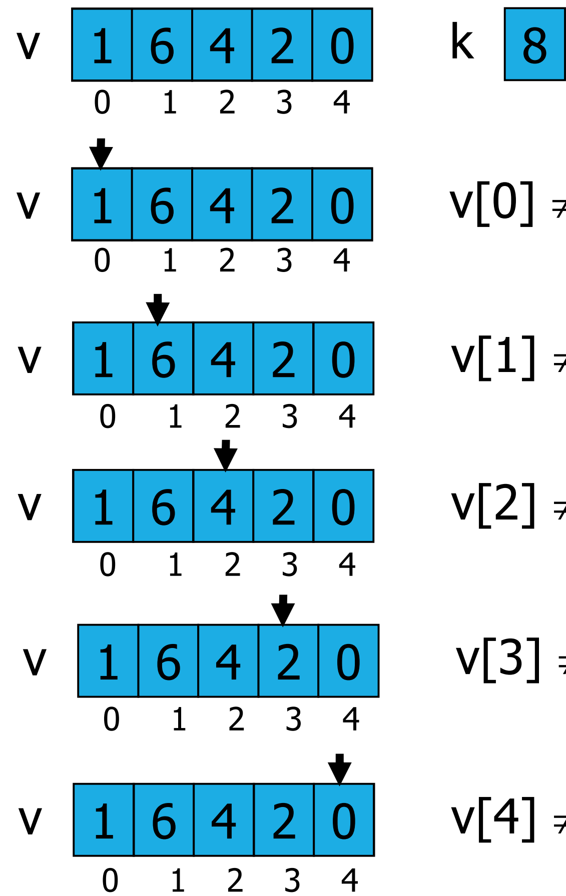
Dato un vettore v[N] di N elementi distinti e una chiave k:

- problema di decisione: la chiave k è presente all'interno di v[N]? Sì/No
- problema di ricerca: se k è presente, in quale posizione (a quale indice del vettore)?

Algoritmo 1: Ricerca sequenziale

Scansione del vettore dal primo potenzialmente fino all'ultimo elemento, confronto ad ogni passo con la chiave k





$v[0] \neq k$

$v[1] \neq k$

$v[2] \neq k$

Ricerca con fallimento

$v[3] \neq k$

$v[4] \neq k$, return indice = -1

Alternative:

- Soluzione 1: scansione del vettore dal primo all’ultimo elemento: sempre N operazioni
- Soluzione 2: uso di un flag: la scansione del vettore può finire anticipatamente, al massimo N operazioni, nel caso peggiore N operazioni.

La complessità asintotica di caso peggiore è la stessa (N), con la seconda alternativa migliora quella del caso medio.

Soluzione 1

```
int LinearSearch1(int v[], int N, int k) {
  int i = 0, index = -1;

  for (i = 0;  i < N; i++)
    if (k == v[i])
      index = i;
  return index;
}
```

Soluzione 2

```
int LinearSearch2(int v[], int N, int k) {
  int i = 0;
  int found = 0;

  while (i < N && found == 0)
    if (k == v[i])
      found = 1;
    else
      i++;

  if (found == 0)
    return -1;
  else
    return i;
}
```

Algoritmo 2: Ricerca binaria o dicotomica

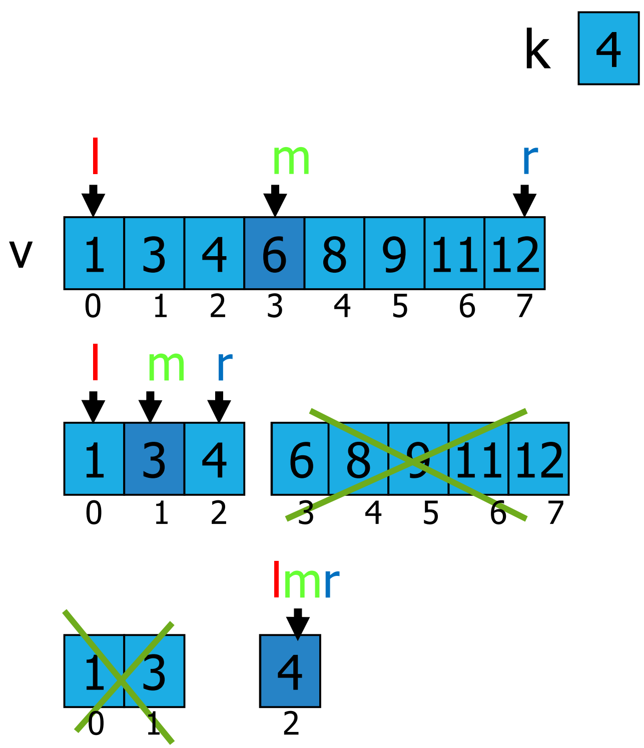
Dato un vettore v[N] ordinato di N elementi distinti e una chiave k:

- problema di decisione: la chiave k è presente all'interno di v[N]? Sì/No
- problema di ricerca: se k è presente, in quale posizione (a quale indice del vettore)?

Si opera su un sottovettore identificato dagli elementi contigui di v compresi un indice estremo di SX (l) e di DX (r). Inizialmente il sottovettore coincide con il vettore di partenza (l = 0 e r = N-1). L'elemento centrale del sottovettore si trova all'indice m= (l+r)/2.

Approccio:

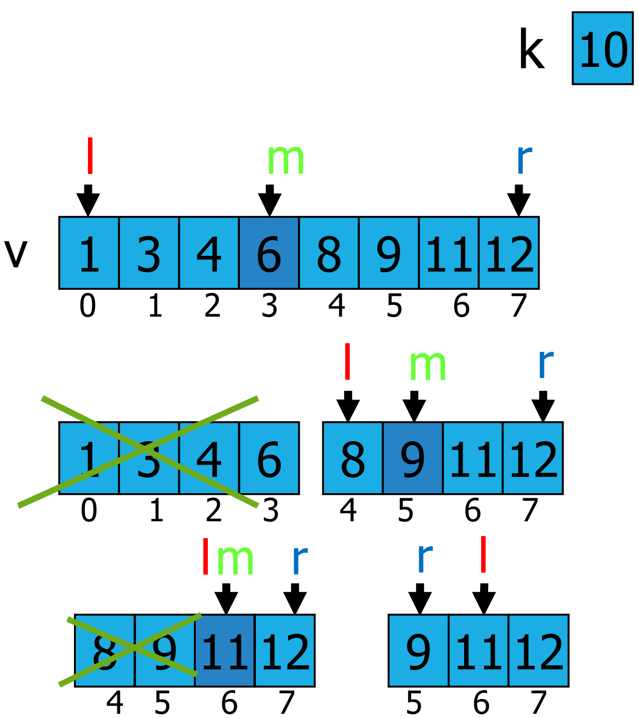
- ciclo in cui ad ogni passo confronto k con v[m], l’elemento centrale del sottovettore
- condizione del ciclo: $l \leq r$ && found==0: la chiave non è ancora stata trovata ed il sottovettore è significativo (l non oltrepassa r)
- corpo del ciclo:
  - se v[m] == k: terminazione con successo, found = 1
  - se v[m] < k: la ricerca prosegue nel sottovettore di DX: l=m+1, r invariato
  - se v[m] > k: la ricerca prosegue nel sottovettore di SX: l invariato, r = m-1
- all’uscita del ciclo si testa found, ritornando -1 per insuccesso o m per successo



l = indice estremo di SX, inizialmente l = 0
r = indice estremo di DX, inizialmente r = N-1
m = indice elemento di mezzo
v[m] = elemento di mezzo

l=0, r=7, m= (l+r)/2=3, v[3]>k, r=m-1
l=0, r=2, m= (l+r)/2=1, v[1]<k, l=m+1
l=2, r=2, m= (l+r)/2=2, v[2]=k

Ricerca con successo, return indice =2



l = indice estremo di SX, inizialmente l = 0

r = indice estremo di DX, inizialmente r = N-1

m = indice elemento di mezzo

v[m] = elemento di mezzo

l=0, r=7, m= (l+r)/2=3, v[3]<k, l=m+1

l=4, r=7, m= (l+r)/2=5, v[5]<k, l=m+1

l=6, r=7, m= (l+r)/2=6, v[6]>k, r=m-1

r < l, uscita dal ciclo, ricerca con fallimento

```c
int BinSearch(int v[], int N, int k) {
  int m, found= 0, l=0, r=N-1;

while(l <= r && found == 0){
    m = (l+r)/2;
    if(v[m] == k)
      found = 1;
    if(v[m] < k)
      l = m+1;
    else
      r = m-1;
  }
  if (found == 0)
    return -1;
  else
    return m;
}
```

Analisi della Ricerca Lineare

- Si esaminano N elementi per ricerca con fallimento e N/2 in media per ricerca con successo
- T(N) cresce linearmente in N.

Analisi della Ricerca Dicotomica

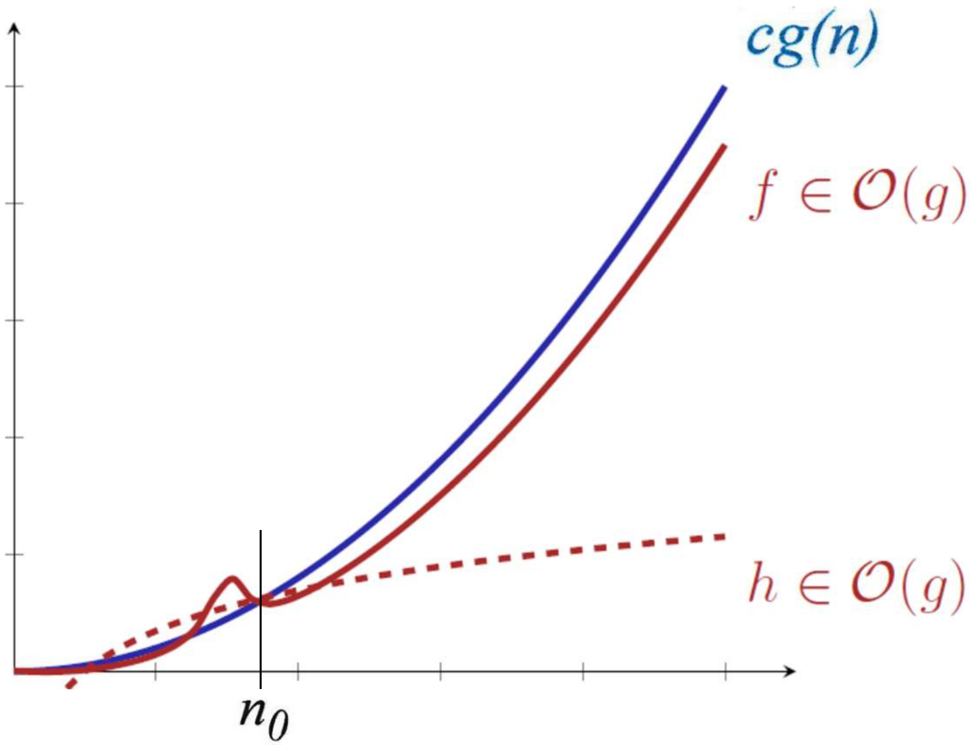
- All'inizio il vettore da esaminare è di N elementi
- Alla seconda iterazione il vettore da esaminare è di circa N/2 elementi
- ....
- Alla i-esima iterazione il vettore da esaminare è di circa $N/2^i$ elementi
- La terminazione avviene quando il vettore da esaminare è ampio 1, quindi $n/2^i = 1$, $i = \log_2(n)$
- T(n) cresce logaritmicamente in n.

