



## **Práctica Final: Jugador automático para “¿Quién es Quién?”**

Dpto. Ciencias de la Computación e Inteligencia Artificial  
E.T.S. de Ingenierías Informática y de Telecomunicación  
Universidad de Granada



## **Estructuras de Datos**

Doble Grado en Ingeniería Informática y  
Matemáticas  
Grado en Ingeniería Informática

# Índice de contenido

<b>1. Introducción</b>	<b>3</b>
<b>2. El problema: Juego del “¿Quién es quién?”</b>	<b>3</b>
<b>3. Definición del tablero de personajes</b>	<b>3</b>
<b>4. Jugador automático para “¿Quién es quién?”</b>	<b>4</b>
4.1. Lectura del tablero.	4
4.2. Representación de las preguntas para adivinar todos los personajes del tablero.	4
4.3. Proceso de juego de una partida de “¿Quién es Quién?”.	5
4.4. Construcción del árbol de preguntas.	6
Construcción básica.	6
Omitir preguntas inútiles.	6
Mejora: Elegir la pregunta más adecuada.	7
4.5. Métodos sobre el árbol de preguntas.	8
Métodos Adicionales	8
<b>5. Se entrega / Se pide.</b>	<b>9</b>
Se entrega	9
Se pide	9
Normas de la entrega	9
<b>6. Referencias</b>	<b>10</b>

## 1. Introducción

El objetivo de este guión de prácticas es utilizar un TDA Árbol Binario (TDA. bintree) para implementar un jugador automático para el juego “¿Quién es quién?”.

Los requisitos para poder realizar esta práctica son:

1. Haber estudiado el Tema 1: Introducción a la eficiencia de los algoritmos
2. Haber estudiado el Tema 2: Abstracción de datos. Templates.
3. Haber estudiado el Tema 3: T.D.A. Lineales.
4. Haber estudiado el Tema 4: STL e Iteradores.
5. Haber estudiado estructuras de datos jerárquicas: Árboles

## 2. El problema: Juego del “¿Quién es quién?”

“¿Quién es quién?” (en inglés “Guess who?”), es un juego de adivinanzas para dos jugadores, concebido y fabricado por primera vez por [Milton Bradley](#) en los años 80 y actualmente fabricado y distribuido por [Hasbro](#).

En este juego, cada jugador dispone de un tablero idéntico que contiene 24 dibujos de personajes identificados por su nombre. El juego empieza al seleccionar cada jugador una carta al azar de una pila separada de cartas, que contiene las mismas 24 imágenes. El objetivo del juego consiste en ser el primero en determinar qué carta seleccionó el oponente. Esto se consigue haciendo preguntas, una por turno, cuya respuesta puede ser sí o no, para eliminar candidatos.

Un ejemplo de pregunta es «¿*Tu personaje tiene bigote?*». La respuesta de nuestro oponente nos permitirá eliminar los personajes que no cumplan el criterio, tumbando estas tarjetas del tablero.

El estudiante debe implementar un jugador automático para jugar al “¿Quién es quién?”. En particular, el jugador automático debe hacer las preguntas, recoger la respuesta “Sí”/“No” del jugador humano, y continuar haciendo preguntas hasta deducir el personaje con el que el jugador humano está jugando.


## 3. Definición del tablero de personajes

Para poder jugar al juego “¿Quién es quién?” primero definiremos un tablero de personajes utilizando un formato de texto plano que se indica a continuación.

```
Atributo1 \t Atributo2 \t ... \t AtributoK \t Nombrepersonaje \n
v11 \t v12 \t ... \t v1k p1 \n
v21 \t v22 \t ... \t v2k p2 \n
...
vN1 \t vN2 \t ... \t vNk pN \n
```

Este fichero define un tablero de  $N$  personajes y  $k$  atributos diferentes, donde  $v_{ij}$  toma un valor en  $\{0, 1\}$  ( $0$ =“No”,  $1$ = “Sí”), indicando si el personaje  $i$ , de nombre  $p_i$ , presenta o no el atributo  $j$ . El número máximo de personajes que pueden definirse con  $K$  atributos binarios es  $2^K$ . Para que un tablero sea válido, es obligatorio que dos personajes distintos tengan, al menos, un atributo distinto.

