Índice

- Introducción a los TADs
- Medida de la eficiencia de las implementaciones
- Estructuras lineales: pilas, colas, listas
- Tablas asociativas: hash
- Árboles
- Grafos

Introducción a los TADs

- Los Tipos Abstractos de Datos nos permiten implementar/usar módulos de software (Tipos Concretos de Datos) en base a su especificación.
- Tenemos una abstración de los datos reales.
- Sabemos **qué** sin saber **cómo**.
- Con los primeros lenguajes estructurados (algol, pascal, ...) aparecieron los tipos de datos: valores que podian usarse como dominio de ciertas operaciones
- TAD = VALORES + OPERACIONES
- VALORES: tipos de datos, estructuras de datos
- OPERACIONES: Comportamiento, propiedades, son las únicas que conocen la implementación del tipo
- Aquí, abstracto significa que podemos usar el TAD olvidando la representación del tipo y la implementación de las operaciones

Introducción a los TADs

- Dada una especificación puede haber muchas implementaciones válidas.
- Escogeremos aquella que sea más eficiente
- Cualquier cambio en la implementación es transparente a los programas que usan el TAD

TAD	Diseño	Eficacia	QUÉ
TCD	Implementación	Eficiencia	CÓMO

Introducción a los TADs

- Esta metodología tiene muchos puntos interesantes:
  - Organización del trabajo: los programadores sólo deben conocer la especificación del TAD
  - Reusabilidad: uso del mismo TAD en distintos contextos
  - Corrección: pruebas unitarias y de integración
  - Legibilidad: se manipulan operaciones
  - Eficiencia: podemos buscar la estructura de datos y algortimos más eficiente para implementar el TAD
  - Seguridad: el usuario del TAD no manipula directamente el tipo de datos

Introducción a los TADs: ejemplo

- Construir un programa que lea una série de números complejos (acabados en 0+0i) y que nos devuelva su suma.
- VALORES: a + bi, donde a y b son reales
- OPERACIONES:

  crear: real x real --> complejo

  preal: complejo --> real

  pimaginaria: complejo --> real

  suma: complejo x complejo --> complejo
- PROPIEDADES: las habituales de los complejos

  (a+bi) + (c+di) = (a+c) + (b+d)i

  ...

Introducción a los TADs: ejemplo

```
algoritmo suma_complejos es
    var r : complejo; a, b : real fvar
    r := crear(0.0, 0.0);
    leerReal(a); leerReal(b);
    mientras (a <> 0.0 o b <> 0.0) hacer
        r := suma(r, crear(a,b));
        leerReal(a); leerReal(b)
    fmientras
    escribirTexto("El resultado es: "); escribirReal(preal(r));
    escribirTexto(" + ");
    escribirReal(pimaginaria(r)); escribirTexto("i.")
falgoritmo
```

Introducción a los TADs: ejemplo

- Hemos hecho el programa sin tener ni idea de como se implementan los complejos, ni de cómo están implementadas las operaciones
- Ahora que sabemos qué operaciones necesitamos, las podemos implementar

Introducción a los TADs: ejemplo

- Hemos hecho el programa sin tener ni idea de como se implementan los complejos, ni de cómo están implementadas las operaciones
- Ahora que sabemos qué operaciones necesitamos, las podemos implementar

Introducción a los TADs: ejemplo

```
tipo complejo es tupla r, i : real ftupla ftipo
función crear (a, b : real) devuelve complejo
    var c : complejo fvar
    c.r := a; c.i := b
    devuelve c
ffunción
función preal (c : complejo) devuelve real
    devuelve c.r
ffunción
función pimaginaria (c : complejo) devuelve real
    devuelve c.i
ffunción
función suma (a, b : complejo) devuelve complejo
    var c : complejo fvar
    c.r := a.r + b.r; c.i := a.i + b.i
    devuelve c
ffunción
```

Implementación de TADs

- Para implementar un TAD tenemos que:
  - Escoger una representación "adecuada" de la estructura de datos
  - Codificar las operaciones visibles/públicas del TAD en función de la representación escogida de forma que las operaciones cumplan las propiedades definidas de la forma más eficiente posible
- Obviamente, la representación más adecuada será aquella que nos permita codificar las operaciones del TAD con la máxima eficiencia *espacial* y *temporal*
- Primar siempre la eficiencia temporal a la espacial
- Primar las operaciones críticas

