

Parcial 1: Tipos concretos y Tipos Abstractos de Datos

Docentes: Daniel Fridlender, Renato Cherini, Diego Lis, y Juan Cruz Rodriguez

1. a) Especificar el TAD *votos*[partido] que permite contabilizar los votos obtenidos por cada partido en una elección. Sus constructores son *inicial*, en la que ningún voto fue aún contabilizado, y *contabilizar_voto* : $\text{votos} \times \text{partido} \rightarrow \text{votos}$ que agrega un voto al partido correspondiente. Cuenta además con las siguientes operaciones: *juntar* : $\text{votos} \times \text{votos} \rightarrow \text{votos}$ que contabiliza la totalidad de votos de ambos argumentos, *total* : $\text{votos} \rightarrow \text{nat}$ que devuelve el número total de votos contabilizados, y *subtotal* : $\text{votos} \times \text{partido} \rightarrow \text{nat}$ que devuelve el número de votos de un partido.
- b) Implementar el TAD *votos* asumiendo que los partidos están identificados con números naturales entre 1 y n .
2. a) Especificá el TAD *posición*, que permite acceder a los subárboles de un árbol binario. El TAD debe tener tres constructores: *parar* que crea la posición inicial (de la raíz), *doblar_der* y *doblar_izq* que indican que se debe tomar hacia la derecha y hacia la izquierda respectivamente. El TAD debe tener además tres operaciones: *es_parar?*, que devuelve *verdadero* si y sólo si la posición es *parar*, *ir_a_izq?*, que devuelve *verdadero* si y solo si la última indicación agregada es doblar a la izquierda, y *avanzar* que elimina la última indicación agregada.
- b) Implementá el TAD *posición* utilizando como estructura de datos el TAD *pila*.
- c) Escribí un programa iterativo para **fun** $\uparrow (t : \text{bintree}, p : \text{position})$ **ret** $s : \text{bintree}$ que devuelve el subárbol de t que se encuentra en la posición p , suponiendo que *bintree* es una implementación del TAD *árbol binario* y *position* es la implementación del TAD *posición* del punto anterior.
3. Programá una función **fun** *es_heap*($t : \text{bintree}$) **ret** $b : \text{bool}$ que verifique si el *árbol binario* t satisface el invariante de ordenación que caracteriza a los *heaps binarios*.
4. A continuación se especifica una variante del TAD *lista*:

TAD *lista*[elem]
constructores
 $[] : \text{lista}$
 $\triangleright : \text{elem} \times \text{lista} \rightarrow \text{lista}$
operaciones
 $. : \text{lista} \times \text{nat} \rightarrow \text{elem}$
 $\text{ins} : \text{elem} \times \text{nat} \times \text{lista} \rightarrow \text{lista}$
 $++ : \text{lista} \times \text{lista} \rightarrow \text{lista}$
ecuaciones
 $(x \triangleright xs).0 = x$
 $(x \triangleright xs).(n + 1) = xs.n$
 $\text{ins}(y, 0, []) = [y]$
 $\text{ins}(y, n + 1, x \triangleright xs) = x \triangleright (\text{ins}(y, n, xs))$
 $[] ++ ys = ys$
 $(x \triangleright xs) ++ ys = x \triangleright (xs ++ ys)$

Los constructores y los operadores $.$ y $++$ son los usuales. El operador *ins* con parámetros y, n y xs , inserta en la posición n de la lista xs el elemento y . Notá que el parámetro n debe cumplir $0 \leq n \leq \#xs$.

Implementá este TAD utilizando la estructura de datos *lista-ligada* (no circular). Cada nodo es una tupla con un campo para almacenar un valor, y un puntero al siguiente nodo; y una lista es una tupla con un puntero al primer nodo de la lista, y un puntero al último nodo de la lista.