Por ejemplo, el siguiente tablero (Tablero 1) define 5 personajes diferentes utilizando 4 atributos ( $N=5$ ,  $k=4$ ):



Mujer	Ojos marrones	Pelo castaño	Tiene gafas	Nombre personaje
0	1	1	1	Ernesto
0	0	0	0	Ana
0	0	1	1	Juan
0	1	0	0	Antonio
1	1	1	0	Pilar

Tablero 1.

Este fichero con la definición del tablero de personajes debe ser conocido por parte del jugador humano y del jugador automático, ya que el juego se desarrollará partiendo de esta descripción de personajes.

## 4. Jugador automático para “¿Quién es quién?”

Para implementar un jugador automático que juegue a “¿Quién es quién?” el estudiante completará la clase QuienEsQuien (que se proporciona en el material adjunto a la práctica) para la que se propone la siguiente representación:

```

Class QuienEsQuien{
private:
    vector<string> personajes;    // nombres de los personajes
    vector<string> atributos;    // nombres de los atributos
    vector<vector<bool>> tablero;
        // tablero de características para cada personaje
        // tamaño: size(personajes) x size(atributos)

    bintree<Pregunta> arbol;    // árbol de preguntas para adivinar personajes
    bintree<Pregunta>::node jugada_actual;    // jugada actual
};

```

En esta sección se describen cada uno de los elementos considerados en esta representación, así como algunos de los métodos de la clase. Toda la información está en la especificación de la clase QuienEsQuien (carpeta *include/quienesquien.h*).

### 4.1. Lectura del tablero.

La lectura de los ficheros de tablero (como el representado en el cuadro Tablero 1) está ya implementada en el método:

```
friend istream& operator >> (istream& is, QuienEsQuien &quienEsQuien);
```

de la clase QuienEsQuien. Este método carga los datos del tablero en las variables *tablero*, *personajes* y *atributos* del objeto de clase QuienEsQuien. Puede consultar su especificación e implementación en los ficheros *quienesquien.h* y *quienesquien.cpp*.

### 4.2. Representación de las preguntas para adivinar todos los personajes del tablero.

Para implementar un jugador automático que juegue a “¿Quién es quién?” el estudiante construirá un árbol binario para representar la sucesión de preguntas necesaria para adivinar cada personaje del tablero. Este árbol binario se almacena en el atributo *árbol* de la clase QuienEsQuien. Para ello, el campo etiqueta de cada nodo *n* de este árbol almacenará elementos de tipo Pregunta, definido como:

```
Class Pregunta{
```

```

string atributo;           // atributo sobre el que se pregunta en este nodo
int num_personajes;       // número de personajes que quedan en el tablero
};

```

El nodo raíz del árbol codifica la primera pregunta que formula el jugador automático. Los nodos hijos del nodo raíz (nodos de nivel 2) representan la segunda pregunta que formula el jugador automático, y así sucesivamente. La Figura 1 muestra un esquema de esta representación aplicada al *Tablero 1*.

