Relazione per il Secondo Progetto di Algoritmi e Strutture Dati e Laboratorio

Alessandro Zanatta
143154
zanatta.alessandro@spes.uniud.it

Christian Abbondo
144033
abbondo.christian@spes.uniud.it

12-09-2020

Indice

1 Introduzione

La seguente relazione si propone di analizzare i tempi di esecuzione di operazione su tre diversi tipi di alberi binari:

- Binary Search Tree (BST)
- Adelson-Velsky and Landis Tree (AVL)
- Red-Black Tree (RBT)

Il linguaggio scelto e utilizzato per implementare le strutture dati e gli algoritmi, nonchè per il calcolo dei tempi di esecuzione, è C, in quanto è indubbiamente uno dei linguaggi più efficienti e veloci, al costo di una maggiore complessità del codice.
In questo modo i tempi di esecuzione ottenuti sono liberi da operazioni sulla memoria (come ad esempio un garbage collector) che vengono invece effettuate in momenti opportuni.

2 Considerazioni teoriche

In questa sezione si discuteranno le aspettative teoriche relative ai tempi di esecuzione degli alberi binari in questione.

2.1 Costo asintotico

Ricordiamo, innanzitutto, il costo asintotico delle operazioni al variare del numero di elementi n presenti nell'albero:

Binary Search Tree
Operazione di ricerca: $\Theta(n)$ nel caso peggiore, $O(\log n)$ nel caso medio.
Operazione di inserimento: $\Theta(n)$ nel caso peggiore, $O(\log n)$ nel caso medio.

Adelson-Velsky and Landis Tree
Operazione di ricerca: $O(\log n)$ in tutti i casi.
Operazione di inserimento: $O(\log n)$ in tutti i casi.

Red-Black Tree
Operazione di ricerca: $O(\log n)$ in tutti i casi.
Operazione di inserimento: $O(\log n)$ in tutti i casi.

2.2 Aspettative teoriche

Si discutono di seguito le aspettative teoriche riguardanti i diversi tipi di alberi.

BST Data la maggior semplicità e i maggiori costi asintotici di questo tipo di albero, ci si aspetta una maggiore inefficienza, accentuata se l'albero si sbilancia. Per input casuali potrebbe comunque eseguire le operazioni in un tempo comparabile agli altri due alberi binari in quanto potrebbe rimanere, a causa della natura casuale delle chiavi dei nodi che vengono inseriti, relativamente bilanciato.
Per input fortemente ordinati, questo albero sarà invece certamente molto inefficiente.

RBT e AVL Sia i RBT che gli AVL impiegano tempo asintotico logaritmico per qualsiasi input e operazione, tuttavia è importante notare le differenze di questi due tipi di alberi.
I RBT tendono a essere meno bilanciati rispetto agli alberi di tipo AVL. Formalmente, dato un nodo x e denotato come $x.left$ e $x.right$ rispettivamente il figlio sinistro e destro di x, si ha che $\mathrm{h}(x.left) \leq 2 \cdot \mathrm{h}(x.right)$ (e viceversa). Negli alberi AVL, invece, la differenza di altezza fra due sottoalberi di un certo nodo x non supera mai l'unità. Ci si aspetta quindi un tempo di ricerca maggiore per i RBT rispetto agli AVL.
Consideriamo inoltre la procedura di inserimento in un RBT. Questa richiede al più due cammini radice-foglia, in particolare ne richiede sempre uno per arrivare in una foglia (in cui verrà inserito il nodo con la chiava appropriata) e ne richiede al più un altro (o solo parte di esso) per ri-bilanciare l'albero. Per quanto riguarda gli AVL, invece, sono sempre richiesti esattaemente due cammini, il primo per l'inserimento e il secondo per ri-bilanciare l'albero, similmente al caso dei RBT. Per quanto riguarda l'operazione di inserimento, quindi, ci si aspetta che i RBT siano maggiormente efficienti degli alberi di tipo AVL.

3 Note implementative

Si discutono ora alcune scelte implementative effettuate.

Generazione di numeri pseudo-casuali Dato che la funzione rand() di C è piuttosto lenta e tende a fornire gli stessi interi dopo un numero relativamente piccolo di iterazioni, si utilizzerà una implementazione dell'algoritmo Mersenne Twister.

