UNIVERSITA’ DEGLI STUDI DI UDINE

Progetto di Algoritmi e Strutture Dati

(Seconda Parte)

Autori:

Dalla Torre Josè Zito
143216
143216@spes.uniud.it

Tracanelli Alessandra
143342
143342@spes.uniud.it

Professori:

Piazza Carla
Puppis Gabriele

Indice

1 Il problema

1.1 Enunciato

Per i tre tipi diversi di alberi binari di ricerca (semplici, di tipo AVL e Red-Black) implementare le operazioni di ricerca ed inserimento, assumendo che ogni nodo di un albero binario di ricerca contenga una chiave numerica (di tipo intero) e un valore alfanumerico (di tipo stringa).

2 Alberi binari di ricerca semplici

2.1 Significato

Un albero binario di questo tipo è una struttura dati che soddisfa la seguente proprietà: se X è un nodo e Y è un qualsiasi nodo nel sottoalbero sinistro di X, allora key[y] ≤ key[x]; inoltre se Y è un qualsiasi nodo nel sottoalbero destro di X allora key[y] ≥ key[x].

2.2 Ricerca

In un normale albero binario, per cercare un elemento, si deve attraversare l'intero albero (nel caso peggiore). Ma in un BST, questo procedimento può essere ottimizzato, vediamo come.

Si supponga appunto di dover cercare un elemento in un BST. Quando lo confrontiamo con la radice dell'albero, ci si possono presentare tre casi:

1. item == root.val: terminiamo la ricerca quando l'elemento viene trovato;
2. item > root.val: controlliamo solo il sottoalbero destro poiché tutti i valori presenti nel sottoalbero sinistro sono sicuramente minori di root.val;
3. item < root.val: qui controlliamo solo il sottoalbero sinistro dal momento che tutti i valori nel sottoalbero di destra sono maggiori di root.val.

In questo modo si va così a ridurre la complessità del tempo di ricerca in larga misura, visto che dobbiamo solo analizzare un sottostruttura e rifiutare l'altra, risparmiandoci la fatica di confrontare una serie di valori.
In particolare nel caso peggiore si ha una complessità di $\Theta(n)$, $O(\log n)$ nel caso medio.

2.3 Inserimento

L'algoritmo Inserimento (T, z) ha un comportamento molto simile alla ricerca.
In breve cerca la corretta posizione di z nell'albero identificando così il nodo y che diventerà padre di z. Infine, attacca z come figlio sinistro/destro di y in modo che sia verificata la proprietà tipica degli alberi binari di ricerca.
Anche qui, come per la ricerca, si ha una complessità di $\Theta(n)$ nel caso peggiore, mentre $O(\log n)$ nel caso medio.

3 Alberi binari di ricerca di tipo AVL

3.1 Significato

Per poter comprendere la definizione di un albero binario di ricerca di questa tipologia, viene fornita la spiegazione di che cosa sia il *coefficiente di bilanciamento*, ovvero la differenza tra le altezze rispettivamente del sottoalbero sinistro e del sottoalbero destro di un nodo:

$$b(n) = h(n.l) - h(n.r)$$

dove n è il nodo di cui si vuole calcolare il coefficiente e n.l e n.r sono rispettivamente il figlio sinistro e il figlio destro di n; b(n) può quindi assumere solo valori interi sia positivi che negativi.
L'albero AVL è un albero binario di ricerca bilanciato in cui il coefficiente di bilanciamento vale 0, 1 oppure – 1.

3.2 Ricerca

La ricerca di un nodo in un albero AVL è la stessa di un qualsiasi BST.
Si inizia dalla radice dell'albero e si confronta la chiave con il valore del nodo. Se la chiave è uguale al valore, restituisce il nodo. Se la chiave è maggiore, si cerca dal figlio di destra, altrimenti si continua la ricerca dal figlio di sinistra. Qui la complessità dell'algoritmo è, sia nel caso peggiore che in quello medio, O(log n).

3.3 Inserimento

Il primo passo dell'inserimento di un elemento in un albero AVL funziona come in quello non bilanciato, si cerca il posto dove deve andare. Se la ricerca finisce su un nodo contenente l'elemento da inserire, l'inserimento è terminato, mentre se finisce in una foglia, la si sostituisce con un nodo contenente l'elemento da inserire. Dopo questa operazione, il giusto bilanciamento dell'albero non è più garantito. L'algoritmo, quindi, aggiorna i coefficienti di bilanciamento e controlla se sul percorso dal nuovo nodo alla radice ci siano nodi dove la condizione di bilanciamento non è soddisfatta. In questo caso viene applicata una serie di rotazioni che ripristina tale invariante. Complessità: O (log n) in tutti i casi.

