

Algoritmi per alberi

Albero: $T = (N, r, B)$

- N : insieme di nodi
- r (radice) $\in N$
- B : relazione binaria su N tale che:
 - $\forall n \in N, (n, r) \notin B$
 - $\forall n \in N, \text{ se } n \neq r, \exists n' \in N : (n', n) \in B$
 - $\forall n \in N, \text{ se } n \neq r \text{ è raggiungibile da } r, \text{ cioè esistono } n_1, \dots, n_k \in N, k \geq 2, \text{ tali che: } n_1 = r, (n_i, n_{i+1}) \in B \forall 1 \leq i \leq k-1, \text{ ed } n_k = n$
 - $\forall n \in N, \text{ se } n \neq r \text{ esistono e sono unici } n_1, \dots, n_k \in N, \text{ con } k \geq 2, \text{ tali che: } n_1 = r, (n_i, n_{i+1}) \in B \forall 1 \leq i \leq k-1, \text{ ed } n_k = n$

la lista è un particolare albero con un solo percorso.

Dati $T = (N, r, B)$ e $n \in N$, un sottoalbero generato da n è definito come:

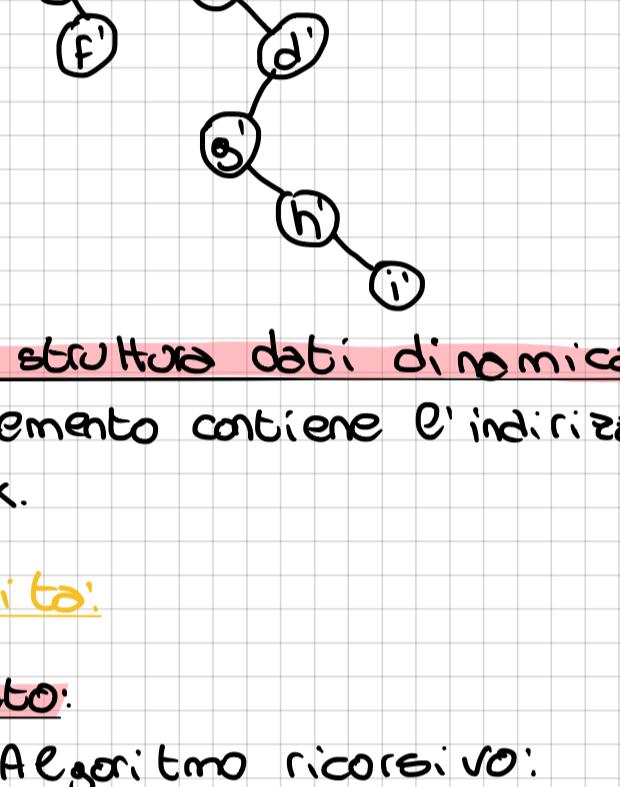
- $T' = (N', r', B')$,
- N' è l'insieme dei nodi raggiungibili da n
- $B' = B \cap (N' \times N')$

Se T è un albero e T_1, T_2 sottoalberi generati da n_1, n_2 :

