Alberi Binari di Ricerca vs Alberi Rosso-Neri

Francesco Baiocchi

Febbraio 2023

Indice

1 Introduzione

Nella presente relazione vengono analizzate le differenze tra gli **Alberi Binari di Ricerca** e gli **Alberi Rosso-Neri** e compresi i vantaggi-svantaggi delle rispettive strutture dati.
Vengono effettuati dei test riguardanti i tempi di inserimento e di ricerca di un elemento sulle strutture sopra indicate. Inoltre, viene anche fornito un confronto tra le altezze al crescere dei nodi.

1.1 Descrizione Strutture dati utilizzate

Un **albero binario di ricerca** è una struttura dati molto utilizzata sui cui è possibile implementare qualsiasi operazione su insiemi dinamici come trovare il massimo, il minimo, fare un inserimento o cancellare un elemento. Le prestazioni di queste operazioni però sono dipendenti **dall'altezza dell'albero stesso** e, se questa è grande, risultano essere poco efficienti.
Gli **alberi rosso-neri**, memorizzando un solo bit in più, quello del colore, tendono a "bilanciare" l'albero accorciando la sua altezza e di conseguenza i tempi d'esecuzione.
Le aspettative pre-test sono quindi quelle di vedere mediamente delle prestazioni migliori da parte degli ARN al crescere del numero dei nodi su algoritmi con complessità computazionale dipendente dall'altezza.

1.2 Specifiche della piattaforma di test

L'insieme dei test è condotto su un elaboratore con le seguenti caratteristiche:

- **Processore** : 1,2 GHz Dual-Core Intel Core m3
- **RAM** : 8 GB 1867 MHz LPDDR3
- **Sistema Operativo** : macOS Ventura 13.2.1

2 Documentazione codice implementato

L'implementazione del progetto, in linguaggio Python, in allegato a questa relazione e raggiungibile al link https://github.com/francescobaio/LabAlg, consente di riprodurre i risultati degli esperimenti sotto presentati.
Nel progetto sono presenti le seguenti classi:

- **ABRNode** : una classe che consente la rappresentazione di un nodo di un albero binario di ricerca con i suoi attributi come chiave,figlio sx e dx e padre.
- **ABR** : la classe che implementa l'albero binario di ricerca con tutti i suoi metodi principali come l'inserimento, cancellazione e ricerca di un elemento.

- **ARNode** : una classe che rappresenta un nodo di un albero rosso-nero estendendo le proprietà di un nodo di un ABR con il bit aggiuntivo del suo colore.
- **ARN** : la classe che implementa l'albero rosso nero che estende le funzionalità dell'ABR tenendo conto delle sue proprietà strutturali per il bilanciamento dell'altezza.

2.1 Classe ABR

Vediamo ora in dettaglio i metodi più interessanti della classe ABR:

- **delete**: permette la cancellazione di un nodo dall'albero, tenendo conto il rispetto della regola di chiave
- **insert**: permette di inserire un nodo nell'albero.
- **findMinimum** e **findMaximum**: restituiscono rispettivamente il nodo con la chiave minima e massima contenuti nell'albero.
- **getHeight**: calcola l'altezza dell'albero e la restituisce come valore di ritorno
- **predecessor** e **successor**: restituiscono rispettivamente il predecessore e il successore del nodo dato in input.

2.2 Classe ARN

Passiamo ora ai metodi della classe ARN di maggior rilevanza:

- **insertFixUp**: permette di fixare le eventuali violazioni, delle regole degli ARN, generate durante l'insert di un elemento nell'albero.
- **leftRotate** e **rightRotate**: procedure utili all'interno dell'insertFixUp per ruotare a sinistra o destra la struttura.

2.3 Test implementati

All'interno del file **test.py** sono presenti vari metodi utili per condurre gli esperimenti sulle strutture.

- **insertTest()** : permette di eseguire il confronto, su vari esperimenti, tra le tempistiche di inserimento dei nodi all'interno delle rispettive strutture, illustrandolo visivamente attraverso la generazione di un grafico.
- **createRandomABR/ARN**: ricevono in ingresso una size, generano un array con elementi interi da 0 fino a $size-1$, randomizzano questa sequenza e creano un ABR o un ARN inserendo via via dei nodi con questi elementi come rispettive chiavi.

- **heightTest**: utilizza la funzione *createRandomABR /ARN* per mostrare attraverso un grafico come, al crescere dei nodi, le altezze degli ARB e ARN crescano in maniera diversa.
- **searchTest**: genera i grafici usati per il confronto nelle tempistiche di ricerca di un nodo all'interno delle due strutture. La chiave del nodo è in ogni esperimento generata in modo randomico nell'intervallo di valori passato a *createRandomABR*.