Notazione asintotica O

Definizione:

$$T(n) = O(g(n)) \Leftrightarrow$$
$$\exists\ c>0,\ \exists\ n_0>0 \text{ tale per cui } \forall n \geq n_0$$
$$0 \leq T(n) \leq cg(n)$$

g(n) = limite superiore lasco di T(n). Al massimo si fanno g(n) operazioni (la costante moltiplicativa non conta nell'analisi asintotica)


$cg(n)$
$f \in \mathcal{O}(g)$
$h \in \mathcal{O}(g)$
n_0

Esempi:

$T(n) = 3n+2 = O(n)$, $c=4$ e $n_0=2$: $3n+2 \leq 4n \quad \forall n \geq 2$

$T(n) = 10n^2+4n+2 = O(n^2)$, $c=11$ e $n_0=5$ $10n^2+4n+2 \leq 11n^2 \quad \forall n \geq 5$

$T(n) = 3n+2 = O(n^2)$, $c=3$ e $n_0=2$ $3n+2 \leq 3n^2 \quad \forall n \geq 2$

Teorema:

se $T(n) = a_m n^m + ... + a_1 n + a_0$

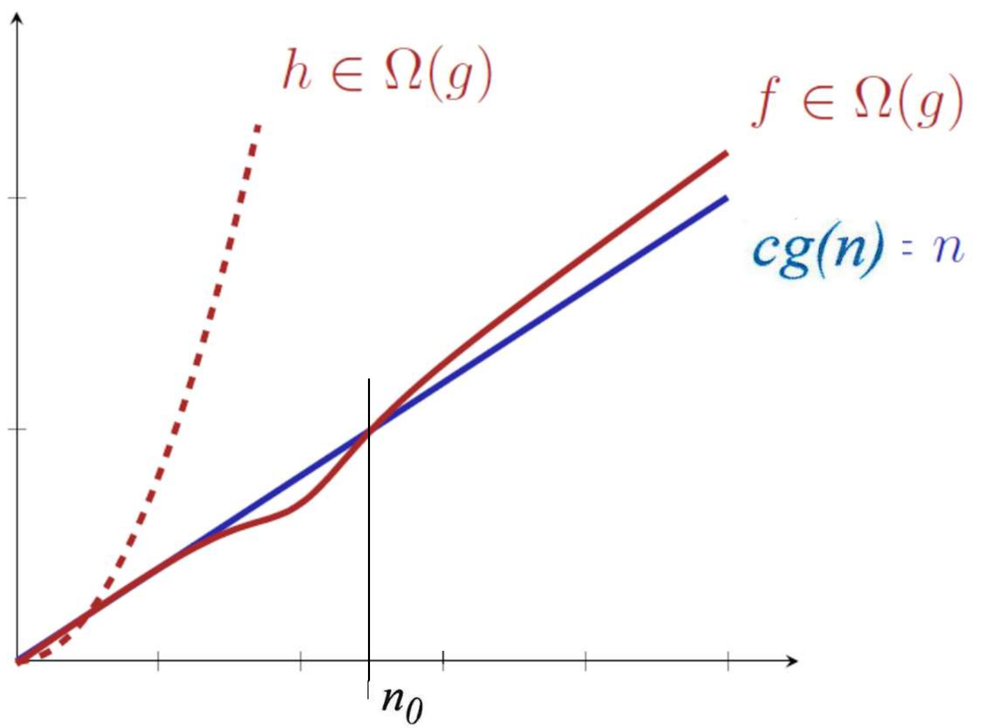
allora $T(n) = O(n^m)$

Notazione asintotica Ω

Definizione:

$$T(n) = \Omega(g(n)) \Leftrightarrow$$
$$\exists\ c>0,\ \exists\ n_0>0 \text{ tale per cui } \forall n \geq n_0$$
$$0 \leq c\ g(n) \leq T(n)$$

g(n) = limite inferiore lasco di T(n). Al minimo si fanno g(n) operazioni (la costante moltiplicativa non conta nell'analisi asintotica)


$h \in \Omega(g)$
$f \in \Omega(g)$
$cg(n) = n$
n_0

Esempi:

$T(n) = 3n+2 = \Omega(n)$, $c=3$ e $n_0=1$ $\qquad 3n \leq 3n+2 \quad \forall n \geq 1$

$T(n) = 10n^2+4n+2 = \Omega(n^2)$, $c=1$ e $n_0=1$ $\qquad n^2 \leq 10n^2+4n+2 \quad \forall n \geq 1$

$T(n) = 10n^2+4n+2 = \Omega(n)$, $c=30$ e $n_0=3$ $\qquad 30n \leq 10n^2+4n+2 \quad \forall n \geq 3$

Teorema:

se $T(n) = a_m n^m + \dots + a_1 n + a_0$

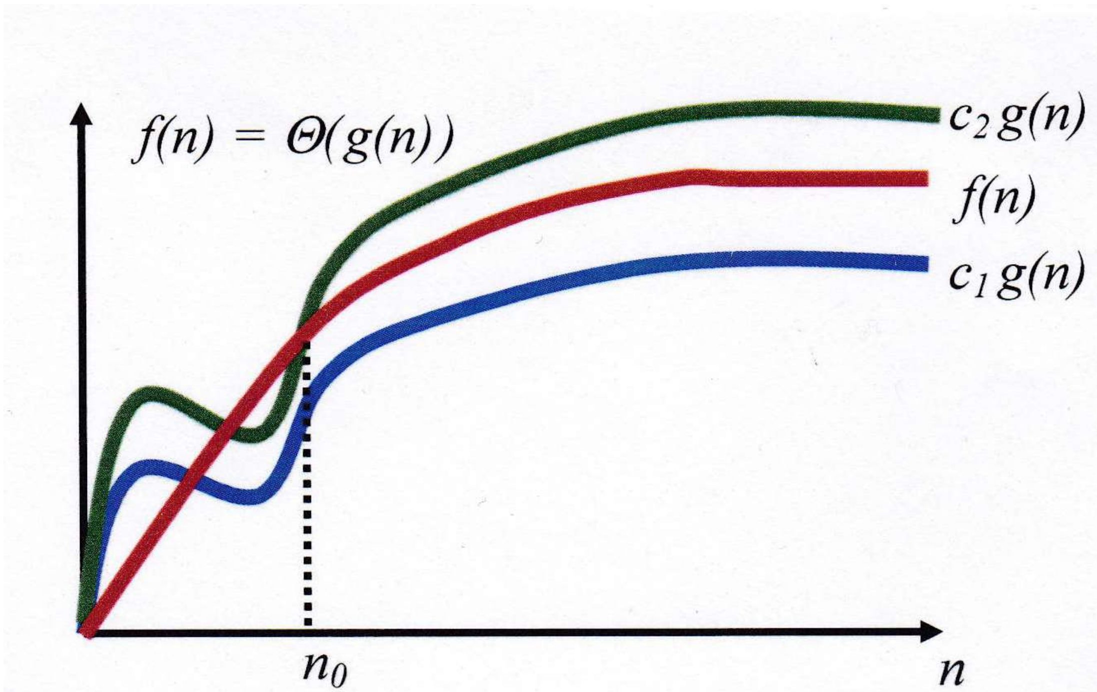
allora $T(n) = \Omega(n^m)$

Notazione asintotica Θ

Definizione:

$$T(n) = \Theta(g(n)) \Leftrightarrow$$
$$\exists\ c_1, c_2 > 0, \exists\ n_0 > 0 \text{ tale per cui } \forall n \geq n_0$$
$$0 \leq c_1\, g(n) \leq T(n) \leq c_2\, g(n)$$

g(n) = limite asintotico stretto di T(n). Si fanno esattamente g(n) operazioni (la costante moltiplicativa non conta nell'analisi asintotica)


$f(n) = \Theta(g(n))$
$c_2 g(n)$
$f(n)$
$c_1 g(n)$
n_0
n

Esempi:

$T(n) = 3n+2 = \Theta(n)$, $c_1=3$, $c_2=4$ e $n_0=2$ $\quad 3n \leq 3n+2 \leq 4n \quad \forall n \geq 1$

$T(n) = 3n+2 \neq \Theta(n^2)$, $T(n) = 10n^2+4n+2 \neq \Theta(n)$

Teoremi:

- Se $T(n) = a_m n^m + \dots + a_1 n + a_0$, allora $T(n) = \Theta(n^m)$
- Date due funzioni g(n) e T(n),

$T(n) = \Theta(g(n)) \Leftrightarrow T(n) = O(g(n))$ e $T(n) = \Omega(g(n))$.

Online Connectivity

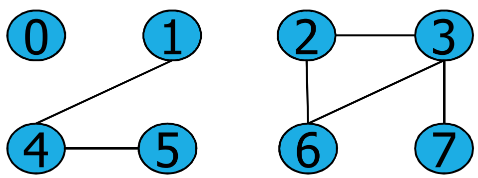
Problema reale per capire l'impatto delle scelte algoritmiche e delle strutture dati sulla complessità:

- Grafo non orientato con interi come vertici e coppie di interi come archi
- Input: sequenze di coppie di interi (p, q)
- Interpretazione: p è connesso con q (c'è un arco tra vertice p e vertice q
- La relazione di connessione è:
  - riflessiva: *p è connesso a p*
  - simmetrica: *se p è connesso a q, q è connesso a p*
  - transitiva: *se p è connesso a q e q è connesso a r, allora p è connesso a r*

quindi è una relazione di equivalenza.

- Output: lista delle connessioni ignote in precedenza (o non implicate transitivamente dalle precedenti):
  - nulla se p e q sono già connessi (direttamente o indirettamente)
  - (p, q) altrimenti

Componente connessa in un grafo non orientato: sottoinsieme massimale dei nodi mutuamente raggiungibili



{0} {1, 4, 5} {2, 3, 6, 7}

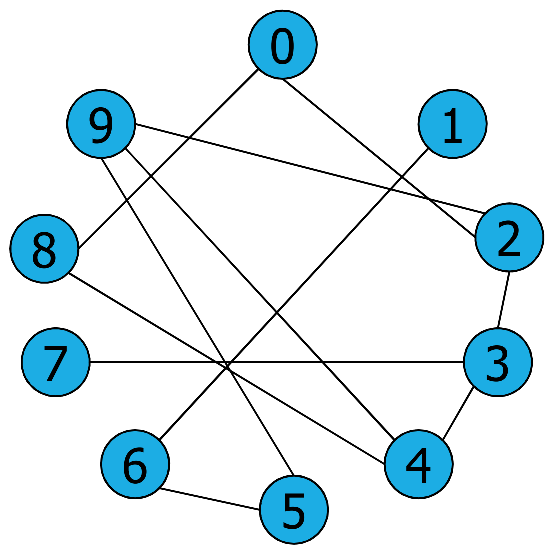
3 componenti connesse

Applicazioni

- Pixel in una fotografia digitale
- Reti di calcolatori (computer, connessioni)
- Reti elettriche (componenti, fili)
- Reti sociali (amici)
- Insiemi matematici
- Variabili in programmi.