Scomputazione dei tempi di inizializzazione Si è ritenuto opportuno, al fine di ottenere misurazioni maggiormente accurate e minormente affette da errori casuali, scomputare il tempo di generazione di n numeri casuali utilizzando l'algoritmo sopra nominato. Si discute il modo in cui si è effettuato ciò nella sezione successiva (4).

4 Calcolo del tempi di esecuzione

Si riportano di seguito alcune considerazioni riguardo il calcolo dei tempi effettuato nell'elaborato.

Risoluzione Si è scelto di utilizzare il clock offerto dalla funzione clock_gettime. Il primo parametro della funzione indica il tipo di clock, mentre il secondo parametro è un puntatore ad una struct contentente il tempo all'istante della chiamata a funzione. Il tipo di clock utilizzato (CLOCK_MONOTONIC_RAW) è, come suggerito dal nome stesso, di tipo monotonico.
Per ottenere una misurazione della risoluzione migliore si è utilizzata la risoluzione mediana, calcolata su un campione di 10000 elementi.

Scomputazione dei tempi di inizializzazione Al fine di scomputare correttamente dai tempi di esecuzione delle operazioni il tempo di generazione di n numeri casuali, si è utilizzata una funzione apposita che calcola il tempo mediano di tale operazione. Al fine di ottenere un errore relativo minore di 0.5% si è ripetuta l'inizializzazione sufficienti volte affinché la seguente equazione fosse soddisfatta:

$$\text{diff}\,(end, start) > resolution \cdot \left(1 + \frac{1}{0.005}\right)$$

dove diff indica l'intervallo temporale trascorso fra *start* e *end* e *resolution* è la risoluzione del clock utilizzato.
Questo procedimento è stato poi iterato un certo numero di volte e dal campione di dati ottenuto si è estratta la mediana.

Tempo di esecuzione degli algoritmi Per ottenere misurazioni con errore relativo minore di 0.5% si è operato nel seguente modo:

- Calcolo del tempo di generazione dei numeri casuali secondo la procedura descritta nel paragrafo precedente

- Calcolo del tempo di esecuzione delle operazioni. Si itera questo passaggio finché non è soddisfatta la seguente equazione:

  $$\text{diff}\,(end, start) > initialization + resolution \cdot \left(1 + \frac{1}{0.005}\right)$$

  dove *initialization* è il tempo calcolato al primo punto.

Errore relativo massimo totale Calcolando i tempi nei modi indicati negli ultimi due paragrafi si è ottenuto un errore relativo complessivo $\leq 1\%$. L'errore relativo totale è, infatti, dato dalla somma degli errori. Essendoci quindi due misurazioni affette da errore[1], entrambe con errore relativo pari o minore a 0.5%, l'errore relativo massimo totale è effettivamente dell'1%.

Problematiche riscontrate Durante le prime misurazioni, si era notata una deviazione standard piuttosto elevata, indice di una grande variabilità nelle misurazioni ottenute. Al fine di ottenere delle misurazioni migliori, cioè meno sensibili ad outliers e rumore, si è scelto di utilizzare, al posto della media, la mediana dei tempi di esecuzione e, al posto della deviazione standard, la deviazione mediana assoluta, definita come $\text{MAD} = \text{median}\left(|X_i - \text{median}\left(X\right)|\right)$. Questi indici, in quanto indici di posizione, sono più robusti della media e della deviazione standard e hanno permesso di ottenere dei tempi sperimentali più accurati e minormente affetti da errori casuali.

[1] La prima è quella relativa al calcolo del tempo di inizializzazione, la seconda è quella relativa al tempo di esecuzione dell'algoritmo.

5 Analisi degli tempi di esecuzione

Per la valutazione dei tre diversi tipi di alberi sono stati utilizzati diversi criteri e sono state considerate diverse casistiche ritenute interessanti. In particolare, ci si soffermerà sui seguenti casi:

- n operazioni di inserimento
- n operazioni di inserimento con il 5% di elementi ordinati
- n operazioni di ricerca
- n operazioni di ricerca con il 10% di elementi ordinati
- n operazioni di ricerca con l'80% di elementi ordinati

5.1 Esecuzione di n operazioni di inserimento

Iniziamo dall'analisi del grafico di un caso generale.

