Implementación del árbol conArrayList

```
public class Cromosoma {

		. . .

	public static String terminales[];
	public static final String terminales6[] = { "A0", "A1", "D0", "D1", "D2", "D3" };
	public static final String funciones[] = { "AND", "OR", "NOT", "IF" };

	private Arbol arbol;
	private double fitness;
	private double fitness_bruto; //Aptitud antes de transformarla
	private double punt;
	private double puntAcum;
	private String fenotipo;

	public Cromosoma(int profundidad, int tipoCreacion, boolean useIf, int tipoMultiplexor) {
		arbol = new Arbol(profundidad, useIf);
		switch(tipoCreacion){
		case 0:
			arbol.inicializacionCreciente(0);
			break;
		case 1:
			arbol.inicializacionCompleta(0,0);
			break;
		case 2:
			int ini = new Random().nextInt(2);
			if(ini == 0) arbol.inicializacionCreciente(0);
			else arbol.inicializacionCompleta(0,0);
			break;
		}
. . .

public class Arbol{

	private String valor;
	private ArrayList<Arbol> hijos;
	private int numHijos;
	private int numNodos;
	private int max_prof;
	private int profundidad;
	private boolean useIF;
	private boolean esHoja;
	private boolean esRaiz;

	. . .

	. . .

	// Devuelve el arbol en forma de array
	public ArrayList<String> toArray(){
		ArrayList<String> array = new ArrayList<String>();
		toArrayAux(array, this);
		return array;
	}
```

```
// Insertar un valor en el arbol (nodo simple)
	public Arbol insert(String v, int index){
		Arbol a = new Arbol(v);
		if(index == -1){
			hijos.add(a);
			numHijos = hijos.size();
		}
		else
			hijos.set(index, a);
		return a;
	}

	// Insertar un arbol en otro arbol.
	public void insert(Arbol a, int index){
		if(index == -1){
			hijos.add(a);
			numHijos = hijos.size();
		}
		else
			hijos.set(index, a);
	}

	public Arbol at(int index){
		return at(this, 0, index);
	}

	private Arbol at(Arbol a, int pos, int index){
		Arbol s = null;
		if(pos >= index) s = a;
		else if(a.getNumHijos() > 0)
		{
			for(int i = 0; i < a.getNumHijos(); i++)
				if(s == null) s = at(a.getHijos().get(i), pos+i+1, index);
		}
		return s;
	}


	private void toArrayAux(ArrayList<String> array, Arbol a){
		array.add(a.valor);
		for(int i = 0; i < a.hijos.size(); i++){
			toArrayAux(array, a.hijos.get(i));
		}
	}

	public int inicializacionCompleta(int p, int nodos){
	int n = nodos;
	int nHijos = 2;
	if(p < max_prof){
		setProfundidad(p);
		Random rnd = new Random();
		int func = 0;

		if(useIF){
			func = rnd.nextInt(Cromosoma.funciones.length);
		}else{
			func = rnd.nextInt(Cromosoma.funciones.length-1);
		}

		this.valor = Cromosoma.funciones[func];
		this.setEsRaiz(true);
```

```
		if(valor.equals("IF")) nHijos = 3;
		if(valor.equals("NOT")) nHijos = 1;

		for(int i = 0; i < nHijos; i++){
			Arbol hijo = new Arbol(max_prof, useIF);
			//hijo.setPadre(this);
			esRaiz = true;
			n++;
			n = hijo.inicializacionCompleta(p+1, n);
			hijos.add(hijo);
			numHijos++;
		}
	}
	else{
		setProfundidad(p);
		Random rnd = new Random();
		int terminal;
		this.setEsHoja(true);
		terminal = rnd.nextInt(Cromosoma.terminales.length);
		valor = Cromosoma.terminales[terminal];
		esHoja = true;
		numHijos = 0;
	}

	setNumNodos(n);
	return n;
}



	private int creaHijos(int p, int nodos) {
		int n = nodos;
		int nHijos = 2;

		if(valor.equals("IF")) nHijos = 3;
		if(valor.equals("NOT")) nHijos = 1;

		for(int i = 0; i < nHijos; i++){
			Arbol hijo = new Arbol(max_prof, useIF);
			//hijo.setPadre(this);
			n++;
			n = hijo.inicializacionCrecienteAux(p+1, n);
			hijos.add(hijo);
			numHijos++;
		}
		return n;
	}

	/**
	 * Devuelve los nodos hoja del árbol
	 * @param hijos Hijos del árbol a analizar
	 * @param nodos Array donde se guardan los terminales
	 */
	public void getTerminales(ArrayList<Arbol> hijos, ArrayList<Arbol> nodos) {

		for(int i = 0; i < hijos.size(); i++){
			if(hijos.get(i).isEsHoja()){
				nodos.add(hijos.get(i).copia());
			}else{
				getTerminales(hijos.get(i).getHijos(), nodos);
			}
```

```
		}
	}

	public int insertTerminal(ArrayList<Arbol> list_hijos, Arbol terminal, int index, int pos){
		int p = pos;
		for(int i = 0; i < list_hijos.size() && p != -1; i++){
			if(list_hijos.get(i).isEsHoja() && (p == index)){
				//terminal.padre = list_hijos.get(i).padre;
				list_hijos.set(i, terminal.copia());
				p = -1;
			}else if(list_hijos.get(i).esHoja && (p != index)){
				p++;
			}else{
				p = insertTerminal(list_hijos.get(i).hijos,terminal, index, p);
			}
		}

		return p;
	}

	public int insertFuncion(ArrayList<Arbol> list_hijos, Arbol terminal, int index, int pos){
		int p = pos;
		for(int i = 0; i < list_hijos.size() && p != -1; i++){
			if(list_hijos.get(i).esRaiz && (p == index)){
				//terminal.padre = list_hijos.get(i).padre;
				list_hijos.set(i, terminal.copia());
				p = -1;
			}else if(list_hijos.get(i).esRaiz && (p != index)){
				p++;
				p = insertFuncion(list_hijos.get(i).hijos, terminal, index, p);
			}
		}
		return p;
	}

	. . .



