Azzolini Riccardo 2019-05-07

Alberi 2-3-4

1 Albero 2-3-4

Un **albero 2-3-4** è un albero con radice ordinato tale che:

- ogni nodo contiene i valori, con $1 \leq i \leq 3$ (v_1, v_2, v_3, con $v_1 < v_2 < v_3$);
- ogni nodo interno con i valori ha $i + 1$ figli;
- tutte le foglie sono sullo stesso livello;
- gli i valori presenti in un nodo interno definiscono gli $i + 1$ intervalli a cui appartengono gli elementi contenuti in ciascuno degli $i + 1$ sottoalberi:
  1. il primo sottoalbero contiene valori minori di v_1;
  2. il j-esimo sottoalbero, con $1 < j \leq i$, contiene valori maggiori di v_{j-1} e minori di v_j;
  3. l'ultimo sottoalbero $(i + 1)$ contiene valori maggiori di v_i.

Esistono quindi 3 tipi di nodi:

	Valori	Primo sottoalbero	Secondo sottoalbero	Terzo sottoalbero	Quarto sottoalbero
Tipo 2	v_1	$x_1 < v_1$	$x_2 > v_1$		
Tipo 3	v_1, v_2	$x_1 < v_1$	$v_1 < x_2 < v_2$	$x_3 > v_2$	
Tipo 4	v_1, v_2, v_3	$x_1 < v_1$	$v_1 < x_2 < v_2$	$v_2 < x_3 < v_3$	$x_4 > v_3$

Gli alberi 2-3-4 si possono considerare una generalizzazione degli alberi binari di ricerca, nella quale ogni nodo può contenere più di un valore. Il loro principale vantaggio rispetto agli alberi 2-3 è l'assenza di etichette da aggiornare dopo l'inserimento/cancellazione.

2 Numero di dati e altezza

Lemma: Sia n il numero di dati in un albero 2-3-4 di altezza h. Si ha allora che:

$$2^{h+1} - 1 \leq n \leq 4^{h+1} - 1$$

Dimostrazione: per induzione su h.

- Se $h = 0$, l'albero è formato da un unico nodo, che per definizione ha un numero di valori

$$\begin{aligned} 1 \leq n \leq 3 \\ 2^{0+1} - 1 \leq n \leq 4^{0+1} - 1 \end{aligned}$$

- Se $h > 0$, allora:
  - la radice ha almeno 1 valore e 2 sottoalberi, T_1 e T_2, di altezza $h - 1$, che implica

$$\begin{aligned} n \geq 1 + N_{\text{dati}}(T_1) + N_{\text{dati}}(T_2) &\geq 1 + 2^h - 1 + 2^h - 1 \\ &= 2^{h+1} - 1 \end{aligned}$$

    per ipotesi d'induzione.
  - la radice ha al massimo 3 valori e 4 sottoalberi, e quindi, per ipotesi d'induzione:

$$\begin{aligned} n \leq 1 + N_{\text{dati}}(T_1) + N_{\text{dati}}(T_2) + N_{\text{dati}}(T_3) + N_{\text{dati}}(T_4) &\leq 3 + 4(4^h - 1) \\ &= 4^{h+1} - 1 \qquad \square \end{aligned}$$

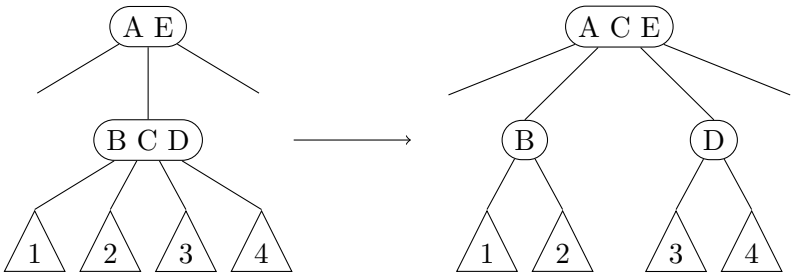
Osservazione: Come conseguenza del lemma, l'altezza di un albero 2-3-4 con n dati è $\Theta(\log n)$, quindi gli alberi 2-3-4 sono **bilanciati**.

3 Inserimento

Per inserire un valore x in un albero 2-3-4:

1. si cerca la posizione in cui inserirlo, scendendo dalla radice fino alle foglie (perché l'inserimento si effettua sempre in una foglia);
2. si aggiunge x ai valori della foglia trovata.

