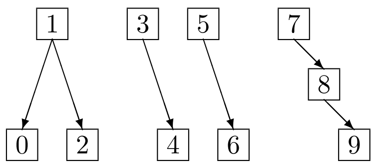
Complementi di Algoritmi e Strutture Dati

(III anno Laurea Triennale - a.a. 2017/18)

Soluzioni della Prova scritta 18 luglio 2018

Esercizio 1 – Union find Considerare la foresta union-find sottostante e un'implementazione di tipo *quick-union* che usa *union-by-size*.

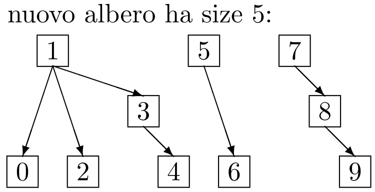


1. Indicare, per ogni radice, il "size" del corrispondente albero.

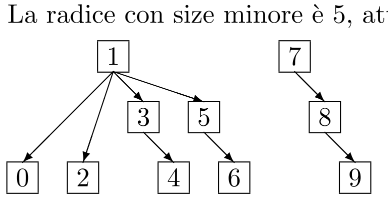
   radice 1: 3
   radice 3: 2
   radice 5: 2
   radice 7: 3

2. Eseguire nell'ordine le seguenti operazioni (ogni operazione va eseguita sul risultato della precedente):
   union(1,3)
   union(3,6)
   union(4,9)
   Per ogni operazione, spiegare che cosa viene fatto e perché, disegnare il nuovo albero e indicare il "size" della sua radice.

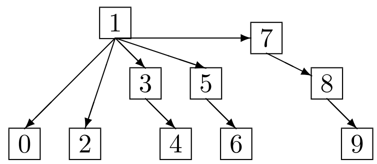
   union(1,3): i due nodi sono radici, quella di size inferiore è 3, attacco 3 come figlio di 1, il nuovo albero ha size 5:

   union(3,6): i due nodi non sono radici, per trovare le rispettive radici risalgo $3 \to 1$ e $6 \to 5$. La radice con size minore è 5, attacco 5 come figlio di 1, il nuovo albero ha size 7:

   union(4,9): i due nodi non sono radici, per trovare le rispettive radici risalgo $4 \to 3 \to 1$ e $9 \to 8 \to 7$. La radice con size minore è 7, attacco 7 come figlio di 1, Il nuovo albero ha size 10:

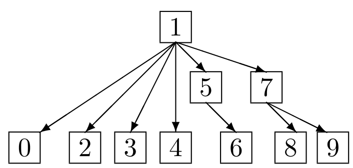


3. Considerare l'esecuzione delle operazioni di cui al punto precedente se l'implementazione adotta la *compressione dei cammini*. Se cambia qualcosa, dire che cosa e perché; se non cambia niente, dire perché.

   union(1,3): non cambia niente perché i due nodi in argomento sono radici.

   union(3,6): non cambia niente perché i due nodi in argomento sono già figli delle radici dei rispettivi alberi.

   union(4,9): nel primo dei due alberi da unire risalendo $4 \to 3 \to 1$ attacco 4 come figlio di 1 (3 lo è già). Nel secondo albero risalendo $9 \to 8 \to 7$ attacco 9 come figlio di 7. Poi unisco i due alberi attaccando 7 come figlio di 1, come prima.

Esercizio 2 – Sorting e alberi AVL Lo schema astratto di heapsort per ordinare n elementi consiste nell'inserire prima tutti i numeri in uno heap, poi tirarli fuori uno alla volta dallo heap, sfruttando il fatto che lo heap li ritornerà ordinati (in modo crescente o decrescente a seconda che sia uno heap a minimo o a massimo).
Possiamo pensare di applicare la stessa idea usando un albero AVL invece che uno heap.

1. Scrivere lo pseudocodice di questo algoritmo "AVL-sort", assumendo la sequenza s implementata ad array. Nota: ovviamente, al contrario dello heap, l'albero AVL non potrà essere costruito "sul posto".

   Occorre indicare le operazioni dell'AVL che vengono usate (dire come si chiamano e che cosa fanno, non dare l'algoritmo).

