Tema 1

Estructuras de Datos, Abstracción y Especificación.

Objetivos

- **Concepto de Estructura de Datos.**
- **Concepto de Abstracción y tipos de abstracción usados en programación.**
- **Concepto de Especificación e Implementación:**
  - Diferenciar especificación e implementación.
  - Formas de especificar: formal e informal.
- **Concepto de TAD:**
  - Tipos genéricos.
- **Complejidad algorítmica. Notación “orden” (O()).**

Estructura de Datos

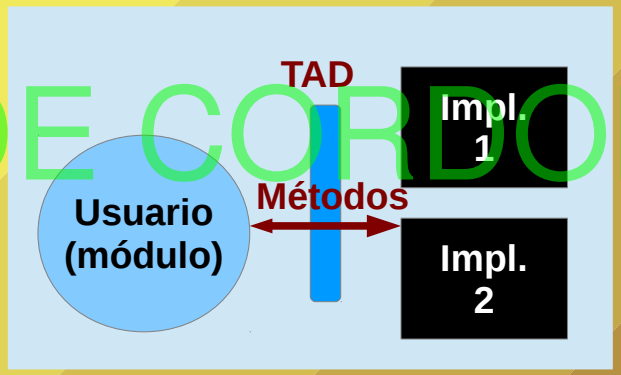
- **Concepto:**
  - Forma concreta de **organizar** datos en la computadora para que puedan ser **almacenados** y **accedidos** de manera **eficiente** (en tiempo y espacio).
  - Varias clasificaciones:
    - Por el tipo de memoria usada: primaria, secundaria.
    - Por el tipo de relación entre los datos:
      - Lineal: 1-1.
      - No lineal:
        - 1-n o jerárquica,
        - N-m o grafos.
  - Utilizaremos el concepto TAD para definirlas.

Tipo Abstracto de Datos

- **Concepto:**
  - Definir un conjunto de valores por las operaciones que puedo hacer con ellos.
  - Abstracción: nos permite centrarme en qué puedo hacer con los valores y no en cómo lo hago.
  - Necesito lenguajes bien estructurados con posibilidad de extender tipos.
  - La POO es la extensión natural de esta idea.

Tipo Abstracto de Datos

- **Especificación vs. Implementación.**
  - Especificación (Qué)
  - Implementación (Cómo)

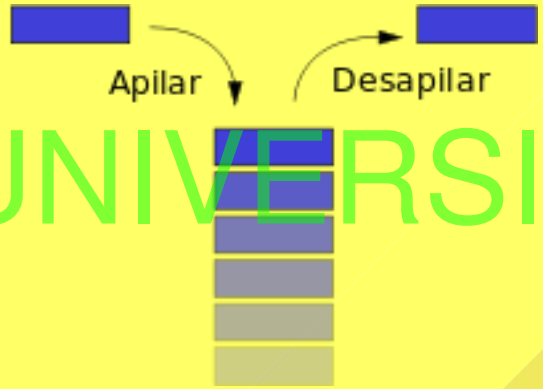


Anatomía de un TAD

- **En un TAD sólo se especifican operaciones:**
  -
  -
  -
- **Cómo se especifican las operaciones:**
  - **Métodos formales**: Lenguaje matemático (lógica de predicados)
  - **Métodos informales**:
    -
    -
    -
    -

Ejemplos

- **Concepto de Pila**



Operaciones con una Pila:

.
.
.
.
.

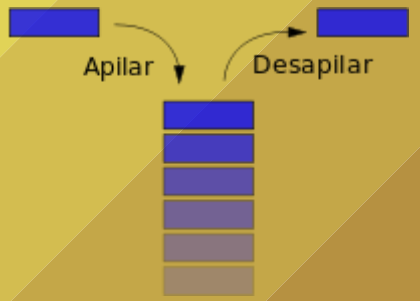
Ejemplos

- **Especificación formal de una pila**

```
TIPO
Pila[G]
FUNCIONES
  apila: Pila[G] x G → Pila[G]
  desapila: Pila[G] ↛ Pila[G]
  cima: Pila[G] ↛ G
  vacía: Pila[G] → Boolean
  crea: → Pila[G]
AXIOMAS
  Para cualquier x:G, p:Pila[G],
  desapila(apila(p, x)) ≡ p
  cima(apila(p, x)) ≡ x
  vacía(crea) ≡ Verdadero
  vacía(apila(p, x)) ≡ Falso
PRECONDICIONES
  desapila(p:Pila[G]) require vacía(p) = Falso
  cima(p:Pila[G]) require vacía(p) = Falso
```

