Esercitazione 2
Alberi n-ari

Corso di Fondamenti di Informatica II

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

A.A. 2013/2014

18 Ottobre 2012

Sommario

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

1 Rappresentazione di un albero n-ario

Un *albero n-ario* è una struttura dati che permette di rappresentare un generico albero in cui ad ogni nodo è associato un valore e in cui ogni nodo ha 0 o più nodi figli. L'implementazione proposta in questa esercitazione rappresenta un *albero n-ario* (non vuoto) con (i) un campo contenente un valore del tipo appropriato (`root`); (ii) una lista di riferimenti agli n sottoalberi (`subtrees`) corrispondenti ai figli di `root`. Ci si riferisca alla figura 1 per maggiori dettagli. Si noti che l'implementazione basata sulla interfaccia `Position` descritta nel libro di testo [1] supporta quanto appena descritto, e sarebbe quindi altrettanto valida per la rappresentazione di alberi n-ari.

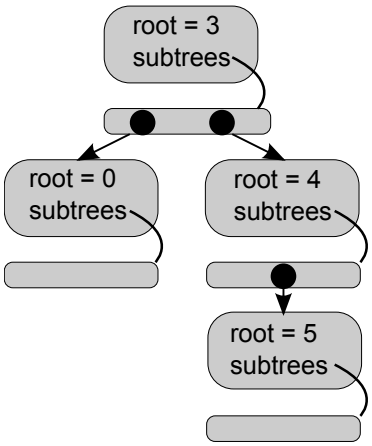

Figura 1: Un albero n-ario con 4 nodi.

Specifiche. Scrivere una classe Java parametrica `Tree<E>` con la seguente interfaccia.

```
public Tree(E val)
```

Costruisce un albero con un singolo nodo di valore `val`.

```
public E getRoot()
```

Ritorna il valore dell'elemento alla radice dell'albero.

```
public void setRoot(E val)
```

Modifica il valore dell'elemento alla radice dell'albero.

```
public List<Tree<E>> getSubtrees()
```

Ritorna una collection di tipo `List` (quale una `LinkedList` o `ArrayList`) contenente i sottoalberi corrispondenti ai figli della radice.

```
public Tree<E> addChild(E val)
```

Aggiunge un nuovo figlio di valore `val` alla radice e restituisce il corrispondente sottoalbero.

```
public void removeChildren()
```

Rimuove dall'albero tutti i sottoalberi corrispondenti ai figli della radice.

Si proceda a testare `Tree<E>` attraverso il driver `Test.java` fornito nella cartella dell'esercitazione e la classe `TreeUtil`, contenente un metodo di stampa generico (`print`) per `Tree<E>`.

2 Visita di alberi n-ari

Per visita di un *albero n-ario* si intende l'operazione di accedere a tutti i suoi nodi eseguendo o meno operazioni su di essi, ad esempio stampandone il valore. Ci sono diversi tipi di visita a seconda dell'*ordine* con cui i nodi vengono visitati.

Specifiche. Si aggiungano i seguenti metodi parametrici alla classe `TreeUtil` fornita nella cartella dell'esercitazione.

```
public static <E> void printPreorder(Tree<E> t)
```

Stampa i nodi dell'albero in preordine.

```
public static <E> void printPostorder(Tree<E> t)
```

Stampa i nodi dell'albero in postordine.

Si proceda a testare i nuovi metodi di `TreeUtil` attraverso il driver `Test.java` fornito nella cartella dell'esercitazione.

3 Visita tramite iteratori

Un iteratore è un oggetto che consente di iterare sugli elementi contenuti in un tipo di dato astratto contenitore (per esempio un insieme, una lista, ...). L'accesso fornito da un iteratore è sequenziale, e può essere effettuato tramite un metodo `next()`. In Java, un tipo iteratore deve implementare l'interfaccia `Iterator<E>`, e di conseguenza implementare i metodi descritti nella documentazione.

Specifiche. Si implementi la classe Java `PreOrderIterator<E>`, implementando l'interfaccia `Iterator<Tree<E>>`. Si faccia uso di un campo dati `Stack<Tree<E>> pila`.

```
PreOrderIterator(Tree<E> t)
```

Inizializza l'iteratore sull'albero `t`, con costo $O(1)$.

```
public boolean hasNext()
```

Ritorna true se l'iterazione non si è conclusa. Il costo deve essere $O(1)$.

```
public Tree<E> next()
```

Restituisce il successivo elemento dell'iterazione, con costo $O(n)$. Con n, in questo caso, ci riferiamo al numero massimo di figli di un nodo (come da definizione di albero n-ario). Lo specifichiamo per non confonderci con il numero totale di nodi dell'albero, a cui spesso ci si riferisce con lo stesso simbolo.

Non si implementi il metodo opzionale `remove`, ovvero si usi:

```
public void remove() {
        throw new UnsupportedOperationException("remove");
}
```

Facoltativo. Si implementi la classe Java `PostOrderIterator<E>`, implementando l'interfaccia `Iterator<Tree<E>>` come si è già fatto precedentemente, ma fornendo accesso sequenziale secondo una visita in postordine. Nota bene: questo esercizio è più complicato del precedente.

Riferimenti bibliografici

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