4 Alberi binari di ricerca di tipo Red-Black

4.1 Significato

Un albero rosso-nero (RBT) è un albero binario di ricerca in cui ad ogni nodo associamo un colore, che può essere rosso o nero. Inoltre un RB tree è una struttura dati che soddisfa le seguenti proprietà: - ogni nodo è rosso o nero;

- la radice è nera;
- ogni foglia è nera;
- se un nodo è rosso, entrambi i suoi figli devono essere neri;
- per ogni nodo n, tutti i percorsi che vanno da n alle foglie sue discendenti contengono lo stesso numero di nodi neri.

4.2 Ricerca

Viene eseguita una ricerca binaria sui record nel nodo corrente.
Se viene trovato un record con la chiave di ricerca, viene restituito quel record.
Se il nodo corrente è un nodo foglia e la chiave non viene trovata, segnala una ricerca non riuscita. Altrimenti si segue il ramo appropriato e si ripete il processo. Complessità: O(log n) in tutti i casi.

4.3 Inserimento

La procedura di inserimento di un elemento nell'albero Red-Black
richiede tempo O(log n) in ogni caso.
In primo luogo si alloca il nodo x contenente la chiave k e si inserisce x nell'albero usando la normale procedura insert per alberi di ricerca binaria. Il nodo inserito è colorato red. In questo tipo di operazione c'è una proprietà che può essere violata: quella che afferma che se un nodo è red allora entrambi i figli sono black. Ovviamente dipende da dove viene inserito il nodo.

Aspettative:

Prima di analizzare i grafici risultanti dal calcolo dei tempi sui diversi algoritmi implementati, sicuramente ci aspettiamo, vista la complessità dei bst maggiore rispetto a quella dei rbt e avl , che ci sia una notevole inefficienza da parte dei primi. Per quanto riguarda gli avl, questi tendono ad essere più bilanciati rispetto ai rbt. Quindi compatibilmente con questo, ci aspettiamo di trovare un tempo di ricerca maggiore per i rbt rispetto agli avl; è per questo, infatti, che l'uso degli avl è fortemente consigliato per applicazioni di alta intensità di ricerca. C'è un caso però in cui è meglio usare i rbt ovvero quando ci si trova di fronte all'inserimento casuale al costo inferiore della rotazione.

5 Calcolo dei tempi

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 contenente 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; rappresenta infatti il tempo assoluto dell'orologio trascorso da un punto fisso arbitrario nel passato. Non è influenzato dalle modifiche all'orologio del sistema.

Per ottenere una misurazione della risoluzione migliore si è utilizzata una campionatura di valori che vanno da 1000 a 1000000.

