

Esercizi sui TDA Priority Queue, Dictionary e Map

1. Scrivere la funzione
`public static <K,V> PositionList<Entry<K,V> > sort(SortedListPriorityQueue<K,V> Q1, SortedListPriorityQueue<K,V> Q2, Comparator <K> c).`

Indicazioni per lo svolgimento dell'esercizio:

- La funzione deve prendere in input due code a priorità Q1 e Q2 e un comparatore c (dello stesso tipo usato dalle due code) e restituire una lista ordinata contenente tutte le entrate di Q1 e tutte le entrate di Q2.
- La funzione deve avere complessità $O(n)$ dove n è il numero totale di elementi di Q1 e Q2.

2. Scrivere la classe **PQStack** che implementa l'interfaccia **Stack**.

Indicazioni per lo svolgimento dell'esercizio:

- La classe deve avere un'unica variabile di istanza. Tale variabile di istanza deve essere di tipo **PriorityQueue**.

3. Implementare **PQ-Sort** con:

- `UnsortedListPriorityQueue` (Selection Sort)
- `SortedListPriorityQueue` (Insertion Sort)

Discutere la complessità computazionale di entrambe le implementazioni

4. Aggiungere alla classe `HeapPriorityQueue` il costruttore

`HeapPriorityQueue(V[] v, K[] k, Comparator<K> c)`

che prende in input, oltre al comparatore C, due array di uguale lunghezza e costruisce l'heap contenente le entrate $(k[i], v[i])$, per $i=0, \dots, \text{length}(v)$. Il metodo deve avere complessità **lineare** nella lunghezza degli array,

5. Aggiungere alla classe **HeapAdaptablePriorityQueue** il metodo

`public Iterable <Position<Entry<K,V>>> insertEntries(K[] S1, V[] S2)`

Indicazioni per lo svolgimento dell'esercizio:

- Il metodo prende in input due array di uguale lunghezza e inserisce nella coda a priorità le entrate $(S1[i], S2[i])$, per $i=0, \dots, \text{length}(S1)$ (si assuma che i due array siano entrambi pieni)
- Il metodo deve restituire in output una collezione iterabile delle posizioni della coda a priorità in cui sono state inserite le nuove entrate.

6. Scrivere la classe `UnsortedListAdaptablePriorityQueue` che implementa `AdaptablePriorityQueue` ed estende la classe `UnsortedListPriorityQueue`

7. Scrivere la funzione

```
public static <K,V> Iterable<Entry<K,V>> sortValues(Dictionary <K,V>
D, Comparator<V> c).
```

Istruzioni per lo svolgimento dell'esercizio:

- La funzione deve restituire una collezione iterabile contenente tutte le entry di **D** ordinate rispetto ai valori delle entrate. La relazione di ordine totale definita sull'insieme dei valori delle entrate di **D** è la stessa specificata dal comparatore **c**.
- La funzione deve lasciare inalterato il contenuto di **D**.
- Se **D** è vuoto allora la funzione deve restituire una collezione vuota.

8. Scrivere la funzione

```
public static <K,V> Iterable <K> subtract( Dictionary<K,V> D1,
Dictionary<K,V>D2).
```

Indicazioni per lo svolgimento dell'esercizio:

- La funzione deve cancellare da **D1** tutte le entrate le cui chiavi compaiono anche in **D2** e deve restituire una collezione iterabile contenente tutte le chiavi cancellate da **D1**. Ciascuna delle chiavi cancellate deve comparire esattamente una volta nella suddetta collezione iterabile.
- La funzione deve lasciare inalterato il contenuto di **D2**

9. Aggiungere il seguente metodo

```
public void updateKey( K key, K newKey)
```

alla classe **ChainingHashTable** che implementa il TDA **Dictionary** mediante una tabella hash in cui le collisioni sono risolte mediante il metodo del chaining.

Indicazioni per lo svolgimento dell'esercizio:

- Il metodo deve sostituire con **newKey** la chiave di tutte le entrate con chiave **key**. Si noti che potrebbe essere necessario spostare le entrate la cui chiave è stata modificata. Il metodo deve lanciare le opportune eccezioni.
- Il metodo **non** deve invocare altri metodi della classe **ChainingHashTable** al di fuori dei metodi **checkKey** e **hashValue**.

10. Aggiungere il seguente metodo

```
public int [ ] indices( K key)
```

alla classe **LinearProbingHashTable** che implementa il TDA **Dictionary** mediante una tabella hash in cui le collisioni sono risolte mediante il metodo del linear probing.

Indicazioni per lo svolgimento dell'esercizio:

- Il metodo deve restituire in output un array di interi che contiene gli indici delle celle della tabella hash in cui si trovano le entrate con chiave **key**. Se la lunghezza dell'array è maggiore del numero di occorrenze di **key** allora le celle inutilizzate devono contenere un **intero negativo**.
- Il metodo deve ispezionare esclusivamente quelle celle della tabella che potrebbero effettivamente contenere entrate con chiave **key** e **non** deve modificare il contenuto della tabella hash in nessun momento della sua esecuzione.

11. Aggiungere il metodo

```
public Entry<K,V> predecessor(Position p)
```

alla classe **BinarySearchTree**.

Istruzioni per lo svolgimento dell'esercizio:

- Il metodo `predecessor` deve restituire l'entrata la cui chiave è il predecessore di quella contenuta in **p**. Se la chiave di **p** non ha predecessori allora il metodo deve restituire `null`.
- La funzione deve avere complessità lineare nell'altezza dell'albero.

12. Aggiungere alla classe **BinarySearchTree** il metodo

```
public Iterable <V> withSmallestKeys(int k)
```

Indicazioni per lo svolgimento dell'esercizio:

- Il metodo deve restituire una collezione iterabile dei **k** valori associati alle **k** chiavi più piccole dell'albero binario di ricerca. Se il dizionario contiene meno di **k** entrate allora la collezione restituita deve contenere i valori di tutte le entrate del dizionario.
- Il metodo deve avere complessità $O(n)$ dove n è il numero delle entrate del dizionario. Metodi con complessità maggiore saranno valutati al più 7 punti.
- Il metodo **NON** deve invocare né i metodi `iterator()` e `positions()` di `LinkedBinaryTree` né i metodi `findAll()` ed `entries()` di `BinarySearchTree` (né metodi con analoghe funzionalità).

13. Scrivere la classe `MDictionary` che implementa l'interfaccia `Dictionary` ed ha un'unica variabile di istanza di tipo `Map`. Si assuma che un utente non inserisca mai entrate con la stessa chiave. Analizzare la complessità dei metodi di `MDictionary`.