Code di priorità

Luca Becchetti

Presentazione tratta dalle slide che accompagnano il testo Data Structures and Algorithms in Java, 6th edition, by M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, Wiley, 2014

Coda di priorità → ADT

- **Collezione di elementi**
- **Ogni elemento → (chiave, valore)**
- **Operazioni fondamentali**
  - insert(k, v)
  - RemoveMin()
    - Rimuove e restituisce elemento con chiave minima
    - restituisce **null** se coda vuota
- **Operazioni ulteriori**
  - min(): restituisce (senza rimuovere) l'elemento con chiave minima o **null** se coda vuota
  - size(), isEmpty()
- Applicazioni
  - Liste di attesa
  - Aste on-line
  - Mercati azionari

Coda di priorità → ADT

- **Collezione di elementi**
- **Ogni elemento → (chiave, valore)**
- **Operazioni fondamentali**
  - insert(k, v)
  - RemoveMin()
    - Rimuove e restituisce elemento con chiave minima
    - restituisce **null** se coda vuota
- **Operazioni ulteriori**
  - min(): restituisce (senza rimuovere) l'elemento con chiave minima o **null** se coda vuota
  - size(), isEmpty()
- Applicazioni
  - Liste di attesa
  - Aste on-line
  - Mercati azionari

Esempio

- **Operazioni e stato della coda di priorità**

Method	Return Value	Priority Queue Contents
insert(5,A)		{ (5,A) }
insert(9,C)		{ (5,A), (9,C) }
insert(3,B)		{ (3,B), (5,A), (9,C) }
min( )	(3,B)	{ (3,B), (5,A), (9,C) }
removeMin( )	(3,B)	{ (5,A), (9,C) }
insert(7,D)		{ (5,A), (7,D), (9,C) }
removeMin( )	(5,A)	{ (7,D), (9,C) }
removeMin( )	(7,D)	{ (9,C) }
removeMin( )	(9,C)	{ }
removeMin( )	null	{ }
isEmpty( )	true	{ }

Ordinamento totale tra le chiavi

- **Chiavi: oggetti arbitrari su un dominio che ammette un ordinamento totale**
- **Elementi distinti potrebbero avere la stessa chiave**

- ***Relazione di ordinamento* ≤**
  - Confrontabilità
    - $x \leq y$ o $y \leq x$
  - Proprietà antisimmetrica
    - $x \leq y$ e $y \leq x \rightarrow x = y$
  - Proprietà transitiva
    - $x \leq y$ e $y \leq z \rightarrow x \leq z$

Elemento di una coda di priorità

- **Il tipo astratto rappresenta coppie (chiave, valore)**
- **Operazioni/ metodi**
  - getKey()
    - Restituisce chiave associata alla coppia
  - GetValue()
    - Restituisce valore associato alla coppia

- **Interfaccia Java**

```
/**
 * Interface for a key-value
 * pair entry
**/
public interface  Entry<K,V>  {
    K getKey();
    V getValue();
}
```

Implementazione di code di priorità
Uso di liste

Rappresentazione con liste

- **Lista non ordinata**
  - Inserimento in testa/coda → costo O(1)
  - removeMin e min hanno costo O(n)
    - Occorre scandire la lista per individuare l'elemento di chiave minima

- **Lista ordinata**
  - Inserimento ha costo O(n) → occorre scandire la lista per individuare la posizione
  - removeMin e min hanno costo O(1)
    - Elemento a chiave minima in prima posizione

Liste non ordinate (Java)

```
1  /** An implementation of a priority queue with an unsorted list. */
2  public class UnsortedPriorityQueue<K,V> extends AbstractPriorityQueue<K,V> {
3    /** primary collection of priority queue entries */
4    private PositionalList<Entry<K,V>> list = new LinkedPositionalList<>();
5
6    /** Creates an empty priority queue based on the natural ordering of its keys. */
7    public UnsortedPriorityQueue() { super(); }
8    /** Creates an empty priority queue using the given comparator to order keys. */
9    public UnsortedPriorityQueue(Comparator<K> comp) { super(comp); }
10
11   /** Returns the Position of an entry having minimal key. */
12   private Position<Entry<K,V>> findMin() {    // only called when nonempty
13     Position<Entry<K,V>> small = list.first();
14     for (Position<Entry<K,V>> walk : list.positions())
15       if (compare(walk.getElement(), small.getElement()) < 0)
16         small = walk;          // found an even smaller key
17     return small;
18   }
19
```

