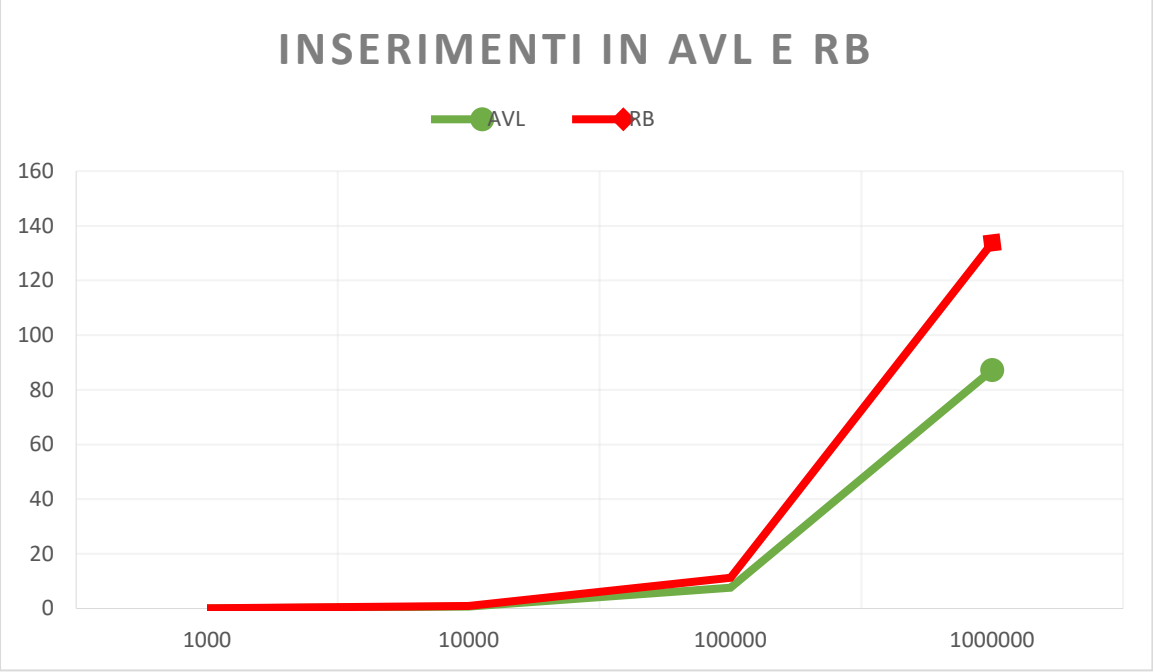
Progetto #2

Nell'implementare il primo esercizio relativo al secondo progetto, abbiamo creato un modulo verifica1.

In questo modulo vi è la funzione test_trees_insertion, la quale, preso in input il numero di elementi, genera una lista casuale di n parole, di lunghezza 10, queste poi sono inserite, secondo chiavi ordinate da 1 a n, rispettivamente, in un albero AVL e RB.

Viene poi stampato, il tempo totale dovuto agli inserimenti nell'albero AVL e quello RB, inoltre, viene stampato il rapporto del tempo impiegato dall'albero RB e quello AVL.

In un campione causale di rispettivamente 1000, 10000, 100000 e 1000000 elementi ordinati otteniamo i seguenti risultati:



n. elementi	1000	10000	100000	1000000
Tempo AVL	0.047	0.657	7.619	87.248
Tempo RB	0.085	0.93	11.275	133.778
Rapporto dei tempi	1.79	1.42	1.48	1.53

Entrambi gli algoritmi di inserimento hanno complessità temporale $O(\log n)$.

Va considerato però, che mentre l'altezza nel caso peggiore di un AVL è $\sim 1.44\ logn$, quella di un RB è $2logn$.

In un inserimento prima dobbiamo cercare il nodo dove inserire la chiave, e il tempo che questa operazione prende dipende dall'altezza dell'albero, poi aggiungiamo il nodo e poi applichiamo operazioni di bilanciamento, le quali ancora una volta richiedono un tempo dipendente dall'altezza dell'albero.

Questo è il motivo per cui, come notato dai precedenti grafici, all'aumentare degli elementi, la differenza di tempo negli inserimenti "si fa sentire".

La ricerca negli AVL, è certamente molto veloce, mentre nei RB la ricerca è meno veloce a causa dell'altezza dell'albero, c'è da dire però, che negli alberi RB l'inserimento potrebbe essere più veloce in quanto il numero di rotazioni da effettuare potrebbe essere inferiore a quello degli alberi AVL.

Asintoticamente dunque, a causa delle differenze di altezze dei due tipi di alberi, e a causa dell'inserimento ordinato di coppie (chiave, valore), anche il numero di rotazioni nell'AVL è costante come quello dei RB, rendendo dunque gli AVL migliori.