   ```
   T = nuovo albero AVL vuoto;
   for (i=0...n-1)
     T.insert(s[i]);
   i = 0;
   while (T non vuoto)
     x = T.min();
     T.remove(x);
     a[i++] = x;
   ```

   dove abbiamo usato le operazioni AVL:
   insert: inserisce un elemento, se necessario ribilancia l'albero eseguendo una rotazione;
   min: cerca l'elemento minimo, che si trova nel nodo più a sinistra, e lo ritorna;
   remove: cancella l'elemento, se necessario ribilancia l'albero eseguendo rotazioni.

2. Quale sarebbe la complessità temporale nel caso peggiore di un tale algoritmo "AVL-sort"? Giustificare la risposta.

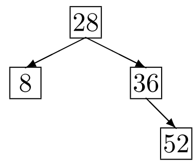
   Le operazioni sullo heap hanno un costo logaritmico nel numero di elementi che lo heap contiene.

   AVL-sort esegue n inserimenti, n accessi al minimo, n cancellazioni, e ciascuna ha costo logaritmico. Quindi la complessità temporale nel caso peggiore è in $O(n \log n)$.

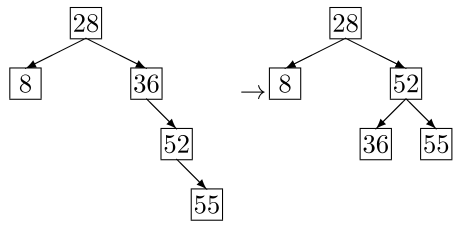
3. Mostrare i passi dell'algoritmo per ordinare la sequenza $s = 28, 8, 36, 52, 55, 1, 30, 32, 15, 90$.

Passi di inserimento nell'albero AVL:

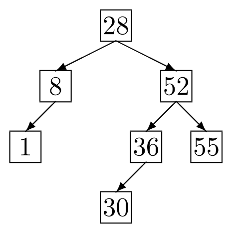
I primi quattro elementi 28,8,36,52 vengono inseriti come in un BST, senza rotazioni:



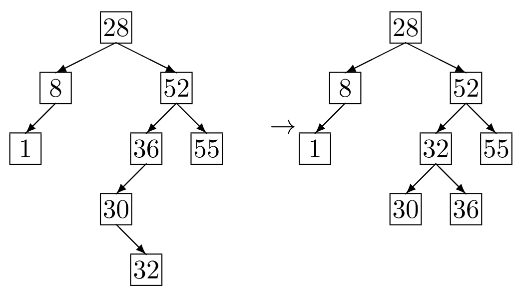
L'inserimento di 55 sbilancia il nodo contenente 36. Per ribilanciare facciamo una rotazione semplice verso sinistra (sono coinvolti i nodi 36,52):



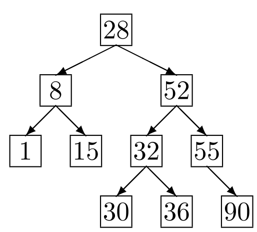
L'inserimento di 1 e poi di 30 non porta a rotazioni:



Inserendo 32, il nodo 36 diventa sbilanciato. Facciamo una rotazione doppia verso sinistra, che coinvolge i nodi 32,30,36:

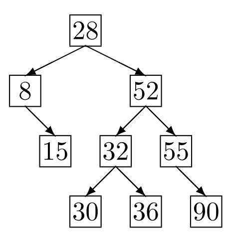


L'inserimento di 15 e 90 non causa rotazioni:

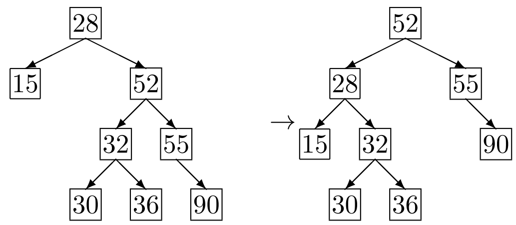


Passi di estrazione del minimo:

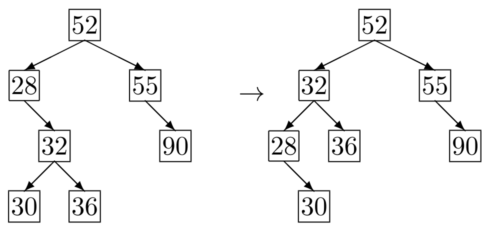
Trovo l'elemento di chiave minima (1) partendo dalla radice e seguendo il puntatore al figlio sinistro finché esiste. Cancello 1, che è foglia, come in un BST, non occorrono rotazioni.



