Alberi Binari di Ricerca e Alberi Rosso Neri

Manuel Cecere Palazzo

21 Marzo 2020

Contents

1 Introduzione

Obiettivo di questa relazione è analizzare le differenze tra Alberi Binari di Ricerca e Alberi Rosso Neri. Per fare ciò vengono effettuati su di essi una serie di test, allo scopo di osservare le differenze nei tempi di esecuzione di alcune operazioni fondamentali e nella struttura degli alberi stessi.

2 Teoria delle strutture dati

2.1 Albero Binario di Ricerca

Un *Albero Binario di Ricerca* è una struttura dati in cui è possibile memorizzare valori dinamicamente. I valori sono inseriti in strutture dette *nodi*. Questi sono collegati tramite puntatori ad altri nodi, che possono avere il ruolo di figlio destro, figlio sinistro o padre. Eccezione a questo è il nodo da cui parte l'albero, detto anche *radice*, che non presenta padre.

Proprietà fondamentale dell'*Albero Binario di Ricerca* è il rapporto fra i figli e il padre: il sottoalbero del figlio sinistro di un nodo x contiene soltanto i nodi con valori minori del valore del nodo x , mentre il sottoalbero del figlio destro di un nodo x contiene soltanto i nodi con valori maggiori del valore del nodo x.

Grazie a questa proprietà le operazioni di ricerca, massimo e successore si effettuano in tempo $\Theta(h)$ dove h è l'altezza dell'albero. L'albero tuttavia non ci da garanzie di essere bilanciato. Un inserimento ordinato dei valori farebbe infatti degenerare questo ad una lista , con la conseguenza di avere le operazioni già citate eseguite in tempo $\Theta(n)$ dove n è il numero degli elementi.

2.2 Albero Rosso Nero

Un *Albero Rosso Nero* è un particolare tipo di Albero Binario di Ricerca, caratterizzato da ulteriori proprietà: i suoi nodi sono dotati di un campo *colore*, rosso o nero. Inoltre l'albero è dotato di nodi "sentinelle", nodi vuoti di colore nero. Queste proprietà ci permettono di modificare le operazioni di inserimento e cancellazione, garantendo che l'albero risulti sempre bilanciato con l'altezza $h{=}\Theta(log\ n)$.

2.3 Aspettative

Dati questi presupposti teorici, ci aspettiamo una prestazione migliore da parte dell'albero rosso nero nel caso di inserimento di valori ordinati. Sia nell'altezza di quest'ultimo, sia di conseguenza nel tempo delle operazioni. Nel caso di inserimenti di valori casuali ci aspettiamo prestazioni analoghe.

3 Esperimenti

Testeremo le nostre strutture dati con 4 tipi di esperimenti:

- **Inserimento di valori**. Misuriamo il tempo necessario ad inserire valori sia in modo casuale, fino a 10000 valori, che ordinato, fino a 5000 valori. Per ogni dimensione dell'input effettuiamo una media, su 80 campioni per l'inserimento casuale e su 10 per il caso ordinato.
- **Altezza degli alberi**. Dopo aver inserito fino a 10000 valori, sia nel caso ordinato che in quello casuale, andiamo a misurare l'altezza degli alberi. Nell'inserimento casuale abbiamo anche calcolato una altezza media su 10

campioni per attenuare gli effetti di casi anomali, come un vettore quasi ordinato.

- **Inorder Walk**. Misuriamo il tempo necessario per effettuare un attraversamento Inorder dei nostri alberi. Lo facciamo sia per il caso ordinato, fino a 2000 valori, sia per quello casuale,fino 5000 valori. Calcoliamo una media su 200 campioni per ogni dimensione dell'input.
- **Ricerca valori**. Dopo aver inserito fino a 2000 valori, misuriamo nel caso ordinato il tempo necessario per ricercare 10 valori casuali. Per ogni dimensione dell'input effettuiamo una media su 100 campioni.

Calcolatore utilizzato

- **Processore:** Intel Core i7-7700 HQ CPU 2.80 GHz, 4 core
- **Sistema operativo:** Windows 10 Home 10.0.18362 64 bit
- **Memoria:** 8GB SDRAM DDR4, 240GB SSD
- **Versione Python:** Python 3.7

4 Documentazione del codice

Il codice è composto dalle classi *Node, ABR e ARN* , usate per implementare le strutture dati e le loro operazioni. Abbiamo poi le varie funzioni di testing per implementare gli esperimenti già citati. Vengono utilizzate la funzione default-timer dal modulo timeit dalla libreria standard di Python per registrare i tempi e il modulo pyplot dalla libreria matplotlib per tracciare i grafici.

5 Risultati sperimentali

5.1 Inserimento di valori

5.1.1 Valori casuali

In questo caso i tempi di esecuzione sono simili per le due strutture dati, come previsto. Il maggior tempo impiegato dell'albero rosso nero può essere attribuito alla sua funzione di inserimento più complessa.

5.1.2 Valori ordinati

Come già abbiamo detto, inserendo valori ordinati l'albero binario di ricerca degenera in una lista, portando ad un aumento dei tempi di inserimento.

