A.E.D. Laboratorio 2

Guillermo Román
guillermo.roman@upm.es

Julio García
juliomanuel.garcia@upm.es

Lars-Åke Fredlund
lfredlund@fi.upm.es

Tonghong Li
tonghong@fi.upm.es

Manuel Carro
mcarro@fi.upm.es

Nicolás Alonso Shirra
nicolas.alonso.shirra@upm.es

Normas

- Fechas de entrega y penalización:

Hasta el martes 14 de octubre, 16:00 horas	0 %
Hasta el miércoles 15 de octubre, 16:00 horas	20 %
Hasta el jueves 16 de octubre, 16:00 horas	40 %

  Después la puntuación máxima será 0
- Se comprobará plagio y se actuará sobre los detectados.
- Usad las horas de tutoría para preguntar sobre programación – son oportunidades excelentes para aprender.

Entrega

- Todos los ejercicios de laboratorio se deben entregar a través de

  `http://deliverit.fi.upm.es`

- El/los fichero(s) que hay que subir es/son `MultiSetList.java`.
- La clase debe estar en el paquete `aed.multisets` .
- La documentación de la API de `aedlib.jar` está disponible en `http://costa.ls.fi.upm.es/teaching/aed/docs/aedlib/`

Tarea para hoy

- El interfaz `MultiSet` representa una estructura de datos "multiconjunto"
- Se pide implementar el interfaz `MultiSet` usando una lista `PositionList`
- Un "multiconjunto" se comporta como un conjunto, excepto que los multiconjuntos admiten elementos repetidos
- Ejemplo: el multiconjunto $\{1,3,1,2\}$ contiene dos enteros 1, un 3, y un 2
- El interfaz `MultiSet` está documentado en el fichero `MultiSet.java`
- Hoy solo hay que modificar, y entregar, el fichero `MultiSetList.java`
- **Notad**: `null` esta permitido como elemento en un multiconjunto

El interfaz MultiSet<E>

```
public interface MultiSet<E>  {
  // Anade n instancias de elem
  void add(E elem, int n);

  // Borra n instancias de elem
  boolean remove(E elem, int n);

  // Devuelve el numero de elementos igual a elem
  int multiplicity(E elem);

  int size();     // Devuelve el numero total de elementos
  int isEmpty();  // Es vacio?

  // Devuelve los elementos (sin repeticion) del multiconjunto
  PositionList<E> elements();

  // Devuelve la interseccion, la union y la diferencia entre 'this' y s
  MultiSet<E> intersection(MultiSet<E> s);
  MultiSet<E> sum(MultiSet<E> s);
  MultiSet<E> minus(MultiSet<E> s);

  // Comprueba si 'this' es un subconjunto, o igual, de s
  public boolean subsetEqual(MultiSet<E> s);
}
```

Ejemplo

```
MultiSet<String> s  =  new MultiSetList<String>();
s.add("a",1);
s.add("a",1);
s.add("b",5);
s.add("b",-1);          ===>          // lanza IllegalArgumentException
s.size();               ===>  7       // {"a","a","b","b","b","b","b"}
s.multiplicity("a");===>  2           // contiene dos "a"
s.remove("a",1);        ===>  true    // se borra un "a"
s.remove("a",1);        ===>  true    // se borra un "a"
s.remove("a",1);        ===>  false   // no se borro un "a"
s.remove("b",3);        ===>  true    // se borro 3 "b"
s.multiplicity("b");===>  2           // {"b","b"}

s.add("b",20);
s.elements();             ===>  ["b"]// Una lista con el elemento "b"

s.add(null,10);                       // Se puede anadir null
s.multiplicity(null); ===> 10         // Y esta presente!
```

Intersection

- s1.intersection(s2) devuelve un multiset nuevo s3, donde para cada elemento $e \in s1 \wedge e \in s2$:

  s3.multiplicity(e) = min(s1.multiplicity(e), s2.multiplicity(e))

- Ejemplo:

```
MultiSet<Integer> s1  =  new MultiSetList<Integer>();
s1.add(0,3);
s1.add(1,1);
s1.add(4,1);   // {0,0,0,1,4}

MultiSet<Integer> s2  =  new MultiSetList<Integer>();
s1.add(0,2);
s1.add(1,3);
s1.add(2,1);   // {0,0,1,1,1,2}

MultiSet<Integer> s3  =  s1.intersection(s2);
---
s3 == {0,0,1}
```

Sum

- s1.sum(s2) devuelve un multiset nuevo s3, donde para cada elemento $e \in s1 \vee e \in s2$:

  $$\text{s3.multiplicity(e)} = \text{s1.multiplicity(e)} + \text{s2.multiplicity(e)}$$

- Ejemplo:

```
MultiSet<Integer> s1  =  new MultiSetList<Integer>();
s1.add(0,3);
s1.add(1,1);
s1.add(4,1);   // {0,0,0,1,4}

MultiSet<Integer> s2  =  new MultiSetList<Integer>();
s1.add(0,2);
s1.add(1,3);
s1.add(2,1);   // {0,0,1,1,1,2}

MultiSet<Integer> s3  =  s1.sum(s2);
---
s3 == {0,0,0,0,0,1,1,1,1,4,2}
```