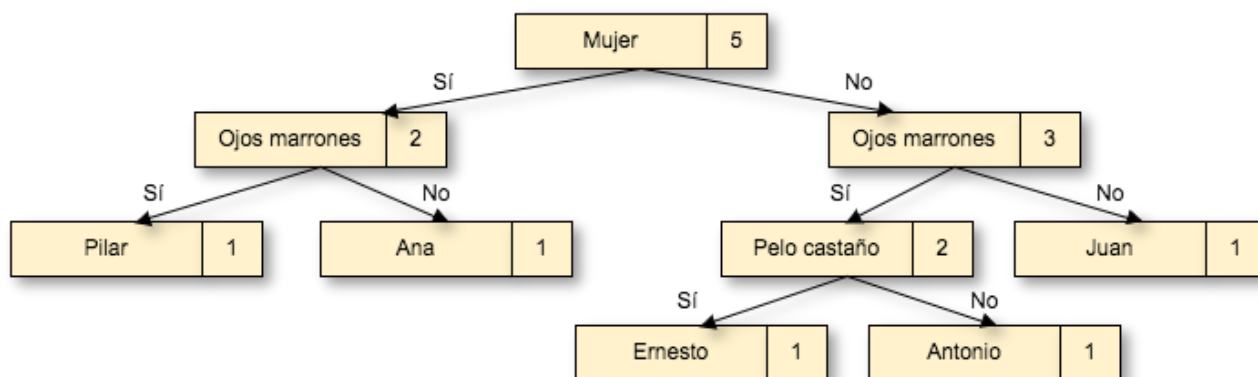


Figura 1. Árbol de preguntas asociadas a Tablero 1. El árbol se ha construido formulando preguntas sobre los atributos de Tablero 1 según el orden en el que vienen dados en el fichero.

Cada nodo  $n$  del árbol codifica una pregunta asociada al juego y el número de personajes que aún no han sido eliminados. El hijo izquierda de  $n$  representa el tablero de personajes que tienen respuesta “Sí” a la pregunta formulada en  $n$ , y el hijo derecha el tablero de personajes que se obtiene con la respuesta “No” a la pregunta. Los nodos hoja del árbol codifican un tablero en el que sólo queda un personaje que no ha sido descartado. En este caso, el campo atributo debe contener el nombre del personaje en cuestión y el campo num\_personajes es igual a 1.

Por tanto, el camino desde la raíz del árbol hasta  $n$  determina la sucesión de preguntas y respuestas obtenidas hasta llegar a ese tablero, y los nodos hoja descendientes de  $n$  determinan los personajes aún no tumbados en el tablero asociado a  $n$ .

### 4.3. Proceso de juego de una partida de “¿Quién es Quién?”.

Una vez construido este árbol con sucesiones de preguntas para identificar cada personaje, una partida del juego se implementa como un recorrido por este árbol, comenzando desde la raíz y hasta alcanzar un nodo hoja. De este modo, durante el transcurso de la partida, será necesario registrar *por cuál nodo del árbol va la partida*. Este es el uso que daremos al atributo *jugada\_actual* de la clase *QuienEsQuien*. Al inicio de la partida, *jugada\_actual* se inicializa con el nodo raíz del árbol. El jugador automático formula la pregunta asociada al este nodo, y la respuesta del usuario determina el camino a seguir (Sí = hijo izquierda, No = hijo derecha). Este proceso se continúa hasta que *jugada\_actual* alcanza un nodo hoja. En ese momento, el jugador automático ya conoce la respuesta, y la imprime en pantalla: “¡Ya lo sé! tu personaje es X.” donde X se reemplaza por el nombre del personaje almacenado en la etiqueta del nodo hoja. Con esto acaba la partida, *jugada\_actual* se reinicia al nodo raíz del árbol y se da al usuario la opción de jugar otra vez.

#### 4.4. Construcción del árbol de preguntas.

##### Construcción básica.

Para implementar un jugador automático básico para el juego del “Quién es quién” construiremos el árbol binario descrito en la sección 4.2 a partir de la información almacenada en los atributos *tablero*, *personajes* y *atributos*. La forma más sencilla de construir el árbol es formular la pregunta asociada al atributo  $i$  (con  $1 \leq i \leq \text{size}(\text{atributos})$ ) en todas las ramas de nivel  $i$  del árbol binario (excepto las hojas), tal y como se muestra en la Figura 1. Es decir, en el primer nivel (nodo raíz), formulamos la pregunta asociada al atributo que ocupa la primera columna del fichero; en el segundo nivel, la pregunta asociada al segundo atributo, y así sucesivamente hasta llegar a los nodos hoja. Nótese que con esta propuesta se hacen siempre las mismas preguntas y en el mismo orden, sean cuales sean las respuestas recibidas (es decir, para todas las ramas del árbol hasta llegar a cada hoja).