Dada la expresión siguiente calcular su resultado:

cima(desapila(apila(desapila(apila(apila(desapila(apila(apila(apila(crea, 1), 2), 3)), cima(desapila(apila(apila(crea, 4), 5)))), 6)), 7)))

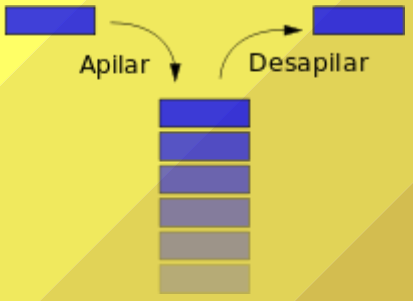


Ejemplos

- **Especificación informal.**

TAD Pila[G]:

- **Constructores**:
  -
    - Post:
- **Observadores**:
  -
    - Prec:
- **Modificadores**:
  -
    - Post:
    - Post:
  -
    - Prec:
    - **Post:**



Ejemplos

- **ADT Point2D:**
  - Points into the Euclidan plane.
  - **Makers**:
    -
  - **Observers**:
    -
    -
    - Bool isEqual (Point2D p) // Are the points the same?.
      - post-c:
  - **Modifiers**:
    - `addPoint(Punto2 d)` //Adds an offset point.
      - Post:
      - Post:
    - `times(Real s)` //Multiplies coordinates by a scalar value.
      - Post:
      - Post:
- **Discusión:**
  - ¿Podría ser interesante introducir un método para calcular la distancia entre dos puntos?
  - ¿Cómo podrías implementar el TAD en C++? ¿Qué representación elegirías? ¿cómo implementar las pre/post condiciones?

Ejemplos

- **ADT: Line2D uses Point2D**
  - Models a line in an Euclidean plane.
  - **Makers**:
    - `Line2D(a,b,c:Real)` //makes a line using the implicit ecuation ax+by+c=0
    - `Line2D(p,q:Point2D)` //makes a line passing through two points.
      - pre-c: not p.isEqual(q)
  - **Observers**:
    - `Real x(Real y)` //returns the x coordinate of a point belonging to the line given its y coordinate.
    - `Real y(Real x)` //returns the y coordinate of a point belonging to the line given its x coordinate.
    - `Real acuteAngle(Line2D s)` // return the acute angle between the two lines.
      - post-c: return value in [0, pi/2]
    - `Point2D crossPoint(Line2D s)` // return the cross point between two lines.
      - pre-c acuteAngle(s)>0
      - post-c: x(return.y())=s.x(return.y()) //the cross point belonging to both lines.
      - post-c: y(return.x())=s.y(return.x()) //the cross point belonging to both lines.
    - `Bool areCoindicent(Line2D s)` //Are the two lines coincident?
      - post-c: areCoindent(s) implies current.acuteAngle(s)=0.0
    - `Real apply(Point2D p)` /Apply the implicit equation to the point.
    - `Real distance(Point2D p)` // Computes the minimum distance of a point to the line.
- **Discusión:**
  - ¿Por qué no hay modificadores?¿Cómo implementarías este TAD en C++?¿Qué representación elegirías?

Notación O

- **Complejidad algorítmica.**
  - ¿Qué es?
  - ¿Por qué es importante?

