Invariante de Representación y Función de Abstracción

Diego y Ramiro (TM) Román (TN)

Algoritmos y Estructuras de Datos

15 de Octubre de 2025

TADs

Al escribir un TAD...

- Definimos las operaciones
- Describimos qué hace cada una (con observadores y lógica)

Dijimos que se pueden implementar de varias maneras, en función de los requerimientos

Diseño

Al diseñar un TAD...

- Elegimos una estructura (usando tipos de implementación)
- Escribimos los algoritmos (en pseudocódigo)

Queremos que la combinación estructura/algoritmos...

- Cumpla con la especificación del TAD
- Sea eficiente (rápida, ocupe poca memoria, etc)
- Sea copada (elegante, linda, fácil de entender, fácil de cambiar)

Ejemplo

Queremos implementar el TAD Conjunto usando arrays

TAD Conjunto$\langle T \rangle$ {

obs elems: conj$\langle T \rangle$

proc conjVacío() : Conjunto$\langle T \rangle$

proc pertenece(in c : Conjunto$\langle T \rangle$, in $e : T$) : Bool

proc agregar(inout c : Conjunto$\langle T \rangle$, in $e : T$)

proc sacar(inout c : Conjunto$\langle T \rangle$, in $e : T$)

proc unir(inout c : Conjunto$\langle T \rangle$, in c' : Conjunto$\langle T \rangle$)

proc restar(inout c : Conjunto$\langle T \rangle$, in c' : Conjunto$\langle T \rangle$)

proc intersecar(inout c : Conjunto$\langle T \rangle$, in c' : Conjunto$\langle T \rangle$)

proc agregarRápido(inout c : Conjunto$\langle T \rangle$, in $e : T$)

proc tamaño(in c : Conjunto$\langle T \rangle$):$\mathbb{Z}$

}

Ejemplo

Conjunto sobre arrays

Módulo ConjuntoSobreArray$\langle T \rangle$ **implementa** Conjunto$\langle T \rangle$ <

```
/* elementos del conjunto */
var datos: array<T>
/* cantidad de posiciones del array
 *  que representan elementos del conjunto */
var cant: int

// . . .
```

>

Observación

`cant` también es la cantidad de elementos del conjunto

Ejemplo

Qué instancias del TAD representan?

Modulo	TAD
$\langle datos : [0,0,0,0,0,0,0], cant : 0\rangle$ $\langle datos : [3,5,0,0,0,0,0], cant : 2\rangle$ $\langle datos : [3,100,-1,4,10,99,0], cant : 0\rangle$	

Ejemplo

Qué instancias del TAD representan?

Modulo	TAD
$\langle datos : [0,0,0,0,0,0,0], cant : 0 \rangle$	$\langle elems : \langle\rangle \rangle$
$\langle datos : [3,5,0,0,0,0,0], cant : 2 \rangle$	$\langle elems : \langle 3,5 \rangle \rangle$
$\langle datos : [3,100,-1,4,10,99,0], cant : 0 \rangle$	$\langle elems : \langle\rangle \rangle$

Función de abstracción

Nos indica, dada una instancia del módulo, a qué instancia del TAD corresponde

Ejemplo

Función de abstracción

- Nos referimos a las variables del módulo y a los observadores del TAD (**NO a las operaciones**)
- Se escribe en lógica, usando los tipos de especificación (seq, conj, dicc)
- Damos por aplicado el `ABS` sobre todas las `var` del módulo

$$
\begin{array}{l}
\texttt{func ABS}(m : \text{ConjuntoSobreArray}\langle T\rangle) : \text{Conjunto}\langle T\rangle \ \{ \\
\quad t : \text{Conjunto}\langle T\rangle \mid \\
\qquad m.cant = |t.elems| \ \wedge \\
\qquad (\forall i : \mathbb{Z})\ (0 \leq i < m.cant \rightarrow_L m.datos.elems[i] \in t.elems) \\
\}
\end{array}
$$

Ejemplo

Qué instancias del TAD representan?