Medida de la eficiencia de las implementaciones

- El tiempo de ejecución de un algoritmo depende de:
  - Ordenador, CPU
  - SO
  - Lenguaje, compilador

  } Factores multiplicativos
  - Datos del problema (tamaño del problema)
  - Condiciones de los datos
- Aproximación empírica:
  - cronometrar el tiempo de ejecución
    - No conseguimos independencia de máquina, lenguaje, etc.
    - No podemos comparar algoritmos “a priori”
    - Resultados difícilmente extrapolabres

Medida de la eficiencia de las implementaciones

- Aproximación teórica:
  - Buscamos expresar el tiempo de ejecución de un algortimo en función del tamaño de datos
  - Nos interesa el orden de crecimiento de la función de coste despreciando factores constantes aditivos y multiplicativos y por tanto independientes de los factores anteriores
- Tenemos que tener en cuenta además las condiciones de los datos, no sólo el tamaño de los datos
  - Análisis del caso peor: límite superior (pesimista)
  - Análisis del caso medio: difícil ... (realista)
  - Análisis del caso mejor: límite inferior (optimista)

Medida de la eficiencia de las implementaciones

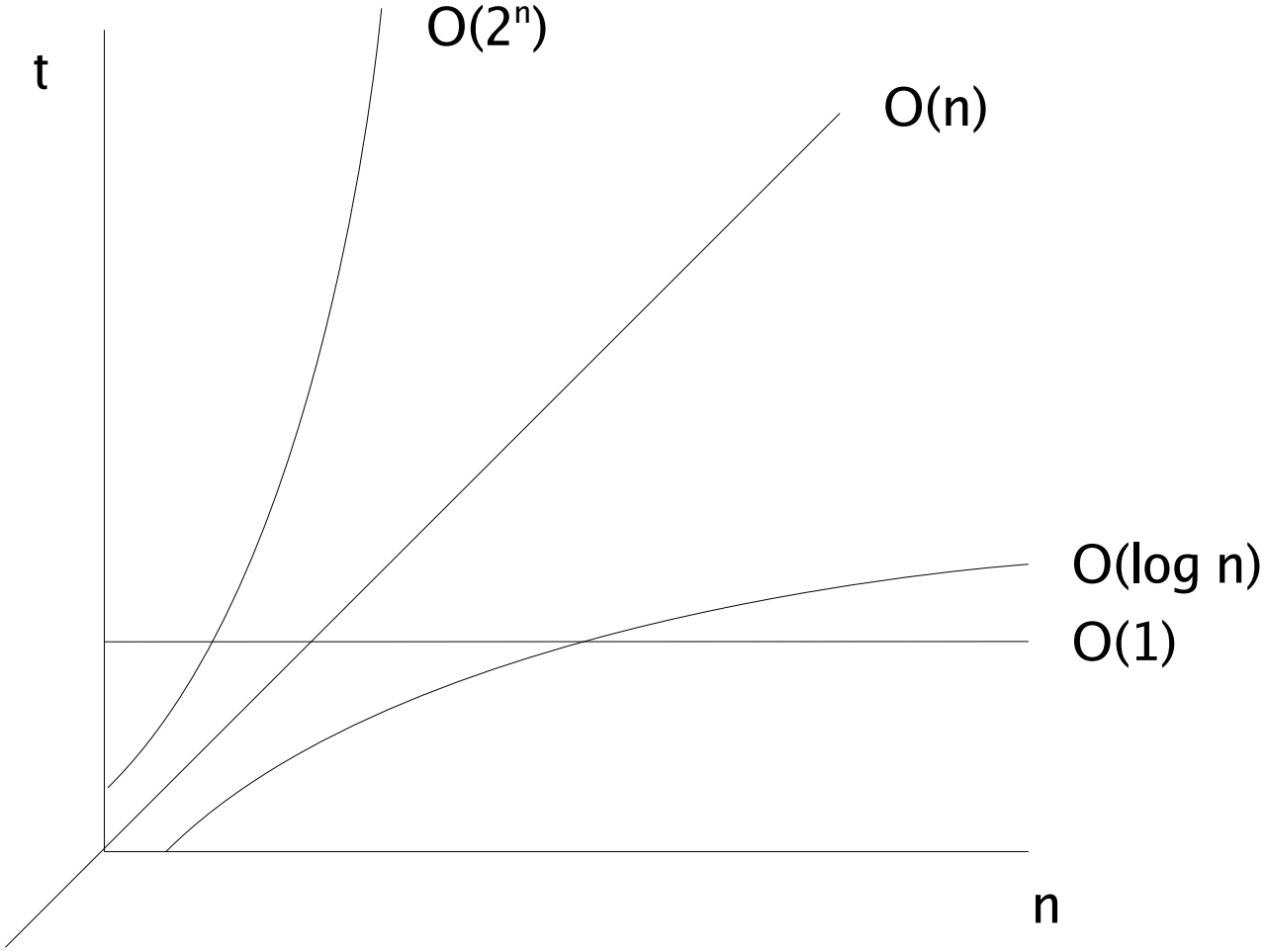
- Búsqueda de un elemento en un vector desordenado de tamaño n
  - Caso peor: n
  - Caso medio: n/2
  - Caso mejor: 1