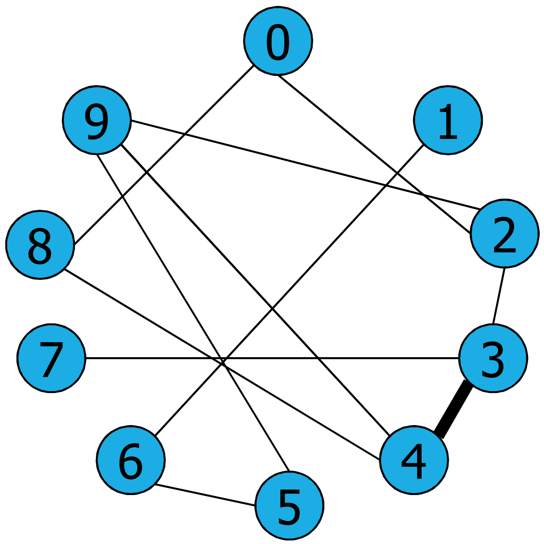
Esempio

Sequenza di input:
3-4, 4-9, 8-0, 2-3, 5-6, 2-9, 5-9, 7-3, 4-8, 5-6, 0-2, 6-1
Grafo corrispondente:



Input

3 4



Output

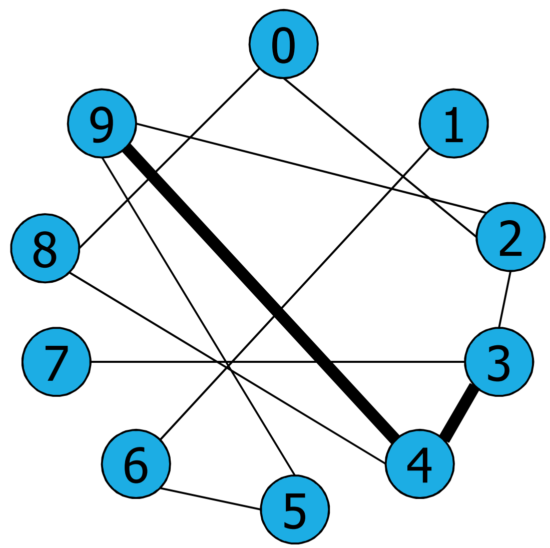
3 4

Input

9 4

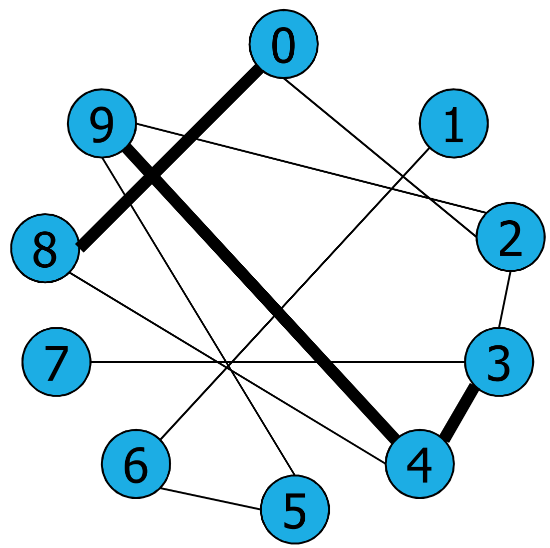
Output

9 4



Input

8 0



Output

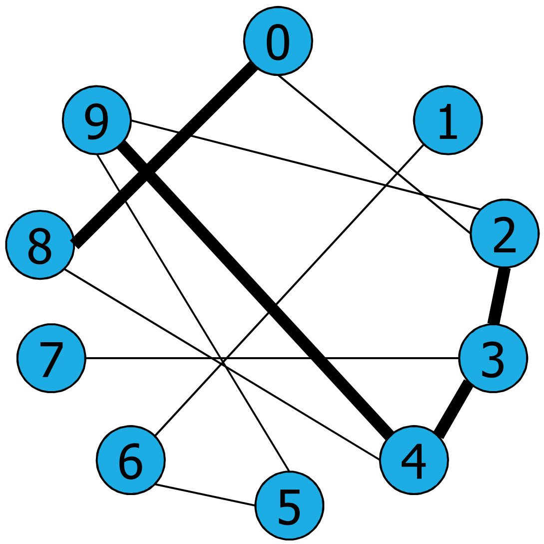
8 0

Input

2 3

Output

2 3

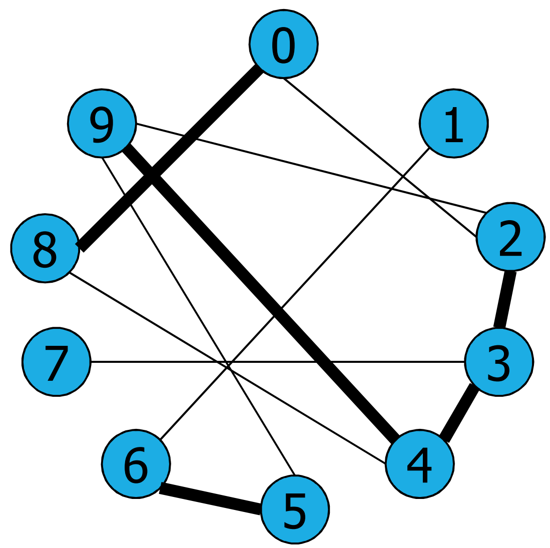


Input

5 6

Output

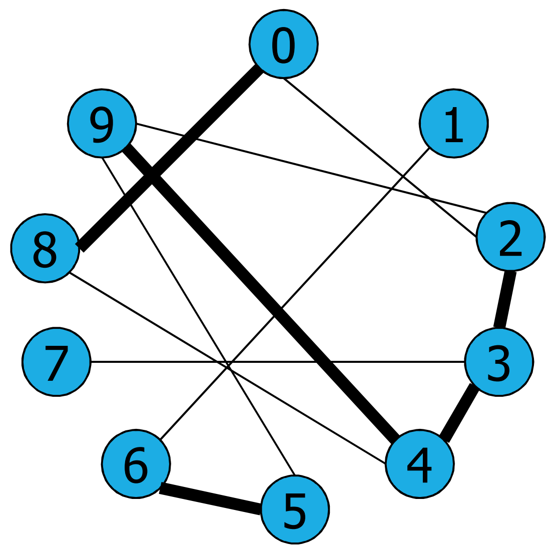
5 6



Input

2 9

Output



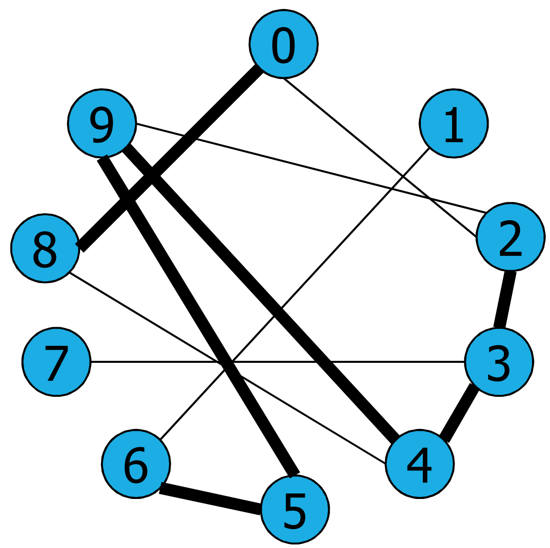
Esiste già il cammino 2-3-4-9

Input

5 9

Output

5 9

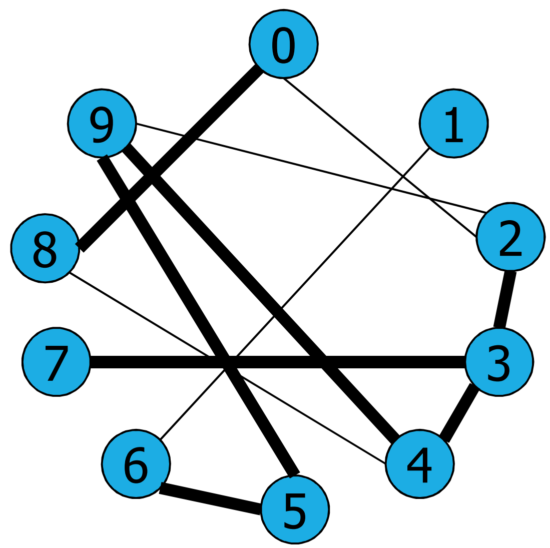


Input

7 3

Output

7 3

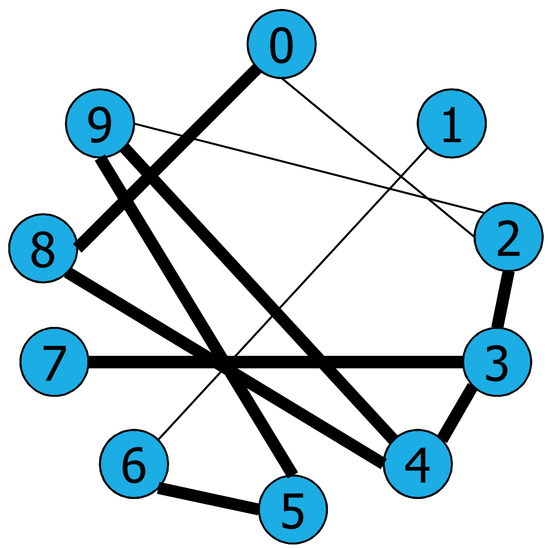


Input

4 8

Output

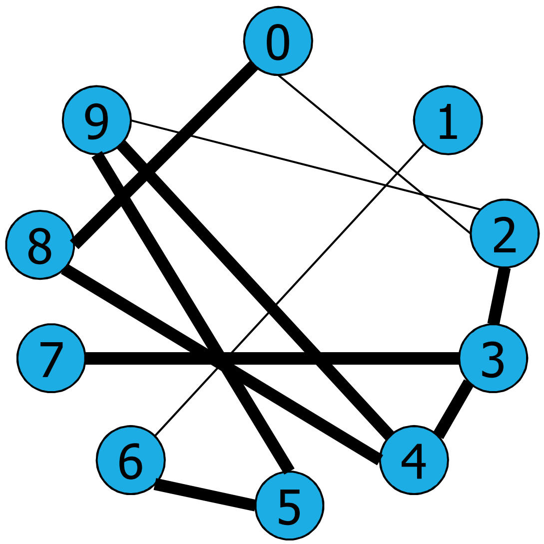
4 8



Input

5 6

Output

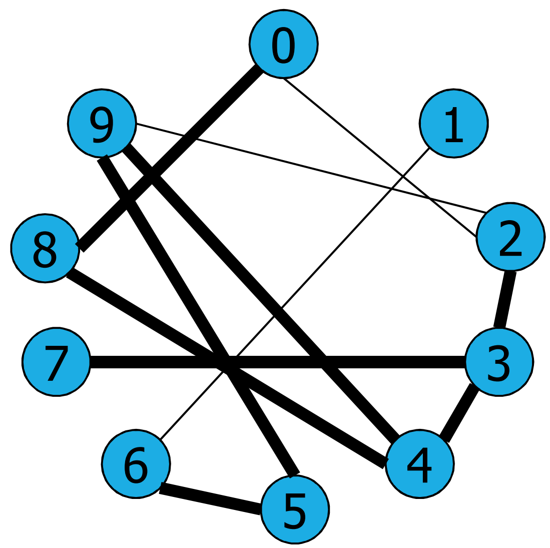


Esiste già il cammino 5-6

Input

0 2

Output



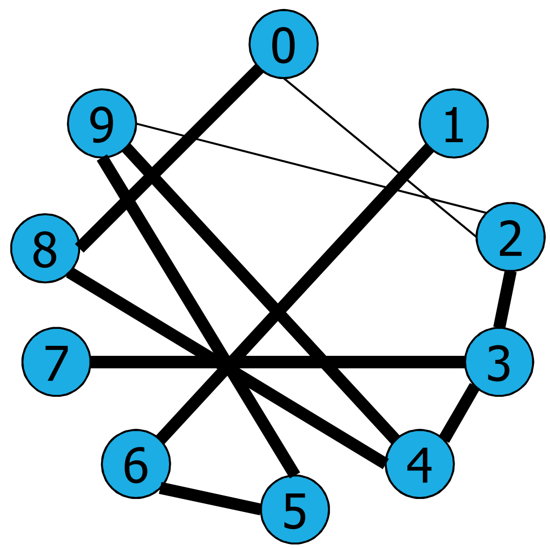
Esiste già il cammino 0-8-4-3-2

Input

6 1

Output

6 1



Approccio on-line

Ipotesi:

