BST vs AVL vs RBT

Matteo Orlandi

February 19, 2026

1 Introduzione

In questa relazione verranno analizzate e confrontate le prestazioni di tre diverse strutture dati utilizzate per la gestione di insiemi dinamici. Le strutture che verranno analizzate sono: gli Alberi Binari di Ricerca (BST), gli alberi AVL e gli Alberi Rosso-Neri (RBT).

La gestione ottimale dei dati è un obiettivo fondamentale quando si parla di strutture dati. Sebbene i BST offrano una struttura semplice e intuitiva per l'inserimento e la ricerca, le prestazioni ottenute dipendono fortemente dall'ordine dei dati in ingresso. Queste possono anche sfociare, nel caso peggiore, in liste collegate, perdendo ogni vantaggio della struttura iniziale. Per ovviare a questo problema, sono state introdotte strutture auto-bilancianti come gli alberi AVL e gli alberi Rosso-Neri, che garantiscono tempi di esecuzione logaritmici, indipendentemente dall'ordine dei dati in ingresso.

L'esperimento che ne deriva mira a verificare sperimentalmente le complessità temporali delle operazioni di **inserimento** e **ricerca**, mettendo a confronto le tre strutture dati negli scenari del caso peggiore e del caso medio.

2 Teoria

Un albero è una struttura dati composta da un insieme di *nodi* e *archi*. Ogni nodo rappresenta un elemento dell'albero ed è composto da una chiave che lo identifica univocamente e da più puntatori, detti archi, che offrono un collegamento verso altri nodi, detti *figli*.

Il nodo principale, da cui ha origine l'intera struttura, prende il nome di *radice*. Ogni nodo che non ha figli viene detto *foglia*. Si dice *cammino* un insieme di nodi e archi che devono essere attraversati per collegare un nodo a un altro. La lunghezza del cammino più lungo rappresenta l'*altezza* dell'albero e viene misurata contando il numero di nodi che si incontrano lungo quel cammino (parte dalla radice e giunge a una foglia)

2.1 Binary Search Tree (BST)

Un Albero Binario di Ricerca è un tipo particolare di albero specializzato per la ricerca di valori. Ogni nodo avrà un puntatore sinistro che permette il collegamento a un sottoalbero contenente solo valori minori e uno destro che permette il collegamento a un sottoalbero contenente solo valori maggiori.

Le operazioni di base richiedono un tempo proporzionale all'altezza h dell'albero, ovvero $O(h)$.

- **Caso Medio:** Se le chiavi vengono inserite in ordine casuale, l'altezza attesa dell'albero è $O(\log n)$.
- **Caso Peggiore:** Se le chiavi vengono inserite in ordine crescente o decrescente, l'albero degenera in una lista concatenata di altezza $h = n - 1$. In questo scenario, le operazioni raggiungono una complessità $O(n)$, rendendo la struttura inefficiente.

2.2 Alberi AVL

Gli alberi AVL sono alberi binari di ricerca **strettamente bilanciati in altezza**. Un albero AVL è bilanciato se, per ogni nodo x, le altezze dei sottoalberi sinistro e destro differiscono al massimo di 1. Per fare ciò, viene introdotto il *Fattore di Bilanciamento* $BF(x)$, definito come:

$$BF(x) = h(x.left) - h(x.right) \tag{1}$$

Perché un albero sia AVL, deve valere che $BF(x) \in \{-1, 0, 1\}$ per ogni nodo. A causa di questo vincolo molto stringente, gli alberi AVL sfruttano al massimo l'ampiezza, ottenendo un'altezza massima limitata a circa $1.44 \lg n$. Nonostante abbiano un costo di inserimento non molto ottimizzato a causa dei frequenti controlli e delle rotazioni, il costo per la ricerca è ottimale.

2.3 Red-Black Tree (RBT)

Gli alberi Rosso-Neri hanno un funzionamento simile a quello degli AVL, ma il bilanciamento viene effettuato secondo limiti meno restrittivi. Ogni nodo contiene un bit supplementare di informazione che indica il colore (rosso o nero). Un albero Rosso-Nero, oltre a soddisfare le caratteristiche di un BST, deve soddisfare le seguenti proprietà:

1. La radice è nera.
2. Ogni nodo è rosso o nero.
3. Ogni foglia (NIL) è nera.
4. Se un nodo è rosso, entrambi i suoi figli sono neri (non possono esserci due nodi rossi consecutivi).
5. Per ogni nodo, tutti i cammini semplici che scendono verso le foglie discendenti contengono lo stesso numero di nodi neri.

In questo modo abbiamo la garanzia che il cammino più lungo non sia mai maggiore del doppio del cammino più breve. L'altezza dell'albero è perciò limitata a $2 \lg(n + 1)$, assicurando che il limite asintotico delle operazioni di inserimento e ricerca sia $O(\log n)$ anche nel caso peggiore.

3 Esperimento e risultati attesi

Tramite questo esperimento si vogliono verificare le complessità asintotiche esposte nella sezione teorica, misurando il tempo di esecuzione $T(n)$ al variare del numero di nodi N.

3.1 Scenari di Test e Complessità Attesa

Vengono definiti due scenari per confrontare le caratteristiche delle diverse strutture dati:

Scenario A: Sequenza Ordinata (Caso Peggiore)

Vengono inseriti elementi in ordine strettamente crescente $(0, 1, ..., n)$.

- **BST:** Il risultato atteso è che l'albero degeneri, con altezza $h = n$. Il risultato asintotico sia per la costruzione che per la ricerca sarà $O(n)$.
- **RBT e AVL:** Le strutture dovrebbero bilanciarsi sfruttando le rotazioni e mantenendo un'altezza logaritmica e prestazioni $O(\log n)$.

