Tema 1

EEDD - GRADO EN ING. INFORMATICA - UCO

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).
  - Características:
    - Tipo de **memoria** usada: contigua/no contigua, primaria/secundaria.
    - Por el tipo de **relación** entre los datos: lineal (1-1) / no lineal (1-n o n-m).
    - ¿Impone un **orden** en los datos?
    - ¿Permite datos **duplicados**?
    - **Operaciones** sobre los datos.
  - Algunos ejemplos:
    - Array, ... listas, pilas, colas

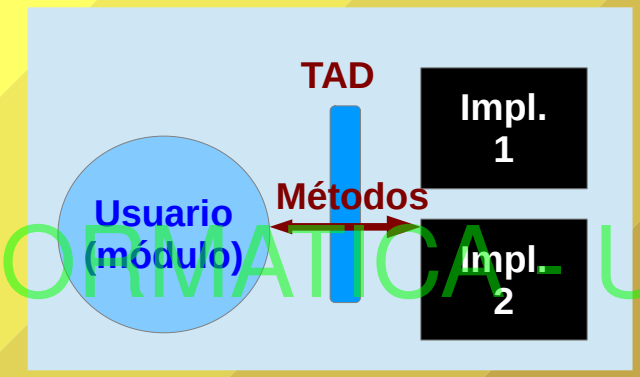
Tipo Abstracto de Datos

- **Concepto.**
  - Es un Tipo de Dato. conjunto de valores sobre los que puedo operar
  - Es Abstracto.
  - Necesitamos soporte del lenguaje.

Tipo Abstracto de Datos

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

```
crea: RealxReal → Vect2D
x: Vect2D → Real
y: Vect2D → Real
sumar: Vect2D x Vect2D → Vect2D
x(crea(v1,v2)) = v1
y(crea(v1,v2)) = v2
x(sumar(v1,v2)) = x(v1)+x(v2)
y(sumar(v1,v2)) = y(v1)+y(v2)
```



  - Implementación (Cómo)

```
//Programación imperativa.
class Vect2D
{
  public:
    Vect2D(float x, float y):
      x_(x), y_(y) {}
    float x() const { return x_;}
    float y() const { return y_;}
    void sumar(Vect2D const& v)
    { x_ += v.x_; y_ += v.y_; }
  private:
    float x_, y_;
};
```

```
//Programación funcional
class Vect2D
{
  public:
    Vect2D(float x, float y):
      v_(2) {v_[0]=x, v_[1]=y;}
    float x() const { return v_[0];}
    float y() const { return v_[1];}
  private:
    std::valarray<float> v_;
};
Vect2D
sumar(Vect2D const&a, Vect2D const&b)
{
   return Vect2D(a.x()+b.x(), a.y()+b.y());
}
```

Anatomía de un TAD

- **En un TAD sólo se especifican las operaciones:**
  - Constructores. generar valores
  - Observadores. ver
  - Modificadores. cambiar
- **Cómo se especifican las operaciones:**
  - **Métodos formales**: Lenguaje matemático (lógica de predicados)
  - **Métodos informales (Diseño por Contrato)**:
    - Interfaz (prototipo) y Documentación.
    - Pre-condiciones.
    - Post-condiciones.
    - Invariantes.

Ejemplos

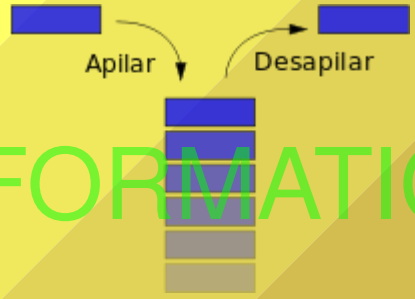
- **Especificación formal de una pila**

genérico

```
TIPO
Pila[G]
FUNCIONES
  crea: → Pila[G]
  vacía: Pila[G] → Bolean
  cima: Pila[G] ↛ G
  apila: Pila[G] x G → Pila[G]
  desapila: Pila[G] ↛ 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
  cima(p:Pila[G]) require vacía(p) = Falso
  desapila(p:Pila[G]) require vacía(p) = Falso
```



Ejercicio: Especifica una operación "tamaño" que nos devuelve el número de elementos que tiene una Pila.