3 Test

3.1 Test dell'altezza

Come primo test analizziamo come al crescere dei nodi le due strutture si bilancino in maniera differente. Facciamo vedere quindi che l'altezza di un albero rosso-nero sia sempre inferiore rispetto a quella di un albero binario di ricerca. Una volta compreso come le procedure *createRandomABR* e *createRandomARN* sopra documentate agiscano, il test risulta essere una semplice misurazione, al crescere dei nodi, dell'altezza delle strutture generate.
Si parte da un minimo di 100 nodi fino ad un massimo di 20 mila con incremento costante, si generano le strutture in maniera randomica attraverso le procedure sopra citate, e ne viene misurata l'altezza. In figura 1 può essere visionato il diverso andamento di crescita delle altezze delle strutture.
Il grafico presenta al crescere dei nodi una media dei risultati ottenuti su un totale di 50 esperimenti.

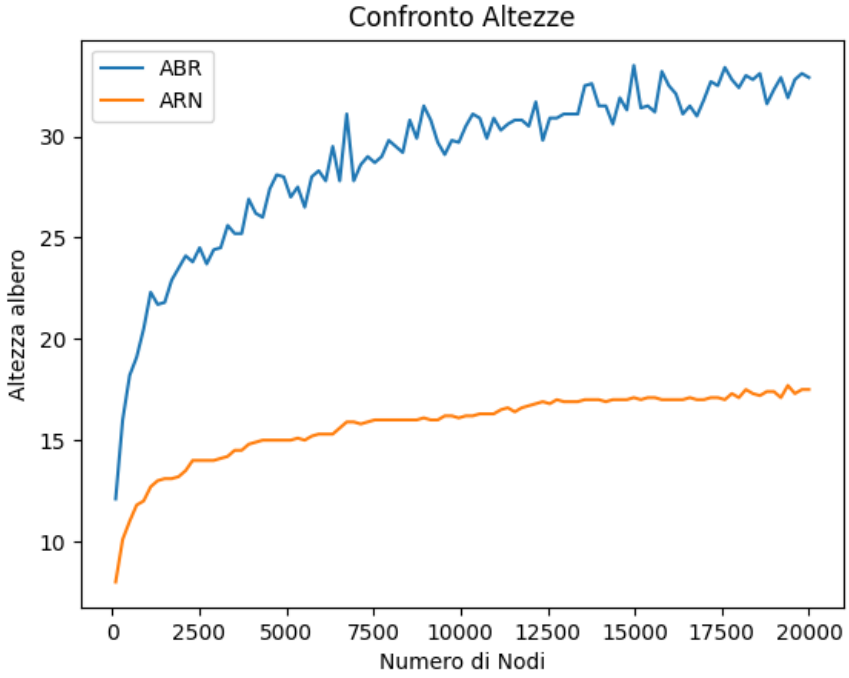

Figura 1: Confronto altezze ABR vs ARN

3.2 Test dell'inserimento

In questa sezione analizziamo il tempo impiegato ,dalle due strutture dati messe a confronto, per inserire i nodi. Al crescere dei nodi, partendo con un numero minimo di 100 e arrivando ad un numero massimo di 20 mila nodi, con un incremento costante, si è potuto verificare quello che si ci aspettava pre-test.
Infatti svolgendo 50 esperimenti diversi e mediando i risultati su questi, si ottiene il grafico in figura 2.
Da questo infatti si evince che, come ci si aspettava, i tempi per l'inserimento di un elemento al crescere dei nodi dell'albero diventano più importanti per gli alberi rosso-neri piuttosto che per gli alberi binari di ricerca. Questo è perchè durante l'inserimento in un albero rosso nero va eseguita la pesante procedura di fixUp che ripristina la validità delle regole della struttura.