##### Omitir preguntas inútiles.

Esta solución puede llevar a situaciones donde se haga una pregunta cuya respuesta sea la misma para todos los personajes que aún no han sido tumbados, por lo que esta pregunta podría ser omitida. Por ejemplo, si en el nivel 1 se pregunta “¿es Mujer?”, no tendría sentido preguntar, a continuación: “¿es Hombre?”. Esta situación se muestra en el *Tablero 2* y gráficamente en la Figura 2.

Mujer	Hombre	Ojos marrones	Ojos negros	Pelo castaño	Personaje
1	0	1	0	1	Pilar
1	0	0	1	1	Ana
0	1	1	0	1	Juan
0	1	0	1	1	Ernesto
0	1	0	1	0	Antonio

Tablero 2.

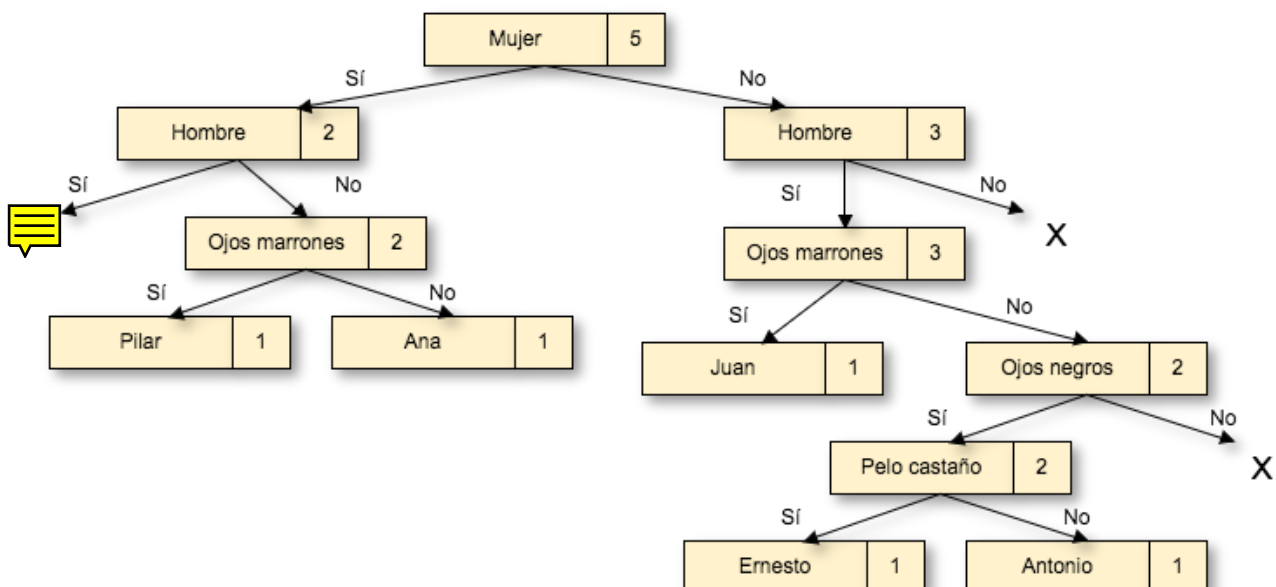


Figura 2. Árbol de partidas asociadas a Tablero 2. El árbol se ha construido formulando preguntas sobre los atributos de Tablero 2 según el orden en el que vienen dados en el fichero.

Se propone una mejora sencilla para omitir preguntas inútiles. Una vez construido el árbol siguiendo las indicaciones de la sección “Construcción Básica”, el alumno puede efectuar un recorrido por todos los nodos del árbol, detectando si existe algún nodo no hoja que sólo tenga un hijo. Si se detecta este caso, ese nodo puede ser reemplazado por su hijo (Figura 3). En la práctica, esto supone que se omiten aquellas preguntas cuyas respuestas no discriminan a, al menos, un personaje no tumbado del resto de personajes no tumbados. El resultado se muestra en la Figura 4.