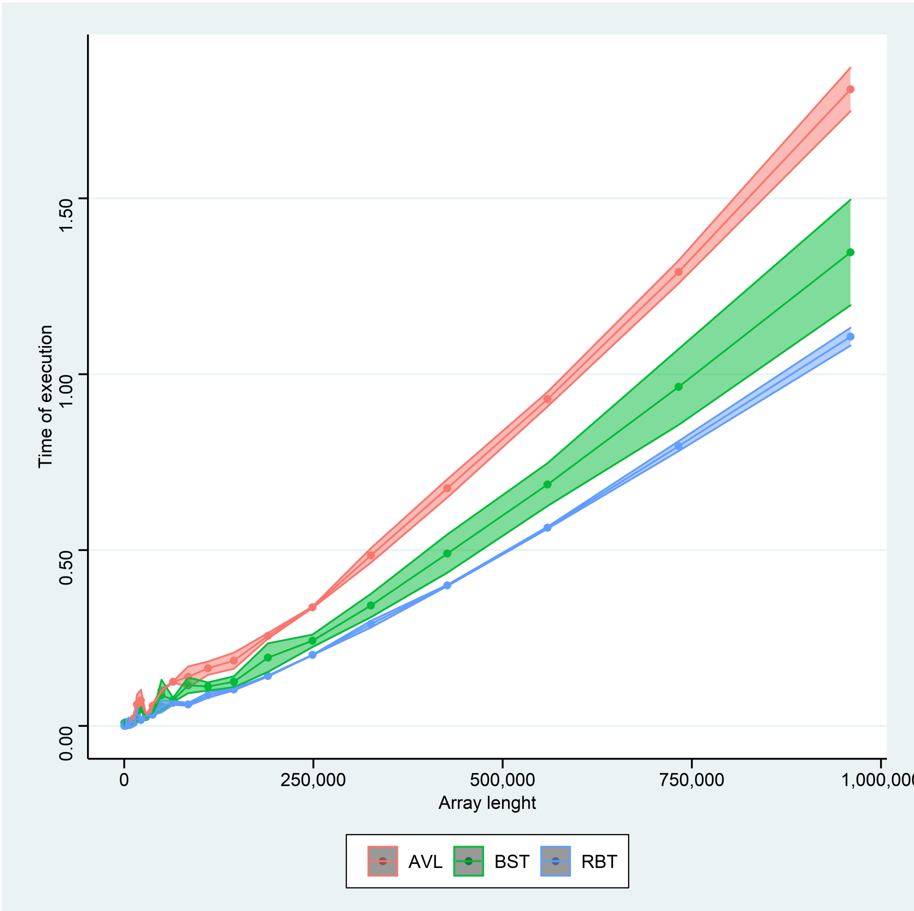

Figura 1: Tempo di esecuzione di n operazioni di inserimento

Le aspettative teoriche sono state principalmente rispettate. Sia gli RBT che gli AVL si sono comportati come ci si aspettava.
Ciò che sorprende di più è l'efficienza dei BST. Questi alberi sono, per input generati casualmente tramite l'algoritmo MT, molto efficienti in quanto non si sbilanciano. Possiamo però notare come la deviazione mediana assoluta sia molto superiore a quella degli altri due alberi, ad indicare come questa efficienza sia molto legata, come già detto, all'input.
La maggiore efficienza dei BST rispetto agli AVL può inoltre essere attribuita all'assenza, per il primo tipo di albero, di procedure di ri-bilanciamento.

