Valutazione performance nella ricerca di elementi in Alberi Rosso Neri e Alberi Binari di Ricerca

Relazione esercizio 2

Laboratorio di Algoritmi e Strutture Dati

Gemma Vaggelli
6348717

Ingegneria Informatica
Università degli studi di Firenze

1 Introduzione

Tra le strutture dati approfondite durante corso, nel seguente esercizio andrò a descrivere i risultati ottenuti dal confronto delle performance nella ricerca di un determinato elemento all'interno di Alberi Binari di Ricerca e Alberi Rosso Neri. Vedremo quale delle due è più veloce per certe operazioni svolte su di esse, confrontando i risultati ottenuti con la teoria. Ci aspettiamo di vedere che, nel caso di una ricerca di un elemento randomico all'interno di un albero, la struttura degli Alberi Rosso Neri permette di raggiungere dei tempi di ricerca minori.

2 Caratteristiche

2.1 Caratteristiche Alberi Binari di Ricerca

Un albero binario di ricerca, o ABR, è una struttura dati collegata dove ogni nodo è un oggetto.
T.root punta alla radice dell'albero.
Ogni nodo è caratterizzato da:

- un campo **chiave** (o eventualmente dati satelliti)
- un campo **left** (che punta al figlio sinistro)
- un campo **right** (che punta al figlio destro)
- un campo **p** (che punta al padre)

Gli Alberi Binari di Ricerca hanno 2 proprietà :

1. se y è nel sottoalbero sinistro di x, allora **y.key** $\leq$ **x.key**
2. se y è nel sottoalbero destro di x, allora **y.key** $\geq$ **x.key**

2.2 Caratteristiche Alberi Rosso Neri

Un albero rosso-nero, o ARN, è un albero binario di ricerca particolare che risulta molto efficiente anche nei casi peggiori, dove resta comunque bilanciato.
Ogni nodo x ha un attributo x.color booleano, che può essere rosso o nero.
Un Albero Rosso Nero ha le seguenti caratteristiche:

- le foglie vuote (NIL) sono **nere**

- ha una sentinella, T.NIL, per le foglie di T
- il padre della radice è T.NIL
- non interessa key in T.NIL

Ha inoltre 5 proprietà fondamentali:

1. Ogni nodo o è **rosso** o è **nero**
2. La radice è **nera**
3. Ogni foglia (T.NIL) è **nera**
4. Se un nodo è **rosso** entrambi i suoi figli sono **neri** (cioè non posso avere due rossi consecutivi lungo un cammino da radice a foglia)
5. Tutti i cammini da ogni nodo alle sue foglie contengono lo stesso numero di nodi **neri**

3 Prestazioni

3.1 Prestazioni Alberi Binari di Ricerca

E' molto efficiente per eseguire operazioni come inserimento, cancellazione e ricerca di un elemento. Sappiamo dalla teoria che le operazioni di base nell' albero binario di ricerca impiega un tempo pari a O(h), con h= altezza dell' albero, quindi anche l' inserimento e la cancellazione.

3.2 Prestazioni Alberi Rosso Neri

Grazie alla sua struttura costantemente bilanciata,in un albero rosso nero le operazioni di:

- Min
- Max
- Successor
- Predecessor

- Search
- Inserimento

vengono eseguite in tempo $O(lgn)$ mentre nell'ABR impiegavano $O(h)$ Andiamo adesso a vedere perché l'albero rosso nero è più performante degli ABR.

4 Documentazione

Durante un inserimento in un ABR l'elemento viene inserito con la possibilità di sbilanciare l'albero, ciò significa che se vengono inseriti elementi con .key crescente(o decrescente) a ogni inserimento, l'ABR potrebbe degenerare in un lista. Questo vuol dire anche che l'altezza sarà quindi coincidente con il numero di nodi e di conseguenza si ha un tempo di esecuzione, per le operazioni sopra descritte, di $O(n)$.Nell'ARN a ogni inserimento si ha un FIX-UP che si occupa di bilanciare l'albero. Questo fa si che l'albero non possa degenerare in alcun modo in una lista.
Andiamo a vedere per via sperimentale se questo fatto è vero, svolgendo 2 tipi di test:

