Tipos Abstractos de Datos

Román Gorojovsky & Valentino Murga

Algoritmos y Estructuras de Datos

3 de Septiembre de 2025

Plan del día

- ¿Qué es un TAD?
- Anatomía de un TAD
- Ejercicio de ejemplo
- Ejercicios entre todos
- Recreo (la mejor parte de la clase)
- Modelado
- Otro ejercicio

¿Qué es un TAD?

Un TAD (Tipo Abstracto de Datos)

- Define un conjunto de **valores** y las **operaciones** que se pueden realizar sobre ellos.
- Es **abstracto**: se enfoca en la funcionalidad, sin importar
  - Representación interna de los datos
  - Implementación de las operaciones
- Especifica **qué** hacen las operaciones y no **cómo** lo hacen

¿Qué partes tiene un TAD?

- Nombre *(y parámetros)*
- Observadores
- Igualdad
- Operaciones
  - Nombre
  - Parámetros con todos sus tipos
  - Requiere
  - Asegura
- Predicados y auxiliares (opcionales)

Ejemplo

```
TAD Punto {
    obs x : ℝ
    obs y : ℝ

    proc igualdad(in P_1 : Punto, in P_2 : Punto) : bool {
        asegura { res = true ↔ ··· }
    }

    proc crearPunto(in x : ℝ, in y : ℝ) : Punto {
        asegura { ··· }
    }
    proc mover(inout p : Punto, in dx : ℝ, in dy : ℝ) {
        requiere { ··· }
        asegura { ··· }
    }

    aux theta(p : Punto) : ℝ = ···

    pred estaEnElOrigen(p : Punto) {
        ···
    }
}
```

Observadores

`obs x` : $\mathbb{R}$
`obs y` : $\mathbb{R}$

- Los Observadores describen una instancia del TAD en un determinado momento en términos de tipos más conocidos.
- La especificación de las operaciones del TAD se escribe en función de los observadores antes y después de su ejecución.
- Los *asegura* de cada operación deben especificar el valor de todos los observadores
- Se pueden usar todos los tipos de nuestro lenguaje de especificación

Operaciones

Las operaciones (`proc`) del TAD

- Se especifican mediante nuestro lenguaje de especificación
- Los parámetros pueden ser in o inout
- Puede haber un valor de salida
- Por convención, el TAD es el primer parámetro del `proc`

La lista de operaciones del TAD son el *contrato* o *interfaz* del mismo: será lo que luego implementemos y lo que usen los *clientes* del TAD.

Operaciones – Ejemplo

```
proc mover(inout p : Punto, in dx : ℝ, in dy : ℝ) {
    requiere { p = P_0 }
    asegura { p.x = P_0.x + dx ∧ p.y = P_0.y + dx }
}
```

- Los requiere y asegura hacen referencia a los observadores
- Para hablar del valor incial de un parámetro inout usamos metavariables
- Definimos la metavariable *para el TAD entero* no para los observadores
- Las metavariables se definen en el `requiere`

En el asegura tenemos que describir el valor de todos los observadores al salir de la operación. Recordemos que la implementación final del TAD sólo debe respetar la especificación y puede modificar cualquier cosa que no se defina.

Operación de creación

- Crea una instancia de un TAD a partir de un conjunto de parámetros
- El TAD es el valor de salida
- Hay que especificar **siempre** el valor de **todos** los observadores

$$\texttt{proc nuevoPunto}(\text{in } x : \mathbb{R}, \text{in } y : \mathbb{R}) : Punto \; \{$$
$$\quad \texttt{asegura} \; \{ \; res.x = x \land res.y = y \; \}$$
$$\}$$

Operación de igualdad

- Compara dos instancias de un TAD
- Devuelve un valor booleano
- Puede alcanzar con comparar los observadores uno a uno
- Vamos a ver ejemplos donde no es así

```
proc igualdad(in P1 : Punto, in P2 : Punto) : bool {
    asegura { res = true ↔ P1.x = P2.x ∧ P1.y = P2.y }
}
```

Ejemplo completo – (Ej 2.a)

Especifique mediante TADs (...) Punto2D, que representa un punto en el plano. Debe contener las siguientes operaciones:

- *nuevoPunto*: que crea un punto a partir de sus coordenadas x e y.
- *mover*: que mueve el punto una determinada distancia sobre los ejes x e y.
- *distancia*: que devuelve la distancia entre dos puntos.
- *distanciaAlOrigen*: que devuelve la distancia del punto $(0, 0)$.

Ejemplo completo – (Ej 2.a)

TAD Punto2D {

obs x : $\mathbb{R}$

obs y : $\mathbb{R}$

proc nuevoPunto(in $x : \mathbb{R}$, in $y : \mathbb{R}$) : $Punto2D$ {

requiere { $True$ }

asegura { $res.x = x \land res.y = y$ }

}

proc igualdad(in $P_1 : Punto$, in $P_2 : Punto$) : bool {

asegura { $res = \text{true} \leftrightarrow P_1.x = P_2.x \land P_1.y = P_2.y$ }

}

proc mover(inout $p : Punto2D$, in $dx : \mathbb{R}$, in $dy : \mathbb{R}$) {

requiere { $p = P_0$ }

asegura { $p.x = P_0.x + dx \land p.y = P_0.y + dy$ }

}

...

