L’analisi di complessità degli algoritmi

Paolo Camurati
Dip. Automatica e Informatica
Politecnico di Torino

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 dimensione intero o più semplicemente 5

- Dipendente dalla dimensione n del problema. Esempi:
  - numero di bit 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: $n \to \infty$:

- 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 dello 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 $n = 1M$:

- Algoritmo 1: 20000 s = 333,33 minuti
- Algoritmo 2: 1000 s = 16,67 minuti

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

Analisi della Ricerca Lineare

- Si esaminano n numeri 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 numeri
- Alla seconda iterazione il vettore da esaminare è di circa n/2 numeri
- ....
- Alla i-esima iterazione il vettore da esaminare è di circa $n/2^i$ numeri
- 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.

Analisi dell'Insertion Sort

Due cicli annidati:

- Esterno: n-1 esecuzioni
- Interno nel caso [progressione aritmetica finita di ragione 1 (Gauss, fine XVII sec.)] all'i-esima iterazione c

  Complessità:

  $T(n) = 1+2+3+ \dots +(n-2)+(n-1)$

  $= \Sigma_{1\leq i < n} i = n(n-1)/2$

  T(n) cresce quadraticamente in n.

- Interno nel caso migliore: 1 esecuzione all'i-esima iterazione di quello esterno
- Complessità

$$T(n) = 1 + \underbrace{1 + 1 + \dots\dots\dots + 1}_{n\text{-}1 \text{ volte}} = n\text{-}1$$

T(n) cresce linearmente in n.

Analiticamente, nel caso peggiore, ipotizzando di costo unitario tutte le operazioni:

```
                                             num. di volte
for(i=1; i<n; i++) {                         n
     x = A[i];                               n - 1
     j = i - 1;                              n - 1
     while (j >= 0 && x < A[j]){             ∑_{k=2}^{n} k
       A[j+1] = A[j];                        ∑_{k=2}^{n} (k-1)
       j--;                                  ∑_{k=2}^{n} (k-1)
     }
     A[j+1] = x;                             n - 1
  }
```

$$T(n) = n+(n-1)+(n-1)+\sum_{k=2}^{n} k+\sum_{k=2}^{n}(k\text{-}1) + \sum_{k=2}^{n}(k\text{-}1)+(n-1)$$

Ricordando che:

$\sum_{k=2}^{n} k$ = n*(n+1)/2 -1

$\sum_{\mathrm{k}=2}^{n}(k$-1) = n*(n-1)/2

T(n) = n + 3*(n-1) + n*(n+1)/2 -1

+ 2*(n*(n-1)/2)

= $3/2n^2 + 7/2n - 4$

T(n) cresce quadraticamente.

Ricordando che:

$\sum_{j=2}^{n} j$ = n*(n+1)/2 -1

$\sum_{j=2}^{n} (j\text{-}1)$ = n*(n-1)/2

T(n) = n + 3*(n-1) + n*(n+1)/2 -1
+ 2*(n*(n-1)/2)
= $3/2n^2 + 7/2n - 4$

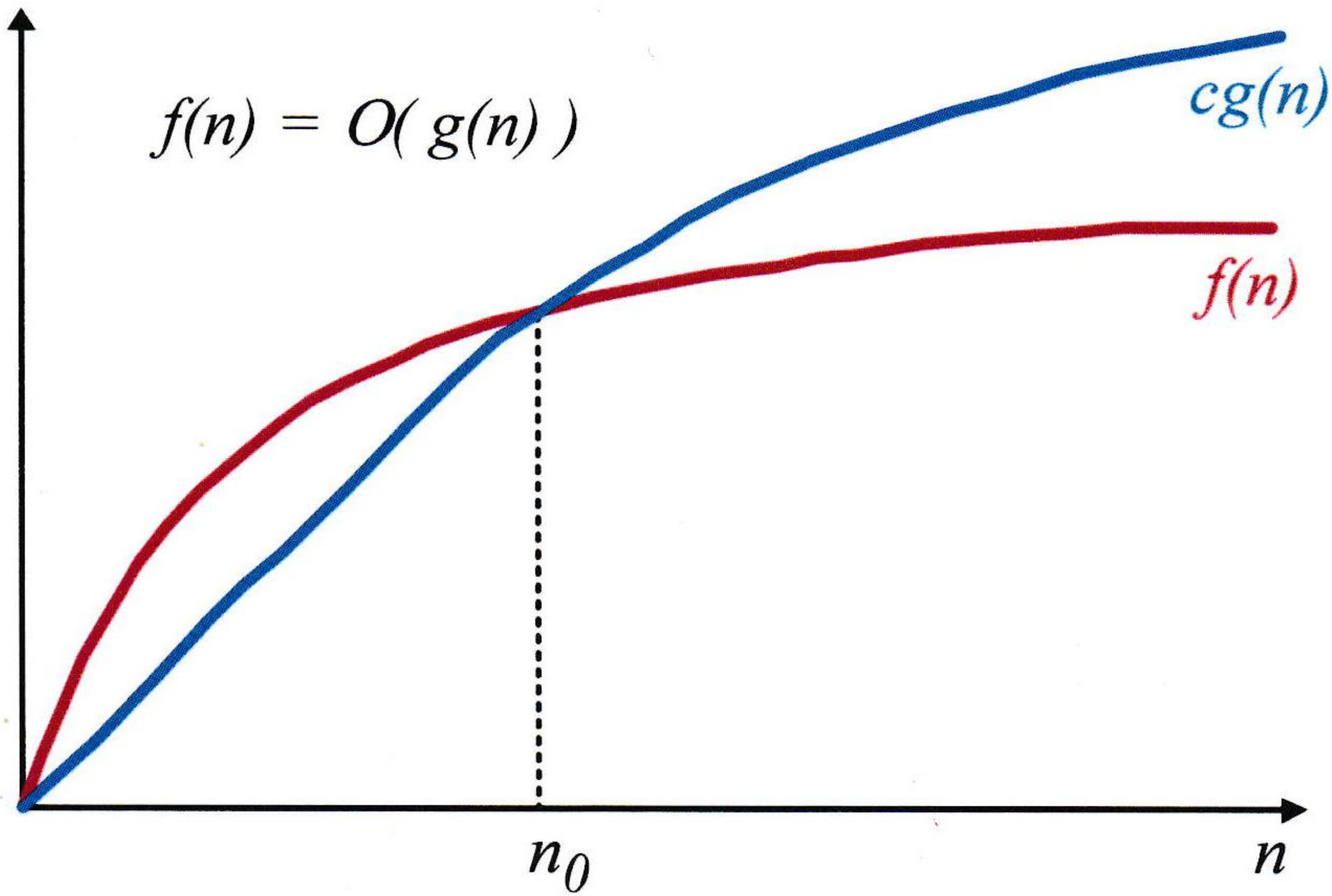
T(n) cresce quadraticamente.

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).


f(n) = O(g(n))
cg(n)
f(n)
n0
n

Esempi:

$T(n) = 3n+2 = O(n)$, $c=4$ e $n_0=2$

$T(n) = 10n^2+4n+2 = O(n^2)$, $c=11$ e $n_0=5$

$T(n) = 3n+3 = O(n^2)$, $c=3$ e $n_0=2$

Teorema:

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

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

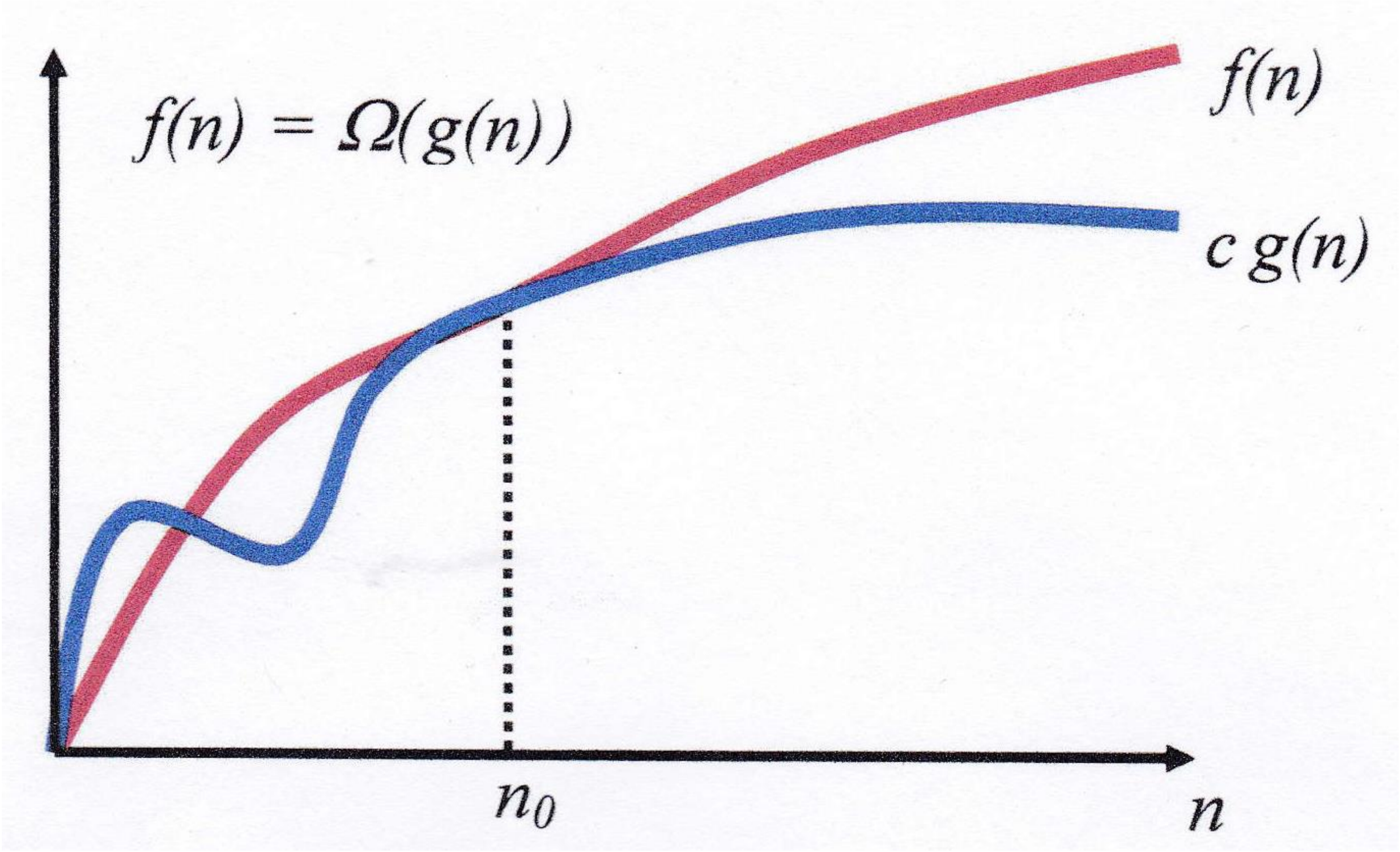
Notazione asintotica Ω

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).

La quantità di tempo <u>sufficiente</u> per eseguire l'algoritmo con qualsiasi dati è g(n).