1. Ricerca di una chiave non appartenente a un **albero binario di ricerca** e a un **albero rosso-nero** creato con **valori random**
2. Ricerca di una chiave non appartenente a un **albero binario di ricerca** e a un **albero rosso-nero** creato con **valori crescenti**

Andiamo a generare 500 ABR e 500 ARN con dimensione crescente e chiavi dei nodi random. Dopodiché inseriamo in un grafico la media dei tempi impiegati per un numero prefissato di search (in questo caso 20) di un elemento non esistente all'interno dell'albero in entrambe le strutture. I due alberi vengono creati con gli stessi elementi, cioè una volta generata una chiave, viene creato un nodo sia per l'ABR che per l'ARN con quella chiave e viene inserito. In questo modo si possono confrontare i tempi sulle due strutture, create con gli stessi valori. I risultati nella tabella e nel grafico sono le medie sui 20 test svolti.
In seguito, andiamo a generare 500 ABR e 500 ARN con dimensione crescente e chiavi dei nodi crescenti. Dopodiché inseriamo in un grafico i tempi impiegati per una ricerca di un elemento non presente negli alberi, per entrambe le strutture. In particolare prendiamo il massimo numero intero rappresentabile dalla macchina.
Come nell'esperimento precedente, andiamo a prendere una media dei dati relativi a più run della ricerca dei nodi.

Hardware I test sono stati eseguiti su un PC Desktop con sistema operativo Windows 10 Home a 64 bit, un processore Intel Core i5-7200U con 8Gb di RAM e Visual Studio Code come IDE.

5 Prestazioni risultati sperimentali

5.1 Prestazioni risultati sperimentali ABR e ARN randomizzati

Caso Random		
Dimensione	Tempo per ricerca ABR	Tempo per ricerca ARN
100	1.120e-05	1.164e-05
200	1.341e-05	1.453e-05
300	1.232e-05	1.430e-05
400	1.179e-05	1.477e-05
500	1.218e-05	1.382e-05

5.2 Prestazioni risultati sperimentali con inserimento di nodi con chiavi crescenti

Caso Peggiore ABR		
Dimensione	Tempo per ricerca ABR	Tempo per ricerca ARN
100	1.153e-04	1.495e-05
200	2.356e-04	1.851e-05
300	3.680e-04	1.998e-05
400	4.839e-04	2.103e-05
500	5.737e-04	3.050e-05

