Algoritmos y Estructuras de Datos

Segundo cuatrimestre - 2025

Departamento de Computación - FCEyN - UBA

Tipos Abstractos de Datos

¿Qué es un TAD?

- TAD quiere decir Tipo Abstracto de Datos
- ¿Qué es un Tipo Abstracto de Datos?
  - Es un tipo de datos porque define un conjunto de valores y las operaciones que se pueden realizar sobre ellos
  - Es abstracto ya que para utilizarlos, no se necesita conocer los detalles de la representación interna ni cómo están implementadas sus operaciones.
  - No conocemos "la forma" de los valores
  - Describe el "qué" y no el "cómo"
  - Son una forma de modularizar a nivel de los datos
- ¿Qué TADs recuerdan de IP?
  - Diccionario
  - Pila
  - Cola

¿Qué es un TAD?

Ejemplo - Pila

- Una pila es una lista de elementos de la cual se puede extraer el último elemento insertado.
- Operaciones básicas
  - **apilar:** ingresa un elemento a la pila
  - **desapilar:** saca el último elemento insertado
  - **tope:** devuelve (sin sacar) el último elemento insertado
  - **vacía:** retorna verdadero si está vacía
- Todo esto es lo que ya sabemos (de IP), pero ...
  - ¿**Qué** hacen **exactamente** estas operaciones.
  - ¿Y si tuvieramos que **demostrar** un programa que usa una Pila?
  - Vamos a tener que especificar estas operaciones.

¿Qué es un TAD?

Ejemplo - Conjunto

- El TAD conjunto es una abstracción de un conjunto matemático, que "contiene" "cosas" (todas del mismo tipo), sus "elementos".
- Hay operaciones para **agregar** y **sacar** elementos y para ver si algo está o no (**pertenece**). Se puede saber cuántos elementos tiene.
- El conjunto no tiene en cuenta repetidos: si en un conjunto de números agregamos el 1, el 5 y otra vez el 1, la cantidad de elementos será 2.

¿Qué es un TAD?

Ejemplo - Punto 2D

- El TAD punto 2D es una abstracción de un punto en el plano cartesiano.
- Tiene operaciones para moverlo, rotarlo sobre el eje o alejarlo del centro, etc

¿Qué caracteriza a un TAD?

- **Instancias:** que pertenecen a su conjunto de valores
- **Operaciones:**
  - para crear una nueva instancia
  - para calcular valores a partir de una instancia
  - para modificar
- **Observadores:**
  - Permiten proyectar una instancia del TAD a un *dominio semántico* más conocido. Vamos a "explicar" el TAD en términos de este dominio.
  - Son funciones que para una instancia de un TAD devuelven una instancia de los tipos de datos del lenguaje de especificación ($\mathbb{Z}$, $\mathbb{R}$, $seq < T >$, $conj < T >$, etc.)
  - Queremos elegir el dominio semántico más natural para el TAD que vamos a especificar. Queremos ser minimales en esa elección.
  - En un instante de tiempo, lo retornado por todos los observadores del TAD para una instancia caracterizan a esa instancia.

Observadores

Ejemplo: Pila

- ¿Qué tipo básico del lenguaje de especificación nos puede ayudar para especificar las operaciones de una pila?
- Una secuencia:

  `obs s: seq<T>`

  - La pila vacía es una secuencia vacía
  - Cuando apilamos le agregamos un elemento a la secuencia (¿Cuál?)
  - Cuando desapilamos le sacamos un elemento a la secuencia (¿Cuál?)
  - Cuando queramos ver el elemento de "arriba", miramos un elemento de la secuencia (¿Cuál?)

Observadores

Ejemplo: Pila v2

- ¿Qué tipo básico del lenguaje de especificación nos puede ayudar para especificar las operaciones de una pila?
- Una secuencia:

  `obs c: conj<`$T \times \mathbb{R}$`>`

  - La pila vacía es un conjunto vacío
  - Cuando desapilamos le sacamos un par al conjunto (¿Cuál?) y devolvemos el primer elemento del par
  - Cuando apilamos $t : T$ le agregamos un par al conjunto (¿Cuál?)
  - Cuando queramos ver el elemento de "arriba"...

