Binary Tree VS Red-Black Tree

Ilda Serjanaj

10 Aprile 2022

Indice

1 Introduzione al problema **2**

2 Caratteristiche teoriche di algoritmi e e strutture utilizzate **2**

3 Prestazioni attese **2**

4 Esperimenti **2**

5 Risultati **3**
5.1 Inserimento - Caso medio (Input Random) 3
5.2 Inserimento - Caso peggiore per RBT (Input in ordine) 4
5.3 Ricerca - Caso medio (input Random) 5
5.4 Ricerca - Caso peggiore per RBT (ricerca su input ordinato) . . 6

6 Conclusioni **6**

1 Introduzione al problema

Lo scopo di questo esercizio e' verificare i vantaggi e gli svantaggi di due strutture dati, consideranto le prestazioni in termini di tempo di inserimento e ricerca nei due casi: **Alberi binari di ricerca** ed **Alberi rosso-neri** studiando il tempo di esecuzione delle varie operazioni tipiche su strutture dati

2 Caratteristiche teoriche di algoritmi e e strutture utilizzate

Le due strutture dati sono ad albero e pertanto prevedono l'esitenza di un nodo detto "radice" (all'apice di essi). Ogni nodo ha un attributo **parent** corrispondente al padre, un attributo **left** corrispondente al proprio figlio sinistro ed un attributo **right** corrispondente al figlio destro
BST: Negli alberi binari di ricerca si ha che il figlio sinistro (left) e' minore del padre, mentre il figlio destro (right) e' maggiore del padre. L'altezza dell'albero non ha un upper bound e dunque, nel caso i valori siano stati inseriti in ordine (crescente o decrescente), l'albero risulta sbilanciato; essendo che la ricerca e l'inserimento scendono l'albero livello per livello, nel caso critico appena citato il tempo richiesto per tali operazioni puo' essere molto elevato.
RBT: Derivano dagli RBT, ma a differenza di essi i nodi che lo compongono sono dotati dell'attributo "color", grazie a cui si puo mantenere bilanciato l'albero (e' dimostrabile che $h_{alberoRN} \leq 2lg(n+1)$): per mantenere tale caratteristica ad ogni inserimento bisogna pero' chiamare la funzione **fix-up**, il che ha un costo a volte non trascurabile.

3 Prestazioni attese

Le prestazioni attese per gli algoritmi relativamente ad BRT ed a RBT sono riportate nella tabella sottostante dove con **h** indichiamo l'altezza dell'albero. Non eseguiremo test di **Delete** poiche' avremmo conclusioni affini con **Search** ed **Insert**

Struttura dati	Search	Insert	Delete
BST	$O(h)$	$O(h)$	$O(h)$
RBT	$O(\log n)$	$\Theta(\log n)$	$O(\log n)$

Nel caso peggiore (albero completamente sbilanciato) quindi la complessità di BST diventa $O(N)$, nel caso migliore (albero bilanciato) la complessità diventa la stessa di RBT $O(\log n)$.

4 Esperimenti

Sono stati effettuati in totale 4 esperimenti (tutti prevedono l'utilizzo di dataset crescenti ad ogni ciclo partendo da 10 fino a raggiungere 12000), in ognuno dei quale vengono confrontati BST ed RBT:

1. Inserimento di dataset di numeri random: viene calcolato il tempo di esecuzione che imipiegano i due algoritmi a inserire dati random nei rispettivi alberi.
2. Inserimento di dataset di numeri ordinati: viene calcolato il tempo di esecuzione che imipiegano i due algoritmi a inserire dati, in questo caso consideriamo il caso peggiore per per gli alberi BST ovvero l'inserimento di dati già ordinati.
3. Ricerca su alberi generati da dataset di numeri random: vengono ricercati dati random su alberi costruiti a partire da dataset random
4. Ricerca su alberi generati da dataset di numeri ordinati ricercati dati random su alberi costruiti a partire da dataset ordinati

5 Risultati

5.1 Inserimento - Caso medio (Input Random)

Dalla Figura 1 possiamo notare come nel caso dell'inserimento di dati random risulta più veloce l'albero Binario.