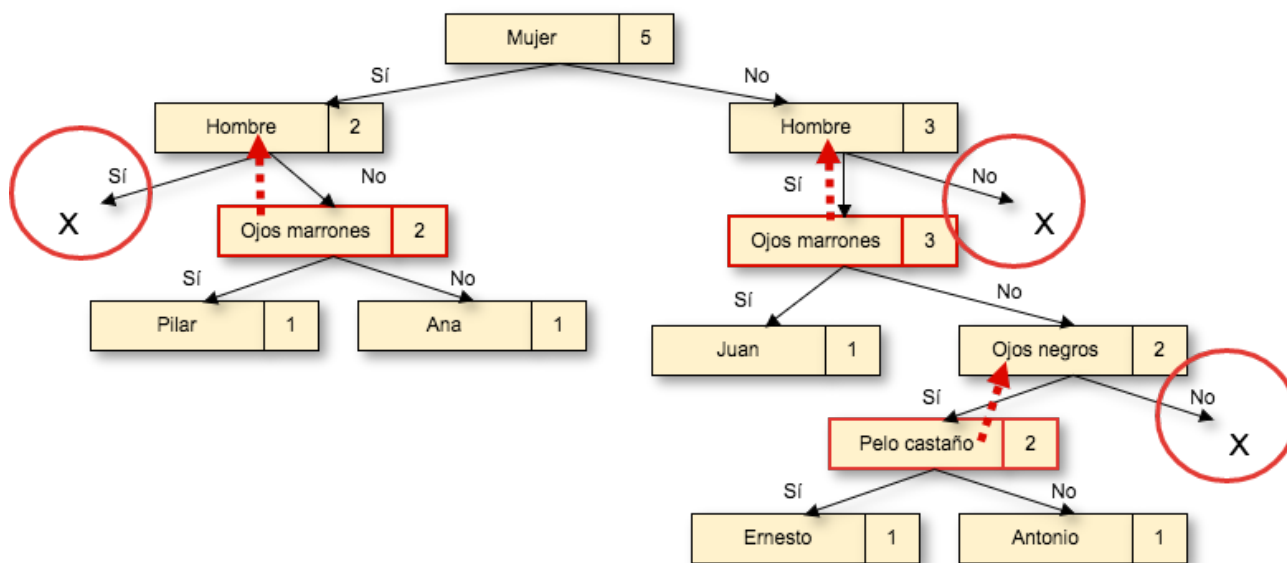


Figura 3. Un recorrido por el árbol binario permite detectar nodos que sólo tienen un hijo. Estos nodos codifican preguntas que podrían omitirse. Para ello, se reemplaza dicho nodo por su hijo.

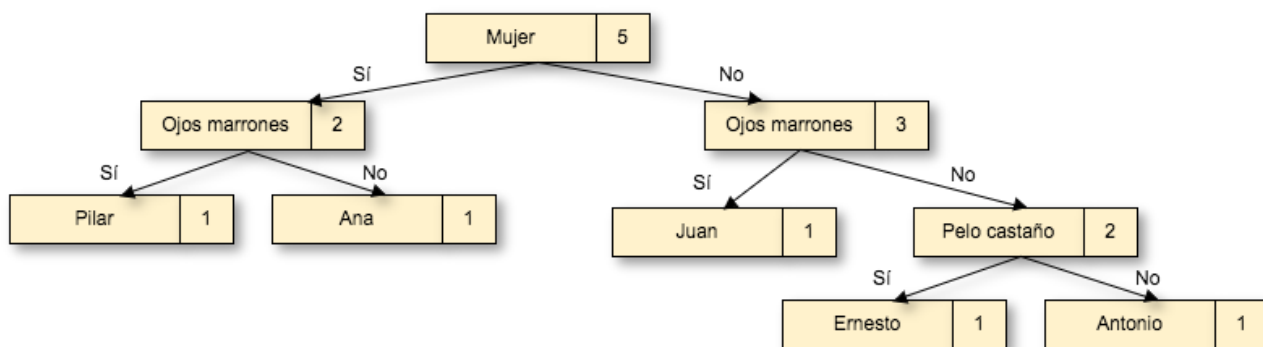


Figura 4. Resultado de la eliminación de nodos no hoja que sólo tienen un hijo. El árbol resultante es más compacto que el de las figuras 2-3.