$$N_1 \cap N_2 = \emptyset \vee N_1 \subseteq N_2 \vee N_2 \subseteq N_1$$

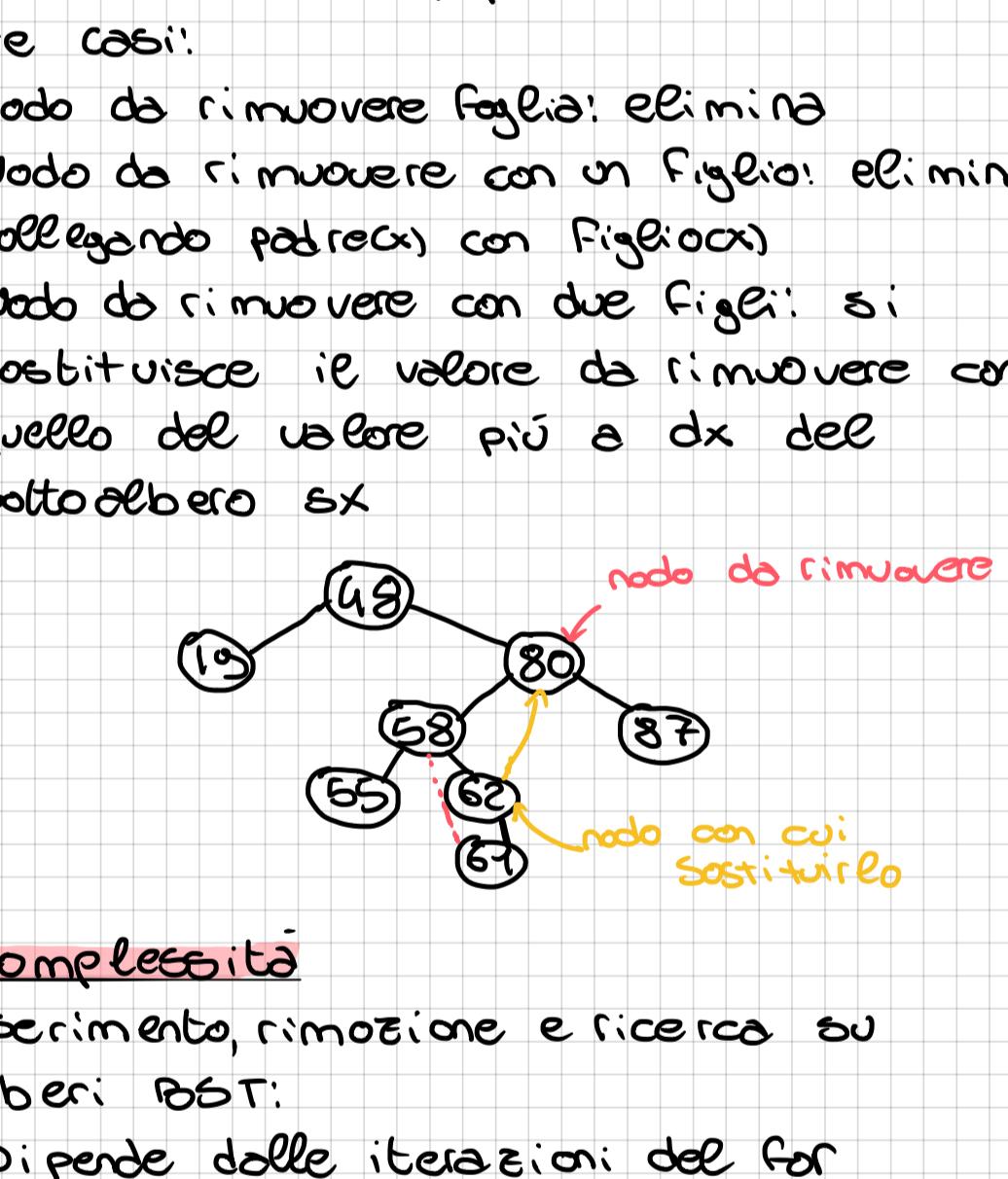
Albero libero: $T = (N, B)$ dove la relazione binaria ha proprietà:

- Antiriflessiva
- Simmetrica
- Connattività
- Aciclicità



Dato $T = (N, B)$ con B antiriflessiva e simmetrica, le seguenti affermazioni sono equivalenti:

- T è un albero libero
- due vertici qualsiasi sono collegati da un cammino semplice ed unico
- T è connesso, ma se viene rimosso un qualsiasi arco definito da B , la struttura risultante non è connessa
- T è aciclico, ma se in qualunque arco viene aggiunto e B la struttura risultante contiene un ciclo
- T è connesso e $|B| = |N| - 1$
- T è aciclico e $|B| = |N| - 1$



Grado: max. nr. Figli di un nodo

Altezza: max. nr. nodi attraversati dalla radice alla foglia più distante

Larghezza: max. nr. nodi allo stesso livello

Un albero di grado $d \geq 1$ e altezza h può contenere un max. nr. di nodi!

