Laboratorio di Algoritmi e Strutture dati

24 marzo 2015

Esercizio `binarysearch`: ricerca binaria *(3 pt)*

Si crei un package *binarysearch*, all'interno del quale si definiscano le due classi seguenti, insieme alle rispettive classi di test.

IntSortedArray (array ordinato di interi parzialmente riempito)

In un file *IntSortedArray.java* si definisca una classe *IntSortedArray* contenente:

1. un campo `elements` di tipo `int[]`, con visibilità *package*, che conterrà gli elementi e sarà in generale solo parzialmente riempito;
2. un campo `size` di tipo `int`, con visibilità *package*, che contiene il numero di elementi effettivamente presenti nell'array (non la capacità!);
3. un costruttore pubblico di default `IntSortedArray()` che crea un array di capacità iniziale 16;
4. un costruttore pubblico `IntSortedArray(int initialCapacity)` che crea un array della data capacità iniziale; `initialCapacity` può essere un intero positivo o anche 0; se è negativo viene lanciata una eccezione `IllegalArgumentException`;
5. un costruttore pubblico `IntSortedArray(int[] a)` che prende come argomento un array `a` di `int`, non necessariamente ordinato, e costruisce un *IntSortedArray* avente come capacità la lunghezza di `a` incrementata di 16, e contenente tutti gli elementi di `a`, ovviamente in ordine;
6. un metodo pubblico `int size()`, che restituisce il numero di elementi effettivamente presenti nell'array (non la capacità!);
7. un metodo con visibilità *package* `int binarySearch(int x)` che realizza la procedura di ricerca binaria iterativa con tutti i raffinamenti visti a lezione, compresa la **restituzione della posizione di inserimento se l'elemento non è presente**;

8. un metodo con visibilità *package* `void reallocate()` che rialloca `elements` in un array di dimensione doppia (viene invocato da `insert` quando l'array è pieno);

9. un metodo pubblico `int insert(int x)` che inserisce `x` in `elements` mantenendolo ordinato, e restituisce l'indice a cui è stato inserito: lo inserisce in ogni caso, anche se l'elemento è già presente (in tal caso si avrà evidentemente un elemento ripetuto); se l'array è pieno, prima rialloca gli elementi in un array di dimensione doppia e poi inserisce l'elemento;

10. un metodo pubblico `int indexOf(int x)` che (usando la ricerca binaria) restituisce l'indice dell'elemento `x` nell'array (o *un* indice, se `x` è presente più volte); se `x` non è presente, restituisce -1.

11. un metodo pubblico `int get(int i)` che restituisce l'i-esimo elemento, oppure solleva l'eccezione `ArrayIndexOutOfBoundsException` se l'i-esimo elemento non esiste;

12. un metodo pubblico `toString()` override dell'omonimo metodo di Object, che produca una stringa rappresentante l'array ordinato, con gli elementi racchiusi fra parentesi quadre e separati da virgola e spazio; esempio: `[5, 13, 25, 25, 43, 61]`.

Si definisca e si esegua poi, tramite *JUnit* lo unit testing del metodo `binarySearch`, e inoltre di tutti i metodi e costruttori pubblici come indicato nelle istruzioni generali per il laboratorio.

Nota Bene. La classe *IntSortedArray* non deve possedere un main, e l'esecuzione deve avvenire solo attraverso lo unit testing.

SortedArrayList (ArrayList generica ordinata)

Questo esercizio è una versione generica dell'esercizio precedente, realizzata per semplicità mediante la classe *ArrayList* di Java. In un file *SortedArrayList.java* si realizzi una classe *SortedArrayList* contenente:

1. un campo `elements` di tipo `ArrayList<E>`, con visibilità *package*;

2. un costruttore pubblico di default `SortedArrayList<E>()` che crea una ArrayList<E> di capacità iniziale 16;

3. un costruttore pubblico `SortedArrayList<E>(int initialCapacity)` che crea una ArrayList<E> della data capacità iniziale;

4. un costruttore pubblico `SortedArrayList<E>(E[] a)` che prende come argomento un array `a` di elementi di tipo `E`, non necessariamente ordinato, e costruisce una ArrayList<E> avente come capacità la lunghezza di `a` incrementata di 16, e contenente tutti gli elementi di `a`;

5. un metodo pubblico `int size()`, che restituisce il numero di elementi effettivamente presenti nell'ArrayList (banalmente richiama il metodo *size* di ArrayList!);

6. un metodo con visibilità *package* `int binarySearch(E x)` che realizza la procedura di ricerca binaria iterativa con i vari raffinamenti visti a lezione compresa la restituzione della posizione di inserimento;

7. un metodo pubblico `int insert(E x)` che inserisce `x` in `elements` mantenendolo ordinato e restituisce l'indice a cui è stato inserito;

8. un metodo pubblico `int indexOf(E x)` che (usando la ricerca binaria) restituisce l'indice dell'elemento `x` nell'array (o *un* indice, se `x` è presente più volte); se `x` non è presente, restituisce -1.

9. un metodo pubblico `E get(int i)` che restituisce l'i-esimo elemento (banalmente richiama la *get* di *ArrayList*!)

10. un metodo pubblico `toString()` override dell'omonimo metodo di Object (banalmente richiama il *toString* di *ArrayList*!)

Nota. Per poter effettuare i confronti fra oggetti di tipo `E` è necessario dichiarare che la classe `E` deve implementare l'interfaccia `Comparable<E>`, con la sintassi:

```
public class SortedArrayList<E extends Comparable<E> >
```

In questo modo si può usare, per effettuare i confronti, il metodo `compareTo`.

Si definisca e si esegua poi, tramite *JUnit*, lo unit testing del metodo `binarySearch`, e di tutti i metodi pubblici come indicato nelle istruzioni generali per il laboratorio.