Modulo	TAD
$\langle datos : [3, 5, 0, 0, 0, 0, 0], cant : 100 \rangle$	
$\langle datos : [3, 0, 0, 0, 0, 0, 0], cant : -1 \rangle$	
$\langle datos : [3, 3, 0, 0, 0, 0, 0], cant : 2 \rangle$	

Ejemplo

Qué instancias del TAD representan?

Modulo	TAD
$\langle datos : [3, 5, 0, 0, 0, 0, 0], cant : 100 \rangle$	inválido!
$\langle datos : [3, 0, 0, 0, 0, 0, 0], cant : -1 \rangle$	inválido!
$\langle datos : [3, 3, 0, 0, 0, 0, 0], cant : 2 \rangle$	inválido!

Invariante de representación

Es un predicado que nos indica qué conjuntos de valores son instancias válidas del módulo

Ejemplo

Invariante de representación

- Para cualquier proc x() del módulo, se tiene que poder verificar
  $$\{InvRep(p)\} procx(p, \cdots) \{InvRep(p)\}$$
- Nos referimos a las variables del módulo (**NO a las operaciones**)
- Se escribe en lógica, usando los tipos de especificación (seq, conj, dict)

$$\texttt{pred InvRep}(m : \text{ConjuntoSobreArray}\langle T\rangle) \ \{$$
$$0 \leq m.cant \leq |m.datos.elems| \ \wedge$$
$$noHayRepetidos(m.datos.elems, m.cant)$$
$$\}$$

$$\texttt{pred noHayRepetidos}(s : \text{seq}\langle T\rangle, cant : \mathbb{Z}) \ \{$$
$$(\forall i, j : \mathbb{Z}) \ (0 \leq i, j < cant \wedge i \neq j \rightarrow_L s[i] \neq s[j])$$
$$\}$$

Algunos algoritmos

Módulo ConjuntoSobreArray$\langle T \rangle$ **implementa** Conjunto$\langle T \rangle$ <

```
  // . . .
  proc tamaño(c: ConjuntoSobreArray<T>): int
    return c.cant

  proc pertenece(c: ConjuntoSobreArray<T>, e : T): bool
    int i = 0
    while (i < c.cant)
      if (c.datos[i] == e)
        return true
      endif
      i = i + 1
    endwhile
    return false

 proc agregar(c: ConjuntoSobreArray<T>, e : T)
   if (pertenece(c, e)
     return
   endif
   c.datos[c.cant] = e // asumimos que c.cant <= |c.datos|
   c.cant = c.cant + 1
```

>

Alternativas

Existen diferentes alternativas de implementación con la misma estructura:

- sin repetidos (la que acabamos de ver)
- con repetidos
- con los elementos ordenados

En cada alternativa cambia el invariante de representación, función de abstracción y algoritmos

Con repetidos

- Se pueden agregar elementos repetidos al array
- Esto hace más eficiente agregar elementos
- `cant` **no es** la cantidad de elementos del conjunto

Ejemplos

Modulo	TAD
$\langle datos : [0,0,0,0,0,0,0], cant : 0\rangle$	$\langle elems : \langle\rangle\rangle$
$\langle datos : [3,5,0,0,0,0,0], cant : 2\rangle$	$\langle elems : \langle 3,5\rangle\rangle$
$\langle datos : [3,100,-1,4,10,99,0], cant : 0\rangle$	$\langle elems : \langle\rangle\rangle$
$\langle datos : [3,5,0,0,0,0,0], cant : 100\rangle$	inválido!
$\langle datos : [3,0,0,0,0,0,0], cant : -1\rangle$	inválido!
$\langle datos : [3,3,0,0,0,0,0], cant : 2\rangle$	$\langle elems : \langle 3\rangle\rangle$
$\langle datos : [3,2,3,2,0,0,0], cant : 4\rangle$	$\langle elems : \langle 2,3\rangle\rangle$
$\langle datos : [3,3,2,2,2,5,0], cant : 2\rangle$	$\langle elems : \langle 3\rangle\rangle$

