# Práctica Final Árboles
# Estructura de datos
# 2017 – 2018

***Grupo:*** *A-2*
***Integrantes:*** *Ángel Barrilao Bensrhir, José Luis Pedraza Román, Jorge García Moreno(GrupoA1)*

# Primeros pasos

Antes de abordar el problema hemos esquematizado y estudiado con papel y lápiz como seria previamente el funcionamiento del árbol, describiendo con todo lujo de detalles los posibles casos que ocurrirían por ejemplo cuando insertamos o podamos una rama.

El principal reto sin duda, ha sido entender y controlar bien la naturaleza de la recursividad ya que no estábamos muy familiarizados con ella. También hay que mencionar la dificultad del método encargado de podar los nodos que posean hijos nulos, debido al numero de posibles casos que se pueden dar.

Por lo general ha sido una practica amena, bastante clara e intuitiva.

Vamos a abordar en este mini-manual los métodos principales de la práctica, **crear_arbol** y **eliminar_nodos_redundantes** ya que una vez entendidos y funcionando, el resto se convierte mas fácil de implementar.

# Métodos principales

## 1.- QuienEsQuien::crear_arbol( )

Este método es el encargado de crear el árbol. Hemos decidido hacerlo iterativo ya que en un principio lo hemos encontrado mas intuitivo que hacerlo de forma recursiva.

Su funcionamiento básicamente consiste en:

1) Recorrer las filas y columnas de la matriz del tablero

2) Por cada posición comprobamos si es verdadero o falso, para determinar en donde vamos a insertar la pregunta.

3)Comprobamos que cuando vallamos a insertar no halla ningún nodo. En caso contrario simplemente avanzamos al siguiente nodo correspondiente.

4)Una vez insertadas las preguntas comprobamos la ultima posición e insertamos el personaje.

5)Finalmente llamamos a la función recursiva para actualizar el numero de jugadores de cada nodo.

**Código:**

```
bintree<Pregunta> QuienEsQuien::crear_arbol()
{

    //TODO :D:D
    //1. El primer atributo lo ponemos como raiz.
    Pregunta pr(atributos[0],personajes.size());
    bintree<Pregunta>aux(pr);
    arbol=aux;

    //2.insertamos los jugadores en las hojas correspondientes, siguiend
    bintree<Pregunta>::node n;
    bintree<Pregunta>::node n_anterior;
    int k,pos;

    for(int i=0;i<tablero.size();i++){
        n=arbol.root();
        pos=1;
        for(int j=0;j<tablero[i].size();j++){
            n_anterior=n;
            if(tablero[i][j]){
                if(n.left().null()){
                    if(pos<atributos.size()){
                        Pregunta pr(atributos[pos],personajes.size());
                        arbol.insert_left(n,pr);
                    }
                }
                n=n.left();
            }else{
                if(n.right().null()){
                    if(pos<atributos.size()){
                        Pregunta pr(atributos[pos],personajes.size());
                        arbol.insert_right(n,pr);
                    }
                }
                n=n.right();
            }
            k=j;
            pos++;
        }
        Pregunta pe(personajes[i],1);
        if(tablero[i][k]){//llegamos a la hoja
            arbol.insert_left(n_anterior,pe);//insertamos el personaje
        }else{
            arbol.insert_right(n_anterior,pe);
        }
    }

    //3.Contamos el numero de jugadores(hojas no nulas) y actualizamos ca
    actualizar_numero_jugadores_activos(arbol.root());

    return arbol;
}
```

## 2.- QuienEsQuien::eliminar_nodos_redundantes()

Este método es fundamental para mejorar sustancialmente el árbol. Consiste en ir visitando los nodos e ir comprobando si alguno de sus hijos es nulo o lo que es lo mismo , si tiene solo un hijo para así podarlo y reconectarlo con su ancestro correspondiente.
Finalmente llamamos a la función actualizar_numero_jugadores_activos para que vuelva a actualizar los nodos con sus jugadores(nodos hojas) tras los ajustes realizados.
**Código:**

```
void QuienEsQuien::eliminar_nodos_redundantes(){
    eliminar_nodos_redundantes(arbol.root());
    actualizar_numero_jugadores_activos(arbol.root());
}
```

Para hacer dicha tarea hemos implementado dos métodos adicionales realacionados con la función **eliminar_nodos_redundantes:**

**1) ajuste_nodos_nulos(bintree<Pregunta>::node n)**: Este método es el encargado de evaluar, podar y conectar el nodo que le pasamos , teniendo en cuenta donde está conectado, según su padre y volverlo a insertar en la rama que le corresponde. Devolverá verdadero o falso si ha ajustado o no el nodo actual.