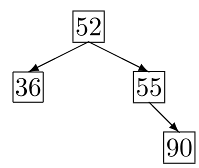
Trovo ancora l'elemento di chiave minima (8). Cancello 8, che ora ha un solo figlio, come in un BST: riattacco suo figlio 15 a suo padre 28. Si sbilancia il nodo 28. Rotazione semplice verso sinistra, sono coinvolti i nodi 28,52:



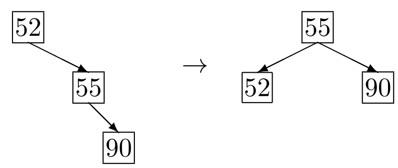
Trovo ora l'elemento minimo 15. Cancellando 15, si sbilancia di nuovo il nodo 28. Facciamo una rotazione semplice verso sinistra (coinvolge i nodi 28,32):



Trovo il minimo 28 e lo cancello (riattaccando 30 come figlio di 20). Trovo il minimo 30 e lo cancello (foglia), poi 32 e lo cancello (riattaccando 36 come figlio di 52), tutto senza rotazioni. Albero fin qui:



Adesso il minimo è 36. La cancellazione di 36 sbilancia il nodo 52. Facciamo rotazione semplice verso sinistra:



Infine, cancello senza necessità di rotazioni 52 (foglia), 55 (il figlio 90 diventa radice), 90 (ultimo nodo, l'albero ora è vuoto).

Esercizio 3 - Relazioni di ricorrenza Per ognuna delle seguenti relazioni di ricorrenza, si dia la soluzione utilizzando il metodo delle sostituzioni successive e si provi la correttezza della soluzione per induzione.

1. $T(n) = \begin{cases} 3\,T(n-1) & \text{se } n > 0 \\ 1 & \text{altrimenti} \end{cases}$

   $T(n) = 3\,T(n-1) =$
   $3(3\,T(n-2)) = 3^2\,T(n-2) = \dots$
   $3^i\,T(n-i) = \dots$
   $3^n \qquad (\text{per } n = i \text{ si ha } 1)$

   Prova per induzione aritmetica:
   $T(0) = 3^0 = 1$ ok
   $T(n) = 3\,T(n-1) =$ (per ip. ind.) $3 \cdot 3^{n-1} = 3^n$ ok

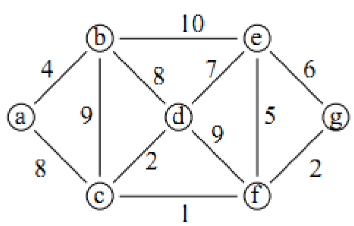
2. $T(n) = \begin{cases} 2\,T(n-1) - 1 & \text{se } n > 0 \\ 1 & \text{altrimenti} \end{cases}$

   $T(n) = 2\,T(n-1) - 1 =$
   $2(2\,T(n-2) - 1) - 1 = 2^2\,T(n-2) - 2^1 - 2^0 = \dots$
   $2^i\,T(n-i) - 2^{i-1} - \dots - 2^0 = \dots \qquad (\text{per } n = i \text{ si ha } 1)$

$2^n - \sum_{i=0}^{n-1} 2^i = 2^n - (2^n - 1) = 1$

Prova per induzione aritmetica:
$T(0) = 1$ ok
$T(n) = 2T(n-1) - 1 =$ (per ip. ind.) $2 - 1 = 1$ ok

Esercizio 4 - Grafi Si esegua, sul seguente grafo:



l'algoritmo di Prim a partire dal nodo a. Inizialmente quindi si avrà `dist`(a)=0 e `dist`=∞ per tutti gli altri nodi. Per ogni iterazione del ciclo while si dia:

- il nodo che viene estratto con la `getMin`
- i nodi per i quali viene modificata `dist` e come
- il minimo albero ricoprente alla fine dell'iterazione, evidenziando chiaramente la parte di albero definitiva.