Observadores

Ejemplo: TAD punto 2D

- El estado del TAD punto 2D puede ser dado por:
  - variables de estado para las coordenadas cartesianas

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

    `obs y:` $\mathbb{R}$
  - o, variables de estado para las coordenadas polares

    `obs rho:` $\mathbb{R}$

    `obs theta:` $\mathbb{R}$
  - ¡Pero no ambas!
- ¿Podríamos tener un solo observador real (por ejemplo, una sola coordenada)?
  - No nos serviría, porque no se puede describir un punto del plano mediante una sola coordenada. No nos alcanza.

Observadores

Ejemplo: TAD conjunto

- El estado del TAD conjunto puede ser:
  - una variable de tipo *conj* $< T >$ (el conjunto de nuestro lenguaje de especificación)

    ```
    obs elems: conj<T>
    ```

    Nota: Formalmente, las variables de estado pueden considerarse también funciones como

    ```
    obs elems(c: Conjunto<T>): conj<T>
    ```

  - O, una variable del tipo secuencia:

    ```
    obs elems: seq<T>
    ```

  - ¿Cuál es la elección más natural?

Observadores

- El conjunto de observadores tiene que ser completo. Tenemos que poder observar todas las características que nos interesan de las instancias.
- A partir de los observadores se tiene que poder distinguir si dos instancias son distintas
- Todas las operaciones tienen que poder ser descriptas a partir de los observadores

Operaciones de un TAD

- Las operaciones del TAD indican qué se puede hacer con una instancia de un TAD
- Las especificamos con nuestro lenguaje de especificación
- Para indicar qué hacen, usamos precondiciones y postcondiciones (requiere y asegura)

```
proc agregar(inout c: Conjunto< T >, in e:T)
      requiere {...}
      asegura {...}
```

- Para eso hablaremos del estado del TAD (o sea, del valor de sus observadores) antes y después de aplicar la operación

Especifiquemos un TAD

Ejemplo - Pila

```
TAD Pila<T> {
        obs s: seq<T>

        proc pilaVacía(): Pila<T>
                asegura {res.s = ⟨⟩}

        proc vacía(in p: Pila<T>): bool
                asegura {res = true ↔ p.s = ⟨⟩}

        proc apilar(inout p: Pila<T>, in e: T)
                requiere {p = P_0}
                asegura {p.s = concat(P_0.s, ⟨e⟩)}

        proc desapilar(inout p: Pila<T>): T
                requiere {p = P_0}
                requiere {p.s ≠ ⟨⟩}
                asegura {p.s = subseq(P_0.s, 0, |P_0.s| − 1)}
                asegura {res = P_0.s[|P_0.s| − 1]}

        proc tope(in p: Pila<T>): T
                requiere {p.s ≠ ⟨⟩}
                asegura {res = p.s[|p.s| − 1]}
}
```

Especifiquemos un TAD

Ejemplo - Pila v2

```
TAD Pila<T> {
        obs c: conj<T x ℝ>

        pred esMax(c:conj<tupla<T x ℝ>>, x:tupla<T x ℝ>)
                {x ∈ c ∧ (∀y:tupla<T x ℝ>)(y ∈ c ∧ x ≠ y → y_1 < x_1)}

        proc pilaVacía(): Pila<T>
                asegura {res.c = {}}

        proc vacía(in p: Pila<T>): bool
                asegura {res = true ↔ p.c = {}}

        proc apilar(inout p: Pila<T>, in e: T)
                requiere {p = P_0}
                asegura {(∃r : ℝ)(p.c = P_0.c ∪ {(e, r)} ∧ esMax(p.c, (e, r)))}

        proc desapilar(inout p: Pila<T>): T
                requiere {p = P_0 ∧ p.c ≠ {}}
                asegura {(∃x : tupla<T x ℝ>)
                                (p.c = P_0.c \ {x} ∧ res = x_0 ∧ esMax(P_0.c, x))}

        proc tope(in p: Pila<T>): T
                requiere {p.c ≠ {}}
                asegura {(∃x : tupla<T x ℝ>)(res = x_0 ∧ esMax(p.c, x))}
(sigue...)
```