/**
	 * Devuelve los nodos internos del árbol
	 * @param hijos Hijos del árbol a analizar
	 * @param nodos Array donde se guardan las funciones
	 */
	public void getFunciones(ArrayList<Arbol> hijos, ArrayList<Arbol> nodos) {

		for(int i = 0; i < hijos.size(); i++){
			if(hijos.get(i).isEsRaiz()){
				nodos.add(hijos.get(i).copia());
				getFunciones(hijos.get(i).hijos, nodos);
			}
		}

	}
```

```
	public Arbol copia(){
		Arbol copia = new Arbol(this.max_prof, this.useIF);

		copia.setEsHoja(this.esHoja);
		copia.setEsRaiz(this.esRaiz);
		copia.setNumHijos(this.numHijos);
		copia.setNumNodos(this.numNodos);
		copia.setProfundidad(this.profundidad);
		copia.setValor(this.valor);
		ArrayList<Arbol> aux = new ArrayList<Arbol>();
		aux = copiaHijos();
		copia.setHijos(aux);

		return copia;
	}

	private ArrayList<Arbol> copiaHijos() {
		ArrayList<Arbol> array = new ArrayList<Arbol>();

		for(int i = 0; i < this.hijos.size(); i++){
			array.add(this.hijos.get(i).copia());
		}

		return array;
	}

. . .

	public int obtieneNodos(Arbol nodo, int n){
		if(nodo.esHoja)
			return n;

		if(nodo.valor.equals("IF")){
			n = obtieneNodos(nodo.hijos.get(0), n+1);
			n = obtieneNodos(nodo.hijos.get(1), n+1);
			n = obtieneNodos(nodo.hijos.get(2), n+1);
		}else if(nodo.valor.equals("AND") || nodo.valor.equals("OR")){
			n = obtieneNodos(nodo.hijos.get(0), n+1);
			n = obtieneNodos(nodo.hijos.get(1), n+1);
		}else{
			n = obtieneNodos(nodo.hijos.get(0), n+1);
		}
		return n;
	}}



public class Cruce {

	private static final double PROB_FUNC = 0.9;

	public Cruce() {    }

	public Cromosoma[] cruzar(Cromosoma padre1, Cromosoma padre2){
		Cromosoma hijos[] = new Cromosoma[2];
		Cromosoma hijo1 = new Cromosoma();
		Cromosoma hijo2 = new Cromosoma();

		ArrayList<Arbol> nodos_selec1 = new ArrayList<Arbol>();
		ArrayList<Arbol> nodos_selec2 = new ArrayList<Arbol>();
```

```
            //Seleccionamos los nodos más relevante según la probabilidad
            //0.9 se cruzará en una función
            //resto se cruzará en un terminal
            nodos_selec1 = obtieneNodos(padre1.getArbol().copia());
            nodos_selec2 = obtieneNodos(padre2.getArbol().copia());

            //obtenemos los puntos de cruce a partir de los nodos seleccionados
            int puntoCruce1 = (int) (Math.random()*nodos_selec1.size());
            int puntoCruce2 = (int) (Math.random()*nodos_selec2.size());

            //copiamos los cromosomas padre en los hijos
            hijo1 = padre1.copia();
            hijo2 = padre2.copia();

            //Cogemos los nodos de cruce seleccionados
            Arbol temp1 = nodos_selec1.get(puntoCruce1).copia();
            Arbol temp2 = nodos_selec2.get(puntoCruce2).copia();

            //realizamos el corte sobre los arboles de los hijos
            corte(hijo1, temp2, puntoCruce1, temp1.isEsRaiz());
            corte(hijo2, temp1, puntoCruce2, temp2.isEsRaiz());

            int nodos = hijo1.getArbol().obtieneNodos(hijo1.getArbol(), 0);
            hijo1.getArbol().setNumNodos(nodos);
            nodos = hijo2.getArbol().obtieneNodos(hijo2.getArbol(), 0);
            hijo2.getArbol().setNumNodos(nodos);

            //Finalmente se evalúan
            hijo1.evalua();
            hijo2.evalua();

            hijos[0] = hijo1;
            hijos[1] = hijo2;

            return hijos;
      }
```

. . .