Con repetidos – InvRep & Abs

$$\texttt{pred InvRep}(m : \text{ConjuntoSobreArray}\langle T \rangle) \ \{$$
$$\quad 0 \leq m.cant \leq |m.datos.elems|$$
$$\}$$

$$\texttt{func ABS}(m : \text{ConjuntoSobreArray}\langle T \rangle) : \text{Conjunto}\langle T \rangle \ \{$$
$$\quad t : \text{Conjunto}\langle T \rangle \mid$$
$$\qquad (\forall e : T) \ ($$
$$\qquad\quad e \in t \leftrightarrow$$
$$\qquad\quad (\exists i : \mathbb{Z}) \ (0 \leq i < m.cant \wedge_L m.datos.elems[i] = e)$$
$$\qquad)$$
$$\}$$

Con repetidos – algoritmos

Módulo ConjuntoSobreArray$\langle T \rangle$ **implementa** Conjunto$\langle T \rangle$ <

```
  // . . .
  proc tamaño(c: ConjuntoSobreArray<T>): int
    array<T> sr = sinRepetidos(c.datos, c.cant)
    return sr.length()

  proc pertenece(c: ConjuntoSobreArray<T>, e : T): bool
    int i = 0
    while (i < c.cant)
      if (c.datos[i] == e)
        return true
      endif
      i = i + 1
    endwhile
    return false

 proc agregar(c: ConjuntoSobreArray<T>, e : T)
   c.datos[c.cant] = e // asumimos que c.cant <= |c.datos|
   c.cant = c.cant + 1
```

>

Con los elementos ordenados (sin repetidos)

- Los elementos se insertan ordenados
- Esto hace más eficiente buscar elementos

Ejemplos

Modulo	TAD
$\langle datos : [0,0,0,0,0,0,0], cant : 0\rangle$	$\langle elems : \langle\rangle\rangle$
$\langle datos : [3,5,0,0,0,0,0], cant : 2\rangle$	$\langle elems : \langle 3,5\rangle\rangle$
$\langle datos : [3,100,-1,4,10,99,0], cant : 0\rangle$	$\langle elems : \langle\rangle\rangle$
$\langle datos : [3,5,0,0,0,0,0], cant : 100\rangle$	inválido!
$\langle datos : [3,0,0,0,0,0,0], cant : -1\rangle$	inválido!
$\langle datos : [3,3,0,0,0,0,0], cant : 2\rangle$	inválido!
$\langle datos : [3,2,0,0,0,0,0], cant : 2\rangle$	inválido!

Con los elementos ordenados – InvRep & Abs

$$\texttt{pred InvRep}(m : \text{ConjuntoSobreArray}\langle T\rangle) \ \{$$
$$\quad 0 \leq m.cant \leq |m.datos.elems| \wedge$$
$$\quad elementosOrdenados(m.datos.elems, m.cant)$$
$$\}$$

$$\texttt{pred elementosOrdenados}(s : \text{seq}\langle T\rangle, cant : \mathbb{Z}) \ \{$$
$$\quad (\forall i : \mathbb{Z})\ (0 < i < cant \rightarrow_L s[i] \geq s[i-1])$$
$$\}$$

$$\texttt{func ABS}(m : \text{ConjuntoSobreArray}\langle T\rangle) : \text{Conjunto}\langle T\rangle \ \{$$
$$\quad t : \text{Conjunto}\langle T\rangle \mid$$
$$\quad m.cant = |t.elems| \wedge$$
$$\quad (\forall i : \mathbb{Z})\ (0 \leq i < m.cant \rightarrow_L m.datos.elems[i] \in t.elems)$$
$$\}$$

Con los elementos ordenados – algoritmos

Módulo ConjuntoSobreArray$\langle T \rangle$ **implementa** Conjunto$\langle T \rangle$ <

```
  // . . .
  proc tamaño(c: ConjuntoSobreArray<T>): int
    return c.cant

  proc pertenece(c: ConjuntoSobreArray<T>, e : T): bool
    /* Busqueda binaria entre 0 y c.cant */
    return busquedaBinaria(c, e)

 proc agregar(c: ConjuntoSobreArray<T>, e : T)
   /* Queda de ejercicio para el lector :)
    * Hay que ``mover'' todos los elementos mayores
    *  que e una posición a la derecha */
```

>