}

Ejemplo completo – (Ej 2.a)

TAD Punto2D {

...

proc distancia(in $p1 : Punto2D$, in $p2 : Punto2D$) : $\mathbb{R}$ {
requiere { $True$ }
asegura { $res = distanciaEntrePuntos(p1.x, p1.y, p2.x, p2.y)$ }
}

proc distanciaAlOrigen(in $p : Punto2D$) : $\mathbb{R}$ {
requiere { $True$ }
asegura { $res = distanciaEntrePuntos(p.x, p.y, 0, 0)$ }
}

aux distanciaEntrePuntos($x_1 : \mathbb{R}, y_1 : \mathbb{R}, x_2 : \mathbb{R}, y_2 : \mathbb{R}$) : $\mathbb{R}$
$= \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$
}

¡Ahora ustedes! – (Ej 2.c)

Especifique mediante TADs (...) Punto2D. Con las mismas operaciones, pero usando como observadores las coordenadas polares

- *nuevoPunto*: que crea un punto a partir de sus coordenadas x e y.
- *mover*: que mueve el punto una determinada distancia sobre los ejes x e y.
- *distancia*: que devuelve la distancia entre dos puntos.
- *distanciaAlOrigen*: que devuelve la distancia del punto $(0, 0)$.

¡Ahora ustedes! – (Ej 1)

Especificar en forma completa el TAD NumeroRacional que incluya las operaciones aritméticas básicas (suma, resta, división, multiplicación) y el predicado de *igualdad* que dados dos números racionales devuelva verdadero si son iguales.

¡Ahora ustedes! – Ej 5.a

Especifique el TAD Conjunto$\langle T \rangle$ con las operaciones

- *nuevoConjunto*: que crea un conjunto vacío
- *agregar*: que agrega un elemento al conjunto
- *pertenece*: que devuelve true si un elemento pertenece al conjunto
- *eliminar*: que elimina un elemento del conjunto

¿Qué observadores vamos a usar?

conj$\langle T \rangle$

Conjunto de tipo T

Operación	Sintaxis
conjunto por extensión	$\{\}$, $\{x, y, z\}$
tamaño	$\lvert c \rvert$
pertenece	$i \in c$
union	$c_1 \cup c_2$
intersección	$c_1 \cap c_2$
diferencia	$c_1 - c_2$

dicc$\langle K, V \rangle$

Diccionario que asocia claves de tipo K con valores de tipo V

Operación	Sintaxis
diccionario por extensión	$\{\}$, $\{\text{“}juan\text{”} : 20, \text{“}diego\text{”} : 10\}$
tamaño	$\lvert d \rvert$
pertenece (hay clave)	$k \in d$
valor	$d[k]$
claves	$claves(d)$
setear valor	$setKey(d, k, v)$
eliminar valor	$delKey(d, k)$

¡Ahora ustedes! – Ej 4

Especifique el TAD Diccionario$\langle K, V \rangle$ con las siguientes operaciones:

- *nuevoDiccionario*: que crea un diccionario vacío
- *definir*: que agrega un par clave-valor al diccionario
- *obtener*: que devuelve el valor asociado a una clave
- *está*: que devuelve true si la clave está en el diccionario
- *borrar*: que elimina una clave del diccionario

¡Recreo!

Un TAD “bien hecho”

Un TAD debería ser

- **Correcto**: Especifica el problema que queremos resolver, cubriendo todos los casos y las restricciones necesarias.
- **Legible**: Quien lee el TAD debería poder entender qué problema se está resolviendo sin dificultad.
- Un buen **Modelo** del problema: Representa la información sobre la que estamos especificando fielmente, sin introducir información extra

Declaratividad – Renombres de tipos

Los nombres que elegimos al definir un TAD deben ser declarativos, es decir, describir para qué se usa o qué hace cada parte del mismo.

Renombres de tipos

Queremos representar alumnes de una escuela con su DNI:

Alumne `ES` $\mathbb{Z}$
`TAD` Escuela {
    ...
    `proc` `calificarAlumne`(inout $e : Escuela$, in $a : Alumne$, in $nota : \mathbb{Z}$) {
        ...
    }
}

Declaratividad – `structs` vs. `tuplas`

Es más declarativo usar `structs` con nombres en lugar de `tuplas`

Alumne como DNI y promedio

Alumne `ES` tupla $\langle \mathbb{Z}, \mathbb{R} \rangle$

```
proc promedio(in a : Alumne) : ℝ {
    asegura { res = a_1 }
}
```

Alumne `ES` struct $\langle DNI : \mathbb{Z}, promedio : \mathbb{R} \rangle$

```
proc promedio(in a : Alumne) : ℝ {
    asegura { res = a.promedio }
}
```

Modelado

Los observadores y operaciones de un TAD definen un **modelo** de un problema. Sólo con leerlos deberíamos poder tener una idea de qué es lo importante del problema, qué información está relacionada con cuál y cómo manipularla.

