

Ayudantía 4: TDA y Complejidad

Estructura de Datos 2020-1

Anastasiia Fedorova, Gonzalo Fernández
Tomás Guttman, Héctor Larrañaga, Martín Salinas

Departamento de Informática
Universidad Técnica Federico Santa María

anastasiia.fedorova@sansano.