$f(n) = \Omega(g(n))$
$f(n)$
$c\,g(n)$
n_0
n

Esempi:

$T(n) = 3n+3 = \Omega(n)$, $c=3$ e $n_0=1$

$T(n) = 10n^2+4n+2 = \Omega(n^2)$, $c=1$ e $n_0=1$

$T(n) = 10n^2+4n+2 = \Omega(n)$, $c=1$ e $n_0=1$

Teorema:

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

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

Notazione asintotica Θ

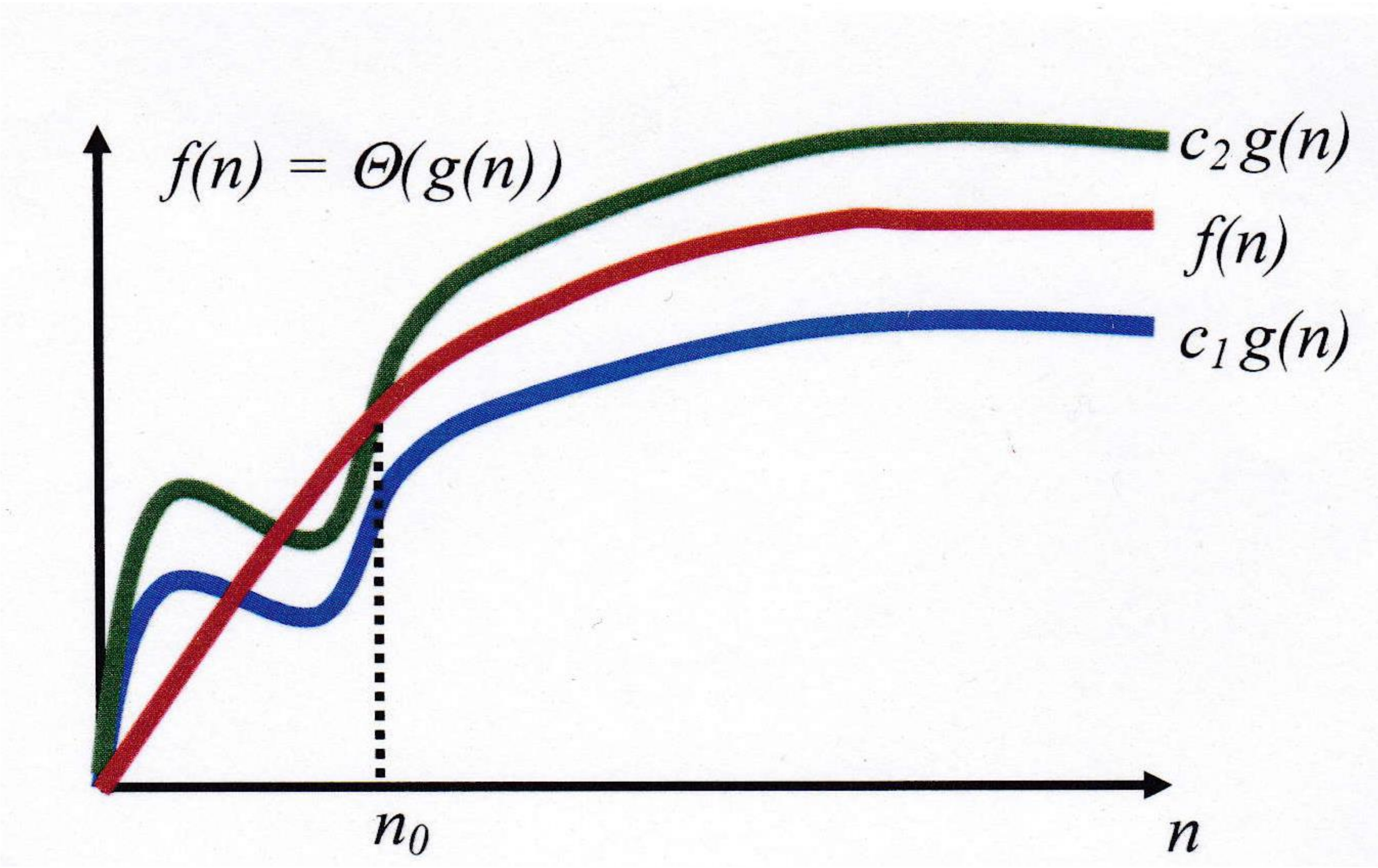
Definizione:

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

$$\exists\ c_1, c_2 > 0,\ \exists\ n_0 > 0 \text{ tale per cui } \forall n \geq n_0$$

$$0 \leq c_1\, g(n) \leq T(n) \leq c_2\, g(n)$$

g(n) = limite asintotico stretto di T(n).


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

Esempi:

$T(n) = 3n+2 = \Theta(n)$, $c_1=3$, $c_2=4$ e $n_0=2$

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

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),

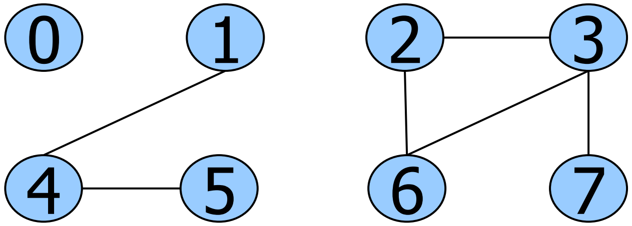
$T(n) = \Theta(g(n)) \Leftrightarrow T(n) = O(g(n))$ e $T(n) = \Omega(g(n))$.

Online Connectivity

- Input: sequenze di coppie di interi (p, q)
- Interpretazione: p è connesso con 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.

Componente connessa: sottoinsieme massimale dei nodi mutuamente raggiungibili



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

3 componenti connesse

- 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

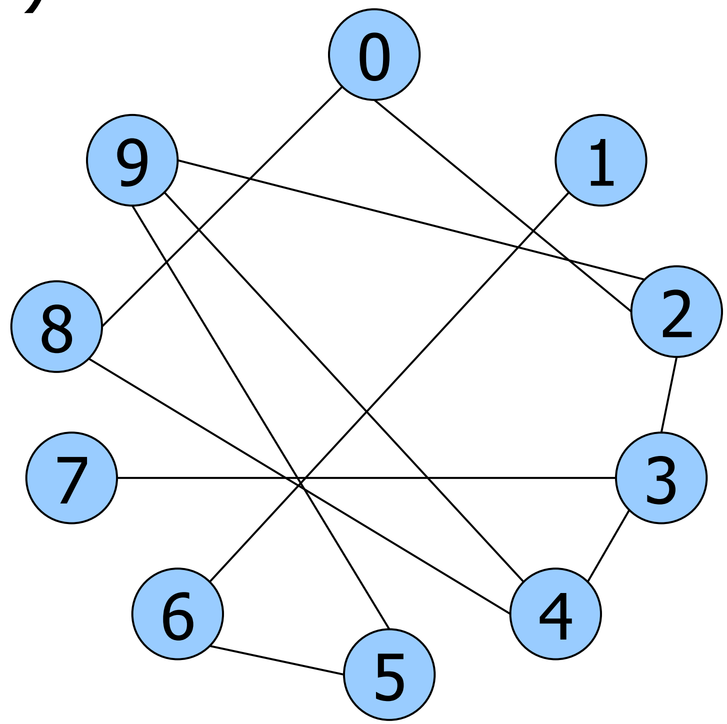
Applicazioni

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

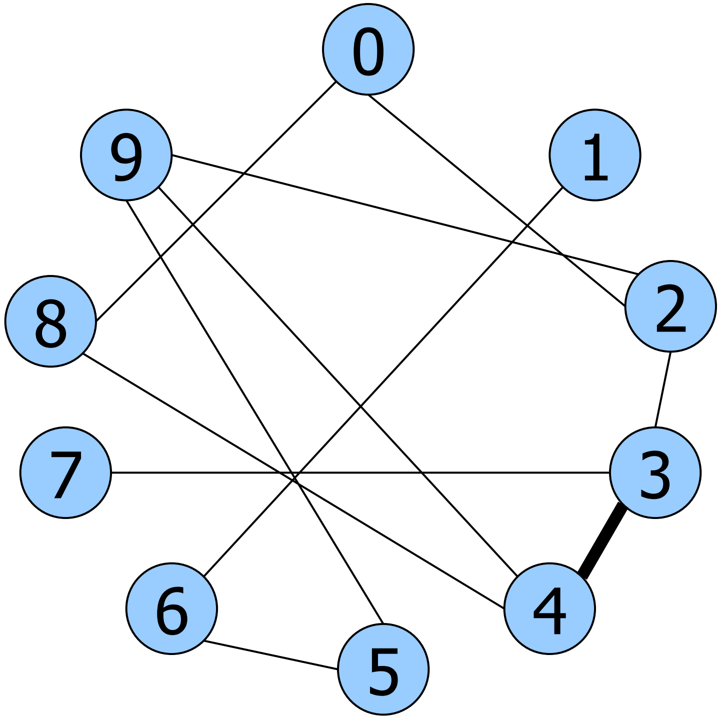
Esempio

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

Grafo: struttura dove sono rappresentati i nodi (vertici) e le loro connessioni (archi).