- non abbiamo a disposizione il grafo
- lavoriamo coppia per coppia (on-line), mantenendo e aggiornando le informazioni necessarie per determinare la connettività
- ogni coppia è formata da 2 interi tra 0 e N-1

Insiemi S_i delle coppie connesse, inizialmente tanti insiemi quanti i nodi, ogni nodo è connesso solo con se stesso

- Operazioni astratte:
  - find: trova l'insieme a cui appartiene un oggetto
  - union: unisci due insiemi

Algoritmo: ripeti per tutte le coppie (p, q)

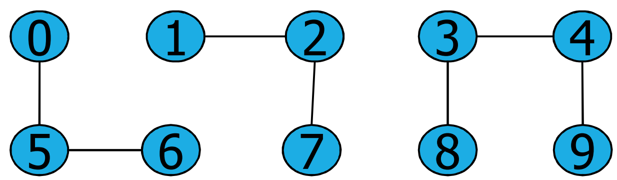
- leggi la coppia (p, q)
- esegui find su p: trova S_p tale che $p \in S_p$
- esegui find su q: trova S_q tale che $q \in S_q$
- se S_p e S_q coincidono, passa alla coppia successiva, altrimenti esegui union di S_p e S_q

Quick find

Rappresentazione degli insiemi S_i delle coppie connesse mediante un vettore id:

- inizialmente id[i] = i (nessuna connessione)
- se p e q sono connessi, id[p] = id[q]

Esempio: il seguente grafo



sarebbe rappresentato così:

id	0	1	1	8	8	0	0	1	8	8
	0	1	2	3	4	5	6	7	8	9

Algoritmo:

- ripeti per tutte le coppie (p, q):
  - leggi la coppia (p, q)
  - se la coppia è connessa (id[p] = id[q]), non fare nulla e passa alla coppia successiva, altrimenti connetti la coppia, scandendo il vettore e cambiando gli elementi che valgono id[p] in id[q]

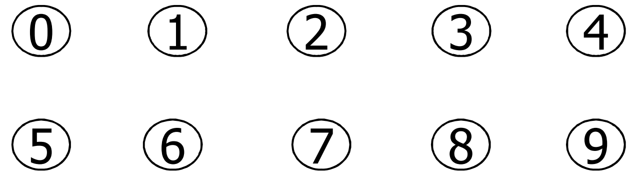
- find: semplice riferimento a una cella del vettore id[indice], costo unitario O(1)
- union: scansione del vettore per cambiare gli elementi che valgono id[p] in id[q], costo lineare nella dimensione del vettore O(n)
- complessivamente il numero di operazioni è legato a

  num. coppie * dimensione del vettore

Rappresentazione ad albero

- Alcuni oggetti rappresentano l’insieme cui essi stessi appartengono
- Gli altri oggetti puntano al rappresentante del loro insieme.

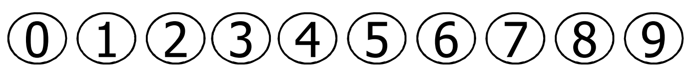
Esempio

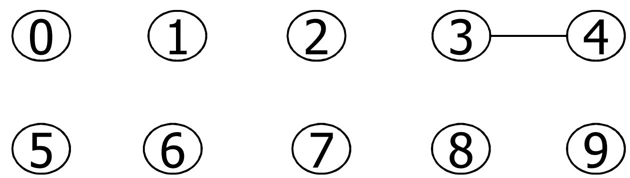


Inizialmente

id										
	0	1	2	3	4	5	6	7	8	9
	0	1	2	3	4	5	6	7	8	9

S_0 = {0}, S_1 = {1}, S_2 = {2}, S_3 = {3}, S_4 = {4}
S_5 = {5}, S_6 = {6}, S_7 = {7}, S_8 = {8}, S_9 = {9}





p q = 3 4

id

0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9

id[p]=3 $\neq$ id[q]=4

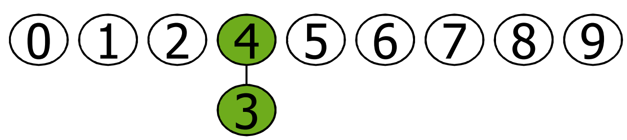
cambia tutti i valori id[p] in id[q]

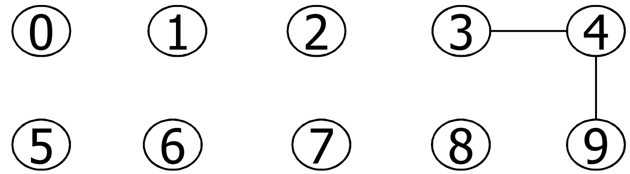
id

0	1	2	4	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9

S_0 = {0}, S_1 = {1}, S_2 = {2}, $S_{3\text{-}4}$ = {3,4},

S_5 = {5}, S_6 = {6}, S_7 = {7}, S_8 = {8}, S_9 = {9}





p q = 4 9

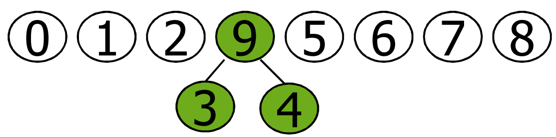
id	0	1	2	4	4	5	6	7	8	9
	0	1	2	3	4	5	6	7	8	9

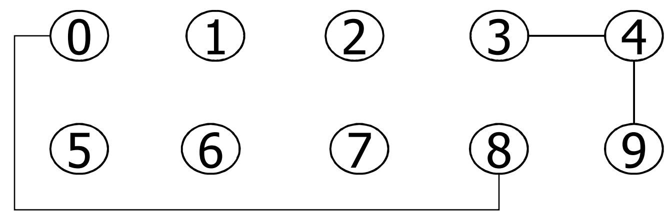
id[p]=4 $\neq$ id[q]=9

cambia tutti i valori id[p] in id[q]

id	0	1	2	9	9	5	6	7	8	9
	0	1	2	3	4	5	6	7	8	9

S_0 = {0}, S_1 = {1}, S_2 = {2}, $S_{3\text{-}4\text{-}9}$ = {3,4,9},
S_5 = {5}, S_6 = {6}, S_7 = {7}, S_8 = {8}





p q = 8 0

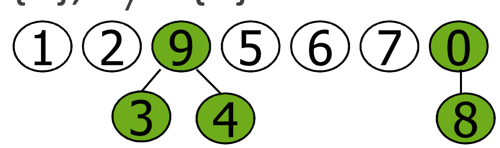
id									
0	1	2	9	9	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9

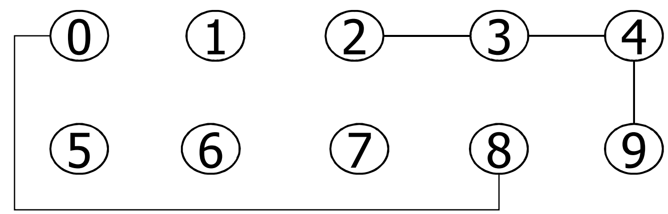
$id[p]=8 \neq id[q]=0$

cambia tutti i valori id[p] in id[q]

id									
0	1	2	9	9	5	6	7	0	9
0	1	2	3	4	5	6	7	8	9

$S_{0\text{-}8} = \{0,8\}$, $S_1 = \{1\}$, $S_2 = \{2\}$, $S_{3\text{-}4\text{-}9} = \{3,4,9\}$,
$S_5 = \{5\}$, $S_6 = \{6\}$, $S_7 = \{7\}$





p q = 2 3

id									
0	1	2	9	9	5	6	7	0	9
0	1	2	3	4	5	6	7	8	9

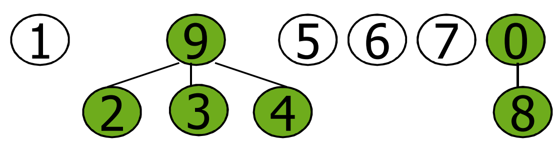
$id[p]=2 \neq id[q]=9$

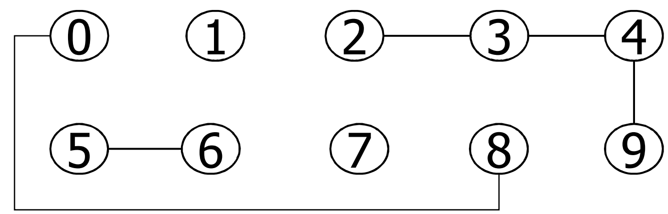
cambia tutti i valori id[p] in id[q]

id									
0	1	9	9	9	5	6	7	0	9
0	1	2	3	4	5	6	7	8	9

$S_{0\text{-}8} = \{0,8\}$, $S_1 = \{1\}$, $S_{2\text{-}3\text{-}4\text{-}9} = \{2,3,4,9\}$,

$S_5 = \{5\}$, $S_6 = \{6\}$, $S_7 = \{7\}$





p q = 5 6

id

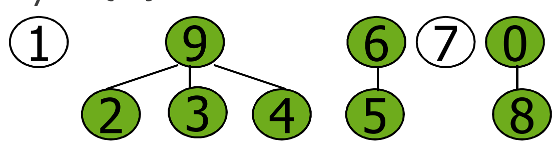
id	0	1	9	9	9	5	6	7	0	9
	0	1	2	3	4	5	6	7	8	9

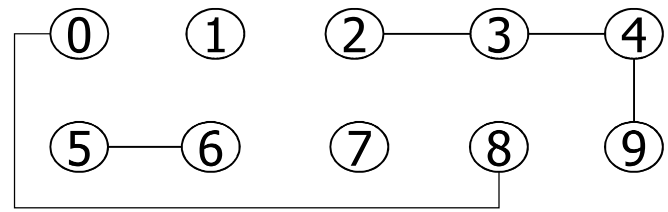
id[p]=5 $\neq$ id[q]=6

cambia tutti i valori id[p] in id[q]

id	0	1	9	9	9	6	6	7	0	9
	0	1	2	3	4	5	6	7	8	9

$S_{0\text{-}8} = \{0,8\}$, $S_1 = \{1\}$, $S_{2\text{-}3\text{-}4\text{-}9} = \{2,3,4,9\}$,
$S_{5\text{-}6} = \{5,6\}$, $S_7 = \{7\}$





p q = 2 9

id	0	1	9	9	9	6	6	7	0	9
	0	1	2	3	4	5	6	7	8	9

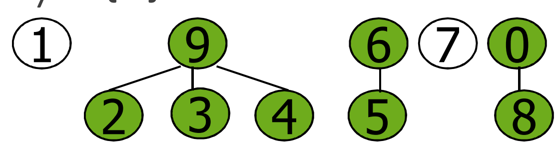
id[p]=9 = id[q]=9

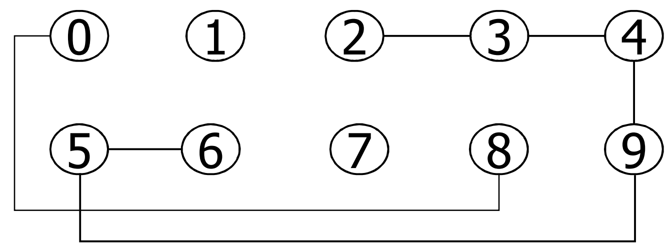
non cambia nulla

id	0	1	9	9	9	6	6	7	0	9
	0	1	2	3	4	5	6	7	8	9

$S_{0\text{-}8} = \{0,8\}$, $S_1 = \{1\}$, $S_{2\text{-}3\text{-}4\text{-}9} = \{2,3,4,9\}$,