Non dovete disegnare lo heap.
La seguente tabella mostra per ogni iterazione: nella prima colonna il nodo che viene estratto; nelle successive i nodi per i quali viene modificata `dist` e come; nell'ultima gli archi dell'albero ricoprente (in grassetto quelli definitivi).

estratto	a	b	c	d	e	f	g	*albero*
	0	∞	∞	∞	∞	∞	∞	
a		4	8					$(a,b), (a,c)$
b				8	10			**(a,b)**, $(a,c), (b,d), (b,e)$
c				2		1		**(a,b)**, **(a,c)**, $(c,d), (b,e), (c,f)$
f					5		2	**(a,b)**, **(a,c)**, (c,d), **(c,f)**, $(f,e), (f,g)$
g								**(a,b)**, **(a,c)**, (c,d), **(c,f)**, (f,e), **(f,g)**
d					7			**(a,b)**, **(a,c)**, **(c,d)**, **(c,f)**, (f,e), **(f,g)**
e								**(a,b)**, **(a,c)**, **(c,d)**, **(c,f)**, **(f,e)**, **(f,g)**

Un'altra tabella possibile è la seguente:

estratto	a	b	c	d	e	f	g	*albero*
	0	∞	∞	∞	∞	∞	∞	
a		4	8					$(a,b), (a,c)$
b				8	10			**(a,b)**, $(a,c), (b,d), (b,e)$
d			2		7	9		**(a,b)**, **(b,d)**, $(d,e), (d,c), (d,f)$
c						1		**(a,b)**, **(b,d)**, (d,e), **(d,c)**, (c,f)
f					5		2	**(a,b)**, **(b,d)**, **(d,c)**, **(c,f)**, $(f,e), (f,g)$
g								**(a,b)**, **(b,d)**, **(d,c)**, **(c,f)**, (f,e), **(f,g)**
e								**(a,b)**, **(b,d)**, **(d,c)**, **(c,f)**, **(f,e)**, **(f,g)**

Esercizio 5 - Ordinamenti Si consideri il seguente algoritmo.

```
ord(a,inf,sup)
  if (inf < sup)
    if (a[inf] > a[sup]) swap(a,inf,sup)
    ord(a,inf + 1,sup − 1)
    if (a[inf] > a[inf + 1]) swap(a,inf,inf + 1)
    ord(a,inf + 1,sup)
```

1. Questo algoritmo ordina correttamente $\mathtt{a}[\mathtt{inf}..\mathtt{sup}]$? Si giustifichi la risposta.

   Sì, lo possiamo provare per induzione aritmetica completa.

   Base Se la lunghezza della sequenza è ≤ 1 ($\mathtt{inf} \geq \mathtt{sup}$) la sequenza è ordinata, e l'algoritmo non fa nulla.

   Passo induttivo Altrimenti, se il primo elemento è maggiore dell'ultimo essi vengono scambiati (quindi vale $\mathtt{a}[\mathtt{inf}] \leq \mathtt{a}[\mathtt{sup}]$). Dopo la prima chiamata ricorsiva, per ipotesi induttiva si ha $\mathit{ordered}(\mathtt{a}, \mathtt{inf} + 1, \mathtt{sup} - 1)$. Se il primo elemento è maggiore del secondo (che è il primo, quindi il minimo, di questa parte ordinata), essi vengono scambiati, quindi si ha $\mathtt{a}[\mathtt{inf}] \leq \mathtt{a}[\mathtt{inf} + 1..\mathtt{sup} - 1]$, e vale ancora $\mathtt{a}[\mathtt{inf}] \leq \mathtt{a}[\mathtt{sup}]$, quindi si ha $\mathtt{a}[\mathtt{inf}] \leq \mathtt{a}[\mathtt{inf} + 1..\mathtt{sup}]$. Dopo la seconda chiamata ricorsiva, per ipotesi induttiva si ha $\mathit{ordered}(\mathtt{a}, \mathtt{inf} + 1, \mathtt{sup})$, e dato che si ha anche $\mathtt{a}[\mathtt{inf}] \leq \mathtt{a}[\mathtt{inf} + 1..\mathtt{sup}]$ si può concludere che tutta la sequenza è ordinata.

2. Si scriva la relazione di ricorrenza che descrive il numero di confronti in funzione della lunghezza $n = \mathtt{sup} - \mathtt{inf} + 1$.

   $T(1) = 0$
   $T(n) = 2 + T(n-1) + T(n-2)$ per $n > 1$

3. Si valuti la complessità dell'algoritmo.

   La relazione di ricorrenza è analoga a quella per gli alberi di Fibonacci (pagina 50 delle note), quindi si tratta di un algoritmo esponenziale.

   quindi $T(n) \geq 2^{\lfloor n/2 \rfloor}$.