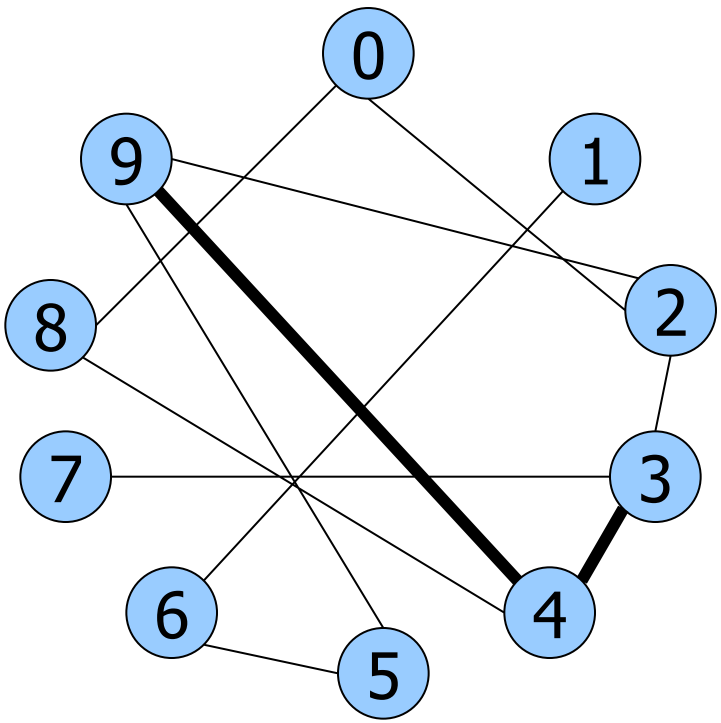


Input
3 4



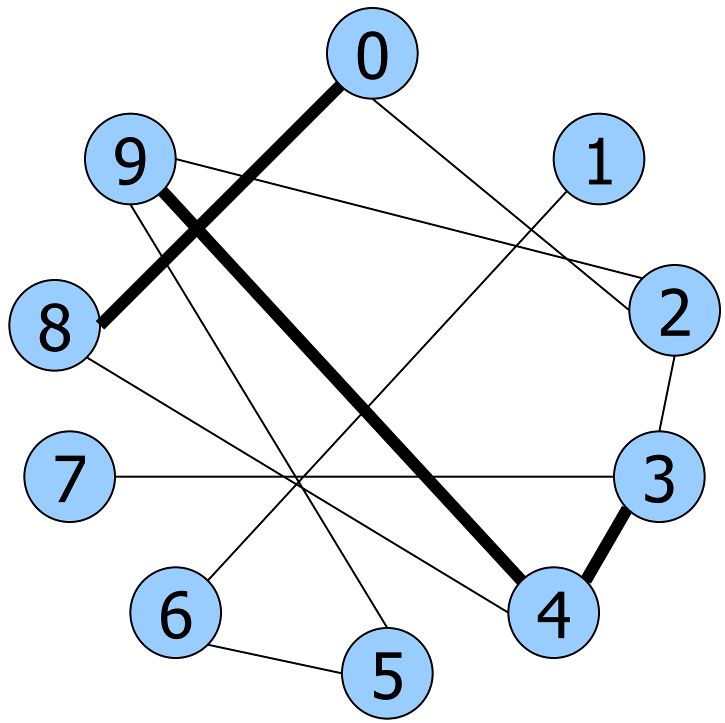
Output
3 4

Input
4 9



Output
4 9

Input
8 0



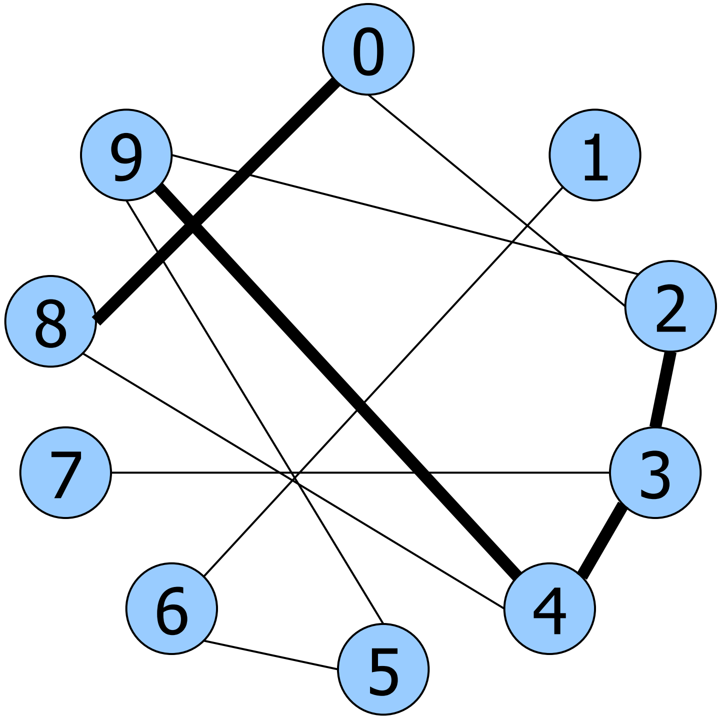
Output
8 0

Input

2 3

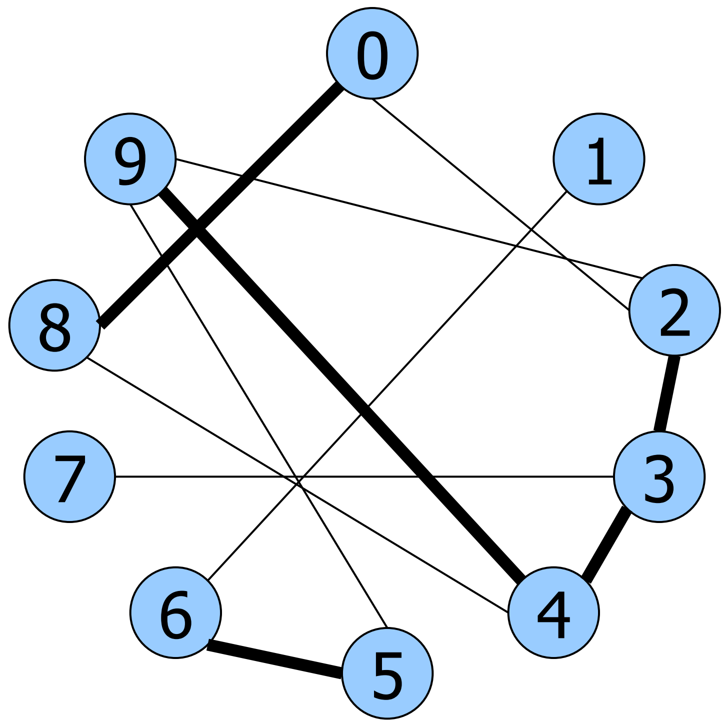
Output

2 3



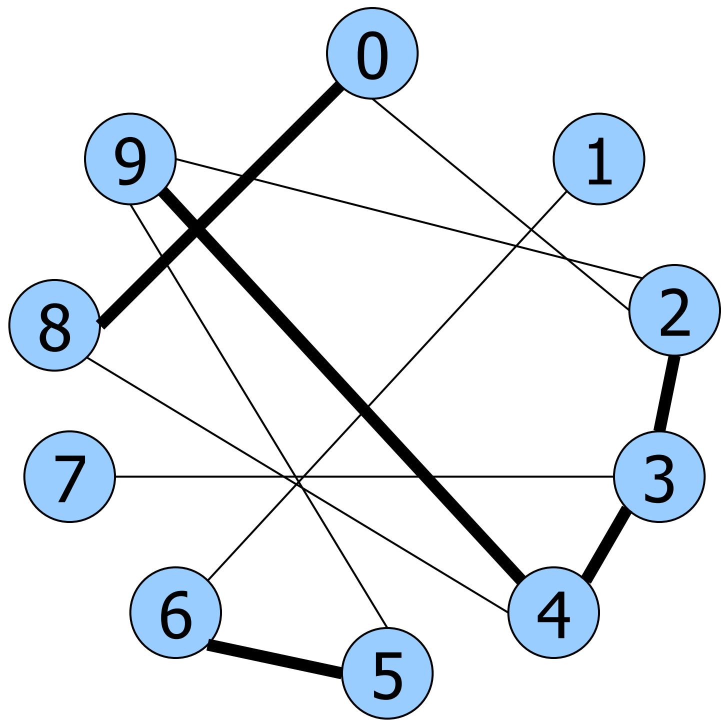
Input
5 6

Output
5 6



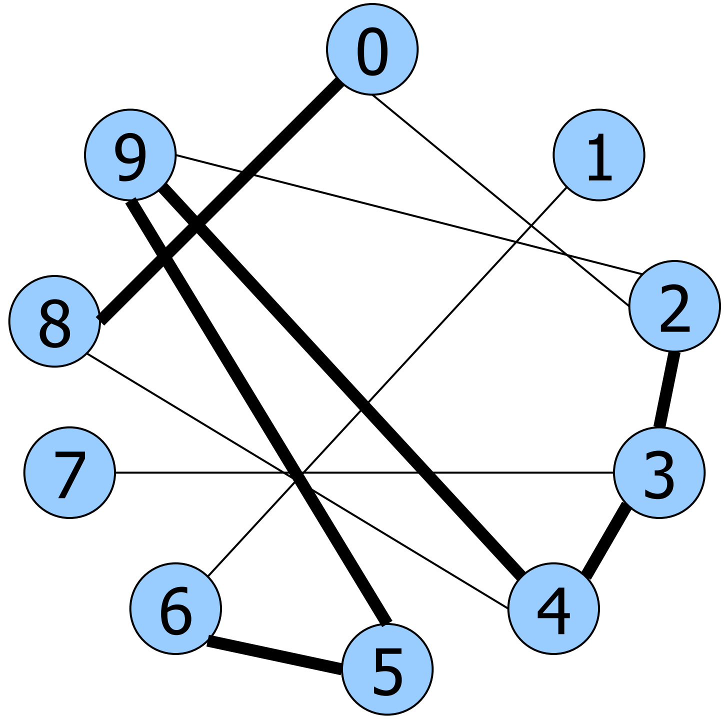
Input
2 9

Output



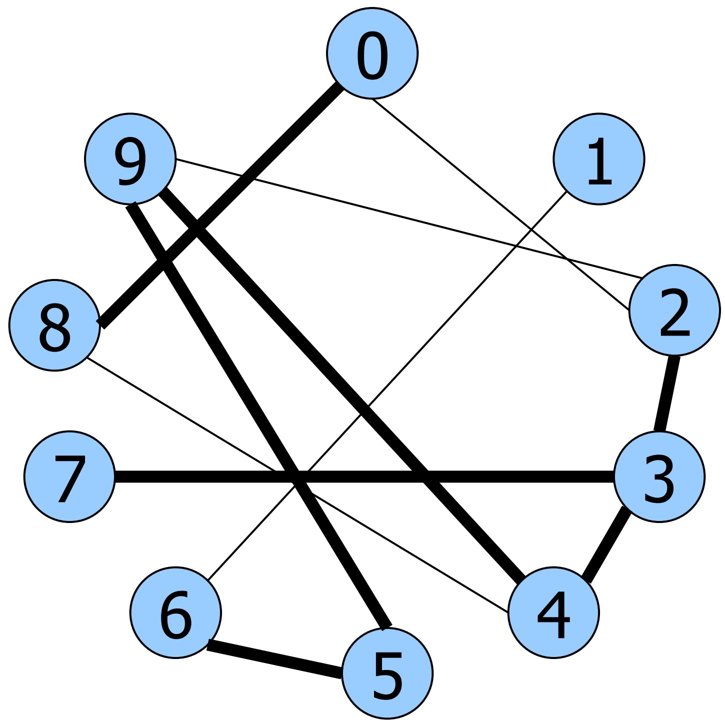
Esiste già il cammino 2-3-4-9

Input
5 9



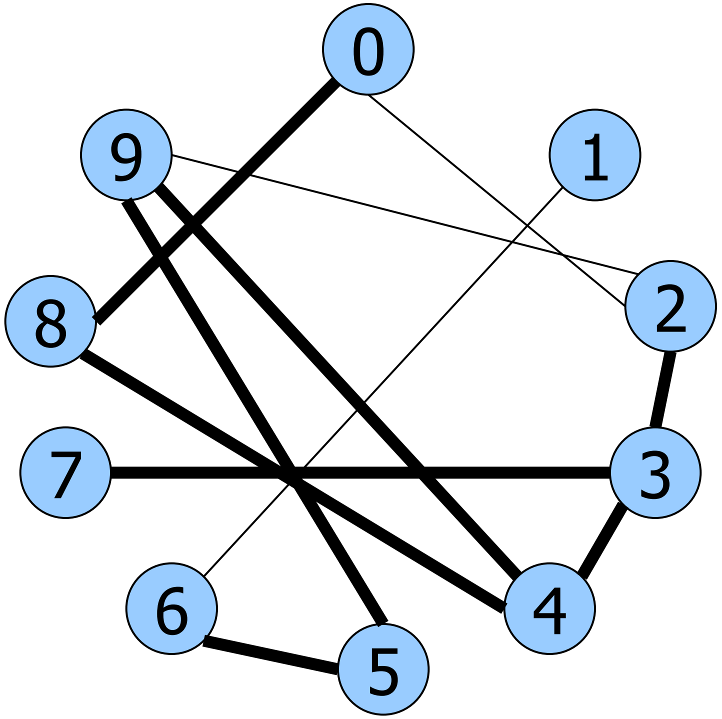
Output
5 9

Input
7 3



Output
7 3

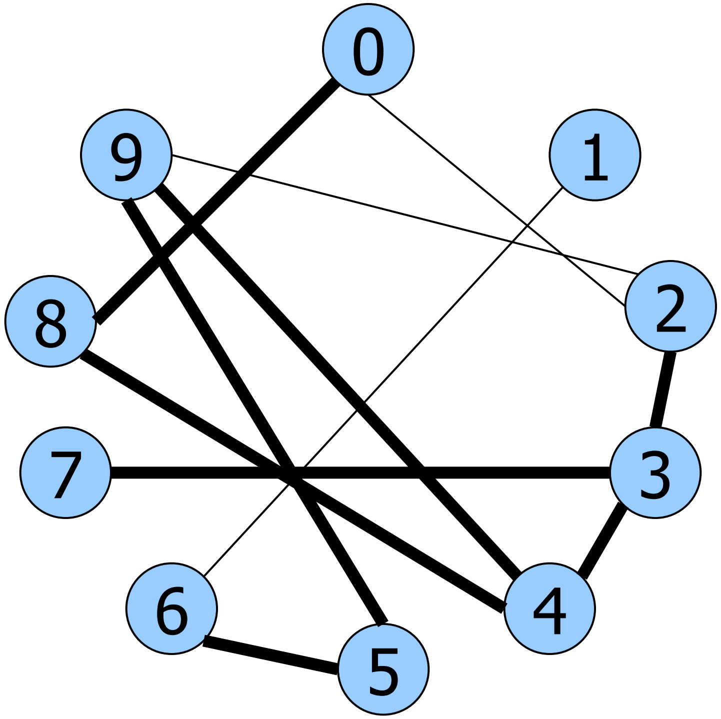
Input
4 8



Output
4 8

Input
5 6

Output

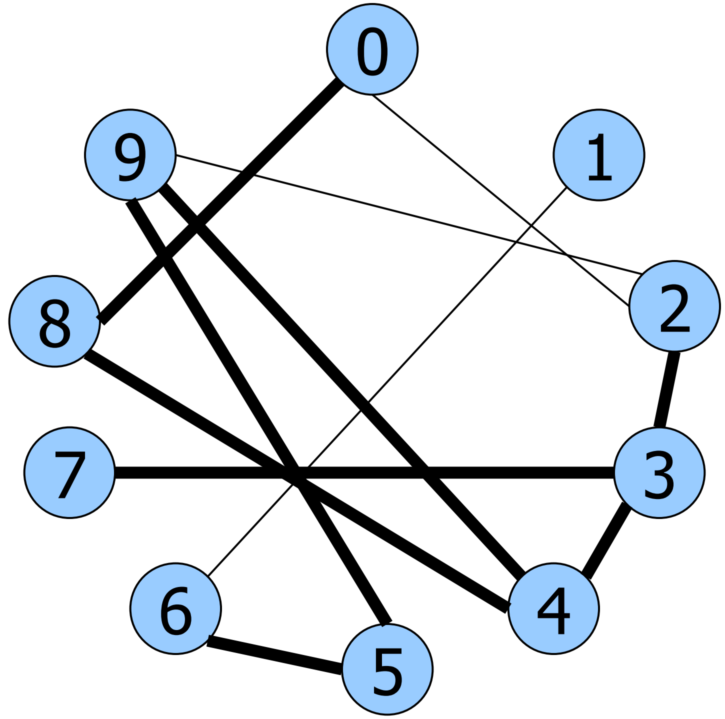


Esiste già il cammino 5-6

Input

0 2

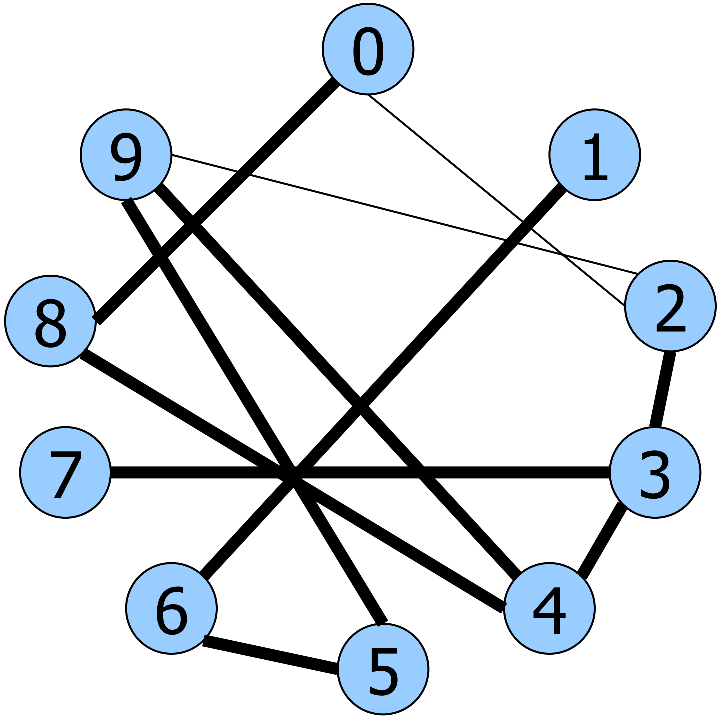
Output



Esiste già il cammino 0-8-4-3-2

Input
6 1

Output
6 1



Approccio

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.