$S_{5\text{-}6} = \{5,6\}$, $S_7 = \{7\}$





p q = 5 9

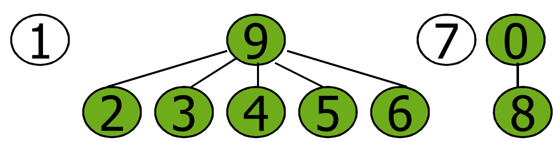
id										
	0	1	9	9	9	6	6	7	0	9
	0	1	2	3	4	5	6	7	8	9

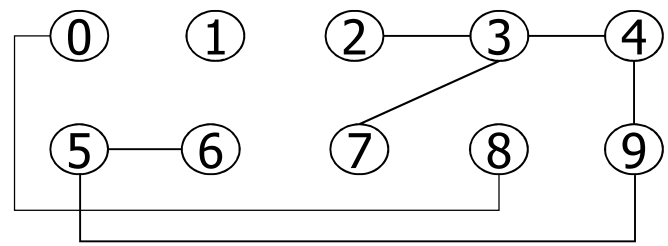
id[p]=6 $\neq$ id[q]=9

cambia tutti i valori id[p] in id[q]

id										
	0	1	9	9	9	9	9	7	0	9
	0	1	2	3	4	5	6	7	8	9

$S_{0\text{-}8}$ = {0,8}, S_1 = {1}, $S_{2\text{-}3\text{-}4\text{-}5\text{-}6\text{-}9}$ = {2,3,4,5,6,9}, S_7 = {7}





p q = 7 3

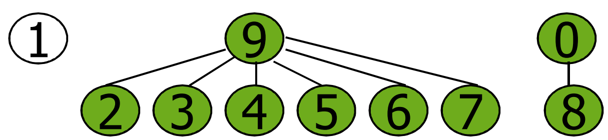
id									
0	1	9	9	9	9	9	7	0	9
0	1	2	3	4	5	6	7	8	9

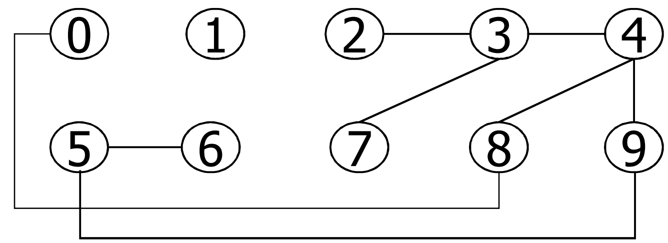
$id[p]=7 \neq id[q]=9$

cambia tutti i valori id[p] in id[q]

id									
0	1	9	9	9	9	9	9	0	9
0	1	2	3	4	5	6	7	8	9

$S_{0\text{-}8} = \{0,8\}$, $S_1 = \{1\}$, $S_{2\text{-}3\text{-}4\text{-}5\text{-}6\text{-}7\text{-}9} = \{2,3,4,5,6,7,9\}$





p q = 4 8

id

0	1	2	3	4	5	6	7	8	9
0	1	9	9	9	9	9	9	0	9

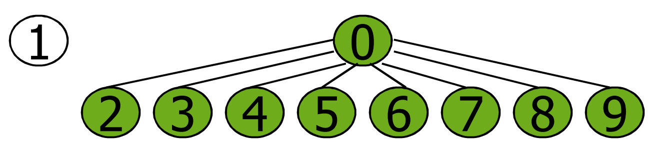
id[p]=9 $\neq$ id[q]=0

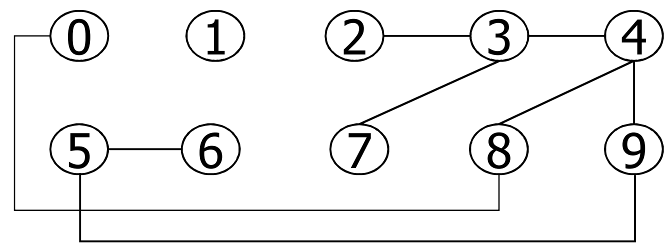
cambia tutti i valori id[p] in id[q]

id

0	1	2	3	4	5	6	7	8	9
0	1	0	0	0	0	0	0	0	0

$S_1 = \{1\}$, $S_{0\text{-}2\text{-}3\text{-}4\text{-}5\text{-}6\text{-}7\text{-}8\text{-}9} = \{0,2,3,4,5,6,7,8,9\}$





p q = 5 6

id

0	1	2	3	4	5	6	7	8	9
0	1	0	0	0	0	0	0	0	0

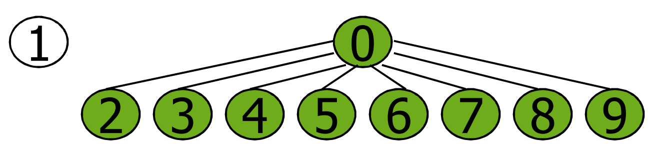
id[p]=0 = id[q]=0

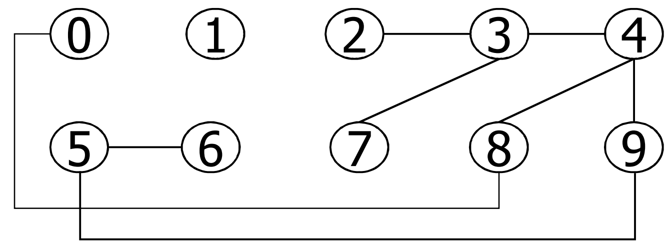
non cambia nulla

id

0	1	2	3	4	5	6	7	8	9
0	1	0	0	0	0	0	0	0	0

$S_1 = \{1\}, S_{0\text{-}2\text{-}3\text{-}4\text{-}5\text{-}6\text{-}7\text{-}8\text{-}9} = \{0,2,3,4,5,6,7,8,9\}$





p q = 0 2

id

0	1	2	3	4	5	6	7	8	9
0	1	0	0	0	0	0	0	0	0

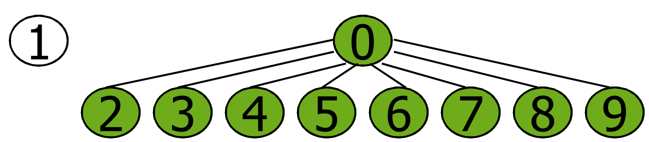
id[p]=0 = id[q]=0

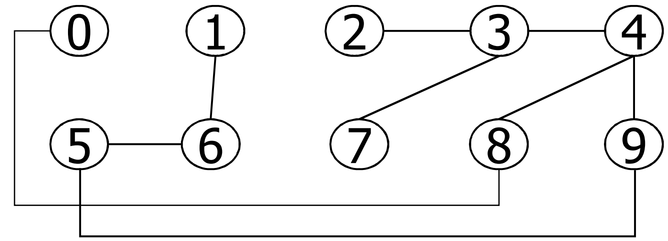
non cambia nulla

id

0	1	2	3	4	5	6	7	8	9
0	1	0	0	0	0	0	0	0	0

S_1 = {1}, $S_{0\text{-}2\text{-}3\text{-}4\text{-}5\text{-}6\text{-}7\text{-}8\text{-}9}$ = {0,2,3,4,5,6,7,8,9}





p q = 6 1

id

0	1	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9

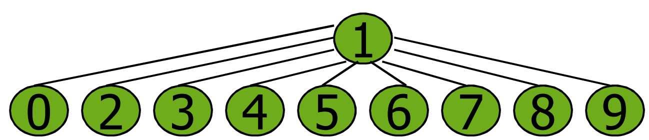
id[p]=0 = id[q]=1

cambia tutti i valori id[p] in id[q]

id

1	1	1	1	1	1	1	1	1	1
0	1	2	3	4	5	6	7	8	9

$S_{0\text{-}1\text{-}2\text{-}3\text{-}4\text{-}5\text{-}6\text{-}7\text{-}8\text{-}9} = \{0,1,2,3,4,5,6,7,8,9\}$



```
#include <stdio.h>
#define N 10000
main() {
  int i, t, p, q, id[N];
  for (i=0; i<N; i++)
    id[i] = i;
  printf("Input pair p q:  ");
  while (scanf("%d %d", &p, &q) ==2) {
    if (id[p] == id[q])
      printf("%d %d already connected\n", p,q);
    else {
      for (t = id[p], i = 0; i < N; i++)
        if (id[i] == t)
          id[i] = id[q];
        printf("pair %d %d not yet connected\n", p, q);
      }
      printf("Input pair p q:  ");
  }
}
```

Quick union

Rappresentazione degli insiemi S_i delle coppie connesse mediante un vettore id:

- inizialmente tutti gli oggetti puntano a se stessi

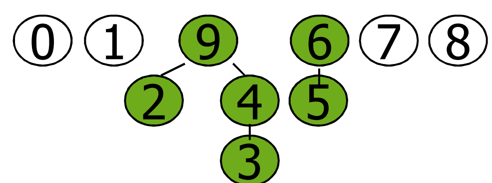
  id[i] = i (nessuna connessione)
- ogni oggetto punta o a un oggetto cui è connesso o a se stesso (no cicli).

Indicando con (id[i])* = id[id[id[… id[i]]]], se gli oggetti i e j sono connessi

(id[i])* = (id[j])*

Esempio:

id	0	1	9	4	9	6	6	7	8	9
	0	1	2	3	4	5	6	7	8	9



Algoritmo:

- ripeti per tutte le coppie (p, q):
  - leggi la coppia (p, q)
  - se (id[p])* = (id[q])* non fare nulla (la coppia è già connessa) e passa alla coppia successiva, altrimenti id[(id[p])*] = (id[q])* (connetti la coppia).