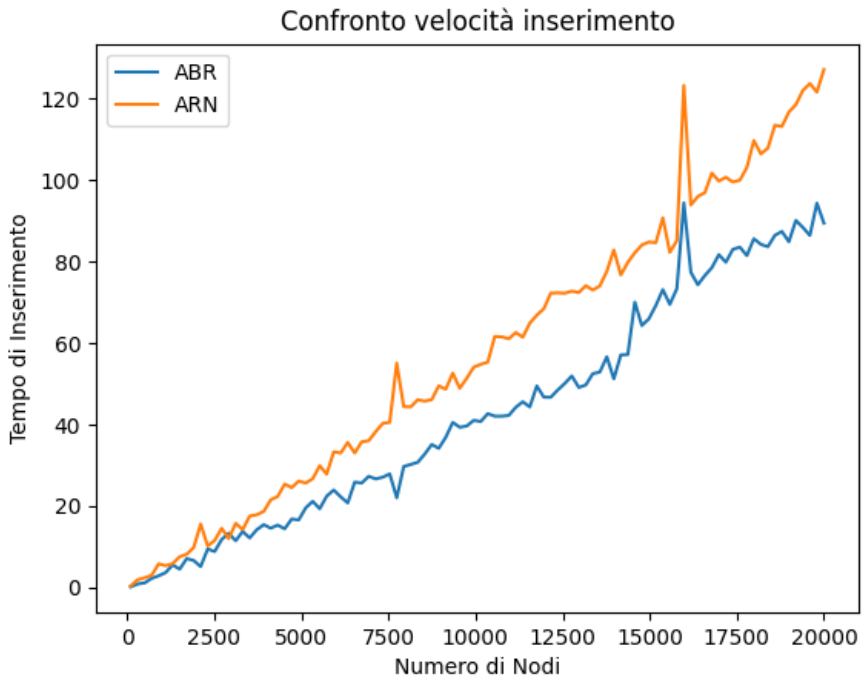

Figura 2: Confronto sul tempo di inserimento ABR vs ARN

3.3 Test della ricerca

Come ultimo test facciamo vedere come in operazioni importanti come la ricerca di un elemento nella struttura, gli alberi rosso-neri siano più efficienti rispetto agli alberi binari di ricerca.
Come nel test dell'inserimento al crescere dei nodi, da 100 a 20 mila con incremento costante, viene mediato su un totale di 50 esperimenti il tempo necessario sulle rispettive strutture per eseguire l'operazione di ricerca. Inoltre allo stesso modo del test di inserimento le strutture sono create con le procedure *randomCreateABR* e *randomCreateARN*.
L'operazione di ricerca viene svolta ogni volta su un numero random nell'intervallo dei valori delle chiavi delle strutture. In figura **??** può essere visionato il loro diverso andamento.
Il grafico presenta al crescere dei nodi una media dei risultati ottenuti su un totale di 50 esperimenti evidenziando come la procedura di ricerca di un elemento sia notevolemente più efficiente negli alberi rosso-neri rispetto agli alberi binari di ricerca.

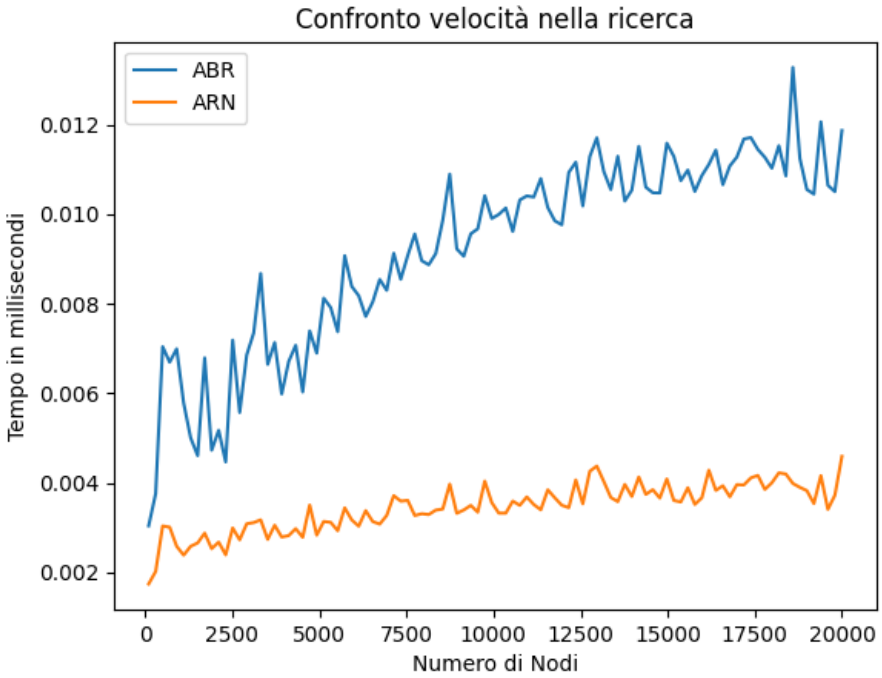

Figura 3: Confronto sul tempo di ricerca ABR vs ARN

4 Conclusioni

Grazie ai test effettuati dunque è possibile affermare che, sebbene nel momento dell'inserimento di un elemento gli alberi rosso-neri siano meno efficienti degli alberi binari di ricerca, i primi mantengano mediamente una struttura più bilanciata. Come conseguenza di questo si ha dunque un risparmio su tutte le operazioni, essendo queste dipendenti computazionalmente dall'altezza dell'albero, come ad esempio la ricerca di un elemento come mostrato nel test dedicato e sopra discusso.