Esercitazione 8
Binary Search Trees

Corso di Fondamenti di Informatica II

BIAR2 (Ing. Informatica e Automatica) e BSIR2 (Ing. dei Sistemi)

A.A. 2010/2011

17 dicembre 2010

Sommario

Scopo di questa esercitazione è implementare e sperimentare metodi di ricerca basati su Binary Search Trees.

1 Binary Search Trees

Un albero binario di ricerca (Binary Search Tree o BST) è un albero binario che memorizza chiavi (o entry chiave-valore) nei suoi nodi interni e che soddisfa la proprietà descritta nel seguito.

Definizione (Proprietà)**.** *In un BST, dati tre nodi u, v e w tali che u è nel sottoalbero sinistro di v e w è nel sottoalbero destro di v, allora vale la seguente:*

$$key(u) \leq key(v) \leq key(w) \tag{1}$$

L'attraversamento *in ordine* (un nodo è visitato dopo il suo sottoalbero sinistro e prima del suo sottoalbero destro) di un albero binario di ricerca, di conseguenza, visita le chiavi in ordine non decrescente. Un albero binario di ricerca può inoltre rappresentare un dizionario o una mappa. Nei dizionari, a differenza delle mappe, ogni chiave può essere associata a più entry (come in un "reale" dizionario ogni parola ha più significati).

Programma Java. Si vogliono implementare le due classi Java `MyBSTree<E>` e `MyBSNode<E>` di tipo generico che rappresentano rispettivamente un albero binario di ricerca contenente entry chiave-valore con chiavi di tipo `int` ed un suo generico nodo. La classe `MyBSTree<E>` dovrà implementare l' interfaccia `BSTree<E>` (fornita nel materiale di supporto) mentre la classe `MyBSNode<E>` dovrà implementare l'interfaccia `BSPosition<E>` (anch'essa disponibile nel materiale di supporto).

Specifiche. La classe `MyBSTree<E>` deve rappresentare una mappa, e quindi l'inserimento va gestito di conseguenza. Il metodo di rimozione implementato nella classe `MyBSTree<E>` deve sostituire il nodo rimosso con il suo *successore* nella visita in ordine.

La ricerca, naturalmente, non deve visitare tutto l'albero ma seguire lo schema della ricerca binaria.

2 Test di correttezza

Dopo aver realizzato le classi per rappresentare l'albero occorre effettuare dei test per vedere se i dati vengono inseriti in maniera corretta.

Programma Java. Si vuole realizzare una classe `MyTest` con un main di test in cui creare un albero binario di ricerca con valori di tipo `String`, inserire, cercare e rimuovere alcuni valori di prova, e poi stamparlo con il metodo stampa della classe `BSTreeUtil` (fornita nel materiale di supporto). Un possibile main da estendere è il seguente.

```
public static void main(String[] args)
```

```
BSTree<String> t              = new MyBSTree<String>();
BSPosition<String> a          = t.insert(1, "Nero");
BSPosition<String> b          = t.insert(2, "Grigio");
BSPosition<String> c          = t.insert(3, "Bianco");
TreeUtil.stampa(t);
remove(2);
TreeUtil.stampa(t);
BSPosition<String> d          = t.find(3);
String val                    = d.getElement();
System.out.println("Trovato " + d.getKey() + " -> " + val);
```

3 Sperimentazione

Data la soluzione dell'Esercizio 1 si vuole implementare un ambiente di sperimentazione per misurare le prestazioni dell'albero binario di ricerca.

Primo livello. Le funzionalità desisderate sono:

- **Calcolo di una metrica.** La metrica usata è un *coefficiente di bilanciamento* $\in [0, n-1]$ che definiamo come la differenza, date le profondità delle foglie, tra la maggiore e la minore.
- **Test su dataset.** Il dataset, in questo caso, è costituito da sequenze significative di inserimento e cancellazione. Un dataset significativo fa variare la misura della metrica il più possibile.

Secondo livello. Le funzionalità desisderate sono:

- **Calcolo di una metrica.** La metrica usata è un *coefficiente di ottimizzazione* che definiamo come il numero di nodi visitati per eseguire una sequenza di ricerca.
- **Test su dataset.** Il dataset, in questo caso, è costituito da diversi alberi contenenti gli stessi elementi ma con coefficienti di bilanciamento diversi[1] e da sequenze di ricerca.

Programma Java. Estendere la classe `MyTest` con i seguenti metodi.

```
public static int coeffBilanciamento(BSTree<String> t)
```

```
public static int coeffOttimizzazione(BSTree<String> t, List<
    Integer> seq)
```

[1] I diversi alberi possono essere costruiti facilmente variando l'ordine degli insermenti in fase di popolamento.

Testing. Eseguire, modificando opportunamente il metodo main implementato nell'Esercizio 2, i test e trovare:

1. una sequenza di inserimento e cancellazione che minimizza il coefficiente di bilanciamento

2. una sequenza di inserimento e cancellazione che massimizza il coefficiente di bilanciamento

3. un albero che, data una sequenza di ricerca, ha basso coefficiente di bilanciamento ma alto coefficiente di ottimizzazione

4. un albero che, data una sequenza di ricerca, ha alto coefficiente di bilanciamento ma basso coefficiente di ottimizzazione

Com'è fatta, in generale, una "buona" sequenza di inserimento se si conosce quale sarà la sequenza di ricerca?

Suggerimento. Le sequenze di ricerca non necessariamente contengono chiavi distinte.

4 Varianti

Si vogliono implementare le seguenti varianti delle specifiche viste fin ora.

- **Rimozione.** Il metodo di rimozione deve sostituire il nodo rimosso con il suo *predecessore*, piuttosto che il successore, nella visita in ordine.

- **Dizionario.** Si vuole rappresentare un dizionario, piutosto che una mappa, con l'albero binario di ricerca.

Suggerimento. La soluzione alla variante *dizionario* richiede di modificare l'interfaccia `BSTree<E>` in un punto, quale?