

Utilidades Práctica 3.

Representación de los individuos

```
tipo TIndividuo = registro{
    TArbol arbol; // estrategia de rastreo
    real adaptación; // función de evaluación
    real puntuación; // puntuación relativa: adaptación/sumadaptación
    real punt_acu; // puntuación acumulada
    booleano elite; // elitismo
    . . .
}
```

Los árboles pueden contener operadores con dos y tres operandos, por lo que podemos considerar tres nodos hijos. Para los operadores con dos operandos el hijo central (Hc) estará a nulo. Por utilidad también almacenamos el número de nodos del árbol y la profundidad. Puedes utilizar la estructura tipo árbol que quieras. Algunos ejemplos:

```
tipo TArbol = registro{
    String dato; // operando u operador
    TArbol Hi; // hijo izquierdo
    TArbol Hc; // hijo central
    TArbol Hd; // hijo derecho
    entero num_nodos; // número de nodos
    entero profundidad; // profundidad del árbol
    . . .
};
```

```
tipo TArbol {
    TArbol padre;
    TArbol[] hijos;
    Tipo tipo;
    int num_nodos; // número de nodos
    int profundidad; // profundidad del árbol
    . . .
}

public static enum Tipo {
    AVANZA, GIRA_DERECHA, GIRA_IZQUIERDA,
    SIC, PROGN2, PROGN3
}
```

Generación de la Población Inicial

- Podemos hacer inicialización creciente o completa y ramped and half

```
funcion creaArbol (TArbol arbol, entero prof_min, entero prof_max)
{
    si prof_min > 0 entonces //no puede ser hoja
    // generación del subarbol de operador
    operador = operador_aleatorio; // símbolo de operador aleatorio
    arbol.dato = operador;
    // se generan los hijos
    HI = construir_arbol(arbol.HI, prof_min - 1, prof_max - 1);
    arbol.num_nodos = arbol.num_nodos + arbol.HI.num_nodos;
    si tres_operandos(operador) entonces
        HC = construir_arbol(arbol.HC, prof_min - 1, prof_max - 1);
        arbol.num_nodos = arbol.num_nodos + arbol.HC.num_nodos;
    eoc // dos operandos
}
```

```

HC = NULL;
HD = construir_arbol(arbol.HD, prof_min - 1, prof_max - 1);
arbol.num_nodos = arbol.num_nodos + arbol.HD.num_nodos;
eoc // prof_min = 0

si prof_max = 0 entonces // sólo puede ser hoja
// generación del subarbol de operando
operando = operando_aleatorio;
// símbolo de operando aleatorio
arbol.dato = operando;
arbol.num_nodos = arbol.num_nodos + 1;
eoc
// se decide aleatoriamente operando u operador
tipo = aleatorio_cero_uno;
si tipo = 1 entonces // se genera operador
// generación del subarbol de operador
{ ... }
eoc // se genera operando
// generación del subarbol de operando
{ ... }

}

```

El Operador de Cruce

El operador de cruce más utilizado en este tipo de problemas es el cruce por intercambio de subárboles: seleccionamos 2 nodos de manera aleatoria e intercambiamos sus subárboles.

```

TArbol subarbol 1, subarbol 2;
entero num_nodos;

num_nodos = minimo(num_nodos(padre1.arbol), num_nodos(padre2.arbol));
nodo_cruce = alea_entero(1, num_nodos);
hijo1.arbol = padre1.arbol;
hijo2.arbol = padre2.arbol;
subarbol 1 = hijo1.arbol.BuscarNodo(nodo_cruce);
subarbol 2 = hijo2.arbol.BuscarNodo(nodo_cruce);
hijo1.arbol.SustituirSubarbol(nodo_cruce, subarbol 2);
hijo2.arbol.SustituirSubarbol(nodo_cruce, subarbol 1);
hijo1.adaptacion = adaptacion(hijo1);
hijo2.adaptacion = adaptacion(hijo2);

...

```

Función de adaptación

La adaptación o aptitud de un individuo es la cantidad de alimento comido por la hormiga dentro de un espacio de tiempo razonable al ejecutar el programa a evaluar. Se considera que cada operación de movimiento o giro consume una unidad de tiempo. En nuestra versión del problema limitaremos el tiempo a 400 pasos. Lo planteamos como un problema de maximización.

```

funcion adaptacion(TIndividuo individuo, TMapa mapa) {
    . . .

    mientras (pasos < 400 y bocados < 90)
        ejecutaArbol(individuo.arbol);

    . . .
}

```

```

public void ejecutaArbol(Arbol A){
    //mientras no se haya acabado el tiempo ni la comida
    si (pasos < 400 && bocados <90) entonces {

        //si estamos encima de comida comemos
        si (matriz[posicionX][posicionY]== 1) entonces {
            matriz[posicionX][posicionY] = 0;
            bocados++;
        }

        //acciones a realizar en función del nodo en el que estemos
        si A.getValor()=="PROGN3"){
            ejecutaArbol(A.getHi());
            ejecutaArbol(A.getHc());
            ejecutaArbol(A.getHd());
        }
        eoc si (A.getValor()=="PPROGN2"){
            ejecutaArbol(A.getHi());
            ejecutaArbol(A.getHc());
        }
        eoc if(A.getValor()=="SIC"){
            si (hayComida()) ejecutaArbol(A.getHi());
            eoc ejecutaArbol(A.getHc());
        }
        eoc si A.getValor()=="AVANZA") Avanza();
        eoc si A.getValor()=="DERECHA") Derecha();
        eoc si (A.getValor()=="IZQUIERDA") Izquierda();
    }
}

```