Apellidos: ***Nombre:***
Grupo: ***DNI:***
Duración: 2:30 h.

Ejercicio 1	Ejercicio 2	Ejercicio 3	TOTAL

1.1: **[1 punto]** Dada una tabla hash de tamaño TAM_TABLA = 16, donde las colisiones se resuelven prueba cuadrática y la función de dispersión viene definida por:

$$H(k) = (\text{DNI } \texttt{mod } 100) \texttt{ mod } (\text{TAM\_TABLA}),$$

Determinar cómo quedaría dicha tabla después de la siguiente secuencia de inserciones

Seq.	Nombre	Apellido	DNI
1	Mariano	Rajoy	41.002.103
2	Alfredo	P. Rubalcaba	8.424.220
3	Cayo	Lara	85.669.039
4	Josep A.	Durán	35.875.692
5	Rosa	Díez	41.002.119

1.2: **[1 punto]** Dada la siguiente secuencia de inserciones -5, -7, -6, 0 y -3, construir un árbol AVL. Eliminar posteriormente, del árbol construido, el nodo -7. Para que el ejercicio pueda ser considerado válido es necesario mostrar gráficamente cada uno de los pasos intermedios

1.3: **[1 punto]** La compañia aérea EDA_URJC se encarga de operar vuelos con orgien en Madrid y destino en algunas de las ciudades más importantes del mundo. Para agilizar algunos procesos, esta compañía quiere almacenar todos sus vuelos en un diccionario ordenado basado en un árbol Rojo-Negro. Asumiendo que la clave de dicho diccionario es la duración del trayecto de Madrid a cualquier otra ciudad, representar gráficamente cómo sería la correspondiente estructura de datos considerando los siguientes vuelos.

NOTA. Considerese que los vuelos se introducen en el diccionario en el orden en el que se presentan en la tabla adjunta

Nº Horas	Destino	Tipo Avión	Nº Pasajeros
2	Barcelona	Boening 727	125
5	Munich	Boeing727	125
10	Tokio	Airbus 340	300
13	Singapure	Airbus 340	300
1	Valencia	Boeing 727	125
3	Paris	Boeing 727	125
12	Pekin	Airbus 340	300
16	Melbourne	Airbus 340	300
14	Tasmania	Airbus 340	300
15	Honolulu	Airbus 340	300

1.4: **[1 punto]** La base de datos IMDB_URJC dispone de un fichero de datos con nombres de actores y sueldo que cobra (en promedio) por cada película en la que participa. El fichero tiene un aspecto como el siguiente:

	Nombre	**Apellido**	**Sueldo (M $)**
0	Abraham	Duarte	1
1	Raúl	Cabido	2
2	José	Vélez	3
3	Ángel	Sánchez	4
4	Hulk	Hoogan	5
5	Santiago	Segura	6
6	Eric	Roberts	7
7	Novak	Djokovic	8
8	Amanda	Ooms	9
9	Nan	Yu	10
10	Randy	Couture	11
11	Scott	Adkins	14
12	Terry	Crews	15
13	Liam	Hemsworth	16
14	Mickey	Rourke	17
15	Jason	Statham	18
16	Jet	Li	19
17	Dolph	Lundgren	20
18	Chuck	Norris	22
19	Jean-Claude	Van Damme	26
20	Bruce	Willis	27
21	Sylvester	Stallone	28
22	Arnold	Schwarzenegger	32

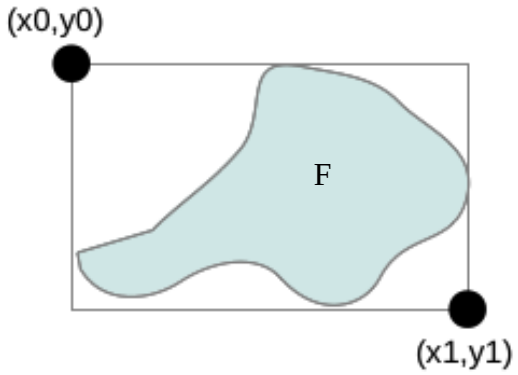
Para poder hacer consultas eficientes, esta empresa quiere costruir índices mediante árboles B en los que la clave sea el sueldo en M$. En la figura adjunta se muestra el índice asociado. Se pide, representar árbol B asociado (con estructura de fichero) después del borrado en el índice de las claves 22, 15 y 1.

NOTA: Para que el ejercicio se evaluado correctamente, es necesario representar todos los pasos intermedios.

