

Diseño I: Elección de estructuras

Clase práctica

Algoritmos y Estructuras de Datos II

Diferencias entre especificación y diseño

- En la etapa de especificación:
 - ▶ Nos ocupamos del '¿Qué?'
 - ▶ Lo explicamos usando TADs
 - ▶ Ese '¿Qué?' se explica bajo el paradigma funcional
- En la etapa de diseño:
 - ▶ Nos ocupamos del '¿Cómo?'
 - ▶ Lo explicamos con *módulos de abstracción*
 - ▶ Ese '¿Cómo?' lo explicaremos usando el paradigma imperativo
 - ▶ Tenemos un *contexto de uso* que nos fuerza a tomar decisiones respecto de la estructura.

Partes de un módulo

Un módulo de abstracción se divide en tres secciones:

Partes de un módulo

Un módulo de abstracción se divide en tres secciones:

- 1 **Interfaz:** sección accesible a los usuarios del módulo (e.g., otros módulos) que detalla cada operación exportada:

Partes de un módulo

Un módulo de abstracción se divide en tres secciones:

- 1 **Interfaz:** sección accesible a los usuarios del módulo (e.g., otros módulos) que detalla cada operación exportada:
 - ▶ Signatura: que recibe y que devuelve.

Partes de un módulo

Un módulo de abstracción se divide en tres secciones:

- 1 **Interfaz:** sección accesible a los usuarios del módulo (e.g., otros módulos) que detalla cada operación exportada:
 - ▶ Signatura: que recibe y que devuelve.
 - ▶ Precondición: condiciones sobre los input.

Partes de un módulo

Un módulo de abstracción se divide en tres secciones:

- 1 **Interfaz:** sección accesible a los usuarios del módulo (e.g., otros módulos) que detalla cada operación exportada:
 - ▶ Signatura: que recibe y que devuelve.
 - ▶ Precondición: condiciones sobre los input.
 - ▶ Postcondición: lo que debe cumplirse después de la ejecución.

Partes de un módulo

Un módulo de abstracción se divide en tres secciones:

- ➊ **Interfaz:** sección accesible a los usuarios del módulo (e.g., otros módulos) que detalla cada operación exportada:
 - ▶ Signatura: que recibe y que devuelve.
 - ▶ Precondición: condiciones sobre los input.
 - ▶ Postcondición: lo que debe cumplirse después de la ejecución.
 - ▶ Descripción (Importante!).

Partes de un módulo

Un módulo de abstracción se divide en tres secciones:

- 1 **Interfaz:** sección accesible a los usuarios del módulo (e.g., otros módulos) que detalla cada operación exportada:
 - ▶ Signatura: que recibe y que devuelve.
 - ▶ Precondición: condiciones sobre los input.
 - ▶ Postcondición: lo que debe cumplirse después de la ejecución.
 - ▶ Descripción (Importante!).
 - ▶ Complejidad.

Partes de un módulo

Un módulo de abstracción se divide en tres secciones:

- 1 **Interfaz:** sección accesible a los usuarios del módulo (e.g., otros módulos) que detalla cada operación exportada:
 - ▶ Signatura: que recibe y que devuelve.
 - ▶ Precondición: condiciones sobre los input.
 - ▶ Postcondición: lo que debe cumplirse después de la ejecución.
 - ▶ Descripción (Importante!).
 - ▶ Complejidad.
 - ▶ Aliasing.

Partes de un módulo

Un módulo de abstracción se divide en tres secciones:

- ❶ **Interfaz:** sección accesible a los usuarios del módulo (e.g., otros módulos) que detalla cada operación exportada:
 - ▶ Signatura: que recibe y que devuelve.
 - ▶ Precondición: condiciones sobre los input.
 - ▶ Postcondición: lo que debe cumplirse después de la ejecución.
 - ▶ Descripción (Importante!).
 - ▶ Complejidad.
 - ▶ Aliasing.
- ❷ **Representación:** sección no accesible a los usuarios del módulo que detalla la estructura de representación y los algoritmos justificando las complejidades deseadas. Define también *Rep* y *Abs*.

Partes de un módulo

Un módulo de abstracción se divide en tres secciones:

- 1 **Interfaz:** sección accesible a los usuarios del módulo (e.g., otros módulos) que detalla cada operación exportada:
 - ▶ Signatura: que recibe y que devuelve.
 - ▶ Precondición: condiciones sobre los input.
 - ▶ Postcondición: lo que debe cumplirse después de la ejecución.
 - ▶ Descripción (Importante!).
 - ▶ Complejidad.
 - ▶ Aliasing.
- 2 **Representación:** sección no accesible a los usuarios del módulo que detalla la estructura de representación y los algoritmos justificando las complejidades deseadas. Define también *Rep* y *Abs*.
- 3 **Servicios usados:** detalla los supuestos sobre los servicios usados de otros módulos. Estos requisitos justifican las complejidades de la sección anterior (que a su vez justifican las de la interfaz).
 - ▶ Ejemplo: cola implementada sobre una lista enlazada.

Apuntes

- No se olviden de los apuntes:
 - ▶ Apunte de diseño.
 - ▶ Apunte de módulos básicos (módulos completos).

Apuntes

- No se olviden de los apuntes:
 - ▶ Apunte de diseño.
 - ▶ Apunte de módulos básicos (módulos completos).
- Veamos un poco estos apuntes...

Apuntes

- No se olviden de los apuntes:
 - ▶ Apunte de diseño.
 - ▶ Apunte de módulos básicos (módulos completos).
- Veamos un poco estos apuntes...
- En general no les vamos a pedir módulos completos.

Apuntes

- No se olviden de los apuntes:
 - ▶ Apunte de diseño.
 - ▶ Apunte de módulos básicos (módulos completos).
- Veamos un poco estos apuntes...
- En general no les vamos a pedir módulos completos.
- Sólo por hoy: diseñar un módulo (casi) completo.

Ejercicio: ¡Subite!