Se si deve inserire un valore in un nodo **saturo**, cioè che contiene già tre valori, tale nodo deve prima essere scomposto, dividendolo in due e inserendo nel padre il valore centrale:



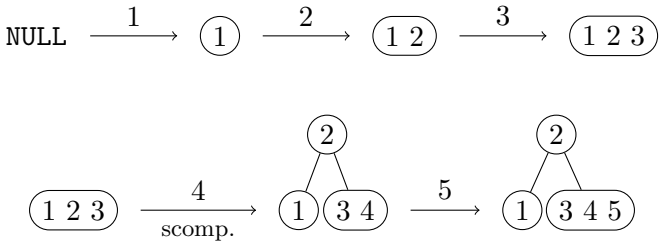
Tale scomposizione si può propagare fino alla radice, se tutti i nodi che si incontrano risalendo l'albero sono saturi. Nell'implementazione, però, si può evitare la necessità di risalire l'albero scomponendo preventivamente i nodi saturi durante la discesa.

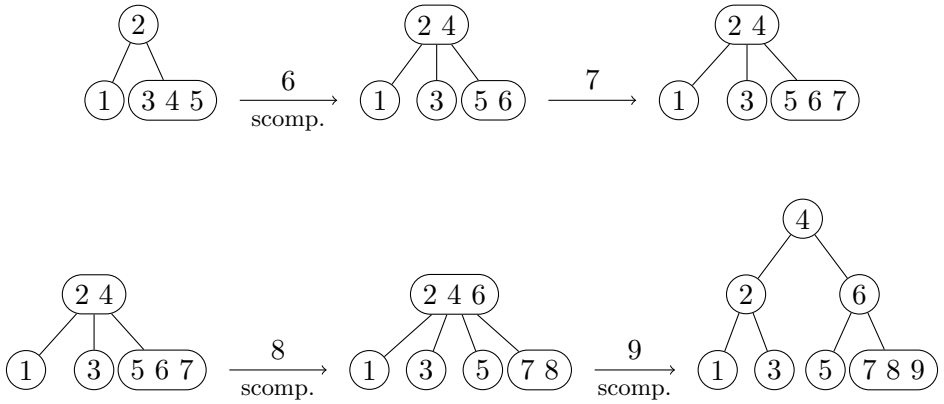
3.1 Complessità

Il costo di un singolo passo, da un livello al successivo, è costante, sia per il semplice attraversamento di un nodo che per la scomposizione di un nodo saturo.

Di conseguenza, l'inserimento richiede in totale tempo di calcolo $\Theta(h) = \Theta(\log n)$ (dove h è l'altezza dell'albero e n il numero di valori contenuti), perché si effettua sempre un cammino dalla radice a una foglia.

3.2 Esempi





4 Complessità di ricerca, minimo e massimo

La complessità in tempo della ricerca è $O(\log n)$, perché

- si arriva a una foglia solo nel caso peggiore: $\Theta(\log n)$;
- nel caso migliore, il dato cercato è alla radice: $O(1)$.

La ricerca del minimo o del massimo, invece, ha sempre costo $\Theta(\log n)$, perché tali valori sono contenuti in delle foglie (rispettivamente la prima a sinistra e l'ultima a destra).

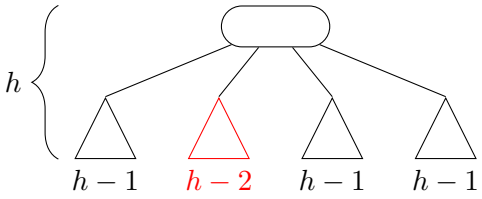
5 Cancellazione

1. Si cerca il nodo contenente il valore x da cancellare.
   - Se tale nodo è una foglia, si elimina direttamente x.
   - Se, invece, è un nodo interno, si cerca un sostituto per x, il massimo tra i valori minori di x o il minimo tra i maggiori: questo viene rimosso dalla foglia in cui è contenuto e messo al posto di x.

   In entrambi i casi, quindi, si rimuove un valore da una foglia α.
2. Se la foglia α rimane senza valori, si effettua la procedura di **ribilanciamento** a partire da α.