H1	**Clave1**	**Pos1**	**H2**	**Clave2**	**Pos2**	**H3**	**Clave3**	**Pos3**	**H4**	**Clave4**	**Pos4**	**H5**	**Padre**
1	3	2	2	6	5	3	-1	-1	-1	-1	-1	-1	4
-1	1	0	-1	2	1	-1	-1	-1	-1	-1	-1	-1	0
-1	4	3	-1	5	4	-1	-1	-1	-1	-1	-1	-1	0
-1	7	6	-1	8	7	-1	9	8	-1	10	9	-1	0
0	11	10	5	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
6	16	13	7	22	18	8	-1	-1	-1	-1	-1	-1	4
-1	14	11	-1	15	12	-1	-1	-1	-1	-1	-1	-1	5
-1	17	14	-1	18	15	-1	19	16	-1	20	17	-1	5
-1	26	19	-1	27	20	-1	28	21	-1	32	22	-1	5

Ejercicio 2 [3 puntos].

Tras el análisis de cierto tipo de imágenes digitales usualmente se obtienen 30000 Bounding Boxes. El *Bounding Box, BB,* de una figura es el menor rectángulo que contiene dicha figura. Cada BB queda completamente descrito por las coordenadas de dos puntos que delimitan el tamaño del rectángulo asociado. Por ejemplo, en la Figura adjunta, los puntos (x0,y0) y (x1,y1) describen un BB de la Figura F.



Tras realizar el análisis de una imagen, se desea almacenar todos los BB obtenidos en una estructura de datos, de manera que luego sea eficiente localizar aquellos BB cuya área sea igual a un área dada.

Se pide:

Implementar la clase `BoundingBox` y la clase `ContenedorDeBoundingBox`, que utilice las estructuras de datos vistas en clase, para contener todas las BB. Dentro de dicha clase, deberá existir un método público que recibirá un entero, representando un área, y devolverá los objetos BB que tengan esa área.

La elección de las estructuras de datos que se usen deberá realizarse de manera que se minimicen los tiempos de búsqueda. El diagrama clases adjunto contiene las estructuras de datos vistas en clase. El diagrama no incluye los parámetros de los métodos ni los tipos devueltos. Cada vez que se use una función deberán suponerse dichos elementos. La calidad de las suposiciones se tendrá en cuenta en la evaluación.

Ejercicio 3 [3 puntos].

El cacheado de ficheros en servidores web reduce el coste de tener que cargar dicho fichero de disco cada vez que se realiza una petición. Una caché MRU (*Most Recently Used*) permite establecer estrategias acerca de qué ficheros (objetos `MyFile`) permanecen en cache, controlando así su tamaño. En concreto, cuando se hace la petición al servidor web de un fichero concreto puede ocurrir que:

- No se encuentra en la cache
  - Si la caché no está llena, el fichero se lee de disco y se guarda en la caché. A continuación se devuelve el fichero consultado (objeto `MyFile`).
  - Si la caché está llena, el fichero cacheado que lleve más tiempo sin ser accedido se elimina. A continuación, el fichero accedido se lee de disco y se guarda en la caché. Por último se devuelve el fichero consultado (objeto `MyFile`).
- Sí se encuentra en la caché. Entonces se recupera y se devuelve el fichero consultado (objeto `MyFile`).

El comportamiento de una cache MRU queda descrito por la clase `MRUCache` que se presenta al final del enunciado. Esta clase contiene un método `getFile()` que dado el nombre de un fichero (`fname`) permite recuperarlo (bien de cache o bien de disco) según los criterios descritos anteriormente. Para poder recuperar el fichero de disco, la clase `MRUCache` contiene un método `readFileFromDisk()` que dado el nombre del fichero es capaz de leer su contenido y almacenarlo en un objeto de la clase `MyFile`. Por último, es posible mostrar el estado de la cache (nombre de los ficheros cacheados en un instante concreto) llamando al método `printMRU()`.

Se pide:

1) Seleccionar la/s estructura/s de datos necesarias para implementar la cache MRU.

2) Añadir los atributos necesarios para que la clase `MRUCache` pueda hacer uso de las estructuras de datos seleccionadas en el punto anterior.

3) Implementar los métodos `getFile()` y `printMRU()`.

NOTA: se requiere que las consultas de cache tengan lugar en el menor tiempo posible, por lo que se tendrá en cuenta la complejidad algorítmica de la solución dada.

```
class MyFile {
  long lastmodified;
  String contents;

  public MyFile(long last, String data) {
    lastmodified=last;
    contents=data;
  }
  public long getLastModified() {return lastmodified;}
  public String getContents() {return contents;}
}

public class MRUCache {

  int cachesize;

  public MRUCache(int max) {cachesize=max;}

  public String getFile(String fname){}
  protected MyFile readFileFromDisk(String fname) {}
  public void printMRU() {}

  public static void main(String args[]) {
    // Number of entries in MRU cache is set to 10
    MRUCache h1=new MRUCache(10);
    for(int i=1;i<=20;i++) {
      // files are stored in a subdirectory called data
      h1.getFile("data"+File.separatorChar+i);
    }
      h1.printMRU();
  }
}
```