5 Grafici

GRAFICO IN SCALA LINEARE

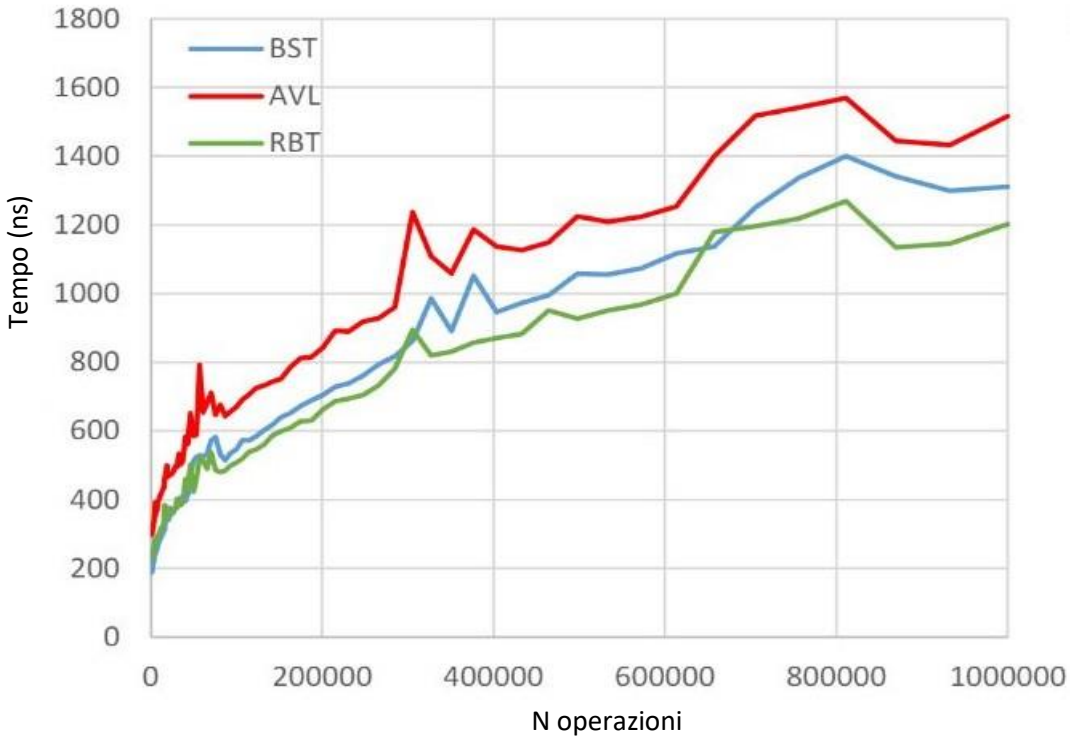
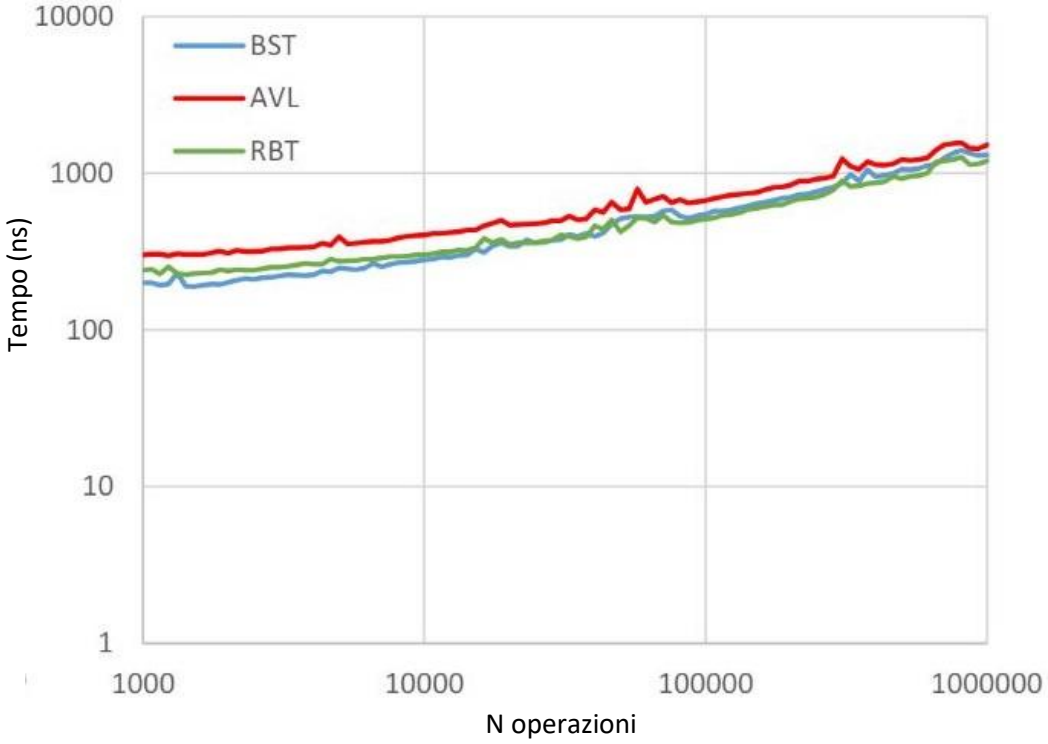


GRAFICO IN SCALA DOPPIAMENTE LOGARITMICA



6 Considerazioni complessive

Analizzando i grafici ottenuti siamo in grado di fare le opportune considerazioni finali sulle tre strutture dati che abbiamo dovuto implementare.

Si può vedere fin da subito come la crescita dei tempi dei tempi di esecuzione nel grafico in scala lineare abbia un andamento pressoché algoritmico per tutti e tre gli algoritmi e questo è dovuto al fatto che per tutti e tre gli alberi la complessità asintotica sia nel caso della ricerca che dell'inserimento è O(log n) nel caso medio. I BST comunque sono quelli che hanno un comportamento più efficiente così come quello dei RBT, il discorso non vale però per gli AVL che hanno raggiunto tempi di esecuzione leggermente più elevati. Sull'efficienza dei BST ovviamente bisogna aggiungere una nota dicendo che sì hanno dimostrato una buna efficienza, ma solo nel caso di input molto causali. Se introduciamo invece input leggermente ordinato, bisogna aspettarsi un incremento sicuro dei tempi.

Infine l'analisi del grafico in scala doppiamente logaritmica, ci va a confermare le affermazioni fatte precedentemente: i BST e i RBT presentano un comportamento molto simile e sicuramente più efficiente rispetto a quello degli AVL.