```
private void corte(Cromosoma hijo, Arbol temp, int puntoCruce, boolean esRaiz) {

      if(!esRaiz){
//dependiendo de qué tipo era el nodo que ya no va a estar, se inserta el nuevo
      hijo.getArbol().insertTerminal(hijo.getArbol().getHijos(), temp, puntoCruce, 0);
            }else{
            hijo.getArbol().insertFuncion(hijo.getArbol().getHijos(), temp, puntoCruce, 0);
            }
      }

      private ArrayList<Arbol> obtieneNodos(Arbol arbol) {
            ArrayList<Arbol> nodos = new ArrayList<Arbol>();

            //Obtenemos una probabilidad al azar
            if(seleccionaFunciones()){//Si devuelve true, el corte se hará en una función
```

```
                        arbol.getFunciones(arbol.getHijos(), nodos);
                        if(nodos.size() == 0){//Si no existen funciones, se seleccionan los terminales
                              arbol.getTerminales(arbol.getHijos(), nodos);
                        }
                  }
                   else{//Si devuelve false, el corte se hará por un terminal
                        arbol.getTerminales(arbol.getHijos(), nodos);
                  }

                  return nodos;
            }

            private boolean seleccionaFunciones(){
                  double prob = Math.random();

                  if(prob < PROB_FUNC){
                        return true;
                  }else{
                        return false;
                  }
            }

}
```

```
. . .

private int evaluar(ArrayList<String> func, int index){
            String funcion = func.get(index);
            int resul;
            if(funcion.equals("IF")){
                  int hijo1 = evaluar(func, index+1);
                  int hijo2 = evaluar(func, index+2);
                  int hijo3 = evaluar(func, index+3);
                  if(hijo1 == 1) resul = hijo2;
                  else resul = hijo3;
            }
            else if(funcion.equals("AND")){
                  int hijo1 = evaluar(func, index+1);
                  int hijo2 = evaluar(func, index+2);
                  if(hijo1 == 1 && hijo2 == 1) resul = 1;
                  else resul = 0;
            }
            else if(funcion.equals("OR")){
                  int hijo1 = evaluar(func, index+1);
                  int hijo2 = evaluar(func, index+2);
                  if(hijo1 == 1 || hijo2 == 1) resul = 1;
                  else resul = 0;

. . .
```

Otro enfoque de evaluación:

```
/**
         * Evalua el arbol de forma recursiva del arbol con el caso de prueba
         * correspondiente.
         *
         * @param caso
         *            Caso de prueba actual.
         *
         * @return La evaluacion recursiva del arbol con el caso de prueba
         *         correspondiente.
         */
        private boolean evaluaRecursivo(boolean[] caso, int numeroEntradas) {

        boolean evaluacion = false;

        // Si es un Terminal
        if (_esHoja) {
                // Devolvemos su valor que le corresponde en el caso de prueba
                evaluacion = caso[_simbolo.devuelvePosTerminal(numeroEntradas)];
          }
        // Si es una Funcion
                else {

                        // Si es la funcion NOT
                        if (_simbolo.getValor().matches("NOT")) {

                                // Devolvemos el valor negado que le corresponde al hijo de la
                                // izquierda
                                evaluacion = !_hijos.get(0).evaluaRecursivo(caso,numeroEntradas);
                        } else
                                // Si es la funcion OR
                                if (_simbolo.getValor().matches("OR")) {
                                        // Devolvemos la OR de sus dos hijos
                                        evaluacion = _hijos.get(0).evaluaRecursivo(caso,numeroEntradas)
                                        || _hijos.get(1).evaluaRecursivo(caso,numeroEntradas);

                                } else
                                        // Si es la funcion AND
                                        if (_simbolo.getValor().matches("AND")) {
                                                // Devolvemos la AND de sus dos hijos
                                                evaluacion = _hijos.get(0).evaluaRecursivo(caso,numeroEntradas)
                                                && _hijos.get(1).evaluaRecursivo(caso,numeroEntradas);
                                        } else
                                                // Si es un IF
                                                if (_simbolo.getValor().matches("IF")) {

                                                        // Si el primer hijo es true
                                                        if (_hijos.get(0).evaluaRecursivo(caso,numeroEntradas))
                                                                // Evaluamos el segundo hijo
                                                                evaluacion =
_hijos.get(1).evaluaRecursivo(caso,numeroEntradas);
                                                        else
                                                                // Y si no evaluamos el tercer hijo
                                                                evaluacion =
_hijos.get(2).evaluaRecursivo(caso,numeroEntradas);
                                                        }
                }
                return evaluacion;
        }
```