5.2 Esecuzione di n operazioni di inserimento con il 10% di elementi ordinati

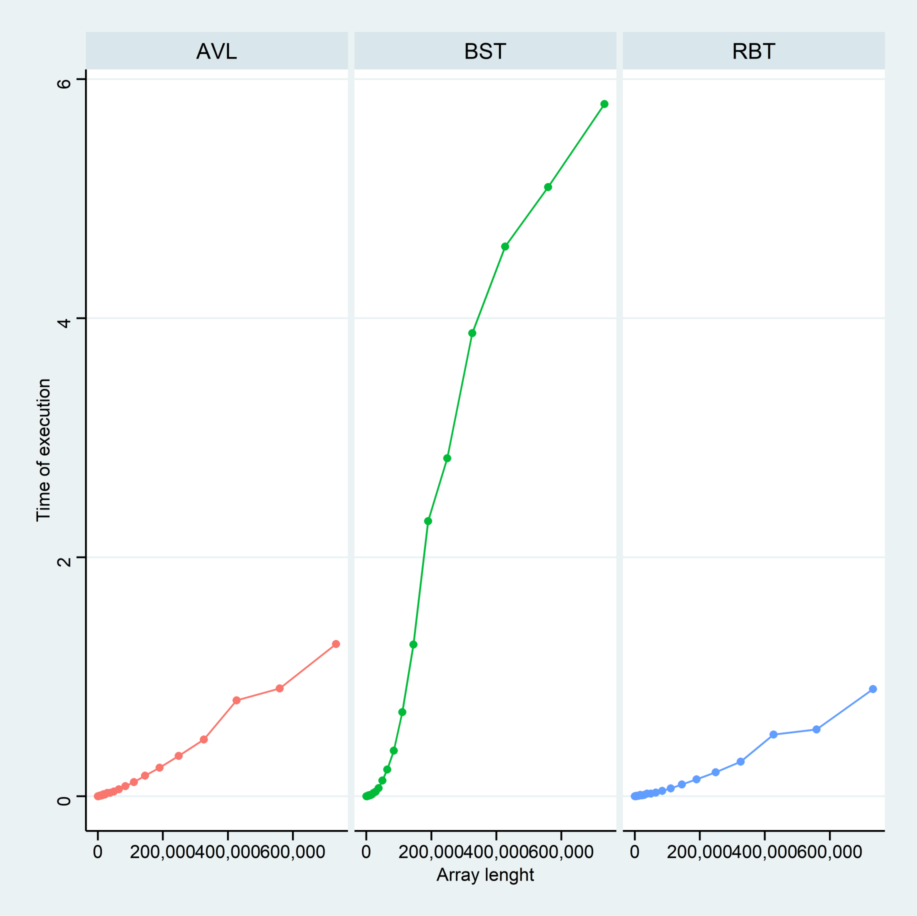

Figura 2: Tempo di esecuzione di n operazioni di inserimento con 10% di elementi ordinati

Anche in questo grafico le aspettative teoriche sono state rispettate. Tutti gli alberi si sono comportati esattamento come predetto.
Possiamo notare come i BST si siano effettivamente rivelati inefficienti rispetto agli altri due alberi. In particolare, si nota molto chiaramente l'andamento lineare nel tempo di esecuzione.
Per quanto riguarda gli altri due alberi, è interessante la maggiore efficienza dei RBT rispetto agli AVL. Come predetto nella sezione 2.2, infatti, i RBT hanno un costo di inserimento minore, dato dal funzionamento della procedura di ri-bilanciamento dell'albero stesso.

