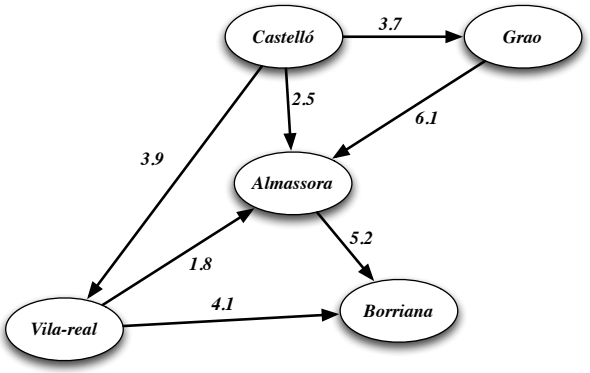
EI1013/MT1013 Estructuras de datos
Examen final
Lunes, 14 de Enero de 2013

1. **(2,5 puntos)** En un fichero de texto se tiene toda la información de un grafo dirigido y ponderado, donde los pesos serán valores enteros. En la primera línea del fichero estará el número de nodos del grafo. A continuación, la etiqueta (`String`) de cada nodo, cada dato en una línea distinta. Finalmente se encuentra la información de las listas de adyacencia de cada nodo. Cada lista de adyacencia está en una línea y hay una línea por cada nodo (aunque no tenga arcos). En cada línea con la lista de adyacencia, en primer lugar, estará el número de arcos y después los arcos. Cada arco consta de etiqueta del nodo destino y peso del arco. Los datos de los arcos en una lista están separados por uno o más espacios en blanco. Si no tiene lista de arcos, la línea correspondiente tendra sólo el número 0 (0 arcos). Por ejemplo, el grafo de la izquierda se describiría mediante el fichero de la derecha:




```
5
Castelló
Grao
Almassora
Borriana
Vila-real
3 Grao 3.7 Almassora 2.5 Vila-real 3.9
1 Almassora 6.1
1 Borriana 5.2
0
2 Almassora 1.8 Borriana 4.1
```

La clase `ListGraph<T,W>` almacena un grafo dirigido con arcos con peso usando listas de adyacencia. La parte privada de su implementación es:

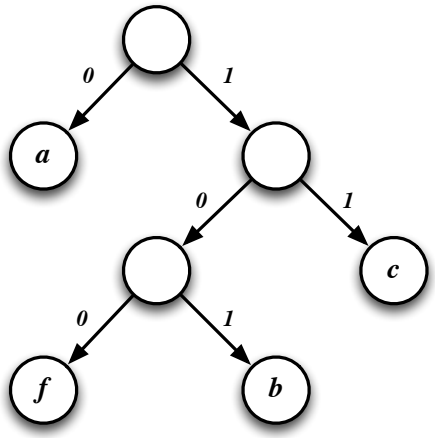
```
public class ListGraph<T,W> implements Graph<T, W> {
        private class Node<T> {
                T data;
                List<EDEdge<W>> lEdges;

                Node (T data) {
                        this.data = data;
                        this.lEdges = new LinkedList<EDEdge<W>>();
                }
        }

        private boolean isDirected;
        private List<Node> nodes = new ArrayList<Node>();
        ...
}
```

Implementar constructor de la clase `ListGraph<String,Double>` que reciba como parámetro el nombre del fichero y construya el grafo correspondiente.

```
public ListGraph (String nomFich);
```

2. **(2 puntos)** Se tiene un árbol binario extendido de caracteres dónde sólo se usan los valores en los nodos hoja (ver la figura). Este árbol representa la codificación binaria de esos caracteres, de tal forma que cada rama hacia la izquierda representa el el carácter '0' y cada rama a la derecha el carácter '1'. Así la cadena que codifica el carácter `'f'` es `'100'`. Se obtiene encadenando los valores asociados a las ramas desde la raíz a la hoja. Si el carácter que se le pasa a la función no está en el árbol devolverá `null`. Escribe una método público estático que reciba un árbol binario como el descrito y un carácter, construya la cadena que lo codifica.

```
public static String codificar (EDBinaryTree<Character> ab, char car);
```



3. (**2.5 puntos**) Tenemos una clase genérica `DoubleLinkedList<T>` que implementa completamente la interfaz `List<T>` mediante nodos doblemente enlazados circularmente. La definición de su parte privada es la siguiente:

```
public class DoubleLinkedList<T> implements List<T> {
        private static class Node<T> {
                public T data;
                public Node<T> next;
                public Node<T> prev;
                ...
        }
        private Node<T> head = null;
        private int size = 0;
    ...
}
```

Añade un método a esta clase para mezclar listas:

```
public void shuffle(List<T> lista)
```

El método tomará una lista `lista` y la mezclará con lista local `this`, modificándola. De tal forma que el primer elemento de `lista` se insertará entre el primero y el segundo de la lista local, el segundo elemento de `lista` entre el segundo y el tercero de la lista local, y así sucesivamente. Si la lista local fuese demasiado corta se insertarían los elemento sobrante de `lista` al final de la lista local.

Ejemplos:

- `lista: [2, 4, 5]; this: [9, 8, 7, 6]` → `this: [9, 2, 8, 4, 7, 5, 6]`
- `lista: [2, 4, 5]; this [99, 100]` → `this: [99, 2, 100, 4, 5]`

4. (**3 puntos)**Deseamos crear un clase que almacene los resultados de las elecciones generales por provincias y partidos, de tal forma que por cada provincia se almacenen los datos obtenidos por cada partido presentado en esa provincia. El número de partidos presentados en cada provincia puede ser variable, y no todos los partidos se presentan en cada provincia. Las estructuras elegidas para representar los datos deberán ser las más eficientes computacionalmente para resolver todas los métodos que se piden en el ejercicio. **Lee detenidamente todos los apartados antes de empezar la solución.**

   a) Define una clase `EleccionesGenerales` que almacene los resultados de todas las provincias. Los métodos deberán optimizar el acceso a los datos por el nombre de provincia. Deberás definir el constructor por defecto de la clase y los siguientes métodos básicos:

   - `public void IncluirProvincia(String provincia)`.
     Añadirá una nueva provincia a la estructura de datos.
   - `public void VotosPartido(String Provincia, String partido, int votos)`.
     Guardará los votos obtenidos por el partido en esa provincia.

   b) Añade un método a la clase que dado el nombre de una provincia muestre por pantalla todos los partidos que han obtenido votos en dicha provincia, así como el número de votos y el porcentaje respecto al total de votos de la provincia. Los partidos deberán aparecer ordenados alfabéticamente.

   ```
   public void MostrarVotosProvincia(String provincia);
   ```

   c) Añade un método que dado un partido calcule el número de votos totales.

   ```
   public int VotosTotal(String partido);
   ```

   d) Añade un método que devuelve una colección con todos los totales de todos los partidos.