Liste non ordinate/2 (Java)

```
20    /** Inserts a key-value pair and returns the entry created. */
21    public Entry<K,V> insert(K key, V value) throws IllegalArgumentException {
22      checkKey(key);      // auxiliary key-checking method (could throw exception)
23      Entry<K,V> newest = new PQEntry<>(key, value);
24      list.addLast(newest);
25      return newest;
26    }
27
28    /** Returns (but does not remove) an entry with minimal key. */
29    public Entry<K,V> min( ) {
30      if (list.isEmpty( )) return null;
31      return findMin( ).getElement( );
32    }
33
34    /** Removes and returns an entry with minimal key. */
35    public Entry<K,V> removeMin( ) {
36      if (list.isEmpty( )) return null;
37      return list.remove(findMin( ));
38    }
39
40    /** Returns the number of items in the priority queue. */
41    public int size( ) { return list.size( ); }
42  }
```

Liste ordinate (Java)

```
1  /** An implementation of a priority queue with a sorted list. */
2  public class SortedPriorityQueue<K,V> extends AbstractPriorityQueue<K,V> {
3    /** primary collection of priority queue entries */
4    private PositionalList<Entry<K,V>> list = new LinkedPositionalList<>();
5
6    /** Creates an empty priority queue based on the natural ordering of its keys. */
7    public SortedPriorityQueue() { super(); }
8    /** Creates an empty priority queue using the given comparator to order keys. */
9    public SortedPriorityQueue(Comparator<K> comp) { super(comp); }
10
11   /** Inserts a key-value pair and returns the entry created. */
12   public Entry<K,V> insert(K key, V value) throws IllegalArgumentException {
13     checkKey(key);     // auxiliary key-checking method (could throw exception)
14     Entry<K,V> newest = new PQEntry<>(key, value);
15     Position<Entry<K,V>> walk = list.last();
16     // walk backward, looking for smaller key
17     while (walk != null && compare(newest, walk.getElement()) < 0)
18       walk = list.before(walk);
19     if (walk == null)
20       list.addFirst(newest);                    // new key is smallest
21     else
22       list.addAfter(walk, newest);              // newest goes after walk
23     return newest;
24   }
25
```

Liste ordinate/2 (Java)

```
26    /** Returns (but does not remove) an entry with minimal key. */
27    public Entry<K,V> min( ) {
28      if (list.isEmpty( )) return null;
29      return list.first( ).getElement( );
30    }
31
32    /** Removes and returns an entry with minimal key. */
33    public Entry<K,V> removeMin( ) {
34      if (list.isEmpty( )) return null;
35      return list.remove(list.first( ));
36    }
37
38    /** Returns the number of items in the priority queue. */
39    public int size( ) { return list.size( ); }
40  }
```

Ordinamento con code di priorità

- **Si inseriscono gli elementi nella coda di priorità con una serie di operazioni *insert***
- **Si rimuovono gli elementi dalla coda di priorità con una sequenza di *removeMin***

```
// Usiamo una coda di priorità Q
Input: un insieme S di coppie (k, v) rappresentato in modo qualsiasi
Output: array a ordinato rispetto alle chiavi degli elementi in S
while (!S.isEmpty())
    Q.insert(S.remove());
    // S.remove() rimuove un elemento (coppia) da S secondo un criterio qualsiasi
while(!Q.isEmpty())
        a.add(Q.removeMin());
```

Esempio: Selection Sort

	Sequenza S	Codia di priorità P	
Input:	(7,4,8,2,5,3,9)	()	
Fase 1			
(a)	(4,8,2,5,3,9)	(7)	O(n)
(b)	(8,2,5,3,9)	(7,4)	
..	.. ..		
(g)	()	(7,4,8,2,5,3,9)	
Fase 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)	$O(n^2)$
(e)	(2,3,4,5,7)	(8,9)	
(f)	(2,3,4,5,7,8)	(9)	
(g)	(2,3,4,5,7,8,9)	()	

Esempio: Insertion 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)	(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)	()

$O(n^2)$

$O(n)$