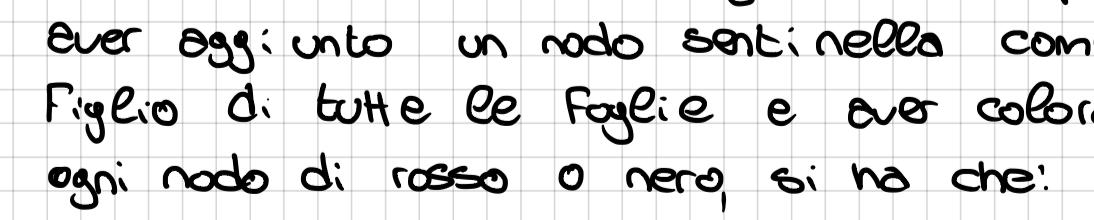
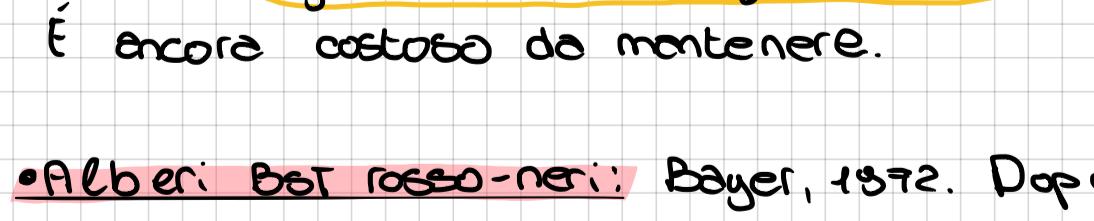
$$L_n = \sum_{i=0}^{h-1} d^i = \frac{d^h - 1}{d - 1}$$

Un albero T è detto ordinato se $\forall n$, le chiavi relative ai suoi figli soddisfano una relazione d'ordine totale.

Alberi binari

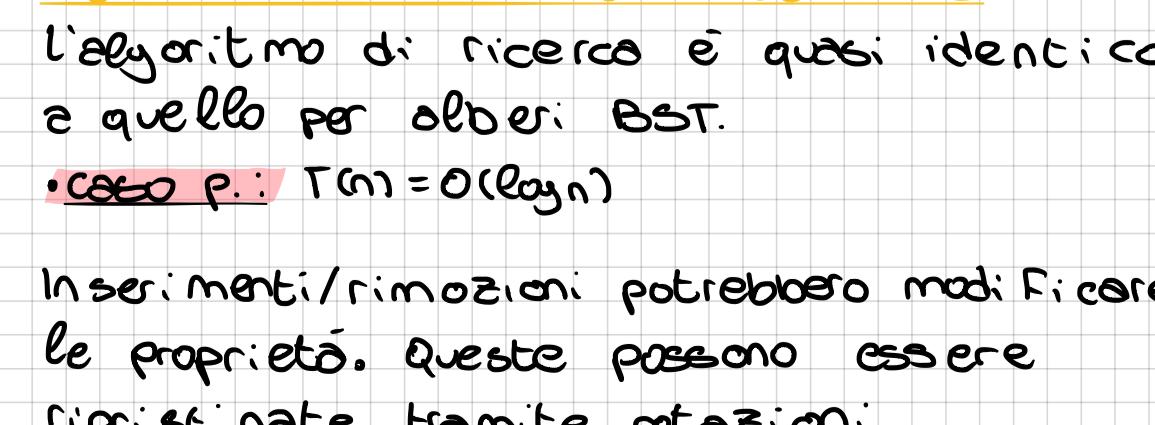
$B = B_{sx} \cup B_{dx}, B_{sx} \cap B_{dx} = \emptyset$

$\forall n, n_1, n_2 \in N, \text{ se } (n, n_1) \in B_{sx} \text{ e } (n, n_2) \in B_{dx} \Rightarrow n_1 < n_2$



Ogni albero non binario è equivalente ad un albero binario cui è applicata la trasformazione fratello-foglio:

- Crea nuovi nodi n'_1, n'_2, \dots, n'_k
- Pon n'_1 Figlio sx di n'
- $\forall i=1, \dots, k-1$ pon n'_{i+1} - Figlio dx di n'_i



Un albero è una struttura dati dinamica gerarchica, ogni elemento contiene l'indirizzo dei Figli dx. e sx.

