Analisi e progettazione di algoritmi

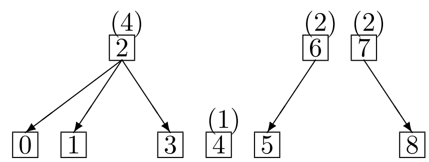
(III anno Laurea Triennale - a.a. 2018/19)

Soluzioni prova scritta 17 luglio 2019

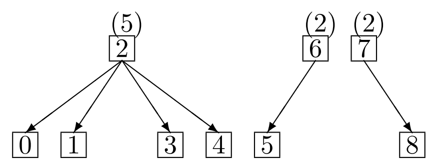
Esercizio – Union find

1. Con union-by-size.

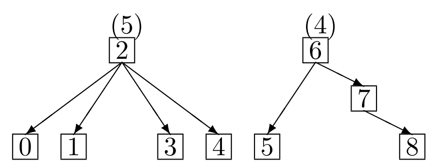
   Situazione iniziale (il size è indicato tra parentesi sopra ogni radice):



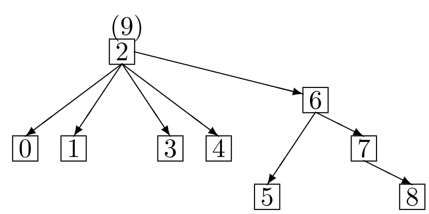
   union(2,4): si esegue *find*(2) che ritorna 2 con size= 4, *find*(4) che ritorna 4 con size= 1. Le due radici sono distinte e si procede all'unione. Sopravvive la radice di size maggiore cioè 2, che ha ora size= 5.



   union(5,7): si esegue *find*(5) che ritorna 6 con size= 2, *find*(7) che ritorna 8 con size= 2. Le due radici sono distinte e si procede all'unione. Avendo le due radici uguale size, per convenzione sopravvive la prima cioè 6, che va ad avere size= 4.



   union(8,1): si esegue *find*(8) che ritorna 6 con size= 4, *find*(1) che ritorna 2 con size= 5. Le due radici sono distinte e si procede all'unione. Sopravvive la radice con size maggiore cioè 6, che ha ora size= 9.

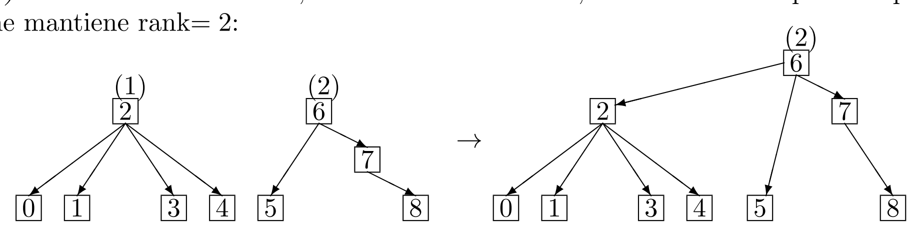


2. Con union-by-rank:

   union(2,4): la radice 2 ha rank= 1, la radice 4 ha rank= 0, sopravvive 2 come prima, 2 mantiene rank= 1.

   union(5,7): la radice 6 ha rank= 1, la radice 7 ha rank= 1, per la convenzione sopravvive 6, che ha ora rank= 2.

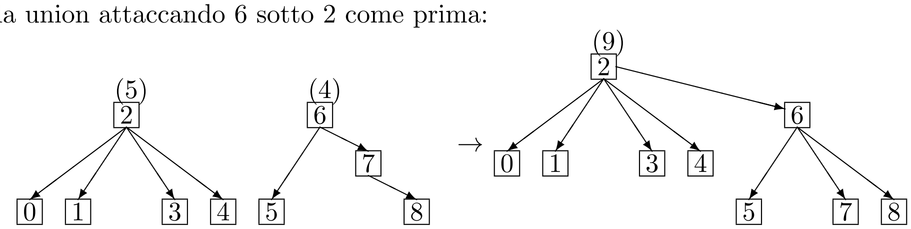
   union(8,1): la radice 6 ha rank= 2, la radice 2 ha rank= 1, diversamente da prima sopravvive 6, che mantiene rank= 2:



3. Con union-by-size e compressione dei cammini:

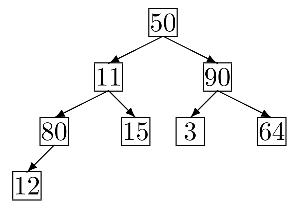
   Le uniche operazioni dove può cambiare qualcosa sono quelle in cui la *find* viene eseguita su un nodo che non è radice e non è figlio diretto della radice.

   Questo avviene solo nella *union*(8,1) eseguendo *find*(8): 8 figlio di 7 a sua volta figlio della radice 6. La *find* ha come effetto collaterale di sganciare 8 da 7 e attaccarlo a 6. Dopo viene eseguita la union attaccando 6 sotto 2 come prima:



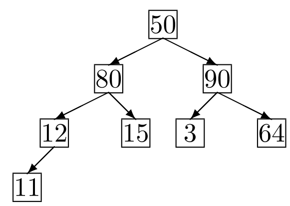
Esercizio 2 – Sorting Pensiamo l'array come se fosse un albero.

