Code di Priorità

TDA Coda di Priorità (§ 8.1.3)

- Una coda di priorità memorizza una collezione di elementi
- Ogni **entry** è una coppia (key, value)
- Metodi principali del TDA Priority Queue
  - insert(k, x)
    inserisce un entry con chiave k e valore x
  - removeMin()
    elimina e restituisce l'entry con chiave minore
- Metodi addizionali
  - min()
    restituisce, ma non rimuove, un entry con chiave minore
  - size(), isEmpty()
- Applicazioni:
  - Voli in attesa di partire
  - Aste
  - Mercato azionario

Relazioni di Ordine Totale (§ 8.1.1)

- Le chiavi di una coda di priorità possono essere oggetti arbitrari su cui è definito un ordine
- Due elementi distinti possono avere la stessa chiave
- Concetto matematico di relazione di ordine totale $\leq$
  - Proprietà riflessiva : $x \leq x$
  - Proprietà asimmetrica: $x \leq y \wedge y \leq x \Rightarrow x = y$
  - Proprietà transitiva: $x \leq y \wedge y \leq z \Rightarrow x \leq z$

TDA Entry (§ 8.1.2)

- Un **entry** in una coda di priorità è semplicemente una coppia key-value
- Le code di priorità permettono un'efficiente inserimento e cancellazione basato su chiavi
- Metodi:
  - key(): restituisce la chiave di un entry
  - value(): restituisce il valore associato con un entry

- Interfaccia Java :

```
/**
  * Interfaccia per un entry

 **/
public interface  Entry  {
   public  Object key();
   public  Object value();
}
```

TDA Comparator (§ 8.1.2)

- Un comparator incapsula l'azione di confronto di due oggetti secondo un dato ordine totale
- Una generica coda di priorità usa un comparator ausiliario
- Il comparator è esterno alle chiavi che vengono confrontate
- Quando la coda di priorità deve confrontare due chiavi usa un comparator

- Il metodo principale di un TDA Comparator è:
  - compare(x, y): Restituisce un integer *i* tale che $i < 0$ se $x < y$, $i = 0$ se $x = y$, è $i > 0$ se $x > y$; un errore viene generato se *x* e *y non possono essere confrontati*.

Esempio di Comparator

- Confronto lessicografico di punti in 2D:

```
/** Comparator per punti in  2D secondo
    l'ordine lessicografico standard. */
public class  Lexicographic  implements
    Comparator  {
   int  xa, ya, xb, yb;
   public int  compare(Object a, Object b)
    throws  ClassCastException  {
     xa = ((Point2D) a).getX();
     ya = ((Point2D) a).getY();
     xb = ((Point2D) b).getX();
     yb = ((Point2D) b).getY();
     if  (xa != xb)
          return  (xb - xa);
     else
          return  (yb - ya);
  }
}
```

- Oggetto Punto in 2D:

```
/** Classe rappresentante un punto nel
    piano con coordinate intere */
public class  Point2D          {
   protected int xc, yc; // coordinates
   public  Point2D(int  x,  int  y)  {
      xc = x;
      yc = y;
  }
   public int  getX()  {
          return  xc;
   }
   public int  getY()  {
          return  yc;
   }
}
```

Coda di priorità Ordinamento (§ 8.1.4)

- Possiamo usare una coda di priorità per ordinare un insieme di elementi confrontabili:
  1. Inserisci gli elementi uno ad uno con una serie di operazioni di insert
  2. Rimuovi gli elementi secondo l'ordine definito con una sequenza di operazioni removeMin
- Il tempo di esecuzione di questo metodo di ordinamento dipende dall'implementazione della coda di priorità

Algorithm *PQ-Sort*(*S*, *C*)
  Input sequence *S*, comparator *C* for the elements of *S*
  Output sequence *S* sorted in increasing order according to *C*
  P ← priority queue with comparator *C*
  while ¬*S.isEmpty* ()
    e ← *S.removeFirst* ()
    P.insert (*e*, *0*)
  while ¬*P.isEmpty*()
    e ← *P.removeMin*().*key*()
    S.insertLast(*e*)

Implementazione Sequenziale di una Coda di Priorità

- Implementazione con una lista non ordinata

- Prestazioni:
  - insert richiede tempo $O(1)$ poichè è possibile inserire un elemento all'inizio o alla fine della sequenza
  - removeMin e min richiedono tempo $O(n)$ poichè è necessario attraversare l'intera sequenza per trovare la chiave più piccola

- Implementazione con una lista ordinata

- Prestazioni:
  - insert richiede tempo $O(n)$ poichè occcorre trovare la posizione dove inserire l'elemento
  - removeMin e min richiedono tempo $O(1)$, poichè la chiave minore si trova all'inizio

Selection-Sort

- Selection-sort è una variante di PQ-sort dove la coda di priorità è implementata con una sequenza non ordinata
- Tempo di esecuzione di Selection-sort:
  1. Inserimento degli elementi nella coda di priorità con n operazioni di insert richiede tempo $O(n)$
  2. Rimozione degli elementi secondo l'ordine con n operazioni removeMin che richiedono tempo proporzionale a

$$1 + 2 + \ldots + n$$

- Selection-sort richiede tempo $O(n^2)$

Esempio di Selection-Sort

	Sequenza *S*	***Coda di priorità*** *P*
Input:	(7,4,8,2,5,3,9)	()
Phase 1		
(a)	(4,8,2,5,3,9)	(7)
(b)	(8,2,5,3,9)	(7,4)
..	.. ..	
.	. .	
(g)	()	(7,4,8,2,5,3,9)
Phase 2		
(a)	(2)	(7,4,8,5,3,9)
(b)	(2,3)	(7,4,8,5,9)
(c)	(2,3,4)	(7,8,5,9)
(d)	(2,3,4,5)	(7,8,9)
(e)	(2,3,4,5,7)	(8,9)
(f)	(2,3,4,5,7,8)	(9)
(g)	(2,3,4,5,7,8,9)	()

Insertion-Sort

- Insertion-sort è una variante di PQ-sort dove la coda di priorità è implementata con una sequenza ordinata
- Tempo di esecuzione di Insertion-sort:
  1. Inserimento degli elementi nalla coda di priorità con n operazioni di insert richiede tempo prporzionale a
     $$1 + 2 + \ldots + n$$
  2. Rimozione degli elementi in sequenza ordinata dalla coda di priorità con una serie di n operazioni di removeMin richiede tempo $O(n)$
- Insertion-sort richiede tempo $O(n^2)$

Insertion-Sort Example

	Sequenza *S*	***Coda di priorità*** *P*
Input:	(7,4,8,2,5,3,9)	()
Phase 1		
(a)	(4,8,2,5,3,9)	(7)
(b)	(8,2,5,3,9)	(4,7)
(c)	(2,5,3,9)	(4,7,8)
(d)	(5,3,9)	(2,4,7,8)
(e)	(3,9)	(2,4,5,7,8)
(f)	(9)	(2,3,4,5,7,8)
(g)	()	(2,3,4,5,7,8,9)
Phase 2		
(a)	(2)	(3,4,5,7,8,9)
(b)	(2,3)	(4,5,7,8,9)
..	..	..
.	.	.
(g)	(2,3,4,5,7,8,9)	()

Insertion-sort sul posto

- Invece di usare una struttura dati esterna, possiamo implementare selection-sort e insertion-sort sul posto
- Una porzione della sequenza di input è utilizzata come coda di priorità
- Per insertion-sort sul posto
  - Manteniamo ordinata la porzione iniziale della sequenza
  - Usiamo degli swaps invece di modificare la sequenza