5.3 Esecuzione di n operazioni di ricerca

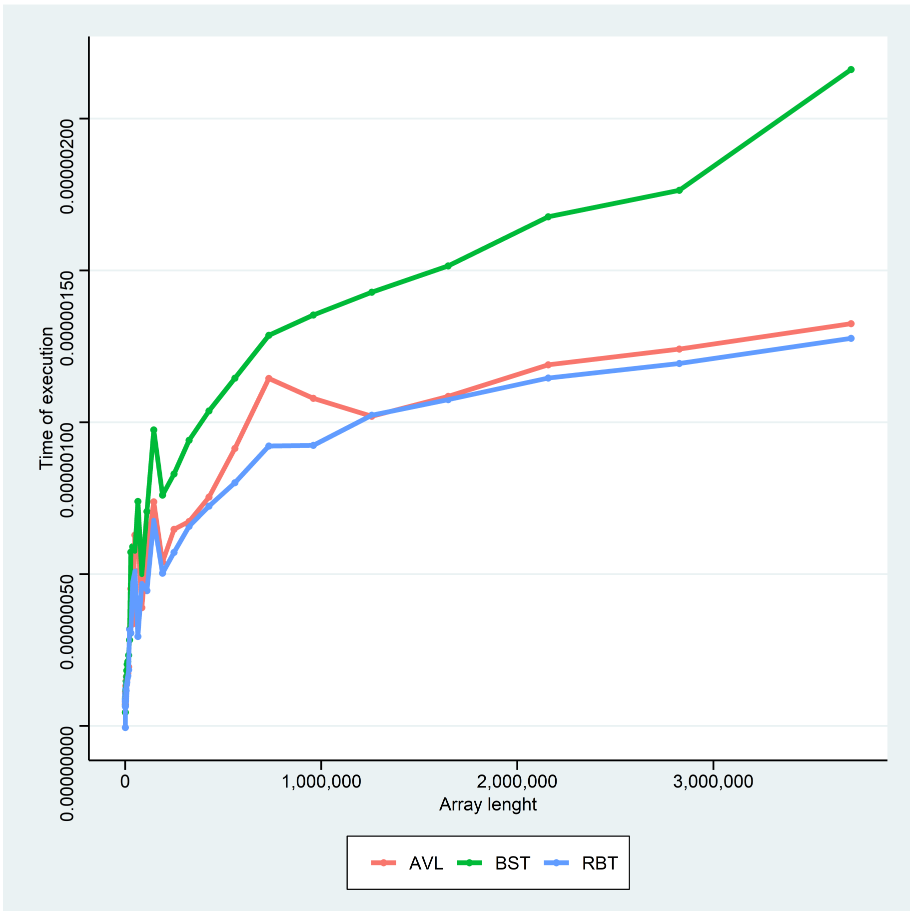

Figura 3: Tempo di esecuzione di n operazioni di ricerca

Nel caso di sole operazioni di ricerca possiamo subito notare come i BST siano meno efficienti degli alberi AVL. Ciò è giustificato dal fatto che gli AVL sono particolarmente efficienti per le operazioni di ricerca in quanto sono effettivamente bilanciati.
Risulta invece piuttosto anomala la loro minor efficienza rispetto ai RBT. Bisogna sottolineare che la differenza fra i due tempi di esecuzione è comunque minima (nell'ordine di 10^{-8}), tuttavia questa rimane costante per svariati valori di n. La congettura riguardo questo anomalia riguarda l'input: essendo questo casuale è possibile che i RBT rimangano abbastanza bilanciati. Ciò tuttavia non spiega il risultato sperimentale ottenuto.