- find: percorrimento di una “catena” di oggetti, costo al massimo lineare nel numero di oggetti, in generale inferiore O(n)
- union: semplice in quanto basta far sì che un oggetto punti all’altro, costo unitario O(1)
- complessivamente il numero di operazioni è legato a

  num. coppie * lunghezza della “catena”

Esempio

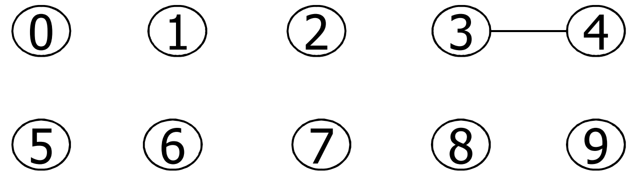
0 1 2 3 4

5 6 7 8 9

Inizialmente

id	0	1	2	3	4	5	6	7	8	9
	0	1	2	3	4	5	6	7	8	9

0 1 2 3 4 5 6 7 8 9



p q = 3 4

id

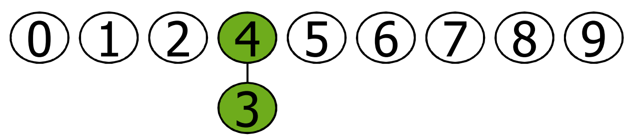
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9

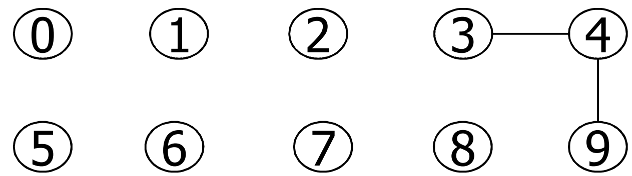
id[p]=3 $\neq$ id[q]=4

faccio in modo che p punti a q: id[p]=4

id

0	1	2	4	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9





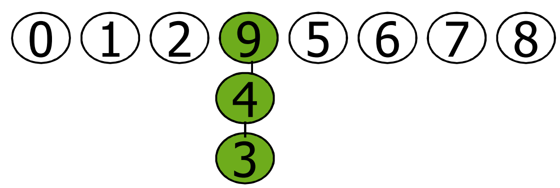
p q = 4 9

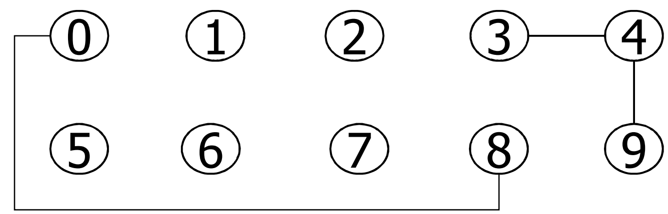
id									
0	1	2	4	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9

id[p]=4 $\neq$ id[q]=9

faccio in modo che p punti a q: id[p]=9

id									
0	1	2	4	9	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9





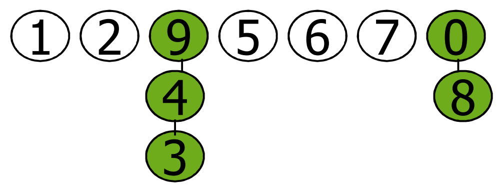
p q = 8 0

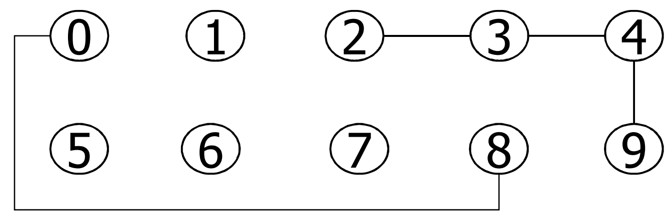
id	0	1	2	4	9	5	6	7	8	9
	0	1	2	3	4	5	6	7	8	9

id[p]=8 $\neq$ id[q]=0

faccio in modo che p punti a q: id[p]=0

id	0	1	2	4	9	5	6	7	0	9
	0	1	2	3	4	5	6	7	8	9





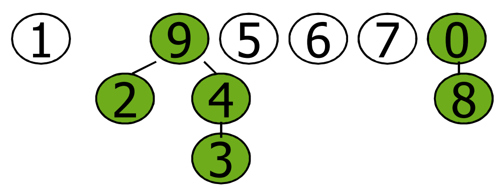
p q = 2 3

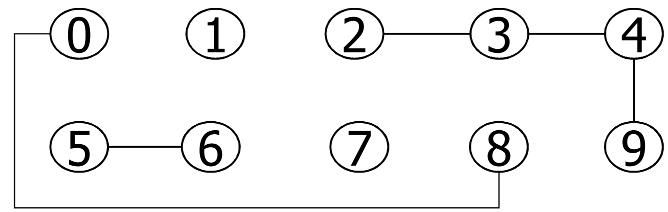
id	0	1	2	4	9	5	6	7	0	9
	0	1	2	3	4	5	6	7	8	9

id[p]=2 $\neq$ id[id[id[q]]]=9

faccio in modo che p punti a q: id[p]=9

id	0	1	9	4	9	5	6	7	0	9
	0	1	2	3	4	5	6	7	8	9





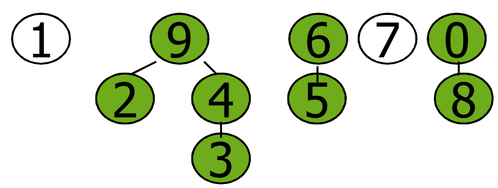
p q = 5 6

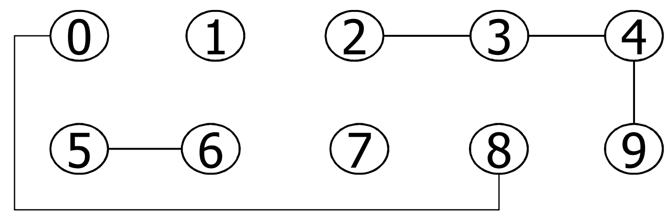
id	0	1	9	4	9	5	6	7	0	9
	0	1	2	3	4	5	6	7	8	9

id[p]=5 $\neq$ id[q]=6

faccio in modo che p punti a q: id[p]=6

id	0	1	9	4	9	6	6	7	0	9
	0	1	2	3	4	5	6	7	8	9





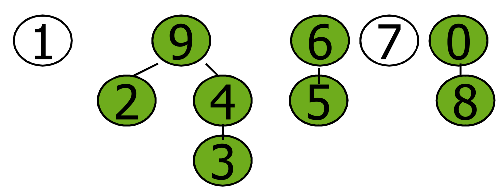
p q = 2 9

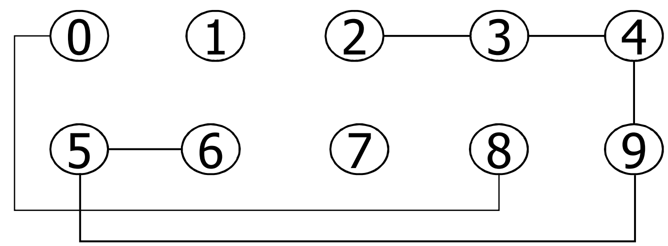
id										
	0	1	9	4	9	6	6	7	0	9
	0	1	2	3	4	5	6	7	8	9

id[id[p]]=9 = id[q]=9

non cambia nulla

id										
	0	1	9	4	9	6	6	7	0	9
	0	1	2	3	4	5	6	7	8	9





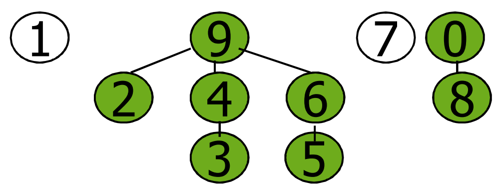
p q = 5 9

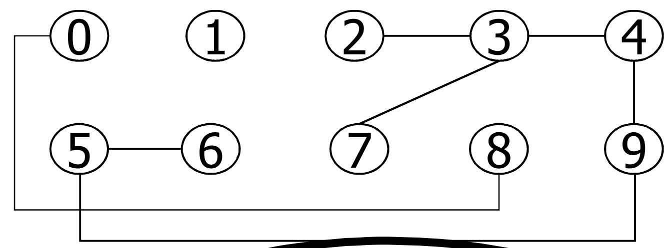
id	0	1	9	4	9	6	6	7	0	9
	0	1	2	3	4	5	6	7	8	9

id[id[p]]=6 $\neq$ id[q]=9

faccio in modo che p punti a q: id[id[p]]=9

id	0	1	9	4	9	6	9	7	0	9
	0	1	2	3	4	5	6	7	8	9





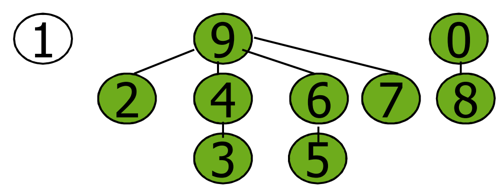
p q = 7 3

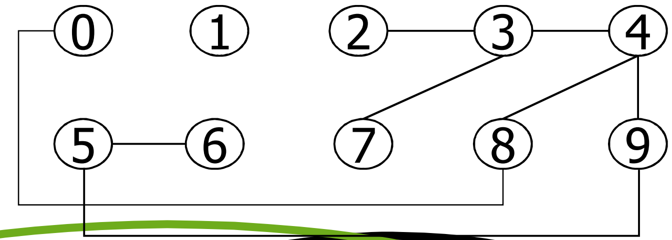
id	0	1	9	4	9	6	9	7	0	9
	0	1	2	3	4	5	6	7	8	9

$id[p]=7 \neq id[id[id[q]]]=9$

faccio in modo che p punti a q: $id[p]=9$

id	0	1	9	4	9	6	9	9	0	9
	0	1	2	3	4	5	6	7	8	9





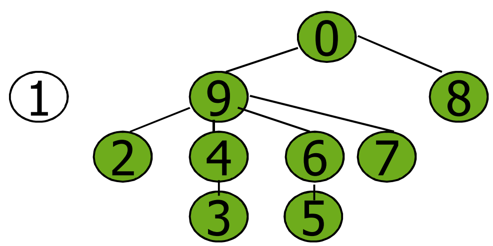
p q = 4 8

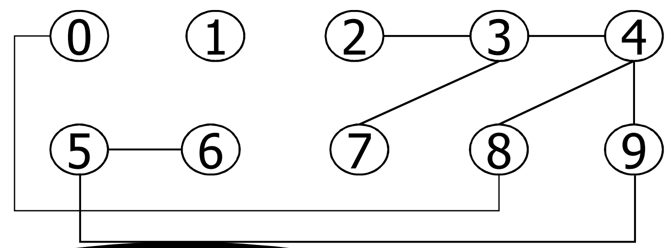
id	0	1	9	4	9	6	9	9	0	9
	0	1	2	3	4	5	6	7	8	9

id[id[p]]=9 $\neq$ id[id[q]]=0

faccio in modo che p punti a q: id[id[p]]=0

id	0	1	9	4	9	6	9	9	0	0
	0	1	2	3	4	5	6	7	8	9





p q = 5 6

id

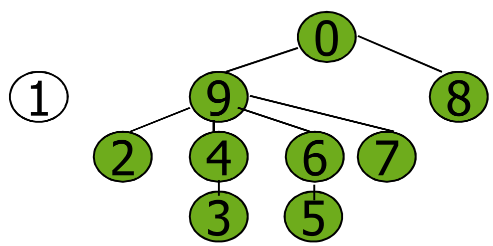
0	1	2	3	4	5	6	7	8	9
0	1	9	4	9	6	9	9	0	0

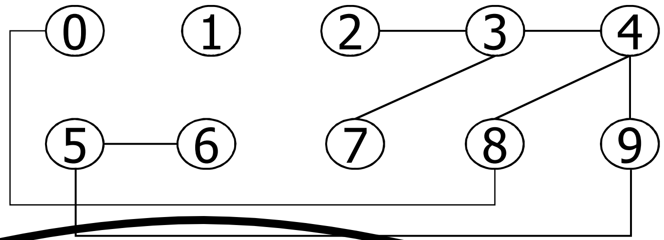
id[id[id[id[p]]]]=0 = id[id[q]]=0

non cambia nulla

id

0	1	2	3	4	5	6	7	8	9
0	1	9	4	9	6	9	9	0	0





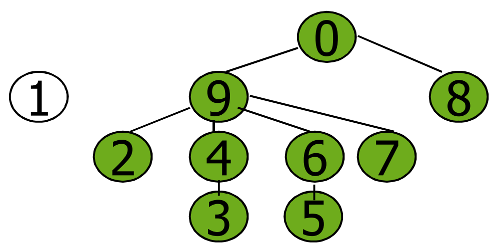
p q = 0 2

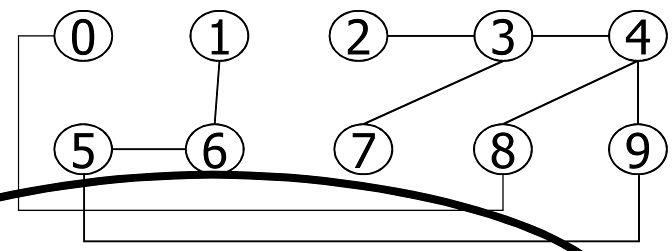
id	0	1	9	4	9	6	9	9	0	0
	0	1	2	3	4	5	6	7	8	9

