Esercitazione 5
Alberi binari

Corso di Fondamenti di Informatica II

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

A.A. 2010/2011

25 Novembre 2010

Sommario

Scopo della esercitazione è quello di realizzare una struttura dati per gestire gli alberi biari ed implementare semplici algoritmi di visita.

1 Rappresentazione di un albero binario

Un *albero binario* è una struttura dati che permette di rappresentare un albero in cui ad ogni nodo è associato un valore e che ha al più 2 nodi figli. Ogni nodo ha un riferimento al genitore, un campo *elemento* che contiene un valore (del tipo appropriato), uno al figlio destro e uno al figlio sinistro. Graficamente, il generico nodo dell'albero è rappresentato in Figura 1.

Ci ri siferisca per i dettagli al libro adottato nel "Corso Fondamenti di Informatica 2 (Algoritmi) - Fabrizio D'Amore" [1].

Programma Java. Si vogliono implementare le due classi Java `MyBTree<E>` e `MyBNode<E>` di tipo generico che rappresentano rispettivamente un albero binario ed un suo generico nodo. La classe `MyBTree<E>` dovrà implementare l' interfaccia `BTree<E>` (fornita nel materiale di supporto) mentre la classe `MyBNode<E>` dovra' implementare l'interfaccia `BPosition<E>` (anch'essa disponibile nel materiale di supporto).

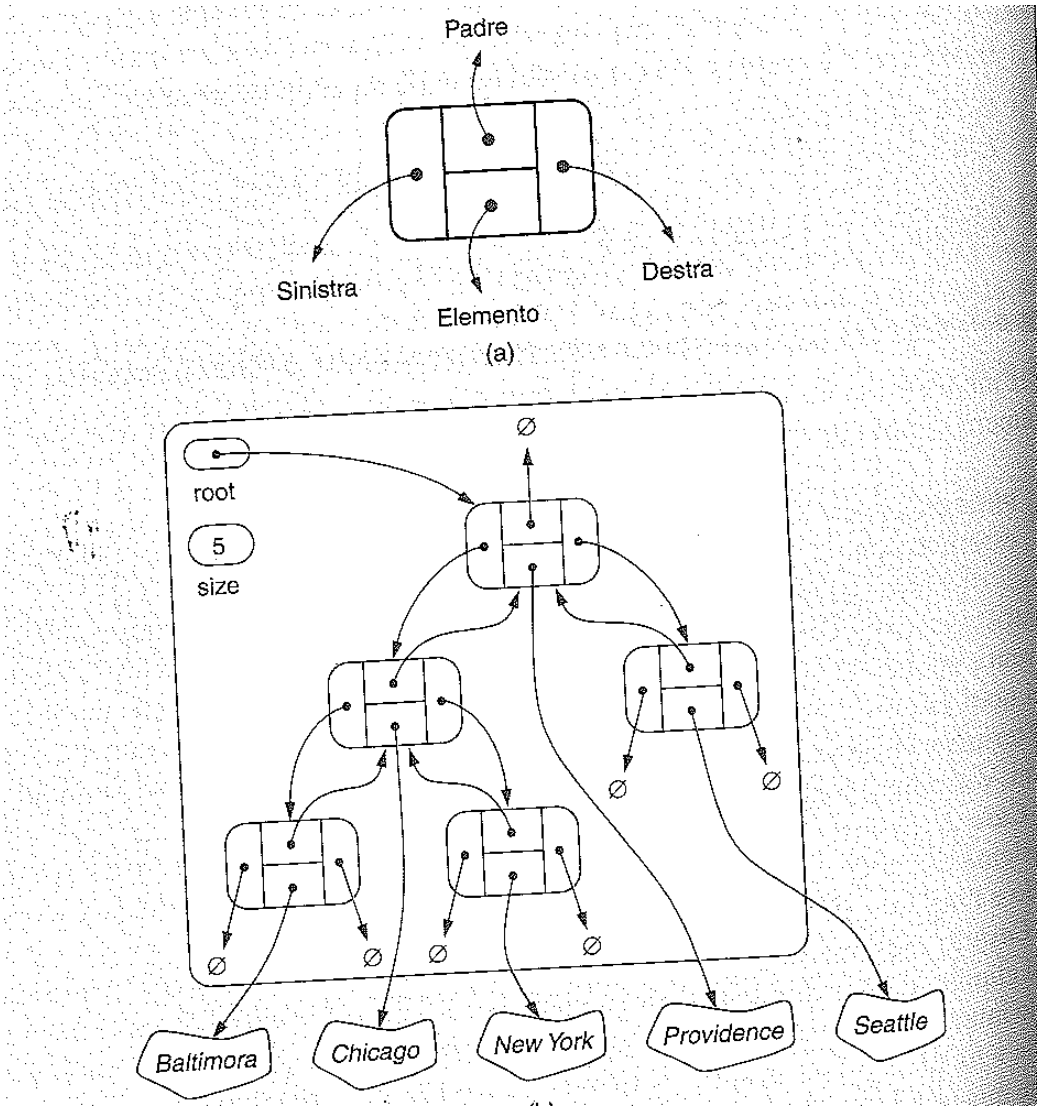

Figura 1: Generico nodo di un albero binario.