5.3 Grafici

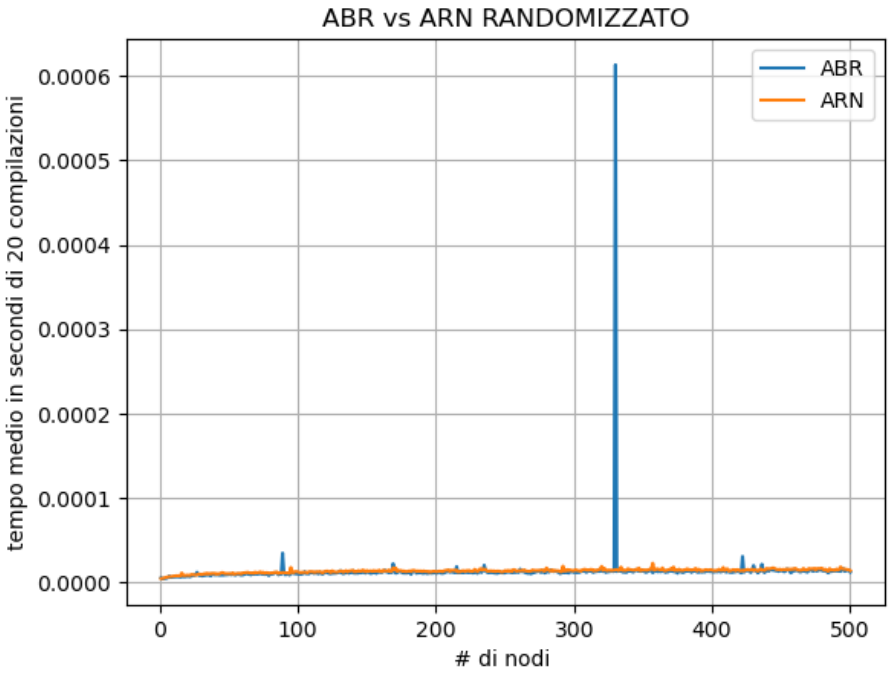

Figura 1: ABR vs ARN con dimensione crescente e chiavi random

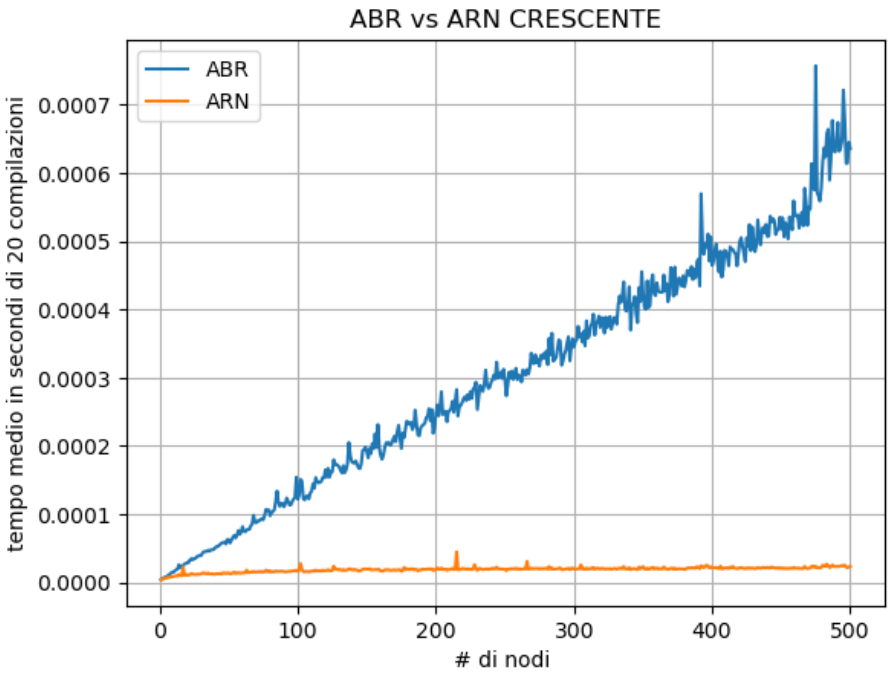

Figura 2: ABR vs ARN con dimensione crescente e chiavi crescenti

6 Analisi risultati sperimentali

6.1 ABR vs ARN randomizzati

Come si può notare dalla Figura 1, l'andamento al crescere della dimensione è quello aspettato.
La ricerca in ABR è $O(h)$ dove h è l'altezza dell'albero, che nel caso random è lgn.
La ricerca in ARN è $O(lgn)$. Siccome stiamo scorrendo tutti i possibili casi poiché si svolgono ricerche di elementi non appartenenti agli alberi, il tempo è esattamente lgn.
Quindi per via sperimentale ci aspettiamo che i tempi di ricerca abbiano all'incirca lo stesso andamento. Questo fatto avviene, dal grafico possiamo dedurre l'andamento logaritmico del grafico.

6.2 ABR vs ARN: con inserimento di nodi con chiavi crescenti

Come si può notare dala Figura 2 l'andamento al crescere della dimensione è quello aspettato.
La ricerca in ABR è $O(h)$ dove h è l'altezza dell'albero, in questo caso h = n..
La ricerca in ARN è $O(lgn)$.
Ci aspettiamo quindi che i tempi di ricerca abbiano un andamento molto diverso: uno lineare, uno logaritmico. Questo perché ci siamo posti nel caso peggiore, in cui l'algoritmo di ricerca scorre tutti i nodi presenti nel cammino dalla radice alla foglia con la chiave più grande. Come previsto, l'ABR risulta più lento per via della sua degenerazione in una lista.L'ARN mantiene come previsto un un andamento logaritmico per la ricerca di un elemento anche in questo caso.

7 Conclusione

Come visto dagli esperimenti l'albero rosso nero risulta una struttura più adatta a svolgere ricerche all'interno di essa, grazie al suo bilanciamento dato dal fix-up al momento dell'inserimento di un nuovo elemento.