Esercizi sui TDA Set, Partition e Graph

1. Aggiungere alla classe OrderedListSet il costruttore
   public<V> OrderedListSet(Comparator <E> c, BinarySearchTree<E,V> D).

Istruzioni per lo svolgimento dell'esercizio:

- Il costruttore deve creare un insieme che contiene tutte le chiavi di **D**. Ovviamente l'insieme deve risultare ordinato.
- Il comparatore ricevuto in input dal costruttore è lo stesso usato dal dizionario **D**.
- Il costruttore non deve invocare né il metodo entries né altri metodi con analoghe funzionalità.

2. Scrivere la classe OrderedListMultiset che implementa la seguente interfaccia Multiset mediante una lista ordinata.

```
public interface Multiset<E> {
  // Restituisce il numero di elementi del multiset.
  public int size();

  //Restituisce true se l'insieme è vuoto.
  public boolean isEmpty();

  //Rimpiazza this con l'unione di this e B.
  //Un elemento appare nell'unione un numero
  //di volte pari al max tra il numero di volte
  //in cui appare in this e il numero di volte in cui appare in B
  public Multiset<E> union(Multiset<E> B);

  //Rimpiazza this con l'intersezione di this e B.
  //Un elemento appare nell'intersezione un numero
  //di volte pari al min tra il numero di volte
  //in cui appare in this e il numero di volte in cui appare in B
  public Multiset<E> intersect(Multiset<E> B);

  //Rimpiazza this con la differenza di this e B.
  //Un elemento appare nell'insieme output un numero
  //di volte pari alla differenza tra il numero di volte
  //in cui appare in this e il numero di volte  in cui appare in B.
  //Se tale differenza è negativa l'elemento non è presente
  //nell'insieme output
  public Multiset <E>subtract(Multiset <E> B);
}
```

3. Scrivere la funzione

```
public static <E> void insertSet(Parition<E> P, Iterable<E>C)
```

Istruzioni per lo svolgimento dell'esercizio:

- La funzione prende in input una partizione **P** e una collezione iterabile **C** che non ha elementi in comune con alcun insieme di **P**.
- La funzione deve aggiungere alla partizione un insieme contenente tutti gli elementi contenuti in **C.** Si noti che **C** potrebbe contenere più occorrenze di uno stesso elemento.
- Se **C** è vuota la funzione non deve apportare alcuna modifica a **P**.
- Ovviamente la funzione può agire su **P** soltanto attraverso i metodi dell'interfaccia Partition.

4. Scrivere la funzione

```
public static<E> Set<E> max(Iterable <E> L, Partition<E> P)
```

che determina l'insieme della partizione P che ha il maggior numero di elementi in comune con L.

5. Scrivere la funzione

```
public static <V,E> Partition<Vertex<V>>connectedComponents(Graph<V,E> G)
```

che restituisce in output una partizione i cui insiemi rappresentano le componenti connesse del grafo.

6. Scrivere la classe AdjacencyListDGraph che implementa l'interfaccia Directed Graph mediante liste di adiacenza

7. Aggiungere il metodo

```
public void split(Vertex<V> u, V x, V y)
```

alla classe AdjecencyListGraph.

Istruzioni per lo svolgimento dell'esercizio:

- Il metodo split rimuove il vertice **u** dal grafo e lo sostituisce con due nuovi vertici **z** e **w** aventi gli elementi settati rispettivamente a **x** e **y**.
- I due nuovi vertici **z** e **w,** oltre ad essere tra di loro adiacenti, devono essere adiacenti a tutti i vertici che risultavano adiacenti al vertice **u** nel grafo originario. I nuovi archi devono avere l'elemento settato a **null**.

8. Scrivere la classe LayerBFS che specializza la classe BFS e computa i vertici a distanza i da start (dove i è un intero che viene fornito in input ad execute).