**Codigo:**

```
bool QuienEsQuien::ajuste_nodos_nulos(bintree<Pregunta>::node n){
    bintree<Pregunta> otroArbol;
    bool control=false;

    if(tengoHijosNulos(n)){
        if(n != arbol.root()){
            //Determino mi posicion con respecto a mi padre
            if(n.parent().left()==n){//Si soy de izq.
                if(!(n.left().null()) && n.right().null()){
                    arbol.prune_left(n,otroArbol);
                    arbol.insert_left(n.parent(),otroArbol);
                    control=true;
                }else if(!(n.right().null()) && n.left().null()){
                    arbol.prune_right(n,otroArbol);
                    arbol.insert_left(n.parent(),otroArbol);
                    control=true;
                }
            }else if(n.parent().right()==n){//Si soy de der.
                if(!(n.left().null()) && n.right().null()){
                    arbol.prune_left(n,otroArbol);
                    arbol.insert_right(n.parent(),otroArbol);
                    control=true;
                }else if(!(n.right().null()) && n.left().null()){
                    arbol.prune_right(n,otroArbol);
                    arbol.insert_right(n.parent(),otroArbol);
                    control=true;
                }
            }
        }else{
            //Determino mi posicion con respecto a mi padre
            if(!(n.left().null()) && n.right().null()){
                arbol.prune_left(n,otroArbol);
                arbol=otroArbol;
                control=true;
            }else if(!(n.right().null()) && n.left().null()){
                arbol.prune_right(n,otroArbol);
                arbol=otroArbol;
                control=true;
            }
        }
    }
    return control;
}
```

**2) eliminar_nodos_redundantes(bintree<Pregunta>::node n):** Se encarga de ir recorriendo el arbol en busca de un nodo que le falte algun hijo. Llamamos a la función **ajuste_nodos_nulos** que nos devuelve si se ha producido ajuste o no . En el caso de que se produjese el ajuste volvemos a empezar desde la raíz para así poder evaluar correctamente el nuevo árbol formado tras el ajuste.

**Código:**

```
void QuienEsQuien::eliminar_nodos_redundantes(bintree<Pregunta>::node n){
    if(!n.null()){
        if(ajuste_nodos_nulos(n)){
            eliminar_nodos_redundantes(arbol.root());
        }
        eliminar_nodos_redundantes(n.left());
        eliminar_nodos_redundantes(n.right());
    }
}
```

Otro método interesante a destacar es **actualizar_numero_jugadores_activos ,** que como su nombre indica , actualiza en cada nodo el numero de jugadores sin revelar que dispone el nodo actual.

Código:

```
int QuienEsQuien::numero_hojas(bintree<Pregunta>::node n){
    if(n.null()){
        return 0;
    }
    if(n.left().null() && n.right().null()){
        return 1;
    }else{
        return numero_hojas(n.left())+numero_hojas(n.right());
    }
}

void QuienEsQuien::actualizar_numero_jugadores_activos(bintree<Pregunta>::node n){
    if(!n.null()){
        int soluciones=numero_hojas(n);
        if(esHoja(n)){
            Pregunta p((*n).obtener_personaje(),soluciones);
            *n=p;
        }else if(esPregunta(*n)){

            Pregunta p((*n).obtener_pregunta(),soluciones);
            *n=p;
        }
        actualizar_numero_jugadores_activos(n.left());
        actualizar_numero_jugadores_activos(n.right());
    }
}

bool QuienEsQuien::esHoja(bintree<Pregunta>::node n){
  return ((!n.null()) && (n.left().null() && n.right().null()));
}
```

También podemos ver el método **numero_hojas** que nos devuelve el numero de hojas de un nodo , y el método **esHoja** que verifica que el actual nodo es no tiene hijos y por lo tanto es hoja.