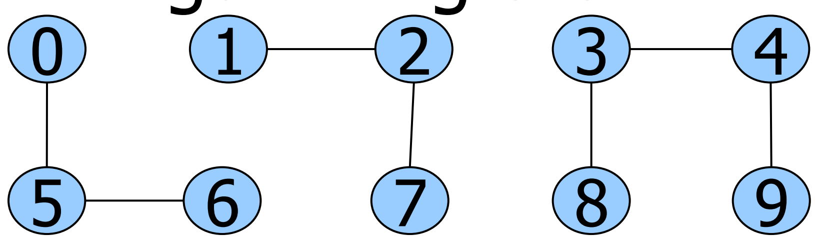
- Insiemi S_i delle coppie connesse, inizialmente tanti insiemi quanti i nodi, ogni nodo è connesso solo con se stesso
- Operazioni astratte:
  - find: trova l’insieme a cui appartiene un oggetto
  - union: unisci due insiemi

- 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 p in q

- find: semplice riferimento a una cella del vettore id[indice], costo unitario
- union: scansione del vettore per cambiare gli elementi che valgono p in q, costo lineare nella dimensione del vettore
- 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.

0 1 2 3 4

Esempio

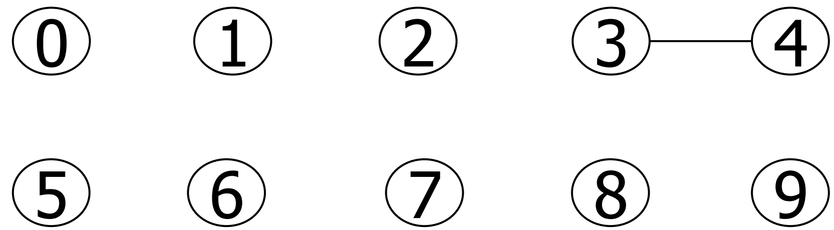
5 6 7 8 9

Inizialmente

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

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

0 1 2 3 4 5 6 7 8 9



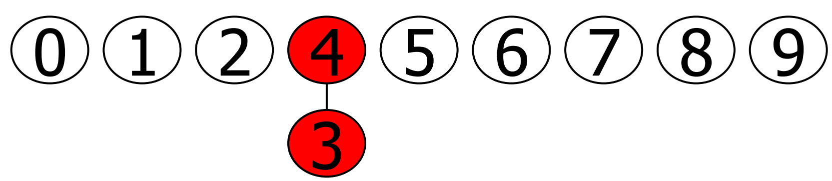
p q = 3 4

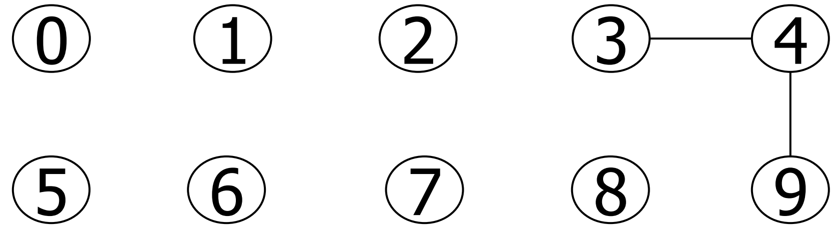
id									
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9

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

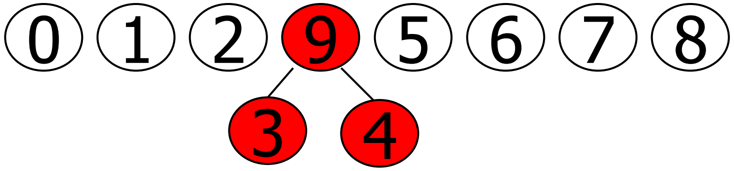
id	0	1	2	4	4	5	6	7	8	9
	0	1	2	3	4	5	6	7	8	9

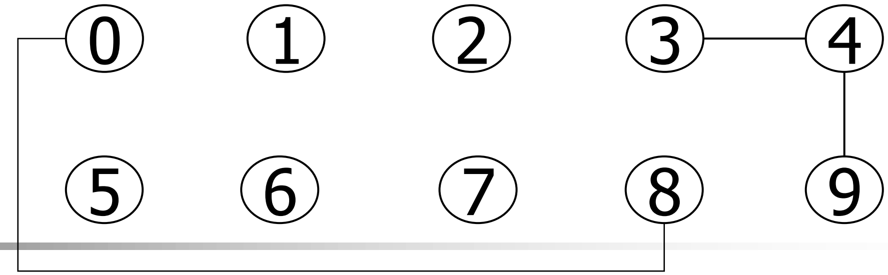
id[p]=4 $\neq$ id[q]=9

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

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





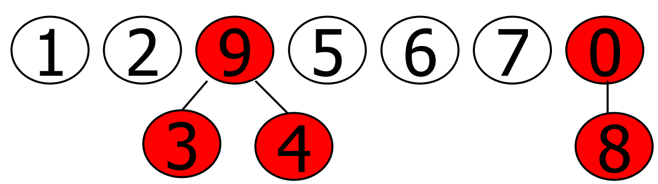
p q = 8 0

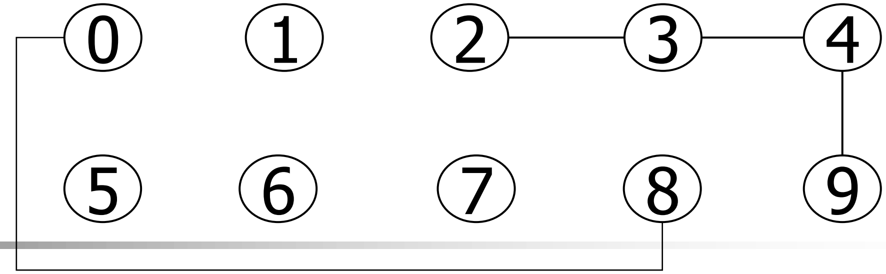
id	0	1	2	9	9	5	6	7	8	9
	0	1	2	3	4	5	6	7	8	9

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

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





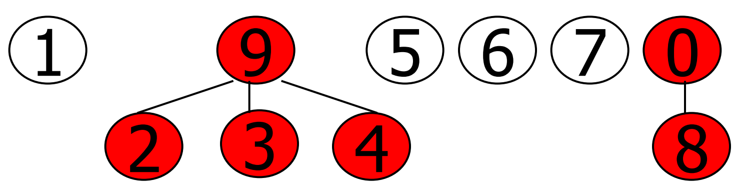
p q = 2 3

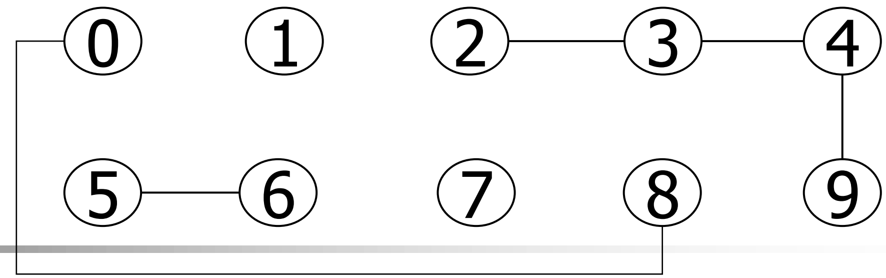
id									
0	1	2	9	9	5	6	7	0	9
0	1	2	3	4	5	6	7	8	9

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

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





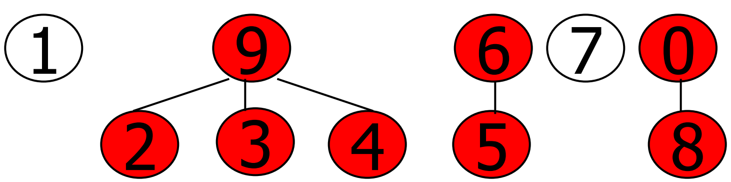
p q = 5 6

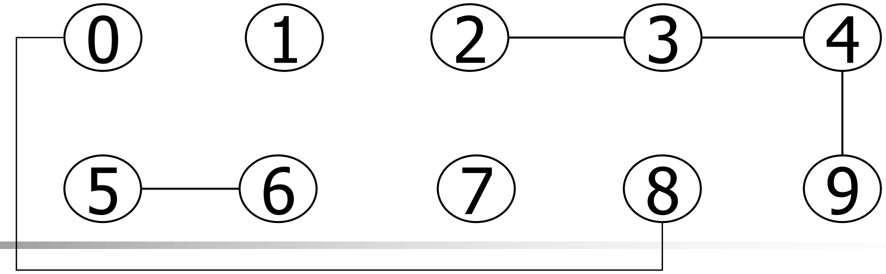
id										
id	0	1	9	9	9	5	6	7	0	9
	0	1	2	3	4	5	6	7	8	9

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

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





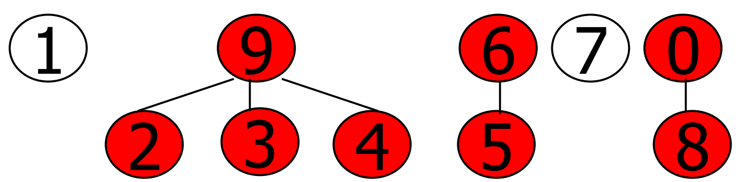
p q = 2 9

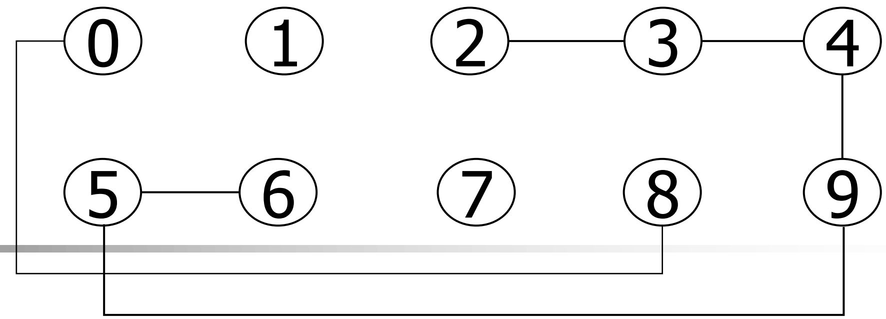
id	0	1	9	9	9	6	6	7	0	9
	0	1	2	3	4	5	6	7	8	9

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

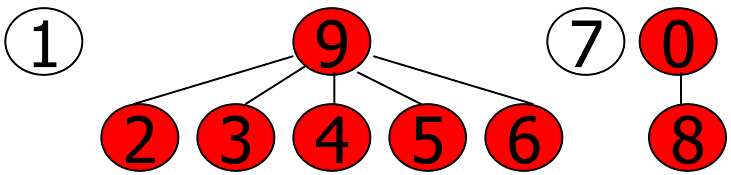
id										
	0	1	9	9	9	6	6	7	0	9
	0	1	2	3	4	5	6	7	8	9