Modelado – Observadores

Los observadores y operaciones de un TAD definen un **modelo** de un problema. Sólo con leerlos deberíamos poder tener una idea de qué es lo importante del problema, qué información está relacionada con cuál y cómo manipularla.

Alumnes de una escuela

- `obs` alumnes : conj$\langle Alumne \rangle$

Modelado – Observadores

Los observadores y operaciones de un TAD definen un **modelo** de un problema. Sólo con leerlos deberíamos poder tener una idea de qué es lo importante del problema, qué información está relacionada con cuál y cómo manipularla.

Alumnes de una escuela

- `obs` alumnes : conj$\langle Alumne \rangle$
- `obs` alumnes : seq$\langle Alumne \rangle$

Modelado – Observadores

Los observadores y operaciones de un TAD definen un **modelo** de un problema. Sólo con leerlos deberíamos poder tener una idea de qué es lo importante del problema, qué información está relacionada con cuál y cómo manipularla.

Alumnes de una escuela

- `obs` alumnes : conj$\langle Alumne \rangle$
- `obs` alumnes : seq$\langle Alumne \rangle$
- `obs` alumnes : dicc$\langle Alumne, Promedio \rangle$

Modelado – Operaciones

Los observadores y operaciones de un TAD definen un **modelo** de un problema. Sólo con leerlos deberíamos poder tener una idea de qué es lo importante del problema, qué información está relacionada con cuál y cómo manipularla.

Alumnes de una escuela

- `proc abrirEscuela()` : $Escuela$

Modelado – Operaciones

Los observadores y operaciones de un TAD definen un **modelo** de un problema. Sólo con leerlos deberíamos poder tener una idea de qué es lo importante del problema, qué información está relacionada con cuál y cómo manipularla.

Alumnes de una escuela

- `proc abrirEscuela`$() : Escuela$
- `proc abrirEscuela`$(\text{in } alumnes : \text{seq}\langle Alumne \rangle) : Escuela$

Modelado – Operaciones

Los observadores y operaciones de un TAD definen un **modelo** de un problema. Sólo con leerlos deberíamos poder tener una idea de qué es lo importante del problema, qué información está relacionada con cuál y cómo manipularla.

Alumnes de una escuela

- `proc abrirEscuela`() : $Escuela$
- `proc abrirEscuela`(in $alumnes$: seq$\langle Alumne \rangle$) : $Escuela$
- `proc abrirEscuela`(in $alumnes$: conj$\langle Alumne \rangle$) : $Escuela$

Modelado

- **No existe** una definición objetiva de qué es “un buen modelo”
- Casi siempre hay más de un modelo correcto para un problema dado
- Casi siempre hay más de un modelo “bueno” para un problema dado

Modelado

- **No existe** una definición objetiva de qué es “un buen modelo”
- Casi siempre hay más de un modelo correcto para un problema dado
- Casi siempre hay más de un modelo “bueno” para un problema dado

¿Cómo se si mi TAD está “bien hecho”?

Modelado

- **No existe** una definición objetiva de qué es "un buen modelo"
- Casi siempre hay más de un modelo correcto para un problema dado
- Casi siempre hay más de un modelo "bueno" para un problema dado

¿Cómo se si mi TAD está "bien hecho"?

Una respuesta poco sastisfactoria

Resolviendo muchos problemas durante mucho tiempo

Modelado

- **No existe** una definición objetiva de qué es "un buen modelo"
- Casi siempre hay más de un modelo correcto para un problema dado
- Casi siempre hay más de un modelo "bueno" para un problema dado

¿Cómo se si mi TAD está "bien hecho"?

Una respuesta poco sastisfactoria

Resolviendo muchos problemas durante mucho tiempo

¡No se asusten!

- Es su primer intento de modelado
- No esperamos que lo hagan perfecto

Buscaminas

El juego del **Buscaminas** consiste en un tablero rectangular dividido en casillas que pueden estar libres o tener una bomba. Al iniciar el juego todas las casillas están "cubiertas", es decir, no se puede distinguir si tienen una bomba o no.

En cada turno hay que descubrir alguna de las casillas cubiertas. Si se descubre una bomba, reventamos y perdemos el juego. Si se descubre una casilla libre podemos seguir jugando. Para las casillas descubiertas el juego nos dice cuántas bombas hay alrededor de esa casilla, un número que puede ir del 0 al 8 aunque en una interfaz gráfica el 0 suele no mostrarse.

Cuando además de estar libre la casilla tiene 0 bombas alrededor, se descubren automáticamente todas sus casillas adyacentes. Esto puede desatar una reacción en cadena y descubrir un grupo grande de casillas en una sola jugada.

El objetivo del juego es descubrir todas las casillas libres, sin reventar en el proceso. El juego muestra el estado actual con una carita que puede ser "muerta" (perdimos) "feliz" (podemos seguir jugando) o "cool" (ganamos!)

Como ayuda se pueden poner banderines en las casillas cubiertas, para marcar que se cree que hay una bomba ahí y evitar descubrirlas. No se permite tratar de jugar una casilla que tiene banderín, pero se pueden sacar los banderines.