- Normalmente estudiaremos el análisis de coste del caso peor
- Las notaciones más usadas para clasificar las funciones de coste de un algoritmo usan la velocidad de crecimiento respecto al volúmen de los datos y definen el coste asimptótico de los algoritmos mirando su comportamiento en el límite
- En notación asimptótica se prescinde de constantes y funciones de orden inferior

Medida de la eficiencia de las implementaciones

▪ $O(1)$	constante	asignación
▪ $O(\log n)$	logarítmico	búsqueda dicotómica
▪ $O(n)$	lineal	búsqueda en vector desordenado
▪ $O(n \log n)$	quasilineal	ordenar un vector
▪ $O(n^k)$	polinómico	árbol de expansión mínimo de un grafo
▪ $O(k^n)$	exponencial	satisfactibilidad de una fórmula

Medida de la eficiencia de las implementaciones



Medida de la eficiencia de las implementaciones

- La tasa de crecimiento de un algoritmo nos da el tamaño máximo de datos que podremos procesar fijando el tiempo

f(n)		10^3 u.t.	10^4 u.t.	
$100n$	$\in O(n)$	10	100	x10
$5n^2$	$\in O(n^2)$	14	44	x3.14
$n^3/2$	$\in O(n^3)$	12	27	x2.25
2^n	$\in O(2^n)$	9	13	x1.44

Cálculo de la complejidad de un algoritmo

- Regla de la suma:

  A1 y A2 son dos acciones

  coste A1 es $T1(n) \in O(f1(n))$

  coste A2 es $T2(n) \in O(f2(n))$

  coste A1; A2 es $T1(n) + T2(n) \in O(max(f1(n),f2(n)))$

- Ejemplos:

  $n+1 \in O(n)$

  $n+n^2 \in O(n^2)$

Cálculo de la complejidad de un algoritmo

- Regla del producto:

  $T1(n) \in O(f1(n))$

  $T2(n) \in O(f2(n))$

  $T1(n) * T2(n) \in O(f1(n) * f2(n))$

- Ejemplos:

  $n * 1 \in O(n)$

  $n * n^2 \in O(n^3)$

Cálculo de la complejidad de un algoritmo

- Asignación, lectura, escritura, comparación tipos elementales O(1)
- Coste de una secuencia: regla de la suma (máximo de costes)
- Coste estructura control alternativa:
  - Coste evaluar la condición + coste de la alternativa peor
- Coste estructura control repetitiva:
  - Suma extendida a todas las iteraciones del coste del cuerpo de la iteración + coste de evaluar la condición del bucle
  - Max número de iteraciones * max coste de cada iteración

Cálculo de la complejidad de un algoritmo: ejemplo

```
acción ordena (ent/sal v : tabla[1..N] de entero) es
    var i, j, k, e : entero fvar
    para i := 1 hasta N-1 hacer
        e := v[i]; k := i;
        para j := i+1 hasta N hacer
            si v[j]<e entonces e:=v[j]; k:=j fsi
        fpara
        v[k]:=v[i];
        v[i]:=e
    fpara
facción
```

O(n-i)

$n^2-n/2 \in O(n^2)$