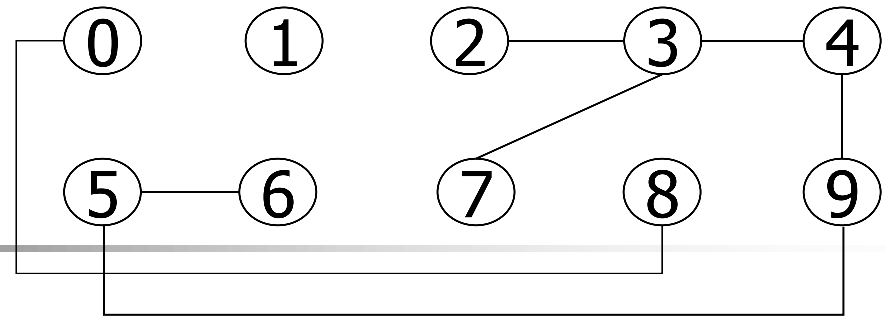
id[p]=6 $\neq$ id[q]=9

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

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





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

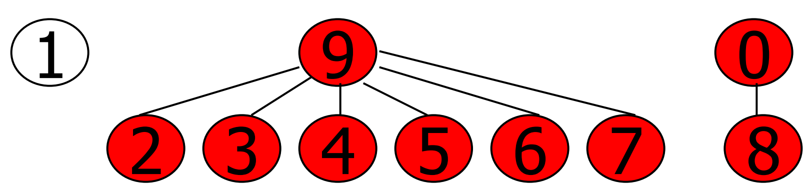
$id[p]=7 \neq id[q]=9$

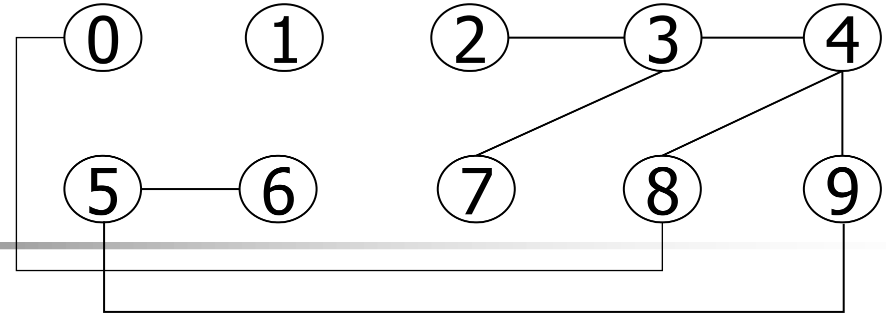
cambia tutti i valori id[p] in id[q]

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

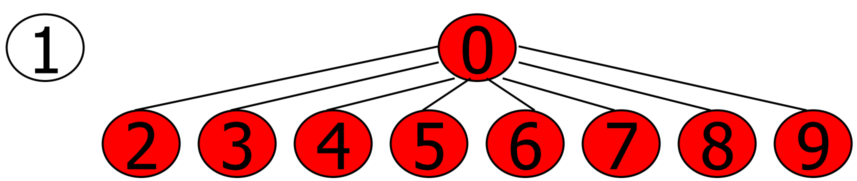
id[p]=9 $\neq$ id[q]=0

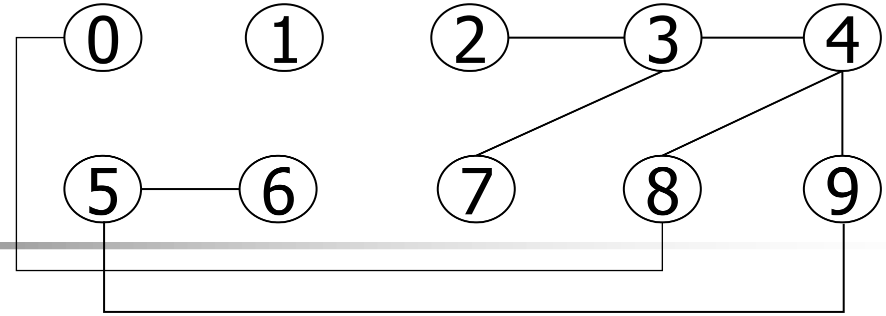
cambia tutti i valori id[p] in id[q]

id

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

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





p q = 5 6

id

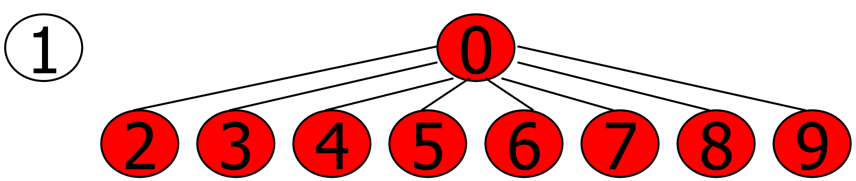
0	1	2	3	4	5	6	7	8	9
0	1	0	0	0	0	0	0	0	0

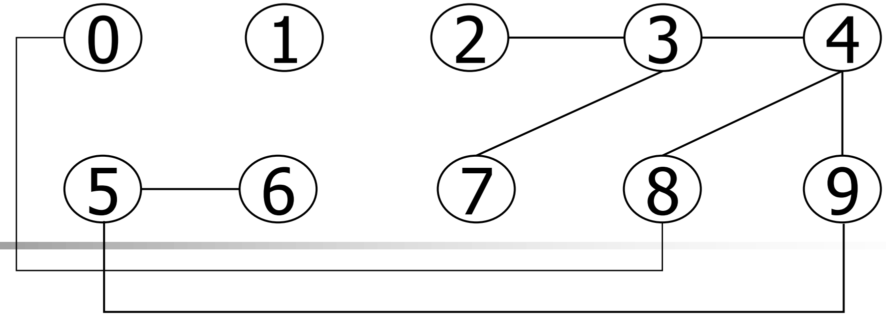
id[p]=0 = id[q]=0
non cambia nulla

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

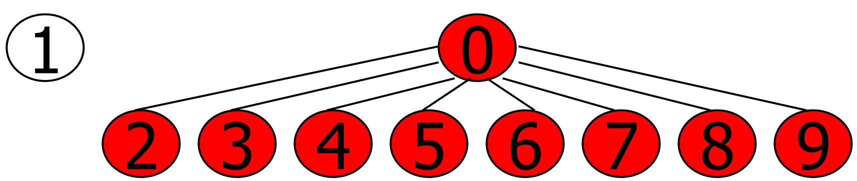
id	0	1	2	3	4	5	6	7	8	9
	0	1	0	0	0	0	0	0	0	0

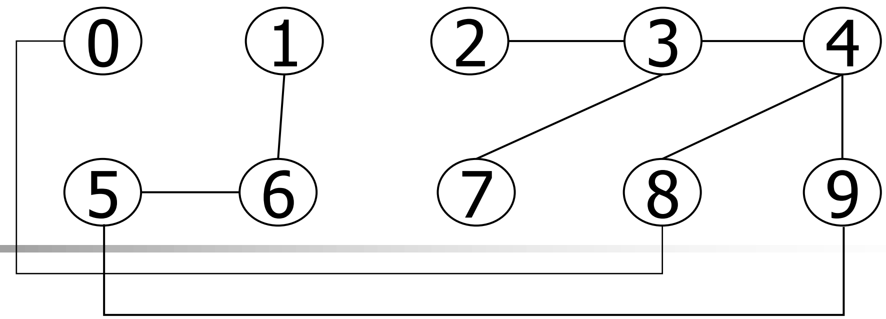
id[p]=0 = id[q]=0

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

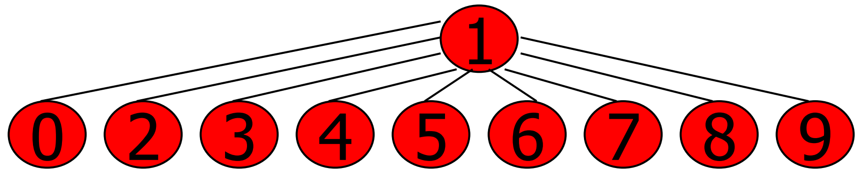
0	1	2	3	4	5	6	7	8	9
0	1	0	0	0	0	0	0	0	0

id[p]=0 = id[q]=1
cambia tutti i valori id[p] in id[q]