tamaño(crea) = 0

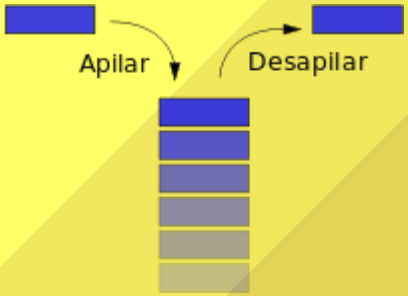
tamaño(apila(p,x)) = tamaño(p) + 1

Ejemplos

- **Especificación informal.**

TAD Pila[G]:

- **Constructores**:
  - crea():Pila[G] #Crea una nueva pila vacía.
    - Post: vacía()
- **Observadores**:
  - vacía():Bool #Está la pila vacía?
  - cima():G #El último elemento apilado.
    - Prec: no vacía()
- **Modificadores**:
  - apila(it:G) #Apila un nuevo elemento.
    - Post: no vacía()
    - Post: cima()==it
  - desapila() #Desapila el último elemento apilado.
    - Prec: no vacía()
    - **Post: vacia() o "cima() es el último elemento apilado".**



Ejercicio: Especifica una operación "tamaño" que nos devuelve el número de elementos que tiene una Pila.

post: tamaño ≥ 0

Notación O

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

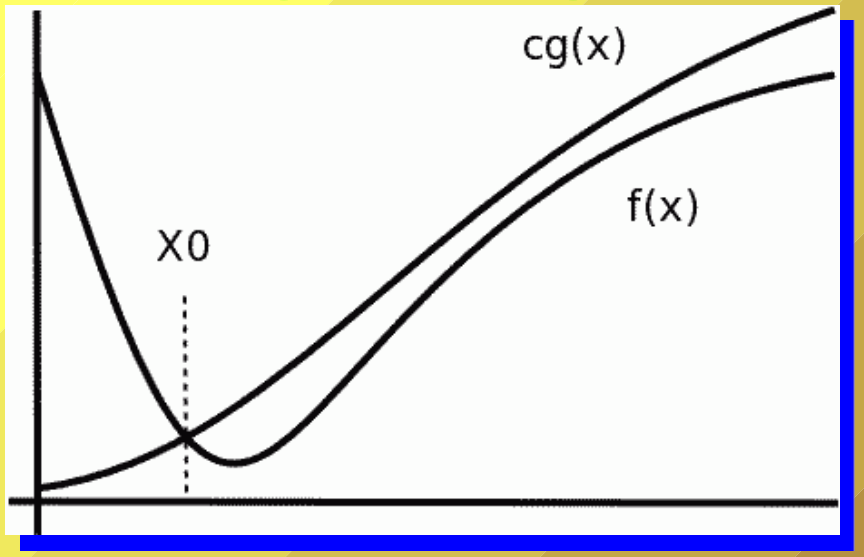
EEDD - GRADO EN ING. INFORMATICA - UCO

Notación O

- **¿En qué consiste la notación O?**
  - Buscar una cota superior asintótica. (eficiencia de algoritmos)