Algoritmi di visita:

In ordine anticipato:

- Algoritmo ricorsivo:
 - Visita testa (elabora)
 - Visita Figlio sx
 - Visita Figlio dx

In ordine simmetrico:

- Algoritmo ricorsivo:
 - Visita (nodo \rightarrow sx)
 - Elabora (nodo \rightarrow valore)
 - Visita (nodo \rightarrow dx)

In ordine posticipato:

- Algoritmo ricorsivo:
 - Visita (nodo \rightarrow sx)
 - Visita (nodo \rightarrow dx)
 - Elabora (nodo \rightarrow valore)

Gli algoritmi di visita possono essere usati anche per la ricerca.

T(n) = O(n) nel caso peggiore (albero vuoto o valore nella radice).

T(n) = O(1) nel caso ottimo (albero vuoto o valore nella radice).

Alberi binari di ricerca

Per ogni nodo contenente una chiave K :

- le chiavi del sottoalbero sx sono $\leq K$
- le chiavi del sottoalbero dx sono $\geq K$

Per trovare un valore V , basta fare un percorso da r ad ogni nodo x t.c.:

- Chiave(x) = $V \rightarrow$ valore trovato
- Chiave(x) $< V \rightarrow$ Proseguire a sx
- Chiave(x) $> V \rightarrow$ Proseguire a dx

La lunghezza media in un T con n nodi è:

$$L_n = \frac{1}{n} \cdot \sum_{i=1}^{n-1} l_i$$

Se la radice contiene la i -esima chiave, B_{sx} contiene le $i-1$ chiavi $\leq i$ e B_{dx} contiene le $n-1$ chiavi $\geq i$. Quindi:

$$L_n = \frac{1}{n} \left(\sum_{j=1}^{i-1} l_j + 1 + \sum_{j=i+1}^n l_j \right) = \frac{1}{n} ((i-1) \cdot L_{i-1} + (n-i) \cdot L_{n-i})$$

Svolgendo una ridicola quantità di calcoli che non mi va di copiare, si ottiene:

$$T(n) = O(\log(n))$$

Criteri di bilanciamento per BST

È possibile ottenere $O(\log(n))$ nel caso peggiore mantenendo l'albero bilanciato.

Inserimento: riempire il più possibile i livelli esistenti prima di creare di nuovi.

Rimozione: ridistribuire i nodi rimasti nei livelli per uniformare.

Se in BST è perfettamente bilanciato $\forall n, i \in N$, il numero di nodi dei sottoalberi sx e dx differiscono al più di uno.

Siccome il perfetto bilanciamento è costoso da mantenere, sono state introdotte forme meno restrittive.

Alberi AVL-bilanciati: Adelson-Velsky e Landis. $\forall n, i \in N$, l'altezza dei sottoalberi differiscono al più di 1.

L'altezza h di un BST di questo tipo soddisfa:

$$\log(n+1) \leq h \leq 1.4404 \cdot \log(n+2) - 0.328$$

È ancora costoso da mantenere.

Alberi BST rosso-neri: Bayer, 1972. Dopo aver aggiunto un nodo sentinella come Figlio di tutte le foglie e aver colorato ogni nodo di rosso o nero, si ha che:

- la radice è nera
- la sentinella è nera
- se n è rosso, i suoi Figli sono neri
- $\forall n, \text{ tutti i percorsi da } n \text{ alla sentinella hanno lo stesso numero di neri.}$

L'altezza h di questi alberi soddisfa:

$$h \leq 2 \log_2(n+1)$$

Inserimento/rimozione in alberi RB

L'algoritmo di ricerca è quasi identico a quello per alberi BST.

Caso p: $T(n) = O(\log(n))$

Inserimenti/rimozioni potrebbero modificare le proprietà. Queste possono essere ripristinate tramite rotazioni.

In un albero RB, ogni nodo contiene l'indirizzo del padre e l'informazione sul colore.

Il padre della radice è la sentinella.

Le proprietà vengono mantenute anche attraverso opportuni cambi di colore.