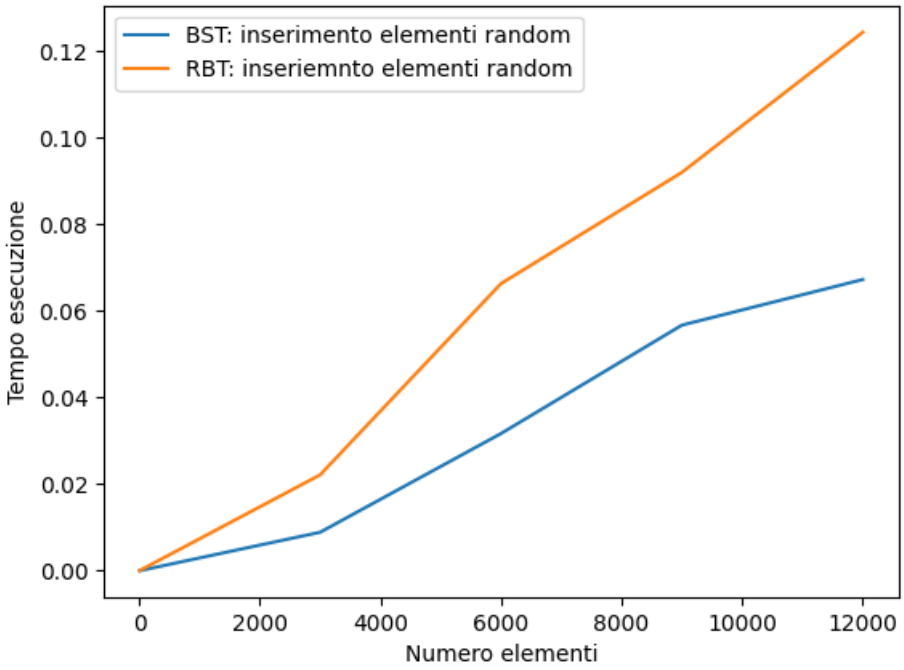

Figura 1: Caso medio per l'inserimento

5.2 Inserimento - Caso peggiore per RBT (Input in ordine)

Grazie alla figura 2 invece notiamo come nel caso in cui inseriamo i dati in ordine, risulta più veloce RBT, in quanto essendo BST non bilanciato l'albero risultante tenderà a crescere in un solo lato aumentando così l'altezza totale dell'albero, al contrario del RBT che invece grazie all'operazione di fix-up tenderà a far risultare bilanciato l'albero risulatante. Questa cosa andrà a influenzare anche la ricerca nel suddetto albero.