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

Recurso: Tiempo/ Espacio

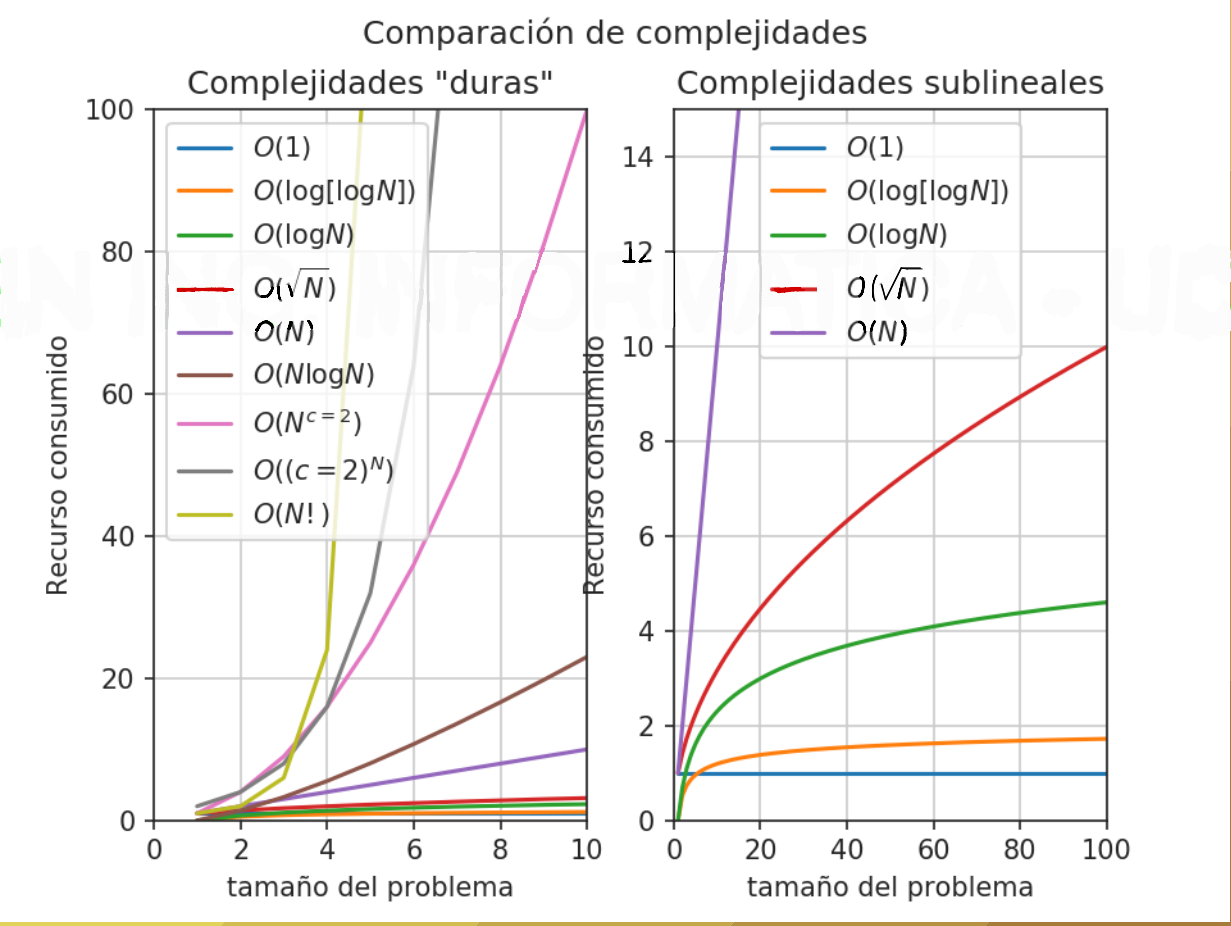


TAM. Problema

Notación O

- **Comparación de complejidades.**

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



Notación O

- **Algunas propiedades.**
  - Suma: $f_1 = O(g_1)\ y\ f_2 = O(g_2) \Rightarrow f_1 + f_2 = O(max\{g_1, g_2\})$
  - Multiplicación: $f_1 = O(g_1)\ y\ f_2 = O(g_2) \Rightarrow f_1 \times f_2 = O(g_1 g_2)$
  - Multiplicar por constante: $f = O(g)\ y\ k \neq 0 \Rightarrow k \times f = O(g)$

Notación O

- **Ejemplo: ordenar un vector de N valores enteros en el intervalo [0, 255].**
  - Método de selección:
    - Tiempo: O(n^2)
    - Espacio: O (1)
  - Método por cálculo de frecuencias:
    - Tiempo: O (n)
    - Espacio: O (n)


```
ordenarSelección(var v:Vector<Integer>)
Aux
  i,j,min : Integer
Comienzo
  Para i de 0 hasta v.tam()-1 hacer
    Para j de i+1 hasta v.tam() hacer
      Si v[i]>v[j] entonces
        intercambia(v[i],v[j])
Fin.
```

```
ordenarFrecuencias(var v:Vector<Integer>)
Aux
  h: Vector<Integer>
  i: Integer
Comienzo
  h ← crea_vector_con_zeros(256)
  Para i de 0 hasta v.tam() hacer
    h[v[i]] += 1
  histograma_a_vector(h, v)
Fin.
```

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
$O(N^2)$	Sumar matriz A+B	Cuadrática
$O(N^3)$	Producto de matrices A*B	Potencial
$O(2^N)$	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 y las operaciones que podemos hacer sobre los mismos.**
- **Los TAD se especifican a partir de sus operaciones y no de su representación.**
- **Una ED es una implementación de un TAD.**
- **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