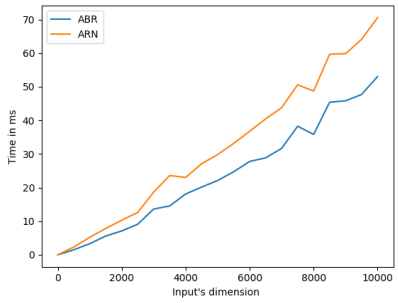

Figure 1: Grafico dei tempi di inserimento di valori casuali

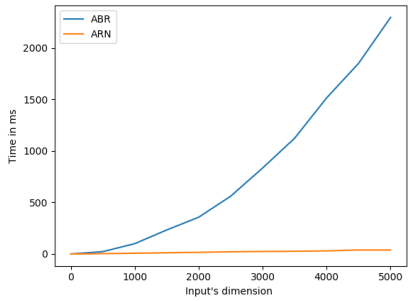

Figure 2: Grafico dei tempi di inserimento di valori ordinati

5.2 Altezze degli alberi

5.2.1 Valori casuali

In questo caso i due alberi presentano lo stesso comportamento asintotico. La minore altezza dell'albero rosso nero è probabilmente dovuta alla funzione di *Fixup*, che assicura una maggiore distribuzione rispetto all'albero binario di ricerca.

5.2.2 Valori ordinati

Nell'inserire valori ordinati, l'albero binario di ricerca degenera in una lista, di conseguenza l'altezza cresce in modo lineare. Qui è più evidente la differenza fondamentale tra i due alberi.

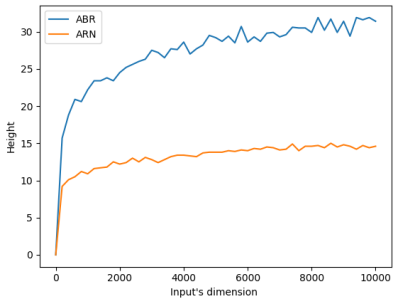

Figure 3: Grafico delle altezze a valori casuali

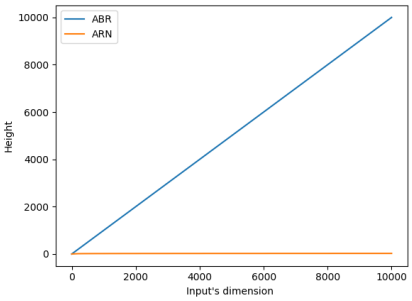

Figure 4: Grafico delle altezze a valori ordinati

5.3 Inorder Walk

5.3.1 Valori Casuali

Nel caso dei valori casuali , il comportamento delle nostre strutture è analogo. Notiamo tuttavia come l'albero rosso nero sembra impiegare sempre un tempo leggermente maggiore. Questo può essere probabilmente dovuto ad una piccola differenza nel codice: Dove nell'albero binario di ricerca si controlla se dei nodi sono vuoti, nell'albero rosso nero si controlla se questi sono dei nodi sentinelle, accedendo all'attributo *self.NIL*. Questi accessi alla memoria, che diventano sempre più numerosi all'aumentare della dimensione dell'input, potrebbero essere la causa dell'aumento dei tempi di esecuzione per l'albero rosso nero.

5.3.2 Valori ordinati

Anche nel caso di valori ordinati, il tempo di esecuzione dei due alberi è simile. Questo tuttavia è comprensibile: l'attraversamento inorder visita necessariamente tutti i nodi dell'albero. Dunque il suo tempo di esecuzione è $\Theta(n)$ sempre, indipendentemente dall'altezza dell'albero.

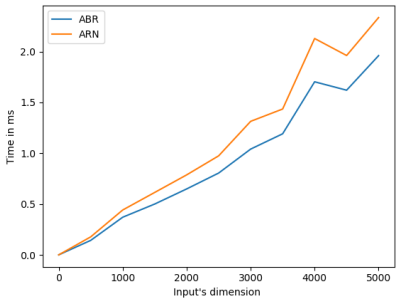

Figure 5: Grafico dei tempi di attraversamento per valori casuali

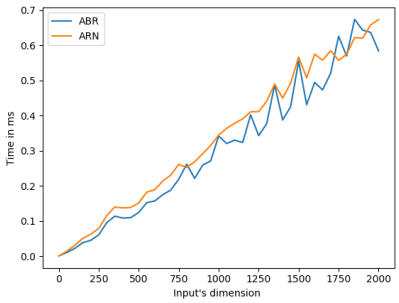

Figure 6: Grafico dei tempi di attraversamento per valori ordinati

5.4 Ricerca valori

Esaminati nel caso ordinato, i risultati sperimentali rientrano nelle previsioni: il tempo di ricerca negli alberi binari è degenerato a lineare, mentre quello negli alberi rosso neri rimane logaritmico.

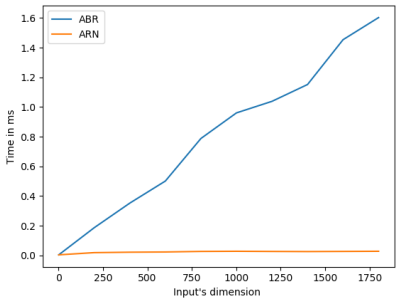

Figure 7: Grafico dei tempi di ricerca per valori ordinati

6 Conclusione

In generale l'esito degli esperimenti rientra nelle nostre aspettative.

Nel caso di valori ordinati, l'albero rosso nero ha tempi di esecuzione migliori rispetto all'albero binario di ricerca, per tutte le operazioni che dipendono dall'altezza dell'albero.

Nel caso di valori casuali, l'albero rosso nero è risultato essere leggermente più lento, a causa delle sue operazioni più complesse rispetto all'albero binario di ricerca. Un trade-off necessario tuttavia per ottenere un albero più distribuito, come notiamo in figura 3.