La codificación Huffman permite representar textos de forma comprimida. Se trata de un código de longitud variable: cada carácter se representa por una secuencia de bits cuya longitud es inversamente proporcional a su frecuencia de aparición. Es decir, los caracteres muy frecuentes son representados por secuencias de pocos bits; mientras que los caracteres menos frecuentes se representan por secuencias de bits más largas. Vamos a desarrollar paso a paso una implementación de la codificación Huffman de un mensaje representado por una cadena (String).

Ejercicio 1. (0.75 puntos) El primer paso para construir un código de Huffman consiste en calcular la frecuencia de aparición o peso de los caracteres del mensaje. Define el método estático

```
public static Dictionary<Character, Integer> weights(String s)
```

que dado un mensaje representado como un String devuelva un diccionario que asocie cada carácter del mensaje con su número de apariciones.

Por ejemplo: `Huffman.weights("abracadabra")` produce el diccionario

```
AVLDictionary(a->5,b->2,c->1,d->1,r->2)
```

Ejercicio 2. Una vez que se ha obtenido la frecuencia de aparición de cada carácter, el siguiente paso consiste en construir un árbol de Huffman. El proceso se describe paso a paso en las transparencias 3-9 (Huffman.pdf). Representaremos los árboles de Huffman mediante la clase `WLeafTree` (árboles binarios con peso e información sólo en las hojas). La implementación está disponible en el fichero `WLeafTree.java`.

La construcción de árboles de Huffman requiere almacenar una colección de árboles. Como colección utilizaremos una cola de prioridad, donde los árboles aparecerán ordenados por peso (primero los más ligeros).

Apartado 2.a (0.75 puntos) Para construir el árbol de Huffman primero debemos obtener la colección inicial de árboles (transparencia 5). Define el método estático

```
public static PriorityQueue<WLeafTree<Character>>  huffmanLeaves(String s)
```

que dada una cadena devuelve una cola de prioridad con árboles hoja. Por ejemplo: `Huffman.huffmanLeaves("abracadabra")` produce la cola de prioridad

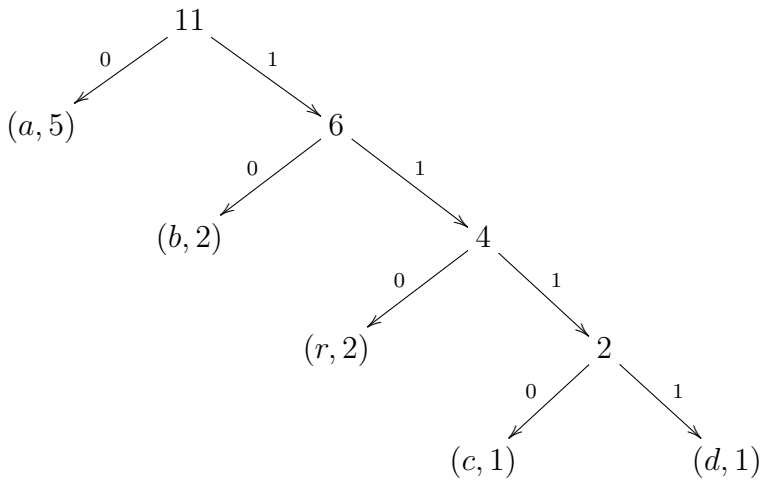
```
BinaryHeapPriorityQueue((c, 1),(d, 1),(r, 2),(b, 2),(a, 5))
```

Apartado 2.b (2.50 puntos) Define el método estático

```
public static WLeafTree<Character> huffmanTree(String s)
```

que dada una cadena con ***al menos 2*** símbolos distintos devuelve el árbol de Huffman correspondiente. Si la cadena no tiene al menos 2 caracteres distintos, el método debe elevar la excepción `HuffmanException`. El método debe implementar el algoritmo de reducción por mezcla descrito en las transparencias 4-9. Al comprobar tu solución ten en cuenta que, como se explica en las transparencias 10 y 11, el árbol de Huffman no es único.

