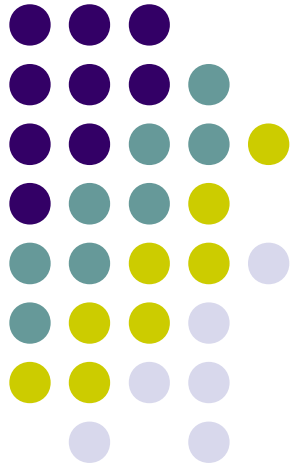
Ejemplo de Diseño

Algoritmos y Estructuras de Datos II

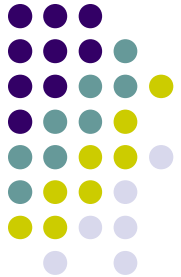
Departamento de Computación

FCEyN – UBA

23 de Septiembre de 2015



¿Qué significa elegir estructuras de datos?



- En la etapa de especificación nos ocupamos de describir '**qué**' queremos lograr.

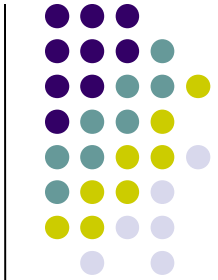
- En la etapa de diseño nos ocupamos del '**cómo**'

  - Lo explicamos con módulos de abstracción
  - Lo hicimos bajo el paradigma imperativo

- Los módulos de abstracción tienen

  - Interfaz pública
  - Estructura de representación
  - algoritmos

Sistema de acceso al subte por tarjetas prepagas multiviajes



- 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 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.
- Se pide tiempo de acceso sublineal en la cantidad total de tarjetas (en caso promedio) para la operación usarTarjeta.

Especificación

TAD Subite

observadores básicos

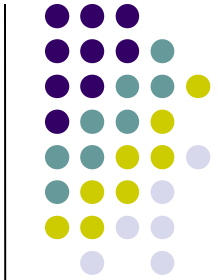
tarjetas	: Subite	→ conj(Tarjeta)
credito	: Subite s × Tarjeta t	→ nat
		{ t ∈ tarjetas(s) }
viajes	: Subite s × Tarjeta t	→ secu(fechaHora)
		{ t ∈ tarjetas(s) }

generadores

Crear	:	→ Subite
IncorporarTarjeta	: Subite s × tarjeta t × nat c	→ Subite
	{ t ∉ tarjetas(s) ∧ c ∈ {1,2,5,10,30} }	
UsarTarjeta	: Subite s × tarjeta t	→ Subite
	{ t ∈ tarjetas(s) ∧ credito(s, t) > 0 }	

Fin TAD

¿Con qué empezamos?



Interfaz SUBITE
Se explica con: Subite
Géneros: Subite
Operaciones:

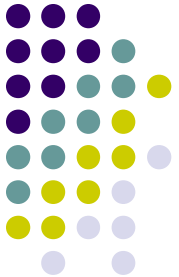
Crear() → res: Subite
Pre:
Post:
Complejidad:
Descripción:
Aliasing:

NuevaTarjeta(*inout* s: Subite, *in* credito: Nat) → res: Tarjeta
...

UsarTarjeta(*inout* s: Subite, *in* i: Tarjeta, *out* credito: Nat) → res: Bool
...

VerViajes(*in* s: Subite, *in* i: Tarjeta, *out* viajes: Secu(FechaHora)) → res: Bool
...

Pre y postcondiciones



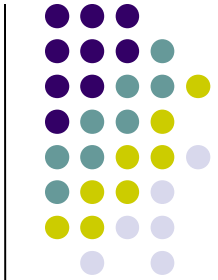
Crear() → res: Subite
{ true }
{ res = Crear }

NuevaTarjeta(*inout* s:Subite, *in* credito: Nat) → res: Tarjeta
{ $s = s_0 \wedge$ credito $\in$ {1,2,5,10,30} }
{ res $\notin$ Tarjetas(s_0) $\wedge$ s = IncorporarTarjeta(s_0,res,credito) }

UsarTarjeta(*inout* s: Subite, *in* i: Tarjeta, *out* credito: Nat) → res: Bool
{ $s = s_0$ }
{ (res $\Leftrightarrow$ (i $\in$ Tarjetas(s_0) $\wedge$ Credito(s_0,i) > 0)) $\wedge$
(s = if res then UsarTarjeta(s_0,i) else s_0 fi) $\wedge$
(res $\Rightarrow$ (credito = Credito(UsarTarjeta(s_0,i))) }

VerViajes(*in* s: Subite, *in* i: Tarjeta, *out* viajes: Secu(FechaHora)) → res: Bool
{ true }
{ (res $\Leftrightarrow$ i $\in$ Tarjetas(s)) $\wedge$ (res $\Rightarrow$ viajes = Viajes(s,i)) }

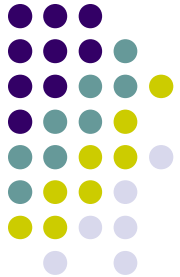
Observaciones sobre servicios exportados



Observación que vale de ahora en más para toda la clase:
n es la cantidad total de tarjetas

Función	Orden
Crear	?
NuevaTarjeta	?
UsarTarjeta	< O(n)
VerViajes	?