id[p]=0 = id[id[id[q]]]=0

non cambia nulla

id	0	1	9	4	9	6	9	9	0	0
	0	1	2	3	4	5	6	7	8	9





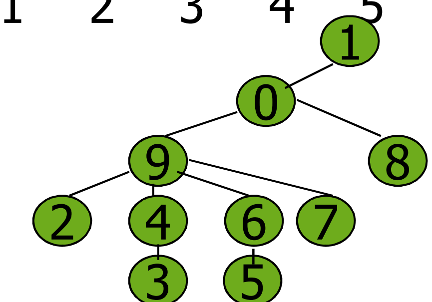
p q = 6 1

id	0	1	9	4	9	6	9	9	0	0
	0	1	2	3	4	5	6	7	8	9

id[id[id[p]]]=0 $\neq$ id[q]=1

faccio in modo che p punti a q: id[id[id[p]]]=1

id	1	1	9	4	9	6	9	9	0	0
	0	1	2	3	4	5	6	7	8	9

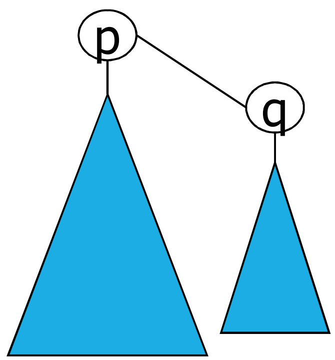


```c
#include <stdio.h>
#define N 10000
main() {
  int i, j, p, q, id[N];
  for(i=0; i<N; i++)
    id[i] = i;
  printf("Input pair p q:  ");
  while (scanf("%d %d", &p, &q) ==2) {
    for (i = p; i!= id[i]; i = id[i]);
    for (j = q; j!= id[j]; j = id[j]);
    if (i == j)
      printf("pair %d %d already connected\n", p,q);
    else {
      id[i] = j;
      printf("pair %d %d not yet connected\n", p, q);
    }
    printf("Input pair p q:  ");
  }
}
```

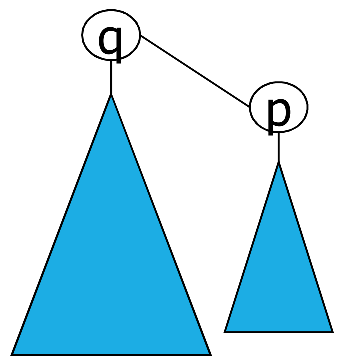
Ottimizzazioni della Quick union

Weighted quick union:

- per ridurre la lunghezza della catena, si mantiene traccia del numero di elementi di ciascun albero (array `sz`) e si collega l'albero più piccolo a quello più grande.
- a seconda di quale tra p e q è l'albero più grande si possono avere le 2 seguenti soluzioni:

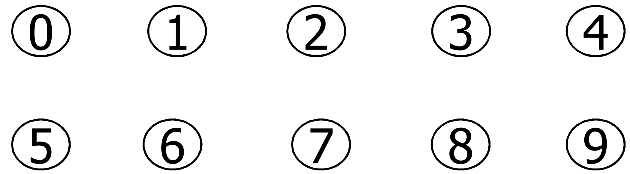


oppure



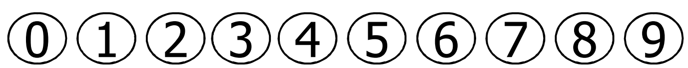
NB: è irrilevante se p appare alla destra o alla sinistra di q.

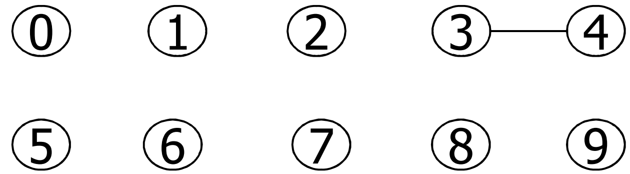
Esempio



Inizialmente

id									
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9





p q = 3 4

id

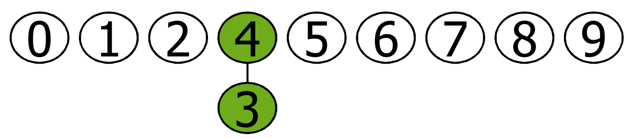
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9

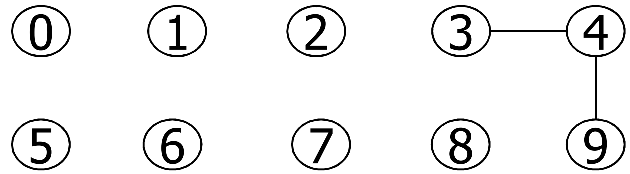
id[p]=3 $\neq$ id[q]=4

faccio in modo che p punti a q: id[p]=4

id

0	1	2	4	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9





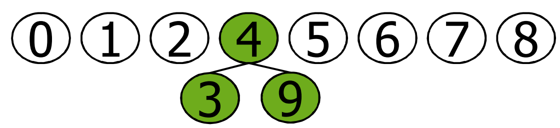
p q = 4 9

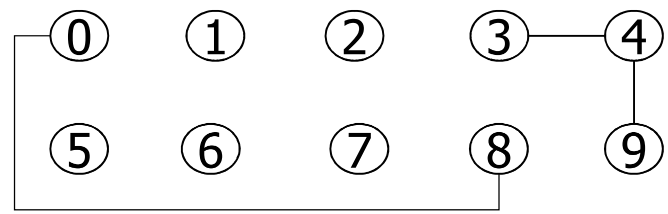
id										
	0	1	2	4	4	5	6	7	8	9
	0	1	2	3	4	5	6	7	8	9

id[p]=4 $\neq$ id[q]=9

faccio in modo che l'albero più piccolo q punti a quello più grande p: id[q]=4

id										
	0	1	2	4	4	5	6	7	8	4
	0	1	2	3	4	5	6	7	8	9





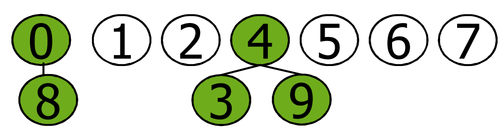
p q = 8 0

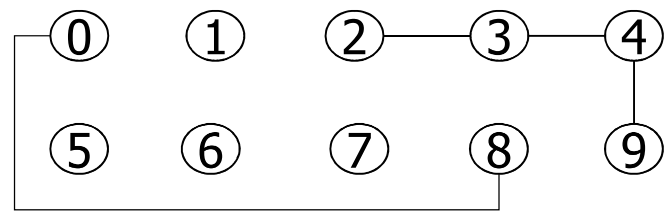
id	0	1	2	4	4	5	6	7	8	4
	0	1	2	3	4	5	6	7	8	9

id[p]=8 $\neq$ id[q]=0

faccio in modo che p punti a q: id[p]=0

id	0	1	2	4	4	5	6	7	0	4
	0	1	2	3	4	5	6	7	8	9





p q = 2 3

id

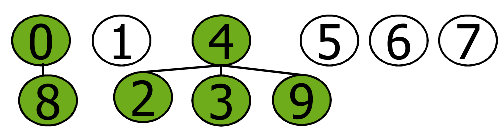
0	1	2	4	4	5	6	7	0	4
0	1	2	3	4	5	6	7	8	9

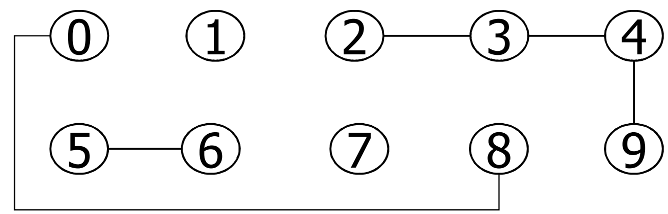
id[p]=2 $\neq$ id[id[q]]=4

faccio in modo che l'albero più piccolo p punti a quello più grande q: id[p]=4

id

0	1	4	4	4	5	6	7	0	4
0	1	2	3	4	5	6	7	8	9





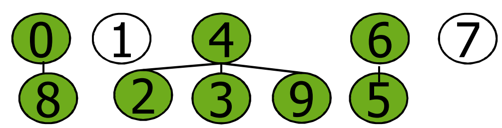
p q = 5 6

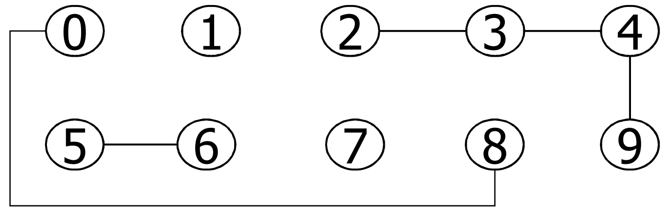
id	0	1	4	4	4	5	6	7	0	4
	0	1	2	3	4	5	6	7	8	9

id[p]=5 $\neq$ id[q]=6

faccio in modo che p punti a q: id[p]=6

id	0	1	4	4	4	6	6	7	0	4
	0	1	2	3	4	5	6	7	8	9





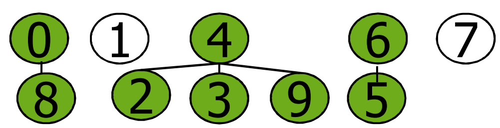
p q = 2 9

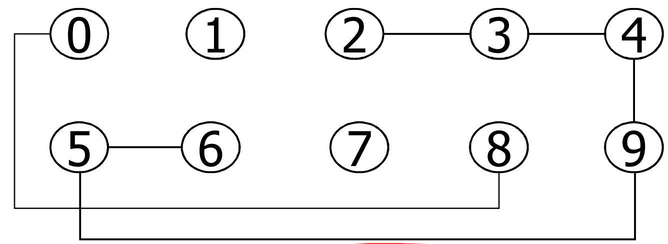
id	0	1	4	4	4	6	6	7	0	4
	0	1	2	3	4	5	6	7	8	9

