Laboratorio di Algoritmi e Strutture dati

5 maggio 2015

⋆ Esercizio `priorityqueue`: coda con priorità *(6 pt)*

Si crei un package *priorityqueue*, all'interno del quale si implementino, insieme alle corrispondenti classi di test, le soluzioni ad almeno uno tra gli esercizi 1 e 2. Opzionalmente si implementi l'esercizio 3.

Esercizio 1 - Code con priorità senza tipi generici

Si definisca:

- un'interfaccia `PriorityQueueStringDouble` che rappresenti il tipo astratto *Coda con Priorità* di elementi di tipo *String* con priorità di tipo *double*, in cui *non sono ammessi elementi ripetuti*:

```
public interface PriorityQueueStringDouble {
  /* se l'elemento è già presente, il metodo non fa nulla e
     restituisce false; se invece l'inserimento va a buon
     fine, restituisce true */
  boolean add(String element, double priority);

  /* restituisce null se la coda è vuota */
  String first();

  /* restituisce null se la coda è vuota */
  String removeFirst();

  boolean isEmpty();

  /* restituisce false se l'elemento non è presente */
  boolean delete(String element);

  /* restituisce false se l'elemento non è presente */
  boolean setPriority(String element, double priority);
}
```

- una classe `PriorityQueueStringDoubleSimple` che implementi l'interfaccia `PriorityQueueStringDouble` in uno dei modi elementari indicati nella slide 26.21 (liste ordinate o non ordinate, array, ecc.);

- una classe `PriorityQueueStringDoubleHeap` che implementi l'interfaccia `PriorityQueueStringDouble` per mezzo della struttura a heap, e contenga almeno un costruttore avente un parametro *int* che designa la capacità iniziale (*initialCapacity*).

Esercizio 2 - Code con priorità con tipi generici

Si definisca:

- un'interfaccia generica `PriorityQueue` che rappresenta il tipo astratto *Coda con Priorità* di elementi di tipo E con priorità di tipo P (in cui non sono ammessi elementi ripetuti):

```
public interface PriorityQueue⟨E, P extends Comparable⟨P⟩⟩ {
  boolean add(E element, P priority);
  E first();
  E removeFirst();
  boolean isEmpty();
  boolean delete(E element);
  boolean setPriority(E element, P priority);
}
```

- una classe generica PriorityQueueSimple⟨E, P **extends** Comparable⟨P⟩⟩ che implementi l'interfaccia precedente in uno dei modi elementari indicati nella slide 26.21 (liste ordinate o non ordinate, array, ecc.);
- una classe generica PriorityQueueHeap⟨E, P **extends** Comparable⟨P⟩⟩ che implementi l'interfaccia precedente per mezzo di una struttura a heap (per evitare i noti problemi degli array generici lo heap può essere realizzato come *ArrayList*; naturalmente il codice Java risulta, in qualunque realizzazione, piuttosto prolisso).

Esercizio 3 - Code con priorità valori in un range limitato

Si implementi una classe `PriorityQueueIntDouble` che realizzi, senza definire una corrispondente interfaccia, il tipo *Coda con priorità* di elementi di tipo *int* compresi in un range limitato, usando una struttura a heap e utilizzando un array delle posizioni invece di una hashmap, come indicato sulle slides:

```
public class PriorityQueueIntDouble {
 /* costruttore di una coda i cui elementi sono compresi nel
    range 0..n-1 */
  PriorityQueueIntDouble(int n)
  boolean add(int element, double priority)
  int first();
  int removeFirst();
  boolean isEmpty();
  boolean delete(int element);
  boolean setPriority(int element, double priority);
}
```