5.1 Ribilanciamento

Per definizione, tutti i sottoalberi di un qualunque nodo devono avere la stessa altezza. Dopo la cancellazione, però, si può verificare uno sbilanciamento, cioè può esistere un nodo con un sottoalbero meno alto degli altri:

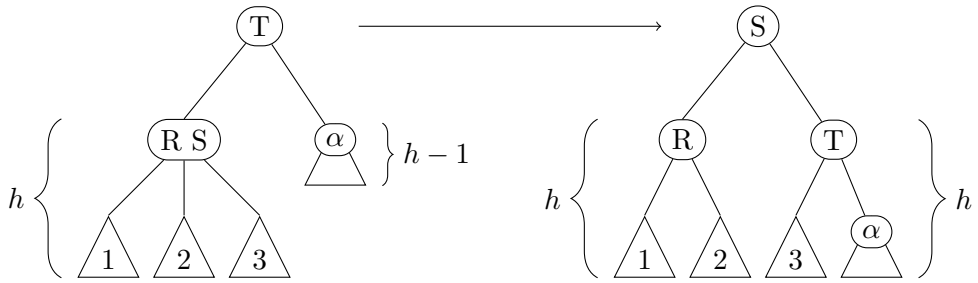


Esso viene corretto mediante una procedura ricorsiva di ribilanciamento, che riceve come argomento (α) la radice del sottoalbero di altezza minore (ad esempio, la foglia che è diventata vuota dopo la cancellazione, la quale è effettivamente un sottoalbero vuoto).

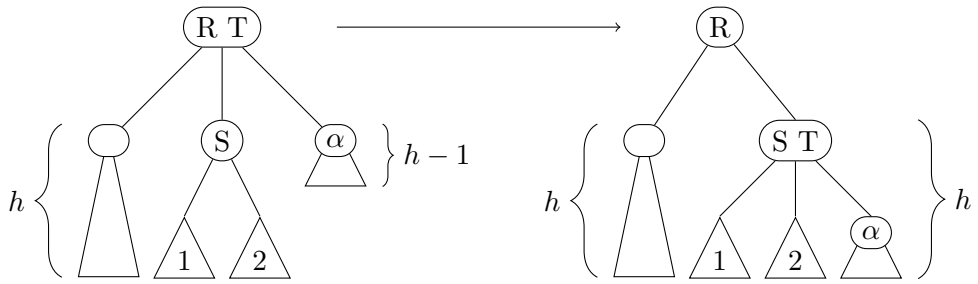
Ci sono tre casi possibili:

1. α ha un fratello con almeno 2 valori;
2. il padre di α contiene almeno 2 valori;
3. il padre e l'unico fratello di α contengono solo un valore ciascuno.

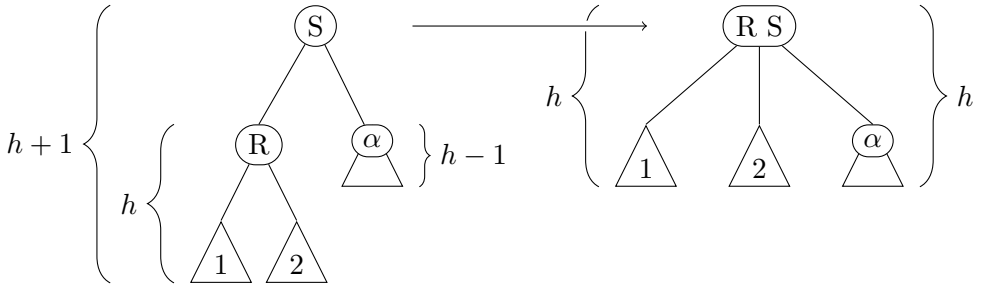
5.1.1 Caso 1



5.1.2 Caso 2



5.1.3 Caso 3



In questo caso, bisogna poi chiamare ricorsivamente la procedura di ribilanciamento sul nodo padre di α (R S), dato che l'altezza dell'intero sottoalbero di cui è radice diminuisce di 1. Se ciò continua a verificarsi fino alla radice dell'intero albero, diminuisce di 1 l'altezza complessiva.

5.2 Complessità

La complessità in tempo della cancellazione è $\Theta(\log n)$ in ogni caso:

1. per la ricerca del valore e del sostituto si percorre un cammino dalla radice a una foglia, quindi il costo è $\Theta(\log n)$;[1]
2. il ribilanciamento ha costo $O(\log n)$:
   - $O(1)$ nel caso migliore, quando non rimane una foglia vuota, e quindi non è necessario alcun ribilanciamento;
   - $\Theta(\log n)$ nel caso peggiore, quando il problema si propaga fino alla radice.

[1] Se, invece, si considera il costo della sola ricerca del valore da eliminare o della sola ricerca del sostituto, questo è $O(\log n)$, perché ciascuna delle due ricerche può percorrere anche solo parte del cammino dalla radice a una foglia. Infatti, la ricerca del sostituto completa il cammino iniziato dalla ricerca del valore, nei casi in cui quest'ultima non si ferma già in una foglia).