Estructura de representación



- Primero descomponemos el problema, usando estructuras de “alto nivel” y buscando contener toda la info necesaria

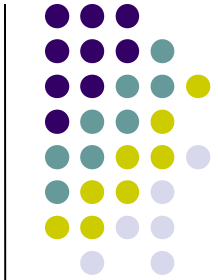
Subite se representa con Dicc**(Tarjeta, DatosTarjeta)**

Tarjeta es **Nat**

DatosTarjeta es **tupla**

< credito: **Nat** ×
viajes: **Secuencia(FechaHora)** >

Estructura de representación



- Ahora tenemos que pensar estructuras para representar cada componente, teniendo en cuenta la complejidad.

- ¿Qué hace usarTarjeta?
  - Busca la tarjeta
  - Le descuenta el saldo
  - Le agrega un viaje

- ¿Cuánto cuesta cada cosa? ¿Cómo hacemos esto en tiempo sublineal?

Algoritmos

- Escribimos el algoritmo para estar seguros…

iUsarTarjeta(*inout* e: estr, *in* i: Tarjeta, *out* credito: Nat) → resultado: Bool

```
Si ¬definido?(e,i)                          O(definido? de dicc).
   devolver false
Si no
   datos ← obtener(e,i)      O(obtener de dicc). Necesitamos que sea con aliasing
   si datos.credito = 0
      devolver false
   si no
      agregar_atrás(datos.viajes,fechahora_actual)     O(agregar de secu)
      datos.credito ← datos.credito – 1              O(1)
      credito ← datos.credito                        O(1)
      devolver true
```

Algoritmos

- Escribimos el algoritmo para estar seguros…

iUsarTarjeta(*inout* e: estr, *in* i: Tarjeta, *out* credito: Nat) → resultado: Bool

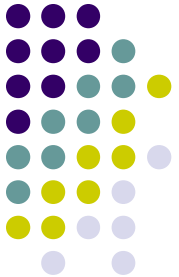
```
Si ¬definido?(e,i)                          O(definido? de dicc).
   devolver false
Si no
   datos ← obtener(e,i)      O(obtener de dicc). Necesitamos que sea con aliasing
   si datos.credito = 0
      devolver false
   si no
      agregar_atrás(datos.viajes,fechahora_actual)     O(agregar de secu)
      datos.credito ← datos.credito – 1              O(1)
      credito ← datos.credito                        O(1)
      devolver true
```

Servicios usados

- Necesitamos un diccionario sublineal en definido? y obtener.
- ¿Qué pasa con la secuencia?

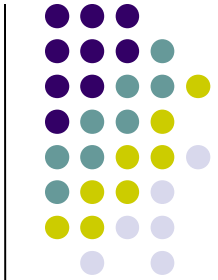
Función	Orden
Obtener	<O(n)
Definido?	<O(n)
Definir	?

Estructura de representación



- ¿Qué estructuras conocen con acceso sublineal?

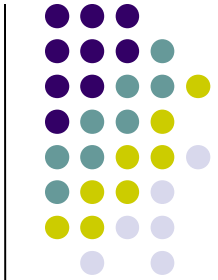
Estructura de representación



- ¿Qué estructuras conocen con acceso sublineal?
  - AVL / ABB
  - Tabla de Hash

¿Cuál usamos?

Estructuras sublineales (en caso promedio)



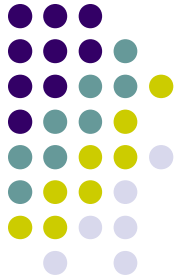
Función	AVL	ABB	Hash
Buscar	O(log n)	O(log n)*	O(1) **
Borrar	O(log n)	O(log n)*	O(1) **
Agregar	O(log n)	O(log n)*	O(1) ***

* Sólo si… Se agregan con distribución uniforme

** Sólo si… La función de hash es buena (genera pocas colisiones - uniforme)

*** Sólo si… El tamaño de la tabla es proporcional a la cantidad de elementos

Estructuras auxiliares



Dicc(Tarjeta,DatosTarjeta) se representa con **AB(Nodo)**

Nodo es <Tarjeta,DatosTarjeta>

Observación: Nodo se iguala y compara por el primer campo.

Secu(FechaHora) se representa con **ListaEnlazada(FechaHora)**

Volvemos a los algoritmos

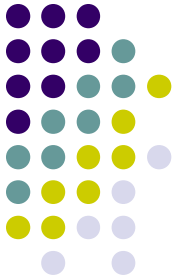
- Escribiendo la complejidad correcta…

iUsarTarjeta(*inout* e: estr, *in* i: Tarjeta, *out* credito: Nat) → resultado: Bool

```
Si ¬definido?(e,i)                                          O(log n)
   devolver false
Si no
   datos ← obtener(e,i)                                     O(log n)
   si datos.credito = 0
      devolver false
   si no
      agregar_atrás(datos.viajes,fechahora_actual)          O(1) (max: 20)
      datos.credito ← datos.credito – 1                     O(1)
      credito ← datos.credito                               O(1)
      devolver true
```

Servicios exportados

Función	Orden
Crear	?
NuevaTarjeta	?
UsarTarjeta	O(log n)
VerViajes	?



Preguntas ¿?