Sistema de tarjetas de transporte para el subte:

- En las boleterías, se venden tarjetas de 1, 2, 5, 10 y 30 viajes
- En la entrada a los andenes, hay molinetes con lectores de tarjetas
- Cuando el usuario pasa su tarjeta por el lector:
 - ▶ Si la tarjeta está agotada o es inválida, se informa de esto en el visor y no se abre el molinete
 - ▶ Si hay viajes en la tarjeta
 - ★ Se abre el molinete
 - ★ Se muestra el saldo restante en el visor
 - ★ Se actualiza en el sistema el nuevo saldo para esa tarjeta
 - ★ Se registra en el sistema el día y la hora del acceso

Ejercicio: ¡Subite!

Sistema de tarjetas de transporte para el subte:

- En las boleterías, se venden tarjetas de 1, 2, 5, 10 y 30 viajes
- En la entrada a los andenes, hay molinetes con lectores de tarjetas
- Cuando el usuario pasa su tarjeta por el lector:
 - ▶ Si la tarjeta está agotada o es inválida, se informa de esto en el visor y no se abre el molinete
 - ▶ Si hay viajes en la tarjeta
 - ★ Se abre el molinete
 - ★ Se muestra el saldo restante en el visor
 - ★ Se actualiza en el sistema el nuevo saldo para esa tarjeta
 - ★ Se registra en el sistema el día y la hora del acceso
- Nos piden tiempo de acceso **sublineal** (en la cantidad total de tarjetas y en peor caso) para la operación de usar la tarjeta.

TAD SUBITE

observadores básicos

tarjetas : subite \longrightarrow conj(tarjeta)

crédito : subite $s \times$ tarjeta $t \longrightarrow$ nat $\{t \in \text{tarjetas}(s)\}$

viajes : subite $s \times$ tarjeta $t \longrightarrow$ secu(FechaHora) $\{t \in \text{tarjetas}(s)\}$

ultimaTarjeta : subite $s \longrightarrow$ tarjeta $\{\# \text{tarjetas}(s) > 0\}$

generadores

Crear : \longrightarrow subite

NuevaTarjeta : subite $s \times$ nat $c \longrightarrow$ subite $\{c \in \{1,2,5,10,30\}\}$

UsarTarjeta : subite $s \times$ tarjeta $t \times$ fechaHora \longrightarrow subite

$\{t \in \text{tarjetas}(s) \wedge \text{crédito}(s,t) > 0\}$

axiomas

...

Fin TAD

Observación: ultimaTarjeta devuelve la última tarjeta creada por NuevaTarjeta.

Diseñado...

¿Qué hacemos ahora?

Diseñado...

¿Qué hacemos ahora?

- ¿Elegir estructura o definir la interfaz?

Diseñado...

¿Qué hacemos ahora?

- ¿Elegir estructura o definir la interfaz?
↔ La interfaz será la misma sin importar la estructura, y además nos puede dar ideas para esta última...

Diseñado...

¿Qué hacemos ahora?

- ¿Elegir estructura o definir la interfaz?
↔ La interfaz será la misma sin importar la estructura, y además nos puede dar ideas para esta última...
- ¿Hay que pensar cuáles serán las operaciones provistas por la interfaz?

Diseñado...

¿Qué hacemos ahora?

- ¿Elegir estructura o definir la interfaz?
↪ La interfaz será la misma sin importar la estructura, y además nos puede dar ideas para esta última...
- ¿Hay que pensar cuáles serán las operaciones provistas por la interfaz?
↪ ¡Sí! No siempre salen directamente del TAD...

Diseñado...

¿Qué hacemos ahora?

- ¿Elegir estructura o definir la interfaz?
↪ La interfaz será la misma sin importar la estructura, y además nos puede dar ideas para esta última...
- ¿Hay que pensar cuáles serán las operaciones provistas por la interfaz?
↪ ¡Sí! No siempre salen directamente del TAD...
- Definamos entonces las operaciones y para cada una definir:
 - ▶ **Descripción**
 - ▶ Precondición
 - ▶ Postcondición
 - ▶ Complejidad (!)
 - ▶ Aliasing (!)

Diseñado...

¿Qué hacemos ahora?

- ¿Elegir estructura o definir la interfaz?
↪ La interfaz será la misma sin importar la estructura, y además nos puede dar ideas para esta última...
- ¿Hay que pensar cuáles serán las operaciones provistas por la interfaz?
↪ ¡Sí! No siempre salen directamente del TAD...
- Definamos entonces las operaciones y para cada una definir:
 - ▶ **Descripción**
 - ▶ Precondición
 - ▶ Postcondición
 - ▶ Complejidad (!)
 - ▶ Aliasing (!)
- ¿Cómo elegimos entonces las operaciones?

Diseñado...

¿Qué hacemos ahora?

- ¿Elegir estructura o definir la interfaz?
↪ La interfaz será la misma sin importar la estructura, y además nos puede dar ideas para esta última...
- ¿Hay que pensar cuáles serán las operaciones provistas por la interfaz?
↪ ¡Sí! No siempre salen directamente del TAD...
- Definamos entonces las operaciones y para cada una definir:
 - ▶ **Descripción**
 - ▶ Precondición
 - ▶ Postcondición
 - ▶ Complejidad (!)
 - ▶ Aliasing (!)
- ¿Cómo elegimos entonces las operaciones?
↪ Depende de muchas cosas (principalmente el contexto de uso).

Diseñado...

Antes que nada arrancamos con la cabecera:

- **Módulo:** Subite
- **Se explica con:** TAD SUBITE
- **Géneros:** subite
- **Operaciones:** ...