## Mejora: Elegir la pregunta más adecuada.

Una mejora sustancial en la construcción del árbol pasaría por elegir, en cada momento, la pregunta que divide al conjunto de personajes no tumbados en dos subconjuntos de tamaño similar. De este modo, el árbol obtenido está mejor balanceado y tendríamos que formular menos preguntas, en promedio, para resolver partidas sobre cualquier personaje. Esta idea intuitiva es la que subyace bajo el concepto de [Entropía](#) en Teoría de la Información y la construcción de [Árboles de Decisión](#) en Inteligencia Artificial. Os invitamos a explorar estos conceptos para implementar una métrica para elegir la mejor pregunta en cada momento.

Para ello, pudiera ser necesario modificar la representación de la clase *QuienEsQuien* o

*Pregunta* para representar las características de los personajes que quedan sin tumbar en cada momento. La representación y la implementación de estas mejoras queda a vuestra elección.

#### **4.5. Métodos sobre el árbol de preguntas.**

Se propone al alumno que implemente los siguientes métodos sobre el árbol de preguntas para la clase `QuienEsQuien`:

##### **`QuienEsQuien::crear_arbol ( )`**

Este método construye el árbol de preguntas para todos los personajes del tablero, según lo descrito en las secciones 4.2 y 4.4.

##### **`QuienEsQuien::iniciar_juego ( )`**

Este método implementa el desarrollo de una partida contra un usuario humano, tal y como se describe en la sección 4.3.

##### **`QuienEsQuien::informacion_jugada( bintree<Pregunta>::node jugada )`**

Este método se aplica sobre un nodo del árbol de preguntas (jugada) para obtener un `set<string>` con los nombres de los personajes que aún no han sido tumbados en el tablero.

##### **`QuienEsQuien::profundidad_promedio_hojas()`**

Este método permite calcular la media de la profundidad de las hojas del árbol. Este valor representa el número (promedio) de preguntas necesarias para adivinar cada personaje. A menor profundidad promedio, mejor solución. Esta métrica es un indicador para evaluar la calidad de vuestra solución.



#### **Métodos Adicionales**

##### **`QuienEsQuien::preguntas_formuladas ( bintree<pregunta>::node jugada )`**

Este método se aplica sobre un nodo del árbol de preguntas (jugada) para obtener una descripción de las preguntas formuladas anteriormente y las respuestas dadas por el usuario hasta ahora. Por ejemplo:

“El personaje oculto tiene las siguientes características:

Mujer - no

Ojos Marrones - si

pero aún no sé cuál es”.

##### **`QuienEsQuien::aniade_personaje (string nombre, vector <bool> caracteristicas)`**

Dado un árbol ya construido para un tablero, imaginemos que tenemos que añadir un nuevo personaje a partir de su nombre y descripción asociada (`vector<bool>`), pero *sin rehacer el árbol completo*. Este método inserta el personaje nuevo en el árbol ya construido. Para ello, es necesario utilizar las características del nuevo personaje para recorrer el árbol desde la raíz y encontrar dónde insertar el nuevo personaje. Típicamente, este recorrido por el árbol nos llevará a un nodo hoja (otro personaje del tablero) y tendremos que añadir un nivel más (una pregunta más) para distinguir al nuevo personaje del nodo hoja que ya estaba en el árbol. Se asume que el `vector<bool>` de características tiene el mismo tamaño y los atributos están en el mismo orden que el especificado en el fichero de entrada.