2 Creazione e popolamento di alberi biari

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 per il test della rappresentazione in cui creare un albero, inserire alcuni valori di prova e poi stamparlo con il metodo stampa della classe `BTreeUtil` (fornita nel materiale di supporto). Un possibile main di prova è il seguente:

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

```
BTree<Integer> t                = new MyBTree<Integer>();
Position<Integer> root          = t.setRoot(100);
Position<Integer> cL            = root.setLeft(20);
Position<Integer> cR            = root.setRight(230);

cL.setLeft(1);
cL.setRight(2);
cR.setLeft(3);

TreeUtil.stampa(t);
```

Suggerimento. Nel momento in cui si devono usare le interfacce ci sono diversi aspetti da considerare, nel seguito descriveremo il pattern di programmazione che useremo nel corso delle esercitazioni.

Si noti come, sia nel main di prova che nei metodi delle classi `MyBNode` e `MyBTree`, queste due classi vengano usate solo per l'istanziazione degli oggetti e mai come dichiarazione del tipo degli oggetti. Questa proprietà del pattern di programmazione deriva dalla filosofia stessa di interfaccia: infatti, volendo un giorno utilizzare le ipotetiche implementazioni, rispettivamente di `Position` e `Tree`, date da `TreeNodeMoltoMeglio` e `MyTreeMoltoMeglio`, basterebbe cambiare solo le istanziazioni degli oggetti mentre il resto del codice resterebbe uguale.

É fuori dallo scopo del corso analizzare dettagli e varianti, che possono essere però discussi con lo studente interessato.

```
public class MyBNode<E> implements BPosition<E> {
    private E element;
    private Position<E> parent;
    private Position<E> left;
    private Position<E> right;

    [...]
}
```

```
public class MyBTree<E> implements BTree<E> {
    private BPosition<E> root;

    public Position<E> setRoot(E n) {
        root = new MyBNode<E>();
        root.setElement(n);
        return root;
    }

    [...]
}
```

3 Visita di alberi binari

Per visita di un *albero binario* si intende l'operazione di accedere a tutti i suoi nodi eseguendo o meno operazioni su di essi, ad esempio stampandone il valore o inserendoli in un insieme. Ci sono diversi tipi di visita dipendentemente dall'*ordine* con cui i nodi vengono visitati.

Programma Java. Si vuole modificare la classe `MyBTree<E>` aggiungendo i seguenti metodi di istanza.

```
public void printPreorder()
```

Stampa i nodi dell'albero in preordine.

```
public List<E> postorder()
```

Restituisce una lista degli elementi dell'albero in postordine.

4 Visita di alberi binari con calcolo di proprietà

La visita di un albero può essere l'operazione di base quando il task è calcolare una particolare proprietà di un *albero binario*, e può essere opportuno (o necessario) utilizzare un tipo di visita piuttosto che un altro.

Programma Java. Si vuole modificare la classe `MyBTree<E>` aggiungendo i seguenti metodi di istanza.

```
public int size()
```

Restituisce il numero di nodi dell' albero binario.

```
public List<E>[] listLevel()
```

Restituisce un array di liste. La lista nella cella `i` contiene gli elementi dei nodi di profondità `i`.

```
public boolean isHeapOrdered(Comparator<E> comparator)
```

Restituisce `true` se l'albero binario è *heap-ordinato* e `false` altrimenti.

Un albero binario è *heap-ordinato* se ogni nodo interno diverso dalla radice, ha elemento maggiore o uguale all'elemento del genitore.

Parametri.

- `comparator`: comparatore per `E`.

```
public boolean isComplete()
```

Restituisce `true` se l'albero binario è *completo* e `false` altrimenti.

Un albero binario è *completo* se, sia h l'altezza, si ha che per $i = 0, 1, ..., h-1$ ci sono 2^i nodi di profondità i e a profondità $h-1$ i nodi interni sono alla sinistra delle foglie.

Nota. Il metodo `isComplete` è abbastanza più difficile degli altri, e la sua realizzazione è facoltativa. Si noti che un albero binario che soddisfa le proprietà verificate da `isHeapOrdered` e `isComplete` è un *heap*.

Programma Java. Si vuole implementare il seguente metodo statico nella classe `MyTest`.

```
public static BTree<Integer> evenTree(BTree<Integer> tree)
```

Restituisce un albero binario che ha tutti e soli i nodi pari di `tree`. Come?

Parametri.

- `tree`: albero binario di `Integer`

Suggerimento. Per realizzare i metodi precedenti è consigliabile ricorrere a metodi privati di supporto in modo da poter aggiungere eventuali parametri aggiuntivi rispetto a quelli indicati nelle signature dei metodi. Ovviamente dovrà essere tali metodi aggiuntivi dovranno essere privati e non accessibili dall'esterno ed utilizzati solo all'interno della classe.

Riferimenti bibliografici

[1] M. T. Goodrich and R. Tamassia. *Strutture dati e algoritmi in Java*. Zanichelli, 2007.