Especifiquemos un TAD

Ejemplo - Pila v2 (continuación)

TAD Pila<T> {

.

.

.

proc igualdad(in p: Pila<T>, in q: Pila<T>): bool

asegura { res $=$ true $\leftrightarrow$

$(\exists m$:dict<$\mathbb{R}$x$\mathbb{R}$>$)($

/* m es una biyección de los reales usados en p y q */

$(\forall x : \texttt{tupla<T x } \mathbb{R}\texttt{>})(x \in p.c \rightarrow x_1 \in claves(m)) \wedge$

$(\forall x : \texttt{tupla<T x } \mathbb{R}\texttt{>})(x \in q.c \rightarrow ((\exists r : \mathbb{R})(m(r) = x_1 \wedge$

$(\forall r' : \mathbb{R})(m(r') = x_1 \rightarrow r = r'))) \wedge$

/* m preserva elementos en T */

$(\forall x : \texttt{tupla<T x } \mathbb{R}\texttt{>})(x \in p.c \rightarrow (x_0, m(x_1)) \in q.c) \wedge$

/* m preserva el orden de reales */

$(\forall x, y : \texttt{tupla<T x } \mathbb{R}\texttt{>})(x \in p.c \wedge y \in p.c \rightarrow x_1 < y_1 \leftrightarrow m(x_1) < m(y_1))$

)

}

Especifiquemos un TAD

Ejemplo - Conjunto

```
TAD Conjunto<T> {
        obs elems: conj<T>

        proc conjVacío(): Conjunto<T>
                asegura {res.elems = ⟨⟩}

        proc pertenece(in c: Conjunto<T>, in e: T): bool
                asegura {res = true ↔ e ∈ c.elems}

        proc agregar(inout c: Conjunto<T>, in e: T)
                requiere {c = C0}
                asegura {c.elems = C0.elems ∪ ⟨e⟩}

        proc sacar(inout c: Conjunto<T>, in e: T)
                requiere {c = C0}
                asegura {c.elems = C0.elems − ⟨e⟩}

        proc unir(inout c: Conjunto<T>, in c': Conjunto<T>):TAREA
        proc restar(inout c: Conjunto<T>, in c': Conjunto<T>):TAREA
        proc intersecar(inout c: Conjunto<T>, in c': Conjunto<T>):TAREA
        proc tamaño(in c: Conjunto<T>): ℤ:TAREA
}
```

Especifiquemos un TAD

Ejemplo - Punto 2D

```
TAD Punto {

        obs x: ℝ
        obs y: ℝ

        proc nuevoPunto(in x: ℝ, in y: ℝ): Punto
                asegura {res.x = x}
                asegura {res.y = y}

        proc coordX(in p: Punto): ℝ
                asegura {res = p.x}

        proc coordY(in p: Punto): ℝ
                asegura {res = p.y}

        proc coordTheta(in p: Punto): ℝ
                asegura {res = safearctan(p.x, p.y)}

        proc coordRho(in p: Punto): ℝ
                asegura {res = sqrt(p.x ** 2 + p.y ** 2)}

        proc mover(inout p: Punto, in deltaX: ℝ, in deltaY: ℝ)
                requiere {p = P0}
                asegura {p.x = P0.x + deltaX}
                asegura {p.y = P0.y + deltaY}

        aux safearctan(x: ℝ, y: ℝ) = ifThenElseFi(x == 0,
π/2*signo(y),arctan(y/x))
```