Por ejemplo: `Huffman.huffmanTree("abracadabra")` produce el siguiente árbol



Ejercicio 3. Una vez construido el árbol de Huffman, se puede obtener el código de Huffman de cada carácter descendiendo por las ramas del árbol. El proceso se describe en las transparencias 12-14. Representaremos el código de Huffman mediante un diccionario de caracteres a listas de bits (enteros).

Apartado 3.a (1 punto) Define el método estático

```
public static Dictionary<Character, List<Integer>>
    joinDics( Dictionary<Character, List<Integer>> d1
            , Dictionary<Character, List<Integer>> d2)
```

que dados dos diccionarios disjuntos (no comparten ninguna clave) devuelve su unión. Por ejemplo, si `d1` y `d2` son los diccionarios

```
d1: AVLDictionary(a->LinkedList(1,2),c->LinkedList(3,4))
d2: AVLDictionary(b->LinkedList(5,6),d->LinkedList(7,8))
```

entonces, `Hamming.joinDics(d1,d2)` produce el diccionario

```
AVLDictionary(a->LinkedList(1,2),b->LinkedList(5,6),
              c->LinkedList(3,4),d->LinkedList(7,8))
```

Apartado 3.b (1 punto) Define el método estático

```
public static Dictionary<Character, List<Integer>>
    prefixWith(int i, Dictionary<Character, List<Integer>> d)
```

que dados un valor `i` y un diccionario cuyos valores son listas devuelva un nuevo diccionario que se obtiene prefijando cada lista del diccionario con `i`. Por ejemplo, si `d1` y `d2` son los diccionarios anteriores, entonces

```
prefix d1 with 0: AVLDictionary(a->LinkedList(0,1,2),c->LinkedList(0,3,4))
prefix d2 with 1: AVLDictionary(b->LinkedList(1,5,6),d->LinkedList(1,7,8))
```

Apartado 3.c (1.50 puntos) Utilizando los métodos anteriores, define el método estático

```
public static Dictionary<Character, List<Integer>>
                            huffmanCode(WLeafTree<Character> ht)
```

que dado un árbol de Huffman devuelve el código correspondiente.

Sugerencia: utiliza recursión sobre el árbol, si bajas por la izquierda prefija con un cero, si bajas por la derecha prefija con un 1.

Por ejemplo, dado el código

```
WLeafTree<Character> ht = Huffman.huffmanTree("abracadabra");
Dictionary<Character, List<Integer>> hc = Huffman.huffmanCode(ht);
```

`hc` será el diccionario

```
AVLDictionary(a->ArrayList(0),b->ArrayList(1,0),c->ArrayList(1,1,1,0),
              d->ArrayList(1,1,1,1),r->ArrayList(1,1,0))
```

Ejercicio 4. (0.50 puntos) Una vez construido el código de Huffman es muy simple codificar un mensaje, basta reemplazar cada carácter por su código, como se ve en la transparencia 15. Define el método estático

```
public static List<Integer>
        encode(String s, Dictionary<Character, List<Integer>> hc)
```

que dados una cadena y un diccionario con el código de Huffman devuelve el mensaje codificado.

Por ejemplo, siendo `hc` el diccionario anterior,

```
List<Integer> codedMsg = Huffman.encode("abracadabra", hc);
```

`codedMsg` será la lista

```
ArrayList(0,1,0,1,1,0,0,1,1,1,0,0,1,1,1,1,0,1,0,1,1,0,0).
```

Ejercicio 5. (2 puntos) Después de recibir el mensaje codificado, el receptor debe descodificarlo. El proceso de descodificación utiliza el árbol de Huffman, tal como se explica en las transparencias 16-25. Define el método estático

```
public static String  decode(List<Integer> bits, WLeafTree<Character> ht)
```

que dados un mensaje codificado y el árbol Huffman correspondiente devuelva el mensaje original.

Por ejemplo, dado la codificación `codeMsg` anterior y el código de Huffman `ht`, el código

```
String decodedMsg = Huffman.decode(codedMsg, ht);
```

produce en `decodedMsg` la cadena ‘`"abracadabra"`.