Le caselle dell'array che sono foglie (15,3,64,12) sono già heap fatti da solo un nodo. Devo esaminare gli altri nodi a ritroso (nell'ordine 80,90,11,50) e rendere heap l'albero di cui sono radice, eseguendo delle *moveDown*.

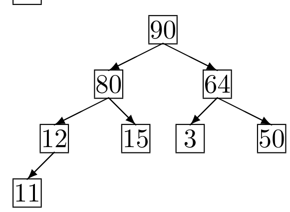


Eseguo *moveDown*(80): $80 > 12$ quindi è già heap.
Eseguo *moveDown*(90): anche in questo caso nessuno scambio.

Eseguo *moveDown*(11): scambio 11 con il figlio di chiave massima (80) e poi di nuovo con il figlio 12.



Eseguo *moveDown*(50): scambio 50 con 90 e poi con 64.



Versione con insert (vale un punto in meno). Nota: mancano i disegni degli alberi.
La prima casella dell'array (contenente 50) è già heap. In questo heap inserisco il numero che sta nella casella seguente (11) e per rendere heap le prime due caselle eseguo *moveUp*(11): siccome $50 > 11$ nessuno scambio (è già heap a massimo).
Adesso sono heap le prime due caselle. Inserisco il numero che sta nella casella seguente (90) eseguendo *moveUp*(90): siccome $50 < 90$, scambio 50 e 90.
Adesso inserisco 80 eseguendo *moveUp*(80): siccome $80 > 11$ scambio, poi $80 < 90$ e ho finito.
Inserisco 15 eseguendo *moveUp*(15) e poi 3 eseguendo *moveUp*(3): nessuno scambio.
Inserisco 64: *moveUp*(64) scambia 64 con 50. Inserisco 12: *moveUp*(12) scambia 12 e 11.
Risultato finale:

90	80	64	12	15	3	50	11

Esercizio 3 - Design e analisi di algoritmi

1. L'algoritmo ricorsivo per calcolarli ha relazione di ricorrenza:

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

   Quindi, dato che ovviamente $T(n+1) > T(n)$ per $n > 2$,

   $T(n) > 3\,T(n-3) + 1$

   Risolviamo:

   $T(n) > 3(3\,T(n-6)+1)+1 = 3^2\,T(n-6)+3^1+3^0 > \ldots > 3^i\,T(n-3i)+3^{i-1}+ \ldots + 3^1 + 3^0$
   (ultimo termine per $i = k$ se $n = 3k$)
   $> 3^k + \ldots + 3^0 = \Omega(3^{\frac{n}{3}})$.

   Si ha quindi un algoritmo esponenziale.

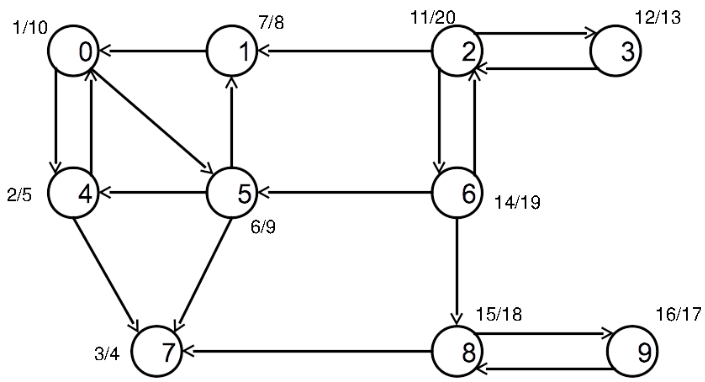
2. Supponiamo di avere la struttura dati "triple" (oppure si può usare un array `trib[0..2]`).

```
trib(n)
  if (n = 0) return 0
  if (n = 1 || n = 2) return 1
  (terzultimo, penultimo, ultimo) = (0,1,1)
    for (i=3; i ≤ n; i++)
      (terzultimo, penultimo, ultimo) =
        (penultimo, ultimo, terzultimo + penultimo + ultimo)
  return ultimo
```

3.
```
trib(0) = (0,1,1)
trib(n) =
  let (terzultimo,penultimo,ultimo) = trib(n-1)
  in return (penultimo,ultimo,terzultimo+penultimo+ultimo)
```

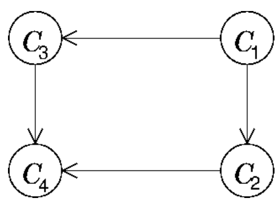
Esercizio 4 - Grafi

1. La visita in profondità assegna i seguenti tempi di inizio e fine visita:



2. La sequenza delle componenti fortemente connesse ottenuta è la seguente:
$\mathcal{C}_1 = \{2,3,6\}, \mathcal{C}_2 = \{8,9\}, \mathcal{C}_3 = \{0,1,5,4\}, \mathcal{C}_4 = \{7\}$.

3. Il grafo quoziente è il seguente:



Esercizio 5 - NP-completezza Se esistesse un algoritmo polinomiale A che decide $\mathcal{P} \setminus \{01, 10\}$, potremmo costruire un algoritmo polinomiale per $\mathcal{P}$ nel modo seguente:

```
input u
if (u = 01) return true
return algo(u)
```