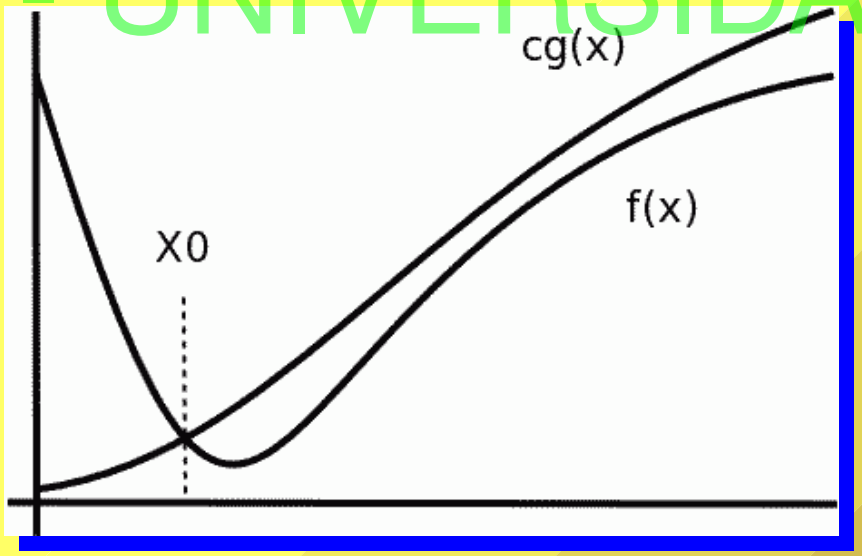
EDI - UNIVERSIDAD DE CORDOBA

Notación O

- **¿En qué consiste la notación O?**
  - Buscar una cota superior asintótica.

$$O(g(x)) = \{ f(x) : \exists c, x_0 > 0 \mid \forall x \geq x_0 : 0 \leq |f(x)| \leq c|g(x)| \}$$

Recurso:
Tiempo/
Espacio



notación	nombre
O(1)	orden constante
O(log log *n*)	orden sublogarítmico
O(log *n*)	orden logarítmico
$O(\sqrt{n})$	orden sublineal
O(*n*)	orden lineal
O(*n* · log *n*)	orden lineal logarítmico
$O(n^c)$	orden potencial
$O(c^n)$, n > 1	orden exponencial
O(*n*!)	orden factorial

TAM. Problema

Notación O

- **Ejemplo: ordenar un vector de N valores enteros en el intervalo [0, 255].**
  - Método de selección:
    - Tiempo: O( )
    - Espacio: O ( )
  - Método por cálculo de frecuencias:
    - Tiempo: O ( )
    - Espacio: O ( )

Notación O

- **Ejemplos típicos (métodos de ordenación).**

Método	Complejidad	Observaciones
Burbuja	$O(N^2)$	N = n. de elementos
Selección	$O(N^2)$	“
Inserción	$O(N^2)$	“
Quicksort	O(NlogN)	“
Frecuencias	O(M)	M= max(N, rango de valores)

Notación O

- **Ejemplos CA en orden creciente.**

Complejidad	Algoritmo	Observaciones
O(1)	Acceso a un elemento en un array	Constante
O(logN)	Búsqueda binaria	logarítmica
O(N)	Búsqueda secuencial	lineal
O(NlogN)	quicksort	Lineal logarítmica
$O(N^2)$	Recorrido matriz N*N	Cuadrática
$O(N^3)$	Producto de matrices N*N	Cúbica
$O(N^4)$	Potencia N-ésima de matriz N*N	
$O(2^N)$	Torres de Hanoi	Exponencial
O(N!)	Viajante de comercio	Factorial

Resumen

- **ED: permite almacenar y acceder de forma eficiente datos en la computadora.**
- **TAD: representación abstracta de un conjunto de valores por las operaciones que podemos hacer sobre los mismos.**

- **Los TAD se especifican a partir de sus operaciones y no de su representación.**
- **La notación O: indica la complejidad algorítmica de una operación. Permite comparar distintas implementación de una misma operación abstracta.**

Referencias

- **Barbara Liskov, Stephen Zilles “PROGRAMMING WITH ABSTRACT DATA TYPES”, 1974.**
- **Caps 1 a 5 de “*Estructuras de Datos*”, A. Carmona y otros. U. Córdoba. 1999.**
- **Caps 1, 4, y 11 de *“Data structures and software develpment in an object oriented domain”*, Tremblay J.P. y Cheston, G.A. Prentice-Hall, 2001.**
- **Cap 6 de “Construcción de Software Orientado a Objetos”, Meyer, B. Prentice-Hall, 1999.**
- **Wikipedia:**
  - Abstraction: https: en.wikipedia.org/wiki/Abstraction_(software_engineering)
  - Abstract Data Type: en.wikipedia.org/wiki/Abstract_data_type
  - Notación O: es.wikipedia.org/wiki/Cota_superior_asint%C3%B3tica