5.4 Esecuzione di n operazioni di ricerca con 10% di elementi ordinati

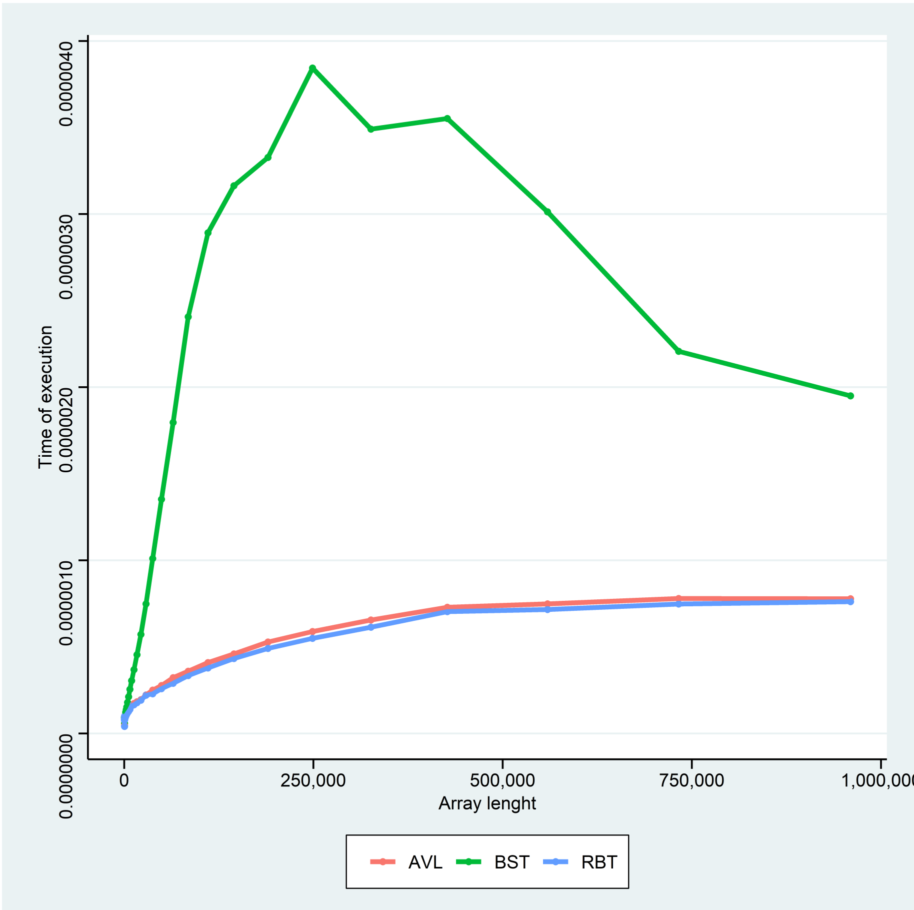

Figura 4: Tempo di esecuzione di n operazioni di ricerca con 10% di elementi ordinati

5.5 Esecuzione di n operazioni di ricerca con 90% di elementi ordinati

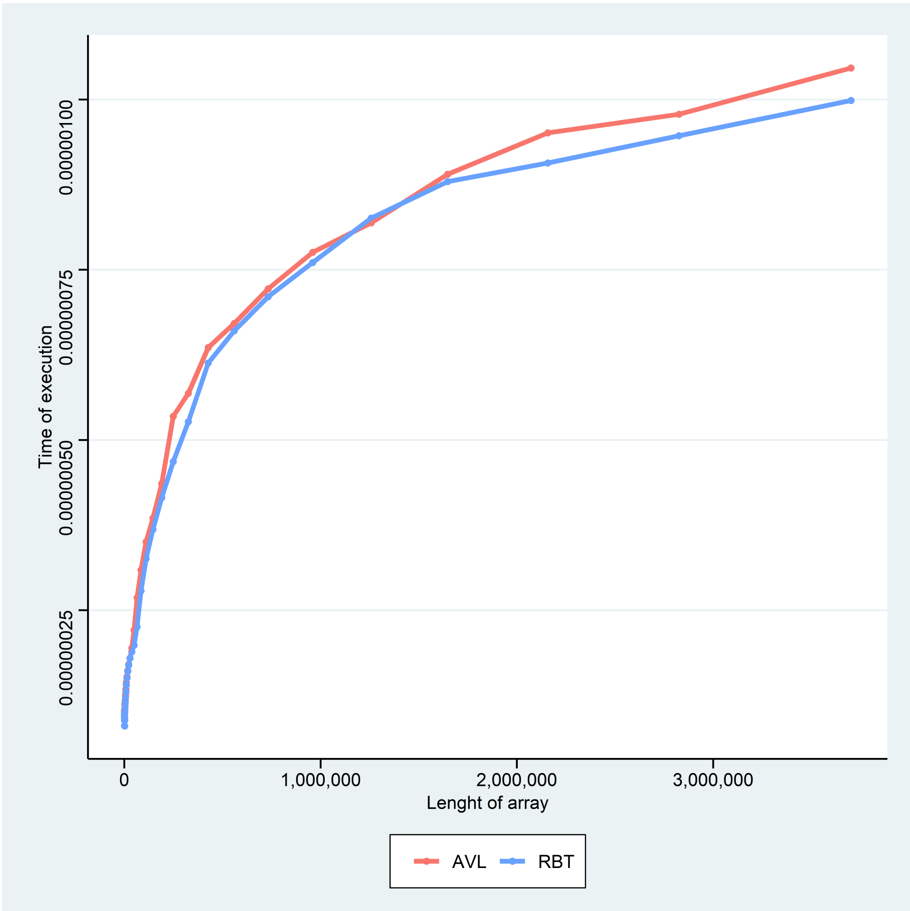

Figura 5: Tempo di esecuzione di n operazioni di ricerca con 90% di elementi ordinati

6 Conclusioni