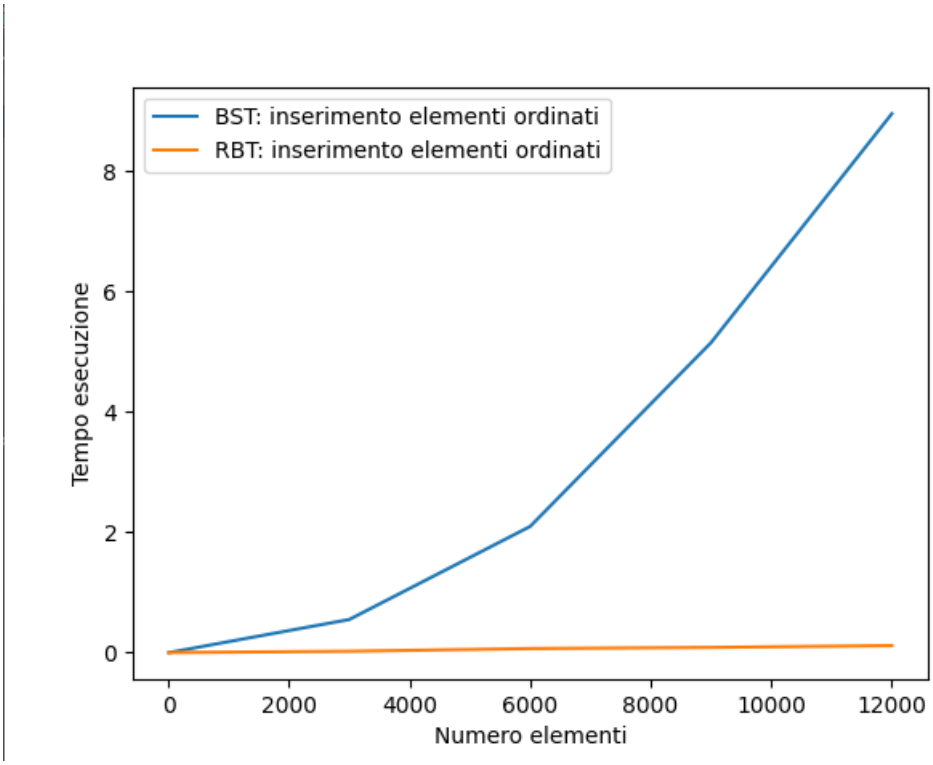

Figura 2: Caso peggiore per l'albero binario di ricerca

5.3 Ricerca - Caso medio (input Random)

Dalla Figura 3

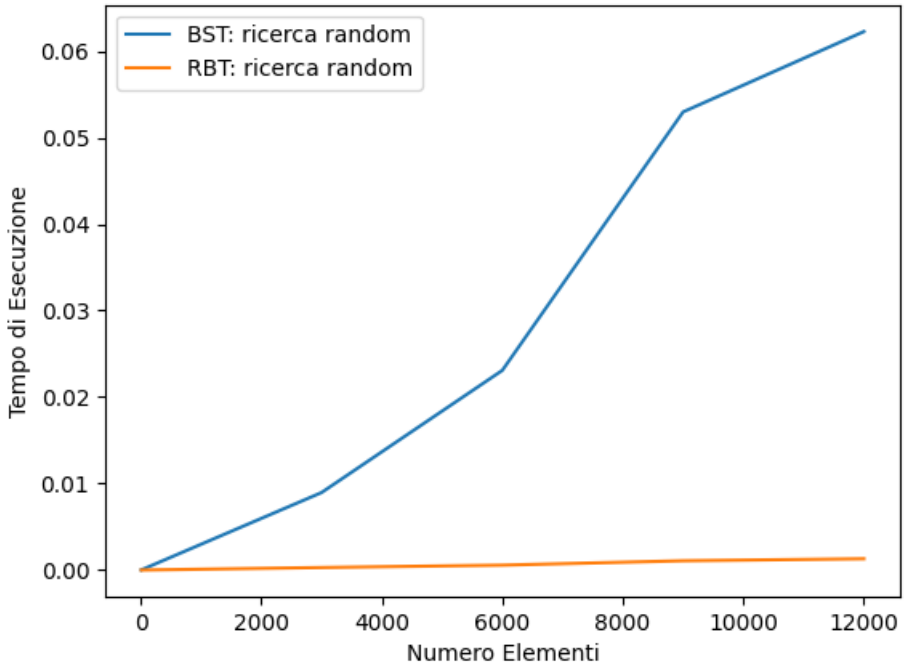

Figura 3: Caso peggiore per l'albero binario di ricerca

5.4 Ricerca - Caso peggiore per RBT (ricerca su input ordinato)

Grazie alla Figura 4

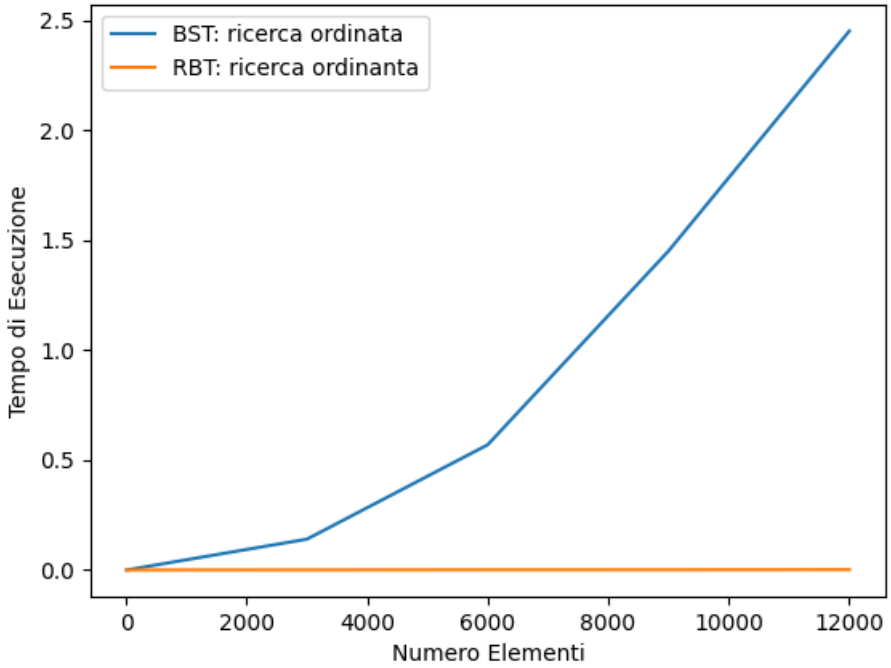

Figura 4: Caso peggiore per l'albero binario di ricerca

6 Conclusioni

E' stato verificato il comportamento asintotico degli algoritmi di inserimento e ricerca: si deduce che il RBT e' preferibile quando si ipotizza un preordinamento (crescente o decrescente) nei valori in input: in tal caso l'BST risulta totalmente sbilanciato e quindi sconveniente sia per inserimento che per ricerca. Se invece i valori in input non sono ordinati, avremo un BST non eccessivamente sbilanciato: l'inserimento sara' spesso piu' veloce rispetto a quello del concorrente poiche', a differenza di esso, non deve eseguire un fix-up.