Minus

- s1.minus(s2) devuelve un multiset nuevo s3, donde para cada elemento $e \in s1 \vee e \in s2$:

  $$\text{s3.multiplicity(e)} = \max(\text{s1.multiplicity(e)} - \text{s2.multiplicity(e)}, 0)$$

- Ejemplo:

```
MultiSet<Integer> s1  =  new MultiSetList<Integer>();
s1.add(0,3);
s1.add(1,1);
s1.add(4,1);   // {0,0,0,1,4}

MultiSet<Integer> s2  =  new MultiSetList<Integer>();
s1.add(0,2);
s1.add(1,3);
s1.add(2,1);   // {0,0,1,1,1,2}

MultiSet<Integer> s3  =  s1.minus(s2);
---
s3 == {0,4}
```

SubsetEqual

- s1.subsetEqual(s2) devuelve un boolean `true` si para cada elemento $e \in s1$. $s2.\texttt{multiplity}(e) \geq s1.\texttt{multiplicity}(e)$
- Ejemplo:

```
MultiSet<Integer> s1  =  new MultiSetList<Integer>();
s1.add(0,3);
s1.add(1,1);
s1.add(4,1);   // {0,0,0,1,4}

MultiSet<Integer> s2  =  new MultiSetList<Integer>();
s1.add(0,2);
s1.add(1,1);   // {0,0,1}

s1.subsetEqual(s2);    => false
s2.subsetEqual(s1);    => true
```

Representación del multiconjunto

- Es facil representar un multiconjunto con una lista:
  `{1,4,9,9,4,9} => [1,4,9,9,4,9]`
- Sin embargo vamos a usar una representación más eficiente: una lista de pares: `Pair(Element,Multiplicity)` – `Multiplicity` indica cuantas occurencias de `Element` hay en el multiconjunto:
  `{1,4,9,9,4,9} => [Pair(1,1),Pair(9,3),Pair(4,2)]`
- Es **obligatorio** usar el atributo `elements` dentro `MultiSetList.java` para guardar los elementos del multiconjunto, y los métodos (`add`, `remove`, ...) deben trabajar sobre este atributo
- Tipo: `PositionList<Pair<Element,Integer>> elements`
  Una lista de pares de elementos y integers

La clase MultiSetList<E>

```
public class MultisetList<E> implements MultiSet<E> {

  // Los elementos, con repeticiones
  private PositionList<Pair<E,Integer>> elements;
  // Tamano del multiset (incluyendo repeticiones)
  private int size;

  // Constructor que crea un multiset vacio
  public MultiSetList() { ... }


  ...
  // A implementar los metodos del interfaz
  ...
}
```

Importante

- **Invariantes** obligatorios (no cambian durante la ejecución):
  - `Multiplicity>0` para todos los objetos `Pair(Element,Multiplicity)` en `elements`, es decir, no puede haber pares que tengan el número de elementos a 0
  - no puede existir dos pares `Pair(Element,Multiplicity1)`, `Pair(Element,Multiplicity2)` con el mismo elemento en `elements`
- **Hint**: hay que borrar pares en el método `remove` para cumplir con el primer invariante
- Por ejemplo, `[Pair(1,5), Pair(8,0)]` no cumple el primer invariante (el `Multiplicity` en `Pair(8,0)` es 0)

Eficiencia

- Es importante que los métodos `size` y `isEmpty` sean *eficientes*
- Por ejemplo: el tiempo necesario para ejecutar una llamada al método `s.size()` no debería depender del tamaño del multiconjunto `s`
  - El tiempo para ejecutar una llamada `s1.size()` cuando `s1` es un multiconjunto con tamaño cero, y el tiempo para ejecutar una llamada `s2.size()` cuando `s2` es un multiconjunto con 100,000 elementos, deberían ser casi el mismo
  - Es decir, la complejidad del método `size()` debería ser $O(1)$

Notas Generales

- El proyecto debe compilar sin errores y debe cumplirse la especificación de los métodos a completar
- Debe pasar todos los test `TesterLab2` correctamente sin mensajes de error
- **Nota:** una ejecución sin mensajes de error y que pase todas las pruebas **no** significa que la implementación sea correcta (es decir, que funcione bien para cada posible entrada)
- Todos los ejercicios se corrigen manualmente antes de dar la nota final

Evaluación

¡Seguir estos consejos os permitirá conseguir mejores resultados!

- Corrección
- Ausencia de código repetido con la misma funcionalidad (podéis usar métodos auxiliares para evitarlo)
- Concisión y claridad del código
- Legibilidad, incluida selección de nombres descriptivos para variables y métodos
- El código debe estar correctamente indentado y con comentarios *útiles* cuando lo veáis necesario
- Eficiencia:
  - Se valorará la complejidad computacional del código
  - Se valorará no iterar innecesariamente en los recorridos de las estructuras de datos