id

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

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



01quick-find.c

```c
#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]]]]

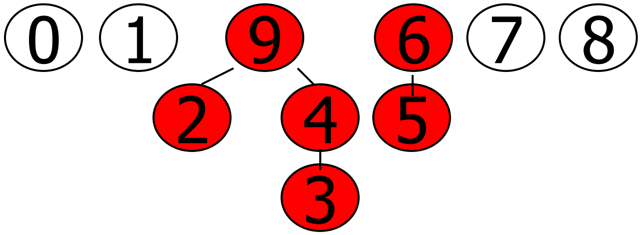
  se gli oggetti i e j sono connessi

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

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
- union: semplice in quanto basta far sì che un oggetto punti all’altro, costo unitario
- complessivamente il numero di operazioni è legato a

  num. coppie * lunghezza della “catena”

0 1 2 3 4

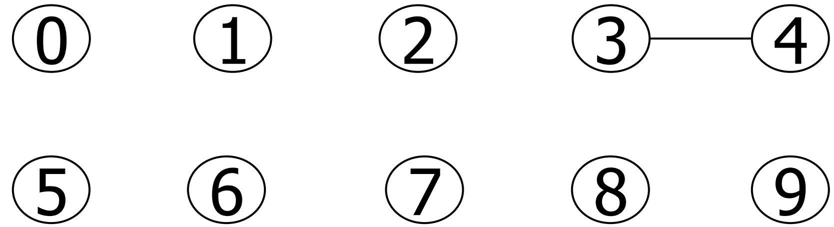
5 6 7 8 9

Esempio

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

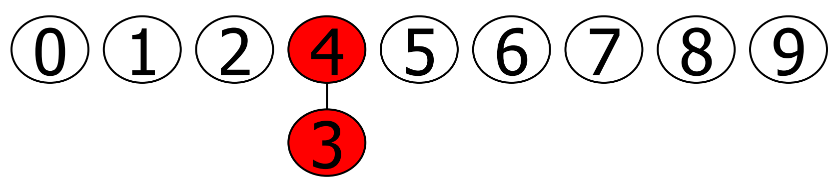
id

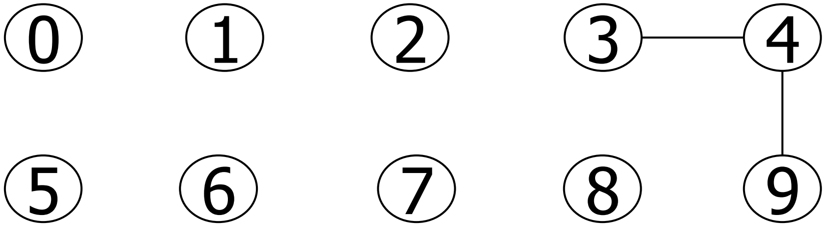
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9

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





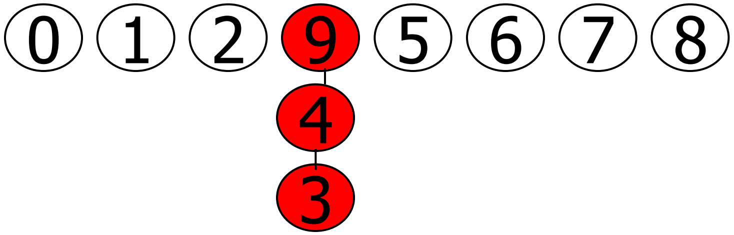
p q = 4 9

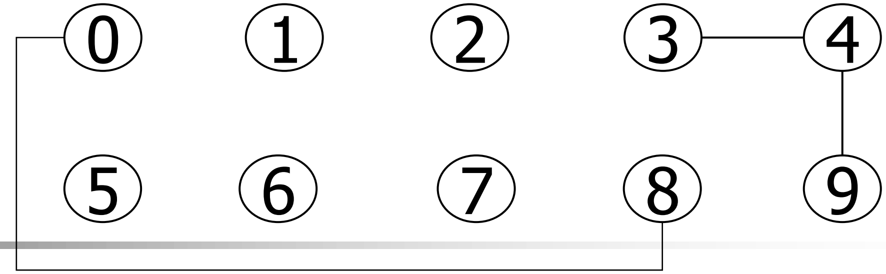
id	0	1	2	4	4	5	6	7	8	9
	0	1	2	3	4	5	6	7	8	9

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





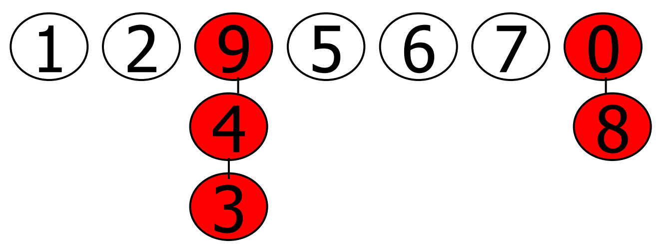
p q = 8 0

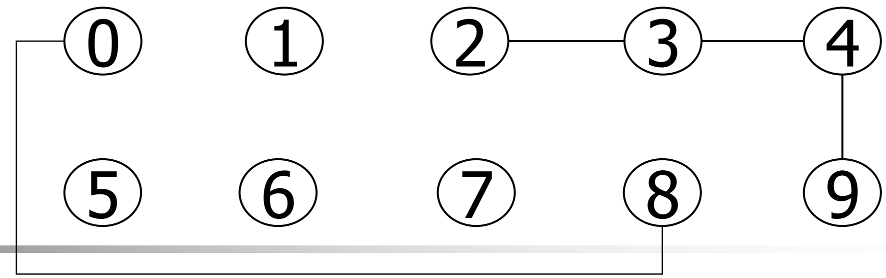
id	0	1	2	4	9	5	6	7	8	9
	0	1	2	3	4	5	6	7	8	9

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





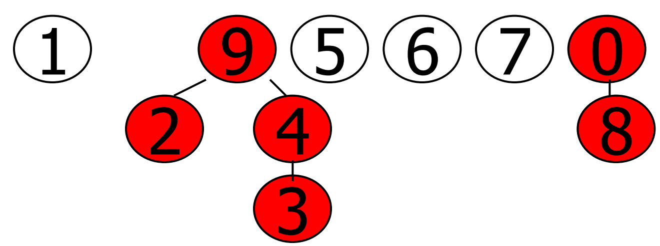
p q = 2 3

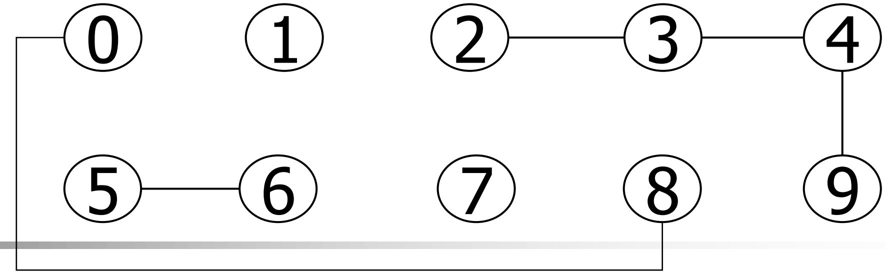
id	0	1	2	4	9	5	6	7	0	9
	0	1	2	3	4	5	6	7	8	9

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





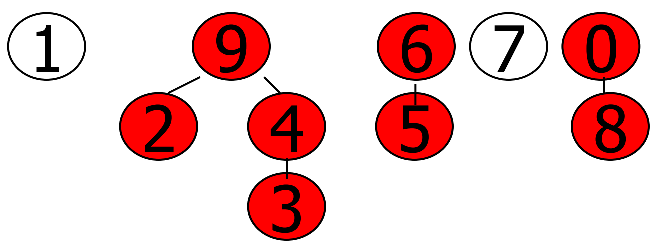
p q = 5 6

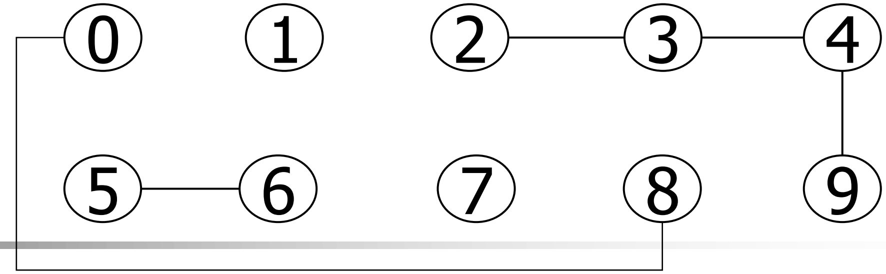
id	0	1	9	4	9	5	6	7	0	9
	0	1	2	3	4	5	6	7	8	9

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





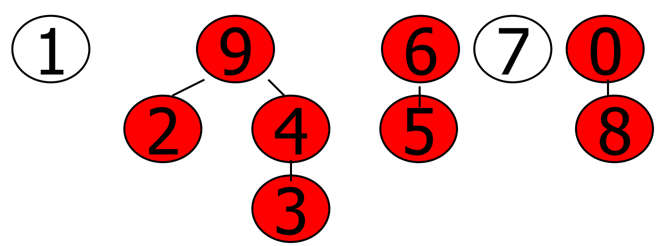
p q = 2 9

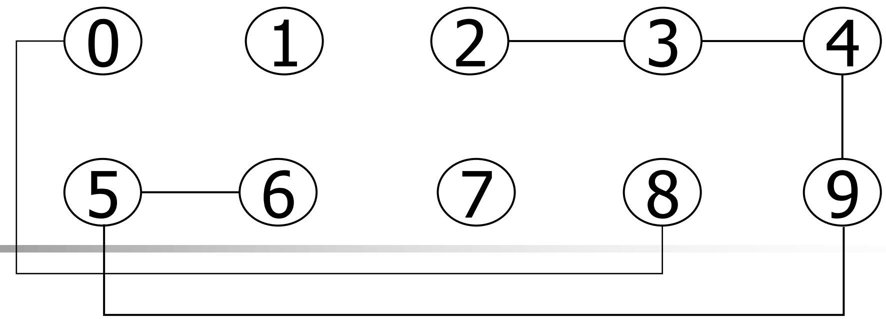
id	0	1	9	4	9	6	6	7	0	9
	0	1	2	3	4	5	6	7	8	9

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

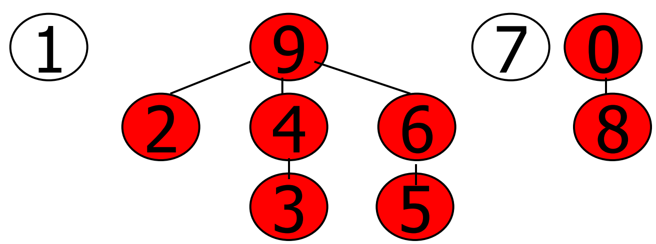
0	1	9	4	9	6	6	7	0	9
0	1	2	3	4	5	6	7	8	9

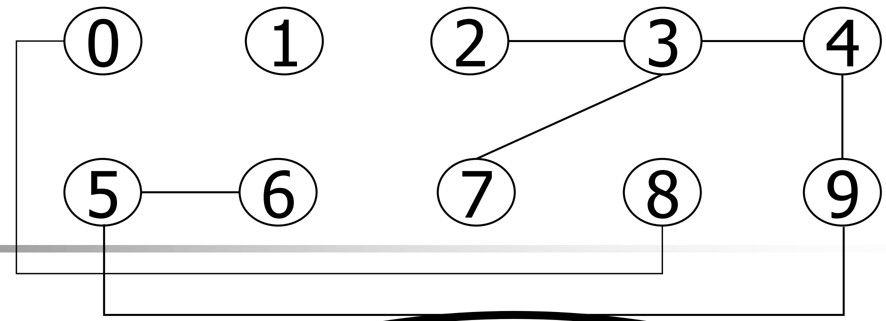
id[id[p]]=6 $\neq$ id[q]=9

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

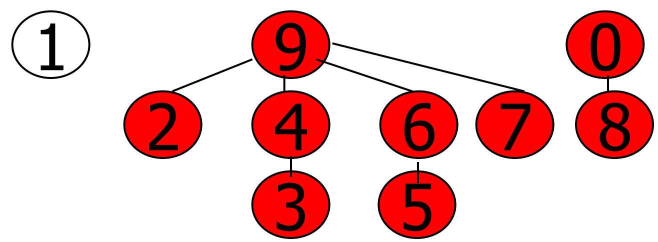
0	1	9	4	9	6	9	7	0	9
0	1	2	3	4	5	6	7	8	9

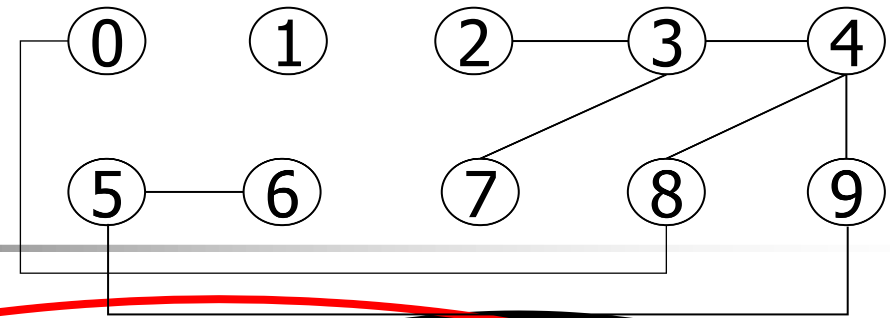
id[p]=7 $\neq$ id[id[id[q]]]=9

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





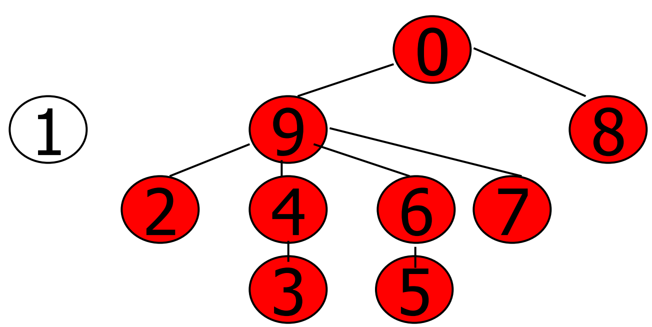
p q = 4 8

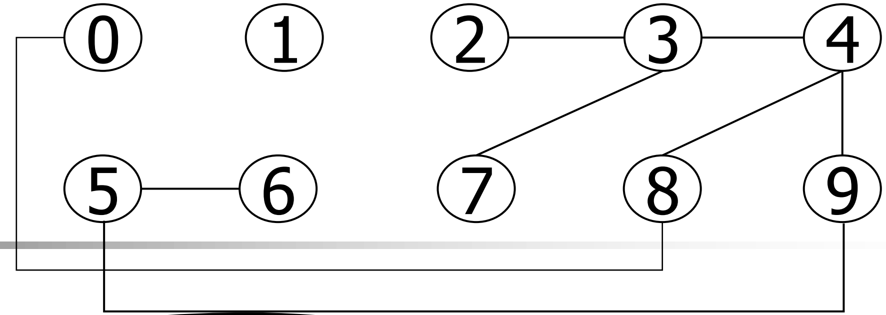
id	0	1	9	4	9	6	9	9	0	9
	0	1	2	3	4	5	6	7	8	9

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





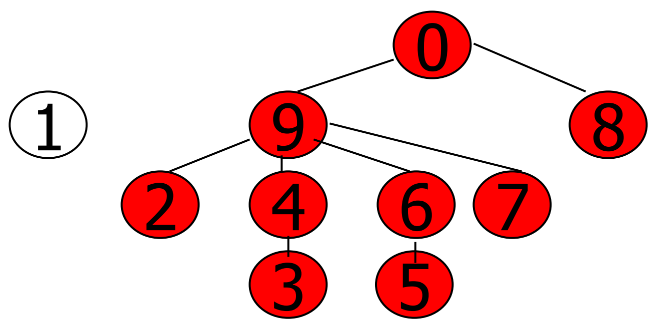
p q = 5 6

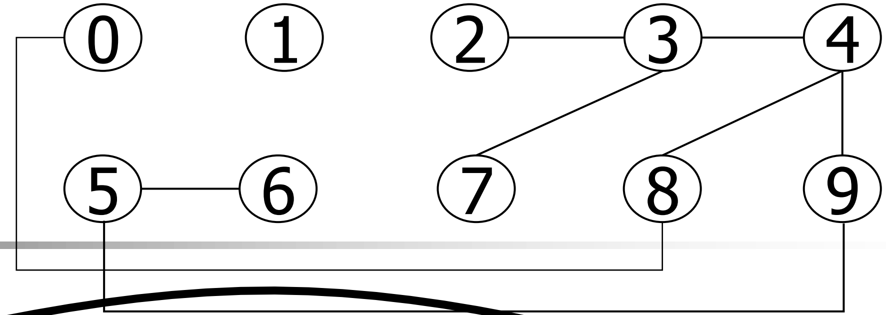
id	0	1	9	4	9	6	9	9	0	0
	0	1	2	3	4	5	6	7	8	9

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



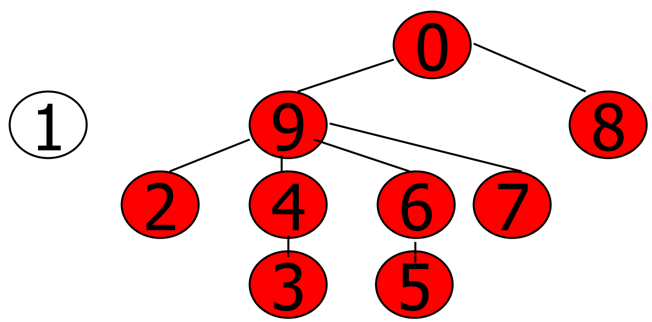


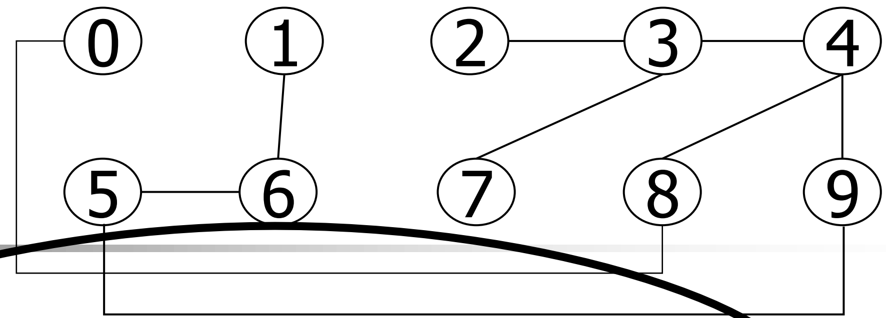
p q = 0 2

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





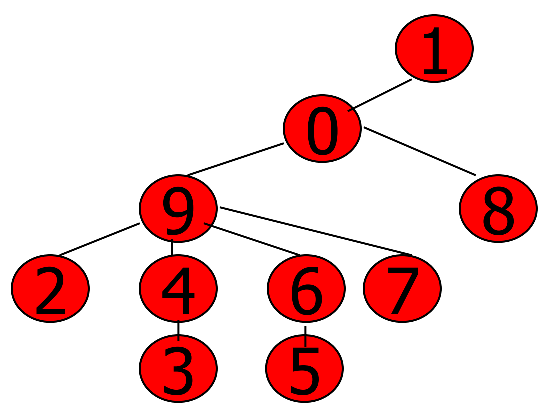
p q = 6 1

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



02quick-union.c

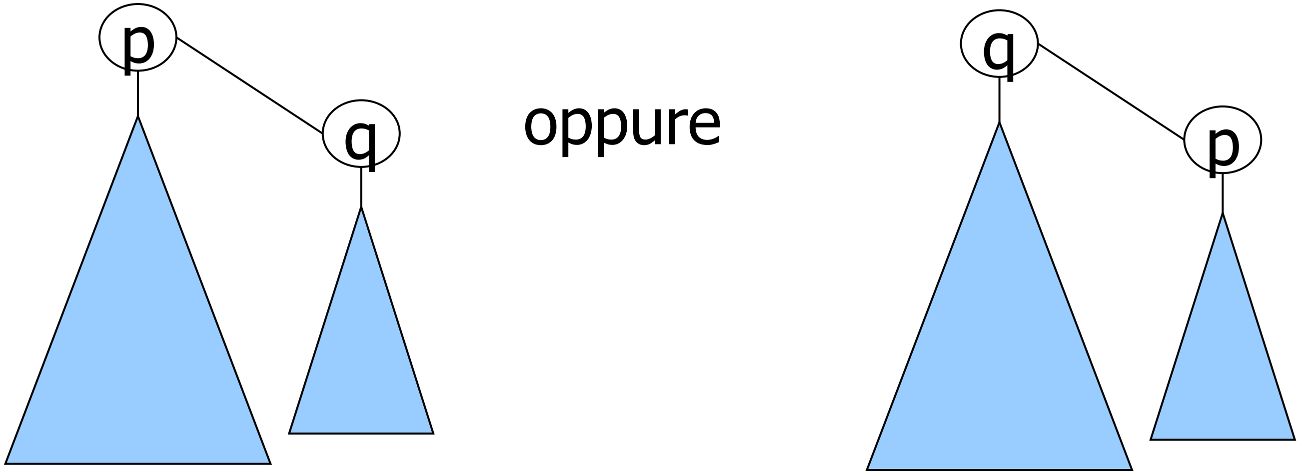
```c
#include <stdio.h>
#define N 10000
main() {
  int i, j, p, q, id[N];
  for(i=0; i<N; i++) id[i] = i;
  printf("Input pair p q:  ");
  while (scanf("%d %d", &p, &q) ==2) {
    for (i = p; i!= id[i]; i = id[i]);
    for (j = q; j!= id[j]; j = id[j]);
    if (i == j)
      printf("pair %d %d already 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