id[id[p]]=4 = id[q]=4

non cambia nulla

id	0	1	4	4	4	6	6	7	0	4
	0	1	2	3	4	5	6	7	8	9





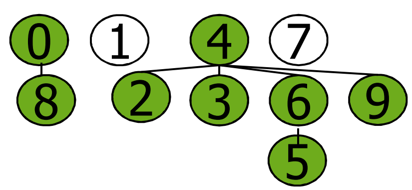
p q = 5 9

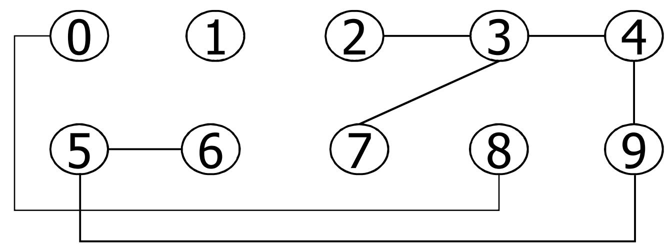
id	0	1	4	4	4	6	6	7	0	4
	0	1	2	3	4	5	6	7	8	9

id[id[p]]=6 $\neq$ id[id[q]]=4

faccio in modo che l'albero più piccolo p punti a quello più grande q: id[id[p]]=4

id	0	1	4	4	4	6	4	7	0	4
	0	1	2	3	4	5	6	7	8	9



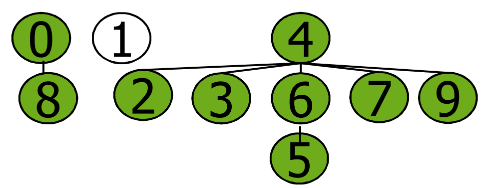


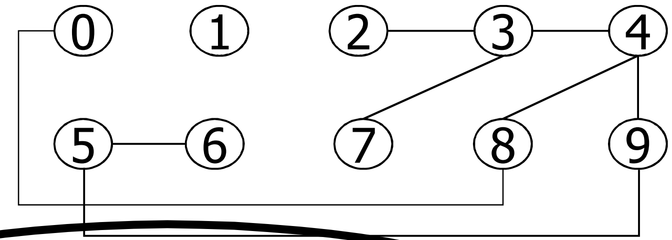
p q = 7 3

id	0	1	4	4	4	6	4	7	0	4
	0	1	2	3	4	5	6	7	8	9

id[p]=7 $\neq$ id[id[q]]=4
faccio in modo che l'albero più piccolo p punti a quello più grande q: id[p]=4

id	0	1	4	4	4	6	4	4	0	4
	0	1	2	3	4	5	6	7	8	9





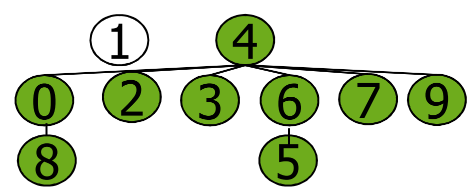
p q = 4 8

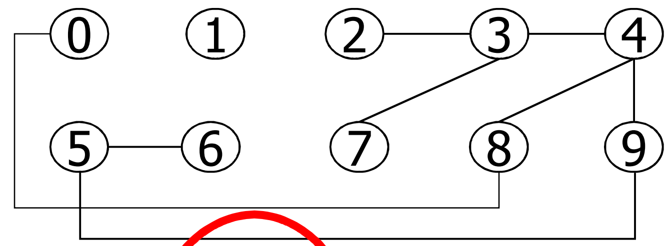
	0	1	2	3	4	5	6	7	8	9
id	0	1	4	4	4	6	4	4	0	4

id[p]=4 $\neq$ id[id[q]]=0

faccio in modo che l'albero più piccolo q punti a quello più grande p: id[id[q]]=4

	0	1	2	3	4	5	6	7	8	9
id	4	1	4	4	4	6	4	4	0	4





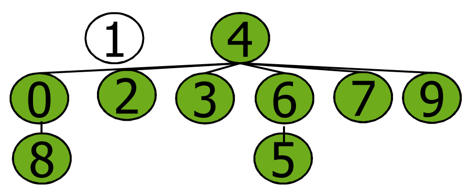
p q = 5 6

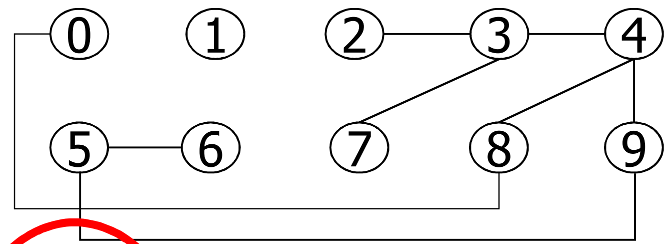
id	4	1	4	4	4	6	4	4	0	4
	0	1	2	3	4	5	6	7	8	9

id[id[id[p]]]=4 = id[id[q]]=4

non cambia nulla

id	4	1	4	4	4	6	4	4	0	4
	0	1	2	3	4	5	6	7	8	9





p q = 0 2

id

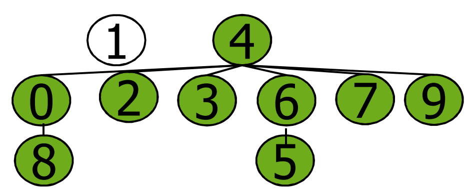
4	1	4	4	4	6	4	4	0	4
0	1	2	3	4	5	6	7	8	9

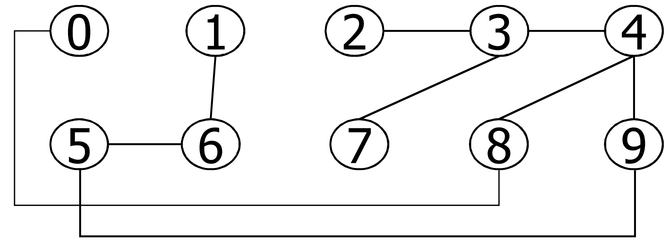
id[id[p]]=4 = id[id[q]]=4

non cambia nulla

id

4	1	4	4	4	6	4	4	0	4
0	1	2	3	4	5	6	7	8	9





p q = 6 1

id

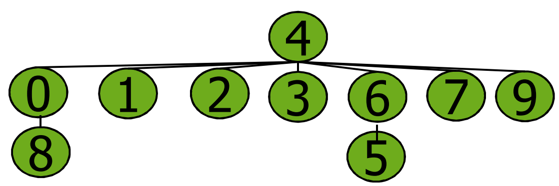
0	1	2	3	4	5	6	7	8	9
4	1	4	4	4	6	4	4	0	4

id[id[p]]=4 $\neq$ id[q]=1

faccio in modo che l'albero più piccolo q punti a quello più grande p: id[q]=4

id

0	1	2	3	4	5	6	7	8	9
4	4	4	4	4	6	4	4	0	4



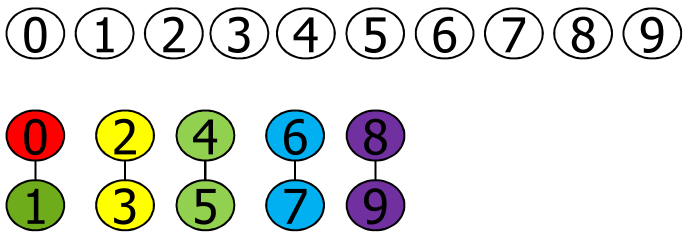
```
...
  int i, j, p, q, id[N], sz[N];
  for(i=0; i<N; i++) { id[i] = i; sz[i] =1; }
  printf("Input pair p q:  ");
  while (scanf("%d %d", &p, &q) ==2) {
    for (i = p; i!= id[i]; i = id[i]);
    for (j = q; j!= id[j]; j = id[j]);
    if (i == j)
      printf("pair %d %d already connected\n", p,q);
    else {
      printf("pair %d %d not yet connected\n", p, q);
      if (sz[i] <= sz[j]) {
        id[i] = j; sz[j] += sz[i]; }
       else { id[j] = i; sz[i] += sz[j];}
    }
    printf("Input pair p q:  ");
  }
...
```

- find: percorrimento di una “catena” di oggetti, costo al massimo logaritmico nel numero di oggetti O(logn)
- union: semplice in quanto basta far sì che un oggetto punti all’altro, costo unitario O(1)
- complessivamente il numero di operazioni è legato a

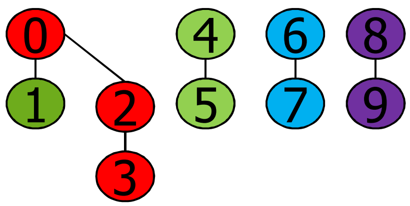
  num. coppie * lunghezza della “catena”
- ma “lunghezza della catena” = altezza dell’albero e quest’ultima cresce in modo logaritmico!

Perché logaritmica?

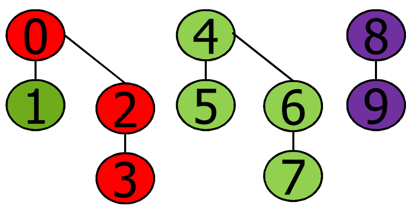
Caso peggiore: dati n elementi, ogni union collega 2 alberi di ugual dimensione:



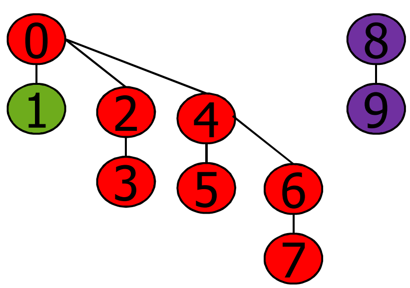
p q = 0 1
p q = 2 3
p q = 4 5
p q = 6 7
p q = 8 9



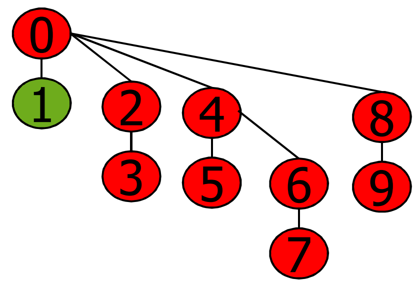
p q = 1 3



p q = 5 7



p q = 3 7

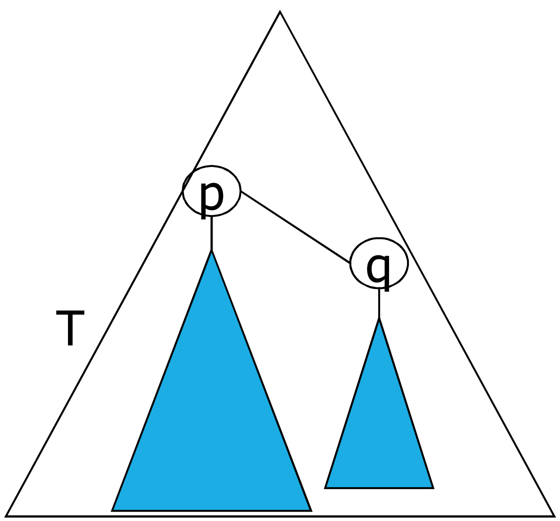


p q = 7 9

Ogni albero che contiene 2^h nodi ha altezza h.
Con la union, nel caso peggiore si fondono 2 alberi con ugual numero di nodi 2^h e si ottiene un albero con 2^{h+1} nodi, quindi con altezza h+1.
L'altezza cresce linearmente con il numero di union effettuate.

Quante union sono necessarie?

Se $T_1 \geq T_2$, ad ogni union di un albero più piccolo in uno più grande si genera un albero la cui dimensione T è almeno il doppio di T_2.



Se ad ogni passo il numero di elementi dell'albero almeno raddoppia e ci sono N elementi, dopo i passi ci saranno almeno 2^i elementi nell'albero. Deve valere $2^i \leq N$, quindi il numero di union è $i \leq \log_2 N$.