Subite: Creación y nuevas tarjetas

Crear : \longrightarrow subite

- Signatura:

Subite: Creación y nuevas tarjetas

Crear : \longrightarrow subite

- Signatura: Crear() \rightarrow res: subite
- Pre:

Subite: Creación y nuevas tarjetas

Crear : \longrightarrow subite

- Signatura: Crear() \rightarrow res: subite
- Pre: { true }
- Post:

Subite: Creación y nuevas tarjetas

Crear : \longrightarrow subite

- Signatura: Crear() \rightarrow res: subite
- Pre: { true }
- Post: { res = Crear } (¿sombbrero?)
- Descripción:

Subite: Creación y nuevas tarjetas

Crear : \longrightarrow subite

- Signatura: Crear() \rightarrow res: subite
- Pre: { true }
- Post: { res = Crear } (¿sombbrero?)
- Descripción: Crea una nueva instancia
- Complejidad: La definiremos más adelante.
- Aliasing: (no aplica)

Subite: Creación y nuevas tarjetas

Crear : \longrightarrow subite

- Signatura: $\text{Crear}() \rightarrow \text{res: subite}$
- Pre: $\{ \text{true} \}$
- Post: $\{ \text{res} = \text{Crear} \}$ (*¿sombbrero?*)
- Descripción: Crea una nueva instancia
- Complejidad: La definiremos más adelante.
- Aliasing: (no aplica)

NuevaTarjeta : $\text{subite } s \times \text{nat } c \longrightarrow \text{subite}$

$\{c \in \{1,2,5,10,30\}\}$

ultimaTarjeta : $\text{subite } s \longrightarrow \text{tarjeta}$

$\{\# \text{tarjetas}(s) > 0\}$

- Signatura:

Subite: Creación y nuevas tarjetas

Crear : \longrightarrow subite

- Signatura: Crear() \rightarrow res: subite
- Pre: { true }
- Post: { res = Crear } (*¿sombbrero?*)
- Descripción: Crea una nueva instancia
- Complejidad: La definiremos más adelante.
- Aliasing: (no aplica)

NuevaTarjeta : subite $s \times$ nat $c \longrightarrow$ subite

$\{c \in \{1,2,5,10,30\}\}$

ultimaTarjeta : subite $s \longrightarrow$ tarjeta

$\{\#tarjetas(s) > 0\}$

- Signatura: NuevaTarjeta(**inout** s : subite, **in** c : nat) \rightarrow **res: tarjeta**
- Pre:

Subite: Creación y nuevas tarjetas

Crear : \longrightarrow subite

- Signatura: $\text{Crear}() \rightarrow \text{res: subite}$
- Pre: $\{ \text{true} \}$
- Post: $\{ \text{res} = \text{Crear} \}$ (¿sombbrero?)
- Descripción: Crea una nueva instancia
- Complejidad: La definiremos más adelante.
- Aliasing: (no aplica)

NuevaTarjeta : $\text{subite } s \times \text{nat } c \longrightarrow \text{subite}$

$\{c \in \{1,2,5,10,30\}\}$

ultimaTarjeta : $\text{subite } s \longrightarrow \text{tarjeta}$

$\{\# \text{tarjetas}(s) > 0\}$

- Signatura: $\text{NuevaTarjeta}(\text{inout } s: \text{subite}, \text{in } c: \text{nat}) \rightarrow \text{res: tarjeta}$
- Pre: $\{ s = s_0 \wedge c \in \{1, 2, 5, 10, 30\} \}$

Subite: Creación y nuevas tarjetas

Crear : \longrightarrow subite

- Signatura: Crear() \rightarrow res: subite
- Pre: { true }
- Post: { res = Crear } (*¿sombbrero?*)
- Descripción: Crea una nueva instancia
- Complejidad: La definiremos más adelante.
- Aliasing: (no aplica)

NuevaTarjeta : subite $s \times$ nat $c \longrightarrow$ subite

$\{c \in \{1,2,5,10,30\}\}$

ultimaTarjeta : subite $s \longrightarrow$ tarjeta

$\{\#tarjetas(s) > 0\}$

- Signatura: NuevaTarjeta(**inout** s : subite, **in** c : nat) \rightarrow **res**: tarjeta
- Pre: { $s = s_0 \wedge c \in \{1, 2, 5, 10, 30\}$ }
- Post:

Subite: Creación y nuevas tarjetas

Crear : \longrightarrow subite

- Signatura: Crear() \rightarrow res: subite
- Pre: { true }
- Post: { res = Crear } (*¿sombbrero?*)
- Descripción: Crea una nueva instancia
- Complejidad: La definiremos más adelante.
- Aliasing: (no aplica)

NuevaTarjeta : subite $s \times$ nat $c \longrightarrow$ subite

$\{c \in \{1,2,5,10,30\}\}$

ultimaTarjeta : subite $s \longrightarrow$ tarjeta

$\{\#tarjetas(s) > 0\}$

- Signatura: NuevaTarjeta(**inout** s : subite, **in** c : nat) \rightarrow **res**: tarjeta
- Pre: { $s = s_0 \wedge c \in \{1, 2, 5, 10, 30\}$ }
- Post: { $s = NuevaTarjeta(s_0, c) \wedge res = ultimaTarjeta(s)$ }

Subite: Creación y nuevas tarjetas

Crear : \longrightarrow subite

- Signatura: $\text{Crear}() \rightarrow \text{res: subite}$
- Pre: $\{ \text{true} \}$
- Post: $\{ \text{res} = \text{Crear} \}$ (¿sombbrero?)
- Descripción: Crea una nueva instancia
- Complejidad: La definiremos más adelante.
- Aliasing: (no aplica)

NuevaTarjeta : $\text{subite } s \times \text{nat } c \longrightarrow \text{subite}$

$\{c \in \{1,2,5,10,30\}\}$

ultimaTarjeta : $\text{subite } s \longrightarrow \text{tarjeta}$

$\{\# \text{tarjetas}(s) > 0\}$

- Signatura: $\text{NuevaTarjeta}(\text{inout } s: \text{subite}, \text{in } c: \text{nat}) \rightarrow \text{res: tarjeta}$
- Pre: $\{ s = s_0 \wedge c \in \{1, 2, 5, 10, 30\} \}$
- Post: $\{ s = \text{NuevaTarjeta}(s_0, c) \wedge \text{res} = \text{ultimaTarjeta}(s) \}$
- Descripción: Agrega una nueva tarjeta al sistema y la retorna.