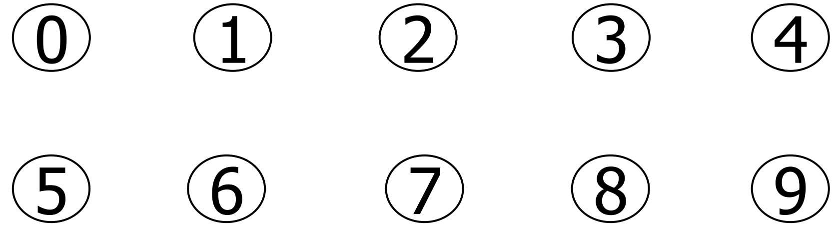
Weighted quick union:

- 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:



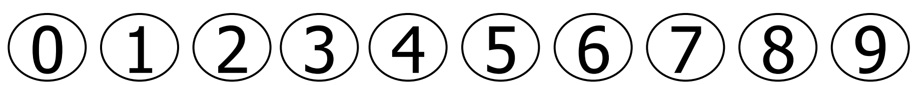
NB: è irrilevante se p appare alla destra o alla sinistra di q.

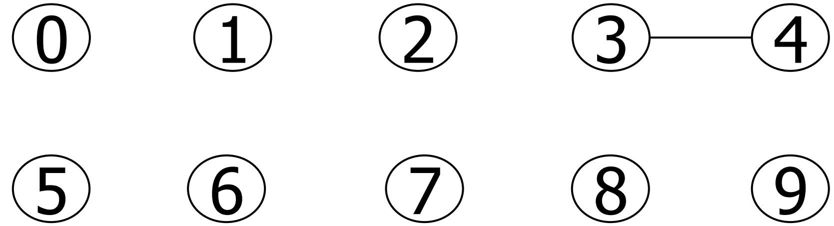


Esempio

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

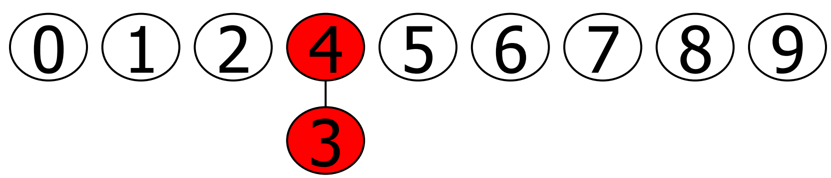
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9

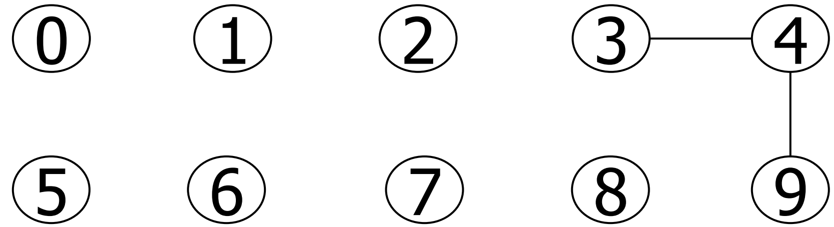
id[p]=3 $\neq$ id[q]=4

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



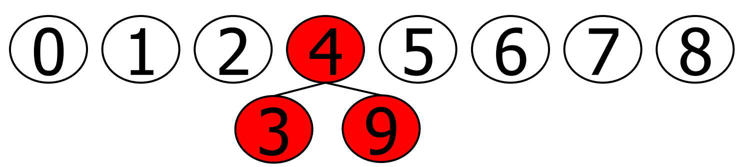


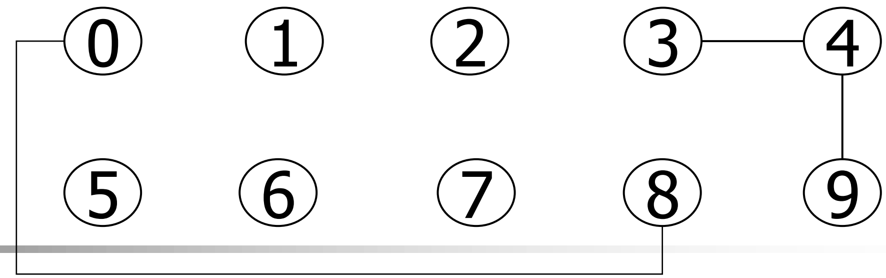
p q = 4 9

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





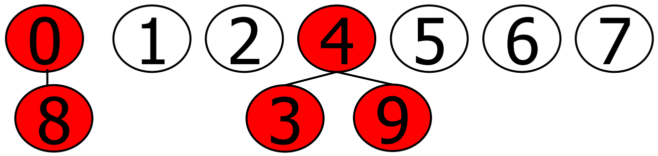
p q = 8 0

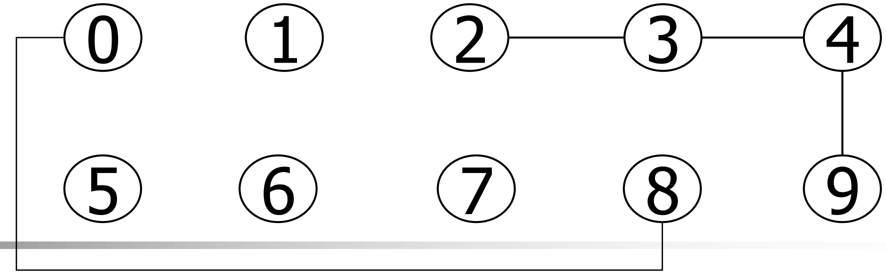
id	0	1	2	4	4	5	6	7	8	4
	0	1	2	3	4	5	6	7	8	9

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





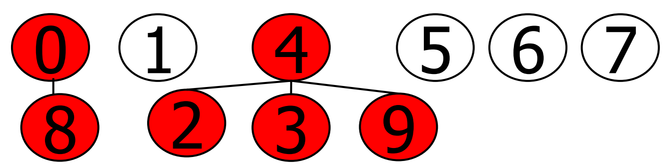
p q = 2 3

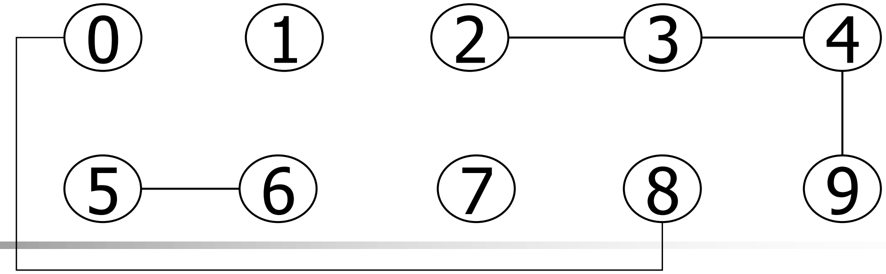
id	0	1	2	4	4	5	6	7	0	4
	0	1	2	3	4	5	6	7	8	9

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





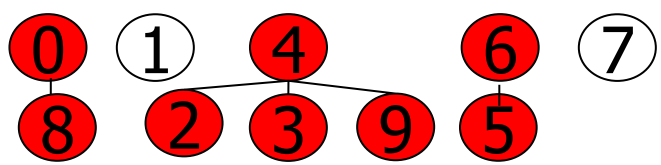
p q = 5 6

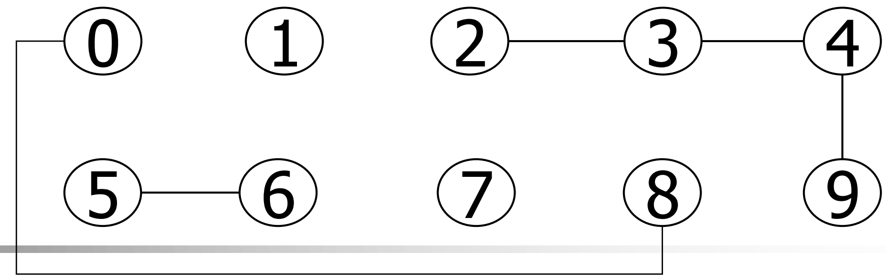
id	0	1	4	4	4	5	6	7	0	4
	0	1	2	3	4	5	6	7	8	9