Scenario B: Sequenza Casuale (Caso Medio)

Vengono inseriti elementi casuali, generati secondo una distribuzione uniforme. Il risultato atteso è che tutte le strutture (BST, AVL e RBT) mantengano un'altezza logaritmica, con prestazioni comparabili $O(\log n)$.

3.2 Tabella riassuntiva

La Tabella che segue riassume i risultati asintotici attesi e che dovranno essere verificati tramite i grafici derivanti dai test.

Mi aspetto che gli AVL, essendo più bilanciati, avranno tempi di ricerca migliori al crescere di N rispetto agli RBT, mentre gli BST risulteranno i più veloci per l'inserimento di valori casuali a causa della mancanza di controlli e di rotazioni.

Struttura	Operazione	Caso Peggiore (Ordinato)	Caso Medio (Random)
BST	Inserimento	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$
BST	Ricerca	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$
AVL	Inserimento	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
AVL	Ricerca	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
RBT	Inserimento	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
RBT	Ricerca	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$

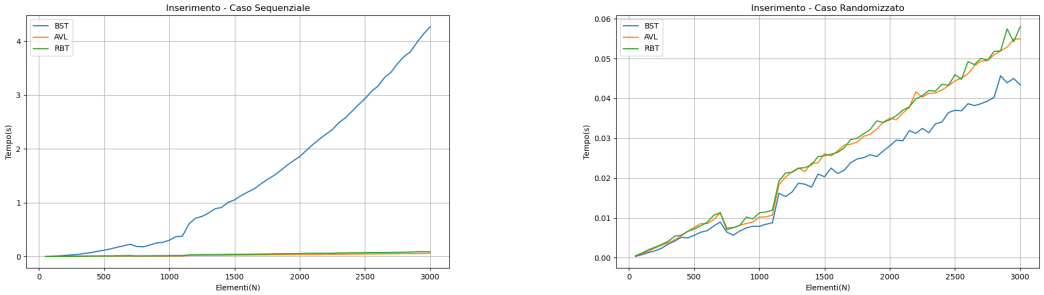

Figure 1: Tempi di inserimento di N elementi. Nel caso a gli elementi sono ordinati, nel caso b gli elementi sono randomici.

4 Risultati esperimento

4.1 Tempi di inserimento

4.2 Altezza alberi

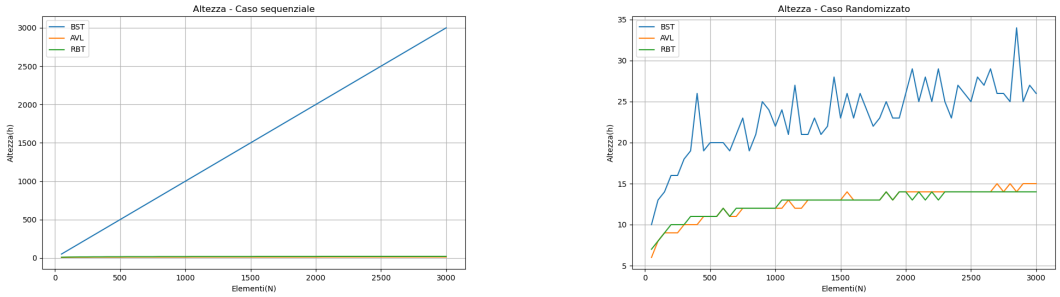

Figure 2: Confronto altezze per gli inserimenti. Nel caso a gli elementi sono ordinati, nel caso b gli elementi sono randomici.

4.3 Tempi di ricerca

5 Conclusioni

Osservando i risultati dei test effettuati, possiamo notare che, per quanto riguarda il caso peggiore, le prestazioni sia per l'inserimento che per la ricerca calano drasticamente se implementate utilizzando un albero binario di ricerca. Lo stesso vale per l'altezza dell'albero, in quanto l'input particolare azzera tutti i vantaggi che un BST offrirebbe, sfociando in una lista concatenata. Analizzando

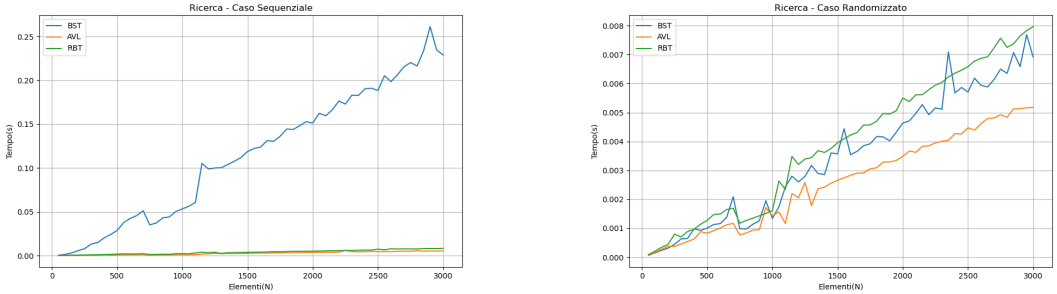

Figure 3: Tempi di ricerca di N elementi. Nel caso a gli elementi sono ordinati, nel caso b gli elementi sono randomici.

invece il caso medio, l'albero binario di ricerca, seppur con prestazioni peggiori, tende a replicare i vantaggi delle altre due tipologie di alberi. Le differenze tra le varie strutture dati si assottigliano molto, ottenendo asintoticamente gli stessi risultati.

Nel complesso, RBT rimane comunque la scelta migliore tra le 3, riuscendo a mitigare le caratteristiche di velocità di un BST, noncurante del bilanciamento della struttura complessiva, con quelle di organizzazione in memoria di un AVL, rigido per ottimizzare il più possibile la ricerca.