Subite: Creación y nuevas tarjetas

Crear : \longrightarrow subite

- Signatura: Crear() \rightarrow res: subite
- Pre: { true }
- Post: { res = Crear } (*¿sombbrero?*)
- Descripción: Crea una nueva instancia
- Complejidad: La definiremos más adelante.
- Aliasing: (no aplica)

NuevaTarjeta : subite $s \times$ nat $c \longrightarrow$ subite

$\{c \in \{1,2,5,10,30\}\}$

ultimaTarjeta : subite $s \longrightarrow$ tarjeta

$\{\#tarjetas(s) > 0\}$

- Signatura: NuevaTarjeta(**inout** s : subite, **in** c : nat) \rightarrow **res**: tarjeta
- Pre: { $s = s_0 \wedge c \in \{1, 2, 5, 10, 30\}$ }
- Post: { $s = NuevaTarjeta(s_0, c) \wedge res = ultimaTarjeta(s)$ }
- Descripción: Agrega una nueva tarjeta al sistema y la retorna.
- Complejidad: La definiremos más adelante.
- Aliasing: (E.g) Se devuelve una referencia no modificable a la nueva tarjeta.

Subite: Usar tarjeta

UsarTarjeta : subite $s \times$ tarjeta $t \times$ fechaHora \longrightarrow subite

$\{t \in \text{tarjetas}(s) \wedge \text{crédito}(s,t) > 0\}$

Supongamos que el contexto de uso nos indica lo siguiente:

- Queremos que UsarTarjeta indique si la misma era válida o no (en lugar de restringir su uso).
- Y además, de usarla, que nos devuelva el nuevo saldo de la tarjeta.

¿Podemos **alterar la signature** para **mejorar la usabilidad** del módulo?

Subite: Usar tarjeta

UsarTarjeta : subite $s \times$ tarjeta $t \times$ fechaHora \longrightarrow subite

$\{t \in \text{tarjetas}(s) \wedge \text{crédito}(s,t) > 0\}$

- Signatura:

Subite: Usar tarjeta

UsarTarjeta : subite $s \times$ tarjeta $t \times$ fechaHora \longrightarrow subite

$\{t \in \text{tarjetas}(s) \wedge \text{crédito}(s,t) > 0\}$

- Signatura:

UsarTarjeta(**inout** s : subite, **in** t : tarjeta, **in** h : fechaHora, **out** c : nat)

\rightarrow **res**: bool

- Pre:

Subite: Usar tarjeta

UsarTarjeta : subite $s \times$ tarjeta $t \times$ fechaHora \longrightarrow subite

$\{t \in \text{tarjetas}(s) \wedge \text{crédito}(s,t) > 0\}$

- Signatura:

UsarTarjeta(**inout** s : subite, **in** t : tarjeta, **in** h : fechaHora, **out** c : nat)

\rightarrow **res**: bool

- Pre: $\{ s = s_0 \}$

Subite: Usar tarjeta

UsarTarjeta : subite $s \times$ tarjeta $t \times$ fechaHora \longrightarrow subite

$\{t \in \text{tarjetas}(s) \wedge \text{crédito}(s,t) > 0\}$

- Signatura:

UsarTarjeta(**inout** s : subite, **in** t : tarjeta, **in** h : fechaHora, **out** c : nat)

\rightarrow **res**: bool

- Pre: $\{ s = s_0 \}$
- Post:

Subite: Usar tarjeta

UsarTarjeta : subite $s \times$ tarjeta $t \times$ fechaHora \longrightarrow subite

$\{t \in \text{tarjetas}(s) \wedge \text{crédito}(s,t) > 0\}$

- Signatura:

UsarTarjeta(**inout** s : subite, **in** t : tarjeta, **in** h : fechaHora, **out** c : nat)

\rightarrow **res**: bool

- Pre: $\{ s = s_0 \}$
- Post: $\{ \text{res} = (t \in \text{tarjetas}(s_0) \wedge \text{crédito}(s_0, t) > 0) \wedge_L$
 $(\text{res} \Rightarrow_L s = \text{UsarTarjeta}(s_0, t, h) \wedge c = \text{crédito}(s, t)) \wedge (\neg \text{res} \Rightarrow s = s_0) \}$

Subite: Usar tarjeta

UsarTarjeta : subite $s \times$ tarjeta $t \times$ fechaHora \longrightarrow subite

$\{t \in \text{tarjetas}(s) \wedge \text{crédito}(s,t) > 0\}$

- Signatura:

UsarTarjeta(**inout** s : subite, **in** t : tarjeta, **in** h : fechaHora, **out** c : nat)

\rightarrow **res**: bool

- Pre: $\{s = s_0\}$
- Post: $\{res = (t \in \text{tarjetas}(s_0) \wedge \text{crédito}(s_0, t) > 0) \wedge_L (res \Rightarrow_L s = \text{UsarTarjeta}(s_0, t, h) \wedge c = \text{crédito}(s, t)) \wedge (\neg res \Rightarrow s = s_0)\}$
- Descripción: Si la tarjeta pasada por parámetro está registrada y puede utilizarla, entonces actualiza su crédito y retorna true
- Complejidad: tiene que ser $< O(n)$, pero lo precisamos luego.
- Aliasing: idem...

Subite: Ver viajes y crédito

$\text{viajes} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{secu}(\text{fechaHora})$	$\{t \in \text{tarjetas}(s)\}$
$\text{crédito} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{nat}$	$\{t \in \text{tarjetas}(s)\}$

Podemos hacer algo parecido para ver los viajes y el crédito....

- Ver viajes:

Subite: Ver viajes y crédito

$\text{viajes} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{secu}(\text{fechaHora})$	$\{t \in \text{tarjetas}(s)\}$
$\text{crédito} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{nat}$	$\{t \in \text{tarjetas}(s)\}$

Podemos hacer algo parecido para ver los viajes y el crédito....

- Ver viajes:
 - ▶ Viajes(**in** s: subite, **in** t: tarjeta, **out** vs: secu(fechaHora)) → res: bool

Subite: Ver viajes y crédito

$\text{viajes} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{secu}(\text{fechaHora})$	$\{t \in \text{tarjetas}(s)\}$
$\text{crédito} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{nat}$	$\{t \in \text{tarjetas}(s)\}$

Podemos hacer algo parecido para ver los viajes y el crédito....