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





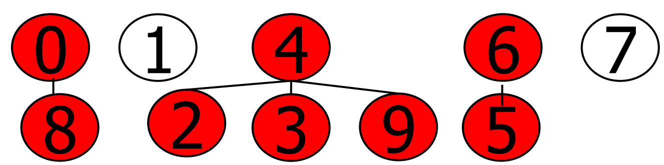
p q = 2 9

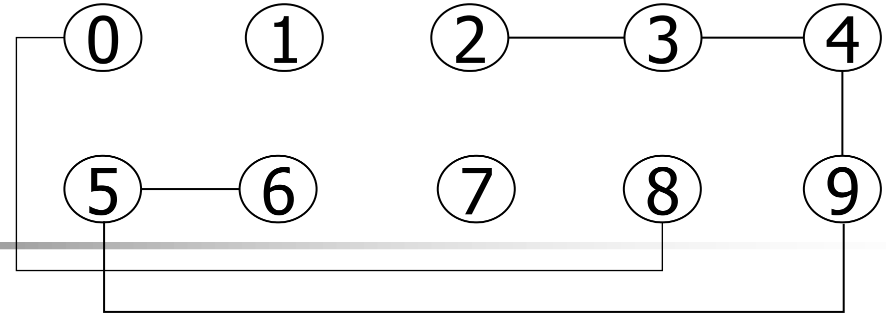
id	0	1	4	4	4	6	6	7	0	4
	0	1	2	3	4	5	6	7	8	9

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





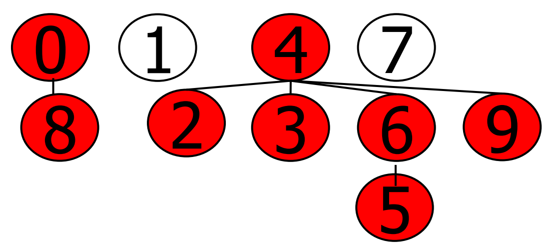
p q = 5 9

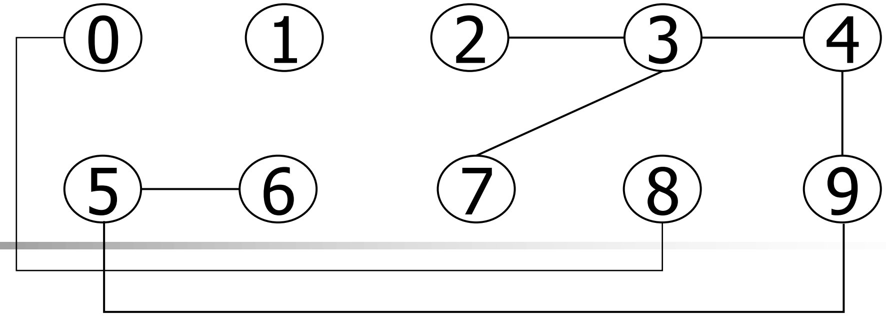
id	0	1	4	4	4	6	6	7	0	4
	0	1	2	3	4	5	6	7	8	9

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

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

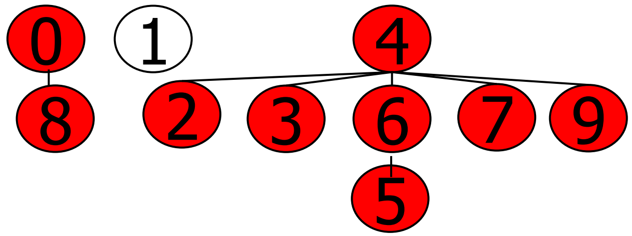
id

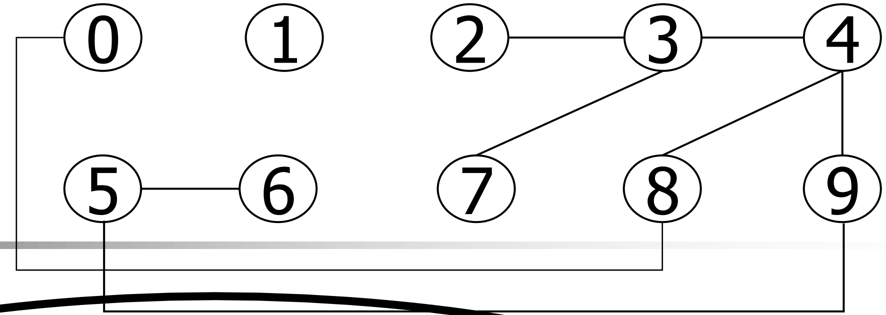
0	1	4	4	4	6	4	7	0	4
0	1	2	3	4	5	6	7	8	9

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



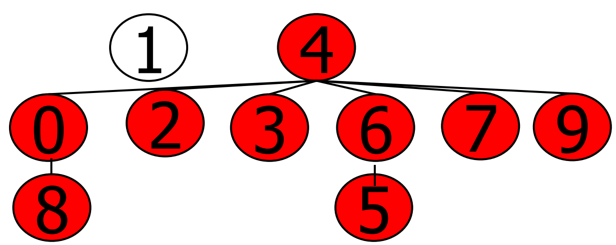


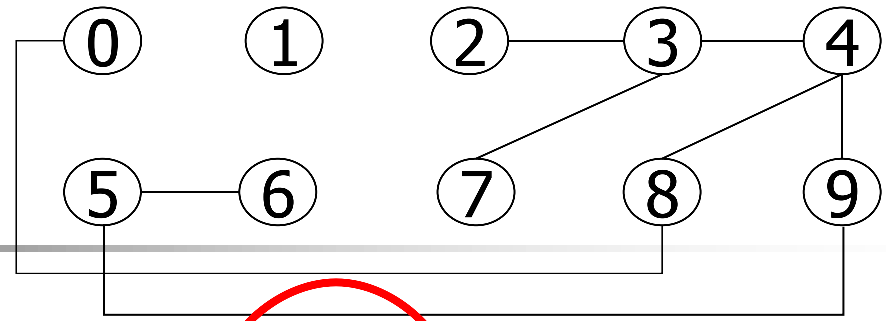
p q = 4 8

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

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

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



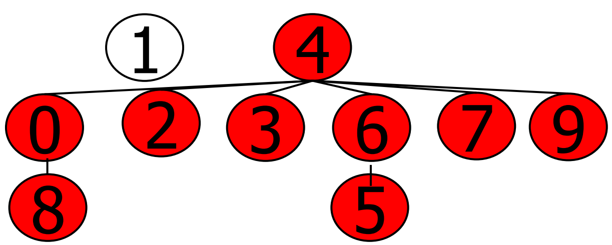


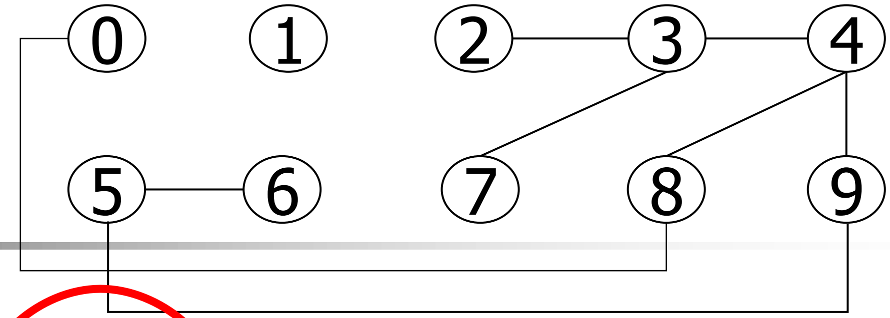
p q = 5 6

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



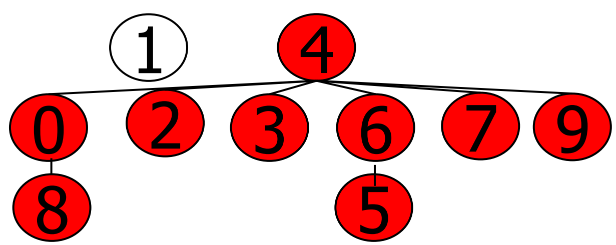


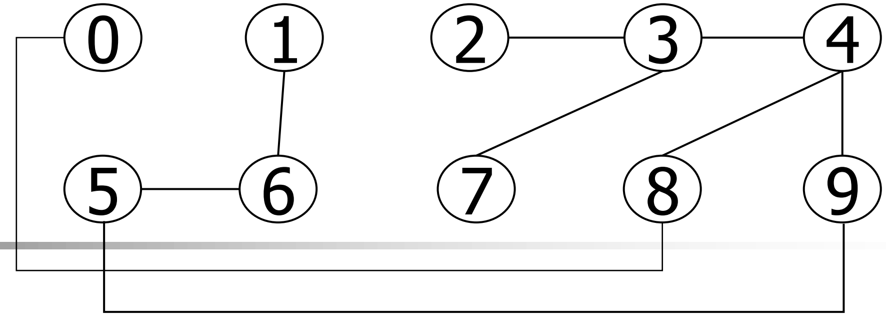
p q = 0 2

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



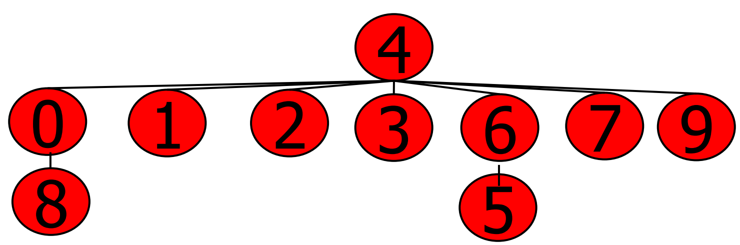


p q = 6 1

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

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



03 weighted-quick-union.c

```
...
  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:  ");
  }
...
```

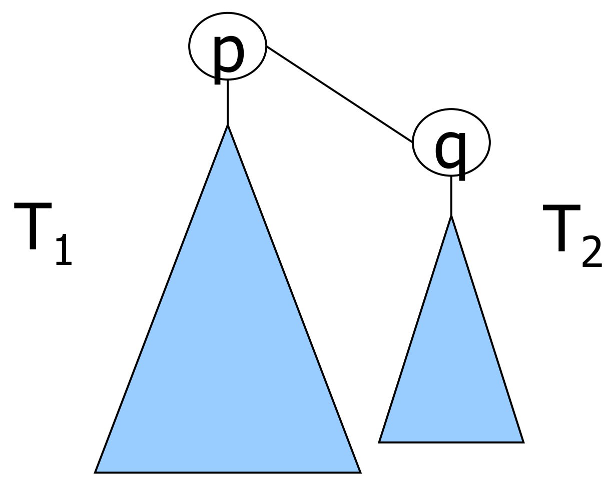
- find: percorrimento di una “catena” di oggetti, costo al massimo logaritmico nel numero di oggetti
- union: semplice in quanto basta far sì che un oggetto punti all’altro, costo unitario
- complessivamente il numero di operazioni è legato a

  num. coppie * lunghezza della “catena”

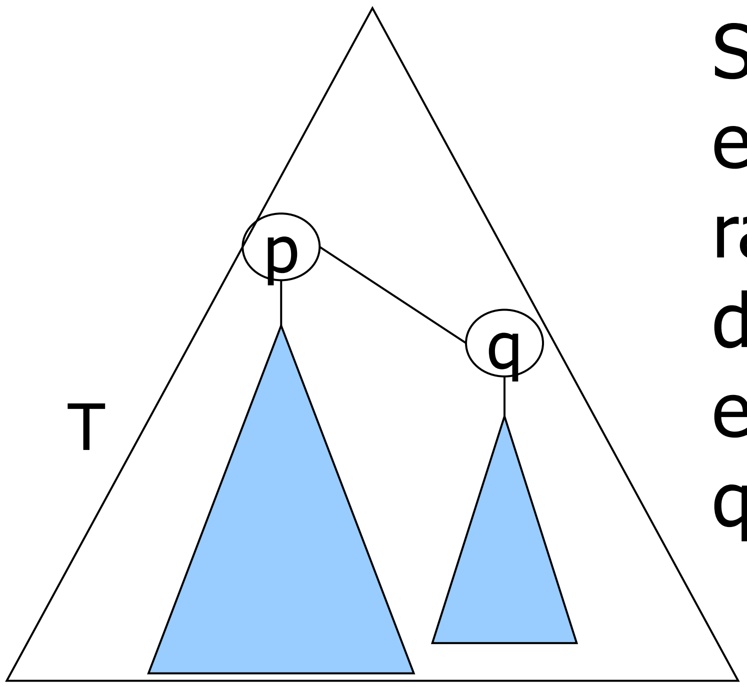
  ma quest’ultima cresce in modo logaritmico!

Perché logaritmico?

Conta distanza massima tra un nodo e la radice. La distanza cresce di 1 quando si collega un albero più piccolo (di dimensione T_2) a un albero più grande (di dimensione T_1).



Ma se $T_1 \geq T_2$ ad ogni connessione di albero piccolo ad albero 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. Deve valere $2^i \leq N$, quindi $i \leq \log_2 N$.

Riferimenti

- Analisi di complessità:
  - Cormen 1.2
  - Sedgewick 2.2
- Notazione asintotica:
  - Cormen 2.1
  - Sedgewick 2.3, 2.4
- Esempi di analisi:
  - Sedgewick 2.6
- Online connectivity e algoritmi Union-Find:
  - Sedgewick 1.2, 1.3