### **QuienEsQuien::elimina\_personaje (string nombre)**

Este método elimina un personaje del árbol ya construido, siguiendo un proceso similar al descrito en el método anterior: utilizaremos las características del personaje para recorrer el árbol desde la raíz y encontrar su nodo hoja asociado. Al eliminarlo, típicamente tendremos que eliminar también a su padre (porque ahora será una pregunta inútil), o mejor dicho: reemplazar a su padre por su otro nodo hijo.

## **5. Se entrega / Se pide.**


### **Se entrega**

Para la implementación de esta práctica, se proporciona el siguiente material:

- Clases bintree y node: implementación del TDA bintree
- Clase QuienEsQuien: almacena una representación del tablero, los personajes y sus atributos, además de un bintree con las preguntas necesarias para adivinar todos los personajes de un tablero dado. **Esta clase está incompleta.**
- Clase Pregunta: representa un nodo del árbol de preguntas de QuienEsQuien.
- Programa principal guesswho.cpp: implementa un programa que lee un fichero con una configuración de tablero y crea un objeto QuienEsQuien para que el usuario juegue contra un jugador automático.
- Ficheros con tableros de ejemplo (carpeta /datos)
- Función tablero\_aleatorio para generar tableros aleatorios de tamaño dado.
- Makefile

### **Se pide**

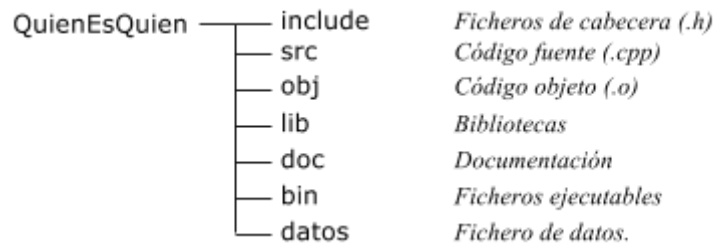
La práctica tiene una puntuación total de 1,5 puntos. Para obtener esta puntuación, se pide:

- Completar la implementación de la Clase QuienEsQuien, en particular, de los métodos descritos en la sección 4.5. **La correcta implementación de estos métodos permitirá obtener una puntuación de 1 punto en la nota de la práctica.**
- Completar la implementación de las “Mejoras” y “Métodos Adicionales” (marcadas con el icono  ) propuestos en las secciones 4.4 y 4.5. **La correcta implementación de estos métodos permitirá obtener una puntuación adicional de 0,5 puntos en la nota de la práctica.**
- Entregar una memoria donde se describa la solución propuesta y cualquier mejora, en su caso, que haya sido implementada para construir el árbol de preguntas.

### **Normas de la entrega**

El alumno deberá empaquetar todos los archivos relacionados en el proyecto en un archivo con nombre “QuienEsQuien.tgz” y entregarlo antes de la fecha que se publicará en la página web de la asignatura. Tenga en cuenta que no se incluirán ficheros objeto, ni ejecutables, ni la carpeta datos. Es recomendable que haga una “limpieza” para eliminar los archivos temporales o que se puedan generar a partir de los fuentes.

El alumno debe incluir el archivo *Makefile* para realizar la compilación. Tenga en cuenta que los archivos deben estar distribuidos en directorios:



Para realizar la entrega, en primer lugar, realice la limpieza de archivos que no se incluirán en ella, y sitúese en la carpeta superior (en el mismo nivel de la carpeta “*QuienEsQuien*”) para ejecutar:

```
prompt% tar zcv QuienEsQuien.tgz QuienEsQuien
```

tras lo cual, dispondrá de un nuevo archivo QuienEsQuien.tgz que contiene la carpeta QuienEsQuien así como todas las carpetas y archivos que cuelgan de ella.

## 6. Referencias

- [GAR06b] Garrido, A. Fdez-Valdivia, J. “*Abstracción y estructuras de datos en C++*”. Delta publicaciones, 2006.