- Ver viajes:
 - ▶ Viajes(**in** s: subite, **in** t: tarjeta, **out vs:** secu(fechaHora)) \rightarrow res: bool
 - ▶ Pre:

Subite: Ver viajes y crédito

$\text{viajes} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{secu}(\text{fechaHora})$	$\{t \in \text{tarjetas}(s)\}$
$\text{crédito} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{nat}$	$\{t \in \text{tarjetas}(s)\}$

Podemos hacer algo parecido para ver los viajes y el crédito....

- Ver viajes:
 - ▶ Viajes(**in** s: subite, **in** t: tarjeta, **out vs:** secu(fechaHora)) \rightarrow **res:** bool
 - ▶ Pre: { true }

Subite: Ver viajes y crédito

$\text{viajes} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{secu}(\text{fechaHora})$	$\{t \in \text{tarjetas}(s)\}$
$\text{crédito} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{nat}$	$\{t \in \text{tarjetas}(s)\}$

Podemos hacer algo parecido para ver los viajes y el crédito....

- Ver viajes:
 - ▶ Viajes(**in** s: subite, **in** t: tarjeta, **out** vs: secu(fechaHora)) \rightarrow res: bool
 - ▶ Pre: { true }
 - ▶ Post:

Subite: Ver viajes y crédito

$\text{viajes} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{secu}(\text{fechaHora})$	$\{t \in \text{tarjetas}(s)\}$
$\text{crédito} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{nat}$	$\{t \in \text{tarjetas}(s)\}$

Podemos hacer algo parecido para ver los viajes y el crédito....

- Ver viajes:
 - ▶ Viajes(**in** s : subite, **in** t : tarjeta, **out vs**: secu(fechaHora)) \rightarrow **res**: bool
 - ▶ Pre: { true }
 - ▶ Post: { $(\text{res} = t \in \text{tarjetas}(s)) \wedge_L (\text{res} \Rightarrow_L \text{vs} = \text{viajes}(s, t))$ }

Subite: Ver viajes y crédito

$\text{viajes} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{secu}(\text{fechaHora})$	$\{t \in \text{tarjetas}(s)\}$
$\text{crédito} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{nat}$	$\{t \in \text{tarjetas}(s)\}$

Podemos hacer algo parecido para ver los viajes y el crédito....

- Ver viajes:

- ▶ Viajes(**in** s: subite, **in** t: tarjeta, **out vs: secu(fechaHora)**) \rightarrow **res: bool**
- ▶ Pre: { true }
- ▶ Post: { $(\text{res} = t \in \text{tarjetas}(s)) \wedge_L (\text{res} \Rightarrow_L \text{vs} = \text{viajes}(s, t))$ }
- ▶ Descripción: Si la tarjeta pasada por parámetro está registrada, retorna true y asigna en *viajes* los viajes realizados
- ▶ Complejidad: La definimos más adelante
- ▶ Aliasing: (E.g) Se devuelve una referencia no modificable a la lista de viajes realizados.

Subite: Ver viajes y crédito

$\text{viajes} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{secu}(\text{fechaHora})$	$\{t \in \text{tarjetas}(s)\}$
$\text{crédito} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{nat}$	$\{t \in \text{tarjetas}(s)\}$

Podemos hacer algo parecido para ver los viajes y el crédito....

- Ver crédito:

Subite: Ver viajes y crédito

viajes	:	subite $s \times$ tarjeta t	\longrightarrow	secu(fechaHora)	$\{t \in \text{tarjetas}(s)\}$
crédito	:	subite $s \times$ tarjeta t	\longrightarrow	nat	$\{t \in \text{tarjetas}(s)\}$

Podemos hacer algo parecido para ver los viajes y el crédito....

- Ver crédito:
 - ▶ Crédito(**in** s : subite, **in** t : tarjeta, **out** cred: nat) \rightarrow res: bool

Subite: Ver viajes y crédito

viajes	:	subite $s \times$ tarjeta t	\longrightarrow	secu(fechaHora)	$\{t \in \text{tarjetas}(s)\}$
crédito	:	subite $s \times$ tarjeta t	\longrightarrow	nat	$\{t \in \text{tarjetas}(s)\}$

Podemos hacer algo parecido para ver los viajes y el crédito....

- Ver crédito:
 - ▶ Crédito(**in** s : subite, **in** t : tarjeta, **out** cred : nat) \rightarrow **res**: bool
 - ▶ Pre:

Subite: Ver viajes y crédito

viajes	:	subite $s \times$ tarjeta t	\longrightarrow	secu(fechaHora)	$\{t \in \text{tarjetas}(s)\}$
crédito	:	subite $s \times$ tarjeta t	\longrightarrow	nat	$\{t \in \text{tarjetas}(s)\}$

Podemos hacer algo parecido para ver los viajes y el crédito....

- Ver crédito:
 - ▶ Crédito(**in** s : subite, **in** t : tarjeta, **out** cred : nat) \rightarrow **res**: bool
 - ▶ Pre: { true }

Subite: Ver viajes y crédito

$\text{viajes} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{secu}(\text{fechaHora})$	$\{t \in \text{tarjetas}(s)\}$
$\text{crédito} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{nat}$	$\{t \in \text{tarjetas}(s)\}$

Podemos hacer algo parecido para ver los viajes y el crédito....

- Ver crédito:
 - ▶ Crédito(**in** s : subite, **in** t : tarjeta, **out** cred : nat) \rightarrow **res**: bool
 - ▶ Pre: { true }
 - ▶ Post:

Subite: Ver viajes y crédito

$\text{viajes} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{secu}(\text{fechaHora})$	$\{t \in \text{tarjetas}(s)\}$
$\text{crédito} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{nat}$	$\{t \in \text{tarjetas}(s)\}$

Podemos hacer algo parecido para ver los viajes y el crédito....

- Ver crédito:

- ▶ Crédito(**in** s : subite, **in** t : tarjeta, **out** cred : nat) \rightarrow **res**: bool
- ▶ Pre: { true }
- ▶ Post: { $(\text{res} = t \in \text{tarjetas}(s)) \wedge_L (\text{res} \Rightarrow_L \text{cred} = \text{crédito}(s, t))$ }

Subite: Ver viajes y crédito

<code>viajes</code>	<code>: subite s × tarjeta t</code>	\longrightarrow	<code>secu(fechaHora)</code>	$\{t \in \text{tarjetas}(s)\}$
<code>crédito</code>	<code>: subite s × tarjeta t</code>	\longrightarrow	<code>nat</code>	$\{t \in \text{tarjetas}(s)\}$

Podemos hacer algo parecido para ver los viajes y el crédito....

- Ver crédito:

- ▶ Crédito(**in** `s`: subite, **in** `t`: tarjeta, **out** `cred`: nat) \rightarrow **res**: bool
- ▶ Pre: { true }
- ▶ Post: { (`res` = `t` \in `tarjetas(s)`) \wedge_L (`res` \Rightarrow_L `cred` = `crédito(s, t)`) }
- ▶ Descripción: Si la tarjeta pasada por parámetro está registrada, retorna true y asigna el crédito en `cred` si no retorna false
- ▶ Complejidad: La definimos más adelante
- ▶ Aliasing: -

Subite: Elección de estructuras

- Elijamos la estructura, sabiendo que la única restricción de complejidad que tenemos es en utilizar la tarjeta ($< O(n)$)

Subite: Elección de estructuras

- Elijamos la estructura, sabiendo que la única restricción de complejidad que tenemos es en utilizar la tarjeta ($< O(n)$)
- Subite se podría representar con `estr`, donde:
 - ▶ `estr` es `dicc(tarjeta, datosTarjeta)`
 - ▶ `tarjeta` es `nat`
 - ▶ `datosTarjeta` es `tupla`
 `crédito: nat,`
 `viajes: secuencia(fechaHora)`

Subite: Elección de estructuras

- Elijamos la estructura, sabiendo que la única restricción de complejidad que tenemos es en utilizar la tarjeta ($< O(n)$)
- Subite se podría representar con `estr`, donde:
 - ▶ `estr` es `dicc(tarjeta, datosTarjeta)`
 - ▶ `tarjeta` es `nat`
 - ▶ `datosTarjeta` es `tupla`
`crédito: nat,`
`viajes: secuencia(fechaHora)`
- ¿Así cumple con las complejidades requeridas?

Subite: Elección de estructuras

- Elijamos la estructura, sabiendo que la única restricción de complejidad que tenemos es en utilizar la tarjeta ($< O(n)$)
- Subite se podría representar con `estr`, donde:
 - ▶ `estr` es `dicc(tarjeta, datosTarjeta)`
 - ▶ `tarjeta` es `nat`
 - ▶ `datosTarjeta` es `tupla`
`crédito: nat,`
`viajes: secuencia(fechaHora)`
- ¿Así cumple con las complejidades requeridas?
- Recordemos que utilizar la tarjeta implica:
 - ▶ **Buscar la tarjeta**
 - ▶ Descontarle el saldo
 - ▶ Agregarle un viaje

Subite: Elección de estructuras

- Elijamos la estructura, sabiendo que la única restricción de complejidad que tenemos es en utilizar la tarjeta ($< O(n)$)
- Subite se podría representar con `estr`, donde:
 - ▶ `estr` es `dicc(tarjeta, datosTarjeta)`
 - ▶ `tarjeta` es `nat`
 - ▶ `datosTarjeta` es `tupla`
`crédito: nat,`
`viajes: secuencia(fechaHora)`
- ¿Así cumple con las complejidades requeridas?
- Recordemos que utilizar la tarjeta implica:
 - ▶ **Buscar la tarjeta**
 - ▶ Descontarle el saldo
 - ▶ Agregarle un viaje
- Escribamos el algoritmo y veamos...

Subite: Algoritmo de UsarTarjeta

function iUSAR_TARJETA(inout e: estr, in t: tarjeta, in h: fechaHora, out crédito: nat)

Subite: Algoritmo de UsarTarjeta

```
function iUSAR_TARJETA(inout e: estr, in t: tarjeta, in h: fechaHora, out crédito: nat)
  if  $\neg$ definido(e, t) then
    res  $\leftarrow$  false
  else
    datos  $\leftarrow$  obtener(e, t)
    if datos.credito = 0 then
      res  $\leftarrow$  false
    else
      datos.credito  $\leftarrow$  datos.credito - 1
      agregar(datos.viajes, h)
      credito  $\leftarrow$  datos.credito
      res  $\leftarrow$  true
    end if
  end if
  devolver res
end function
```

Subite: Algoritmo de UsarTarjeta

```
function iUSAR_TARJETA(inout e: estr, in t: tarjeta, in h: fechaHora, out crédito: nat)
  if  $\neg$ definido(e, t) then                                     ▷  $O(\text{definido})$ 
    res  $\leftarrow$  false
  else
    datos  $\leftarrow$  obtener(e, t)                                ▷  $O(\text{obtener})$ 
    if datos.credito = 0 then
      res  $\leftarrow$  false
    else
      datos.credito  $\leftarrow$  datos.credito - 1
      agregar(datos.viajes, h)                                ▷  $O(\text{agregar})$ 
      credito  $\leftarrow$  datos.credito
      res  $\leftarrow$  true
    end if
  end if
  devolver res
end function
```

Ejercicio: Requisitos de complejidad

- Ok, entonces *definido*, *obtener* y *agregar* tienen que ser sublineales...

Ejercicio: Requisitos de complejidad

- Ok, entonces *definido*, *obtener* y *agregar* tienen que ser sublineales...
- Para el agregar, si implementamos la secuencia con una lista enlazada, el agregar al final cuesta $O(1)$.

Ejercicio: Requisitos de complejidad

- Ok, entonces *definido*, *obtener* y *agregar* tienen que ser sublineales...
- Para el agregar, si implementamos la secuencia con una lista enlazada, el agregar al final cuesta $O(1)$.
- ¿Y el diccionario?
 - ▶ Vectores de tuplas

Ejercicio: Requisitos de complejidad

- Ok, entonces *definido*, *obtener* y *agregar* tienen que ser sublineales...
- Para el agregar, si implementamos la secuencia con una lista enlazada, el agregar al final cuesta $O(1)$.
- ¿Y el diccionario?
 - ▶ Vectores de tuplas
 - ▶ Tabla de Hash (lo van a ver en la teórica y en el labo):
Permite buscar «casi» en $O(1)$

Ejercicio: Requisitos de complejidad

- Ok, entonces *definido*, *obtener* y *agregar* tienen que ser sublineales...
- Para el agregar, si implementamos la secuencia con una lista enlazada, el agregar al final cuesta $O(1)$.
- ¿Y el diccionario?
 - ▶ Vectores de tuplas
 - ▶ Tabla de Hash (lo van a ver en la teórica y en el labo):
Permite buscar «casi» en $O(1)$
 - ▶ Árbol ABB:
Si agregan con una distribución uniforme permite buscar en $O(\log n)$

Ejercicio: Requisitos de complejidad

- Ok, entonces *definido*, *obtener* y *agregar* tienen que ser sublineales...
- Para el agregar, si implementamos la secuencia con una lista enlazada, el agregar al final cuesta $O(1)$.
- ¿Y el diccionario?
 - ▶ Vectores de tuplas
 - ▶ Tabla de Hash (lo van a ver en la teórica y en el labo):
Permite buscar «casi» en $O(1)$
 - ▶ Árbol ABB:
Si agregan con una distribución uniforme permite buscar en $O(\log n)$
 - ▶ Árbol AVL:
Permite buscar en $O(\log n)$

Ejercicio: Requisitos de complejidad

- Ok, entonces *definido*, *obtener* y *agregar* tienen que ser sublineales...
- Para el agregar, si implementamos la secuencia con una lista enlazada, el agregar al final cuesta $O(1)$.
- ¿Y el diccionario?
 - ▶ Vectores de tuplas
 - ▶ Tabla de Hash (lo van a ver en la teórica y en el labo):
Permite buscar «casi» en $O(1)$
 - ▶ Árbol ABB:
Si agregan con una distribución uniforme permite buscar en $O(\log n)$
 - ▶ Árbol AVL:
Permite buscar en $O(\log n)$
- Si elegimos implementar la secuencia con una lista enlazada y el diccionario con un árbol AVL, entonces... touché!

Subite: Algoritmo de UsarTarjeta (con complejidades)

```
function iUSARTARJETA(inout e: estr, in t: tarjeta, in h: fechaHora, out crédito: nat)
  if  $\neg$ definido(e, t) then                                     ▷  $O(\log n)$ 
    res  $\leftarrow$  false
  else
    datos  $\leftarrow$  obtener(e, t)                               ▷  $O(\log n)$ 
    if datos.credito = 0 then
      res  $\leftarrow$  false
    else
      datos.credito  $\leftarrow$  datos.credito - 1
      agregar(datos.viajes, h)                                ▷  $O(1)$ 
      credito  $\leftarrow$  datos.credito
      res  $\leftarrow$  true
    end if
  end if
  devolver res
end function                                                 ▷  $T(n) \in O(\log n)$ 
```

Subite: estructura elegida

Finalmente:

- subite se representa con `estr`, donde:
 - ▶ `estr` es `diccAVL(tarjeta, datosTarjeta)`
 - ▶ `tarjeta` es `nat`
 - ▶ `datosTarjeta` es `tupla`
`crédito: nat,`
`viajes: listaEnlazada(fechaHora)`
- Y donde `diccAVL(a,b)` implementa un diccionario definido sobre un árbol binario con invariante de AVL, y provee las complejidades requeridas.

Subite: estructura elegida

Finalmente:

- subite se representa con `estr`, donde:
 - ▶ `estr` es `diccAVL(tarjeta, datosTarjeta)`
 - ▶ `tarjeta` es `nat`
 - ▶ `datosTarjeta` es `tupla`
 - `crédito: nat,`
 - `viajes: listaEnlazada(fechaHora)`
- Y donde `diccAVL(a,b)` implementa un diccionario definido sobre un árbol binario con invariante de AVL, y provee las complejidades requeridas.
- Deberíamos diseñar `diccAVL`, pero eso lo damos por hecho... ;-)

Subite: estructura elegida

Finalmente:

- `subite` se representa con `estr`, donde:
 - ▶ `estr` es `diccAVL(tarjeta, datosTarjeta)`
 - ▶ `tarjeta` es `nat`
 - ▶ `datosTarjeta` es `tupla`
 - `crédito: nat,`
 - `viajes: listaEnlazada(fechaHora)`
- Y donde `diccAVL(a,b)` implementa un diccionario definido sobre un árbol binario con invariante de AVL, y provee las complejidades requeridas.
- Deberíamos diseñar `diccAVL`, pero eso lo damos por hecho... ;-)
- Restan completar el resto de los algoritmos...
- Resta escribir el invariante de representación y la función de abstracción (queda de tarea)

Subite: adicionales para discutir (1)

Supongamos que el TAD Subite especificaba también la siguiente operación:

`tarjetaMasUsada : subite \longrightarrow tarjeta` $\{ \neg \emptyset(\text{tarjetas}(t)) \}$

Observación: si hay más de una posible respuesta, devuelve la que se usó por último.

- ¿Podemos responder esto con nuestra estructura?

Subite: adicionales para discutir (1)

Supongamos que el TAD Subite especificaba también la siguiente operación:

`tarjetaMasUsada : subite \longrightarrow tarjeta` $\{ \neg \emptyset(\text{tarjetas}(t)) \}$

Observación: si hay más de una posible respuesta, devuelve la que se usó por último.

- ¿Podemos responder esto con nuestra estructura?
- ¿Qué complejidad tendría esa operación?

Subite: adicionales para discutir (1)

Supongamos que el TAD Subite especificaba también la siguiente operación:

`tarjetaMasUsada : subite \longrightarrow tarjeta` $\{ \neg \emptyset(\text{tarjetas}(t)) \}$

Observación: si hay más de una posible respuesta, devuelve la que se usó por último.

- ¿Podemos responder esto con nuestra estructura?
- ¿Qué complejidad tendría esa operación?
- ¿Se puede hacer mejor? ¿Se podría hacer en $O(1)$? ¡Ojo! Sin empeorar las otras operaciones...

Subite: adicionales para discutir (1)

Supongamos que el TAD Subite especificaba también la siguiente operación:

`tarjetaMasUsada : subite \longrightarrow tarjeta` $\{ \neg \emptyset(\text{tarjetas}(t)) \}$

Observación: si hay más de una posible respuesta, devuelve la que se usó por último.

- Una opción posible:
 - ▶ `estr` es tupla \langle
 `tarjetas`: `diccAVL(tarjeta, datosTarjeta)`,
 `masUsada`: `tarjeta` \rangle
 - ▶ `tarjeta` es `nat`
 - ▶ `datosTarjeta` es tupla \langle
 `crédito`: `nat`,
 `viajes`: `listaEnlazada(fechaHora)` \rangle

Subite: adicionales para discutir (1)

Supongamos que el TAD Subite especificaba también la siguiente operación:

`tarjetaMasUsada : subite \longrightarrow tarjeta` $\{ \neg \emptyset(\text{tarjetas}(t)) \}$

Observación: si hay más de una posible respuesta, devuelve la que se usó por último.

- Una opción posible:
 - ▶ `estr` es tupla \langle
 `tarjetas`: `diccAVL(tarjeta, datosTarjeta)`,
 `masUsada`: `tarjeta` \rangle
 - ▶ `tarjeta` es `nat`
 - ▶ `datosTarjeta` es tupla \langle
 `crédito`: `nat`,
 `viajes`: `listaEnlazada(fechaHora)` \rangle
- ¿Cómo serían los algoritmos?

Subite: adicionales para discutir (1)

Supongamos que el TAD Subite especificaba también la siguiente operación:

`tarjetaMasUsada : subite \longrightarrow tarjeta` $\{ \neg \emptyset(\text{tarjetas}(t)) \}$

Observación: si hay más de una posible respuesta, devuelve la que se usó por último.

- Una opción posible:
 - ▶ `estr` es tupla \langle
 `tarjetas`: `diccAVL(tarjeta, datosTarjeta)`,
 `masUsada`: `tarjeta` \rangle
 - ▶ `tarjeta` es `nat`
 - ▶ `datosTarjeta` es tupla \langle
 `crédito`: `nat`,
 `viajes`: `listaEnlazada(fechaHora)` \rangle
- ¿Cómo serían los algoritmos?

Subite: adicionales para discutir (2)

Supongamos que el TAD Subite especificaba también la siguiente operación:

`tarjetasMasUsadas : subite \longrightarrow conj(tarjeta)`

- ¿Podemos responder esto con nuestra estructura?

Subite: adicionales para discutir (2)

Supongamos que el TAD Subite especificaba también la siguiente operación:

`tarjetasMasUsadas : subite \longrightarrow conj(tarjeta)`

- ¿Podemos responder esto con nuestra estructura?
- ¿Qué complejidad tendría esa operación?

Subite: adicionales para discutir (2)

Supongamos que el TAD Subite especificaba también la siguiente operación:

`tarjetasMasUsadas : subite \longrightarrow conj(tarjeta)`

- ¿Podemos responder esto con nuestra estructura?
- ¿Qué complejidad tendría esa operación?
- ¿Se puede hacer mejor? ¿Se podría hacer en $O(1)$? ¡Ojo! Sin empeorar las otras operaciones...

Subite: adicionales para discutir (2)

Supongamos que el TAD Subite especificaba también la siguiente operación:

`tarjetasMasUsadas : subite \longrightarrow conj(tarjetas)`

- Una opción posible:
 - ▶ `estr` es tupla \langle
 `tarjetas: diccAVL(tarjeta, datosTarjeta),`
 `masUsadas: conj(tarjeta) \rangle`
 - ▶ `tarjeta` es nat
 - ▶ `datosTarjeta` es tupla \langle
 `crédito: nat,`
 `viajes: listaEnlazada(fechaHora)\rangle`

Subite: adicionales para discutir (2)

Supongamos que el TAD Subite especificaba también la siguiente operación:

`tarjetasMasUsadas : subite \longrightarrow conj(tarjetas)`

- Una opción posible:
 - ▶ `estr` es tupla(
 `tarjetas`: `diccAVL(tarjeta, datosTarjeta)`,
 `masUsadas`: `conj(tarjeta)`)
 - ▶ `tarjeta` es nat
 - ▶ `datosTarjeta` es tupla(
 `crédito`: nat,
 `viajes`: `listaEnlazada(fechaHora)`)
- ¿Qué complejidades tendríamos? ¿Qué tendríamos que pedirle al conjunto?

Subite: adicionales para discutir (2)

Supongamos que el TAD Subite especificaba también la siguiente operación:

`tarjetasMasUsadas : subite \longrightarrow conj(tarjetas)`

- Una opción posible:
 - ▶ `estr es tupla<`
 `tarjetas: diccAVL(tarjeta, datosTarjeta),`
 `masUsadas: conj(tarjeta) >`
 - ▶ `tarjeta es nat`
 - ▶ `datosTarjeta es tupla<`
 `crédito: nat,`
 `viajes: listaEnlazada(fechaHora)>`
- ¿Qué complejidades tendríamos? ¿Qué tendríamos que pedirle al conjunto?
- ¿Es realmente cierto que esto no empeora la complejidad de **peor caso** de `usarTarjeta()`? (¿Qué hay que tener en cuenta al reemplazar un conjunto de varias tarjetas por uno de una sólo?) :-)

Fin