



Cuestión 1

1.5 puntos

Un puzzle móvil es un juego que consiste en ordenar las piezas de un tablero, en el que siempre hay un hueco, hasta conseguir una figura determinada. Se desea diseñar una variante de este juego, con las siguientes características:

- El tablero tiene un tamaño de 5x5, es decir, podemos poner hasta 24 piezas.
- Al comenzar el juego, el tablero está vacío.
- Una vez colocada una pieza, ésta puede desplazarse en los 4 sentidos (derecha, izquierda, arriba y abajo). Para simplificar sólo se especificará el desplazamiento hacia la derecha.

Para facilitar el diseño del juego, se desea crear el TAD *PUZZLE MÓVIL*, con un conjunto de operaciones, cuya signatura es la siguiente:

espec PUZZLE MÓVIL

usa cadenas, naturales, booleanos

género puzM

operaciones

puzVacío: \rightarrow puzM

{crea un puzzle vacío}

parcial ponerP: puzM natural natural \rightarrow puzM

{coloca una pieza en la coordenada (x,y). No se puede poner una pieza en una coordenada ocupada}

numP: puzM \rightarrow natural

{devuelve el número de piezas que tiene actualmente el puzzle}

parcial ocupada: puzM natural natural \rightarrow booleano

{devuelve verdadero si la posición (x,y) está ocupada y falso en caso contrario}

parcial moverD: puzM natural natural \rightarrow puzM

{desplaza a la derecha la pieza que se encuentra en la posición (x,y). Se produce un error en el caso de que la posición de la derecha esté ocupada y que la pieza que se desea mover no está en el puzzle}

parcial quitarP: puzM natural natural \rightarrow puzM

{quita la pieza que se encuentra en la posición (x,y)}

Considerando $G = \{\text{puzVacío}, \text{ponerP}\}$ el conjunto de operaciones generadoras, completar la especificación algebraica del TAD *PUZZLE MÓVIL* con los dominios de definición de las operaciones parciales y las ecuaciones.

Cuestión 2

1 punto

Supongamos la siguiente representación dinámica para almacenar un Árbol Binario de Búsqueda.

```
tipo ABB = puntero a nodo;  
nodo = registro  
        valor: entero;  
        izq, der: ABB;  
registro;
```

Escribir, en lenguaje algorítmico y de forma recursiva, las siguientes operaciones:

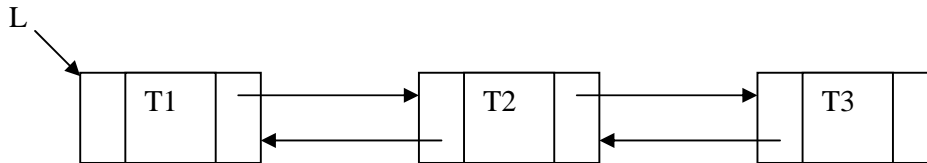
```
función suma (A: ABB): entero;  
{calcula la suma de los elementos del árbol A}
```

```
función equilibrado (A: ABB): booleano;  
{devuelve verdad si el árbol A está equilibrado y falso en caso contrario.  
Suponer que se dispone de una operación que devuelve la altura  
de un árbol, cuya sintaxis es: función altura (A: ABB): integer.  
Suponer también que el árbol vacío está equilibrado }
```

Cuestión 3

1.5 puntos

Dada una Lista Doblemente Enlazada de Tablas (LDET) como la mostrada en el siguiente ejemplo:



donde los datos T1, T2 y T3 son tablas definidas de la forma: **tabla** [1..10] **de** entero, Las posiciones de las tablas que no están ocupadas (vacías) tienen almacenado un **cero (0)**.

Se Pide:

- a) Escribir, en lenguaje algorítmico, la acción **Singulares**, que dada una lista, elimina todos los enteros repetidos en las distintas tablas de la lista.

acción Singulares (**var** L: LDET);

Ejemplo:

Una lista con dos nodos, cuya información es T1(0,1,2,0,0,0,1,0,4,0) T2(0,0,2,0,3,1,0,2,0,4) quedaría de la siguiente forma:

T1(0,1,2,0,0,0,0,4,0) T2(0,0,2,0,3,1,0,0,0,4)

- b) Escribir, en lenguaje algorítmico, la acción **Hace**, expresada en un pseudocódigo de muy alto nivel, y utilizando la representación del tipo LDET dada a continuación:

Tipo LDET = **puntero a** nodo
nodo = **registro**
T: **tabla** [1..10] **de** enteros;
sig, ant: LDET;
fregistro

acción *hace* (**var** L: LDET; d: entero)

var C: conjunto de enteros

{ las operaciones válidas sobre el tipo *conjunto de enteros* son: *poner*(C, d) y *esta?*(d, C) }

m: entero

(a1) Calcular el entero mayor de los existentes en L y almacenarlo en m

para todo L **hacer**

guardar en C todos los enteros distintos de L que no sean 0

fpara

(a2) Eliminar el entero m de todos los nodos de L menos del primer nodo en el que lo encuentre

faccion

IMPORTANTE:

- Los apartados (a1) y (a2) del pseudocódigo deben ser implementados mediante una función y una acción respectivamente.
- No sólo se valorará el correcto funcionamiento de la acción *hace*. Se tendrá muy en cuenta la correcta metodología de programación utilizada para su implementación.

Cuestión 4

1.5 puntos

Dada una tabla Hash implementada con el siguiente tipo de datos:

```
Constante Max = 100;  
Tipo  
nodo = registro  
Clave : clave;  
Valor : valor;  
PosDesb: entero;  
registro;  
  
tablaH = tabla [1..Max] de nodo;
```

Se Pide:

Diseñar la acción **InsMod** (**var t: tablaH; c: clave; v: valor**), que inserta el nuevo par **<c,v>** en la tabla **t** en el caso de que la clave **c** no exista en la tabla, o modifica el valor asociado a la clave **c** por el nuevo valor **v** en caso que exista la clave **c** en la tabla.

En caso de *colisión*, se utilizará el mecanismo de resolución de zona de desbordamiento en el propio vector, para lo que se utilizará la función de recolocación *R*.

El procedimiento a seguir es el siguiente: "Si al intentar insertar un nuevo par **<c,v>**, la posición devuelta por la función de dispersión ya está ocupada, se utilizará la función de recolocación para calcular el nuevo hueco en el vector. La posición devuelta por *R* se almacena en el campo PosDesb de la posición colisionada de la tabla. Si dicha posición también está ocupada, se vuelve a repetir este proceso hasta un máximo de 4 veces".

Agotados los 4 intentos de inserción, el nuevo par no se puede insertar, y se mostrará el mensaje "Par no insertado: posible factor de carga alto".

Notas:

- Se supone que existen las funciones de dispersión (H) y recolocación en el vector soporte (R), con la siguiente sintaxis:

H(c: clave) devuelve 1..Max

R(c: clave) devuelve 1..Max

- El campo *posDesb* estará a **ceros (0)** en caso de posiciones en las que no se ha producido colisión
- No sólo se valorará el correcto funcionamiento de la acción *InsMod*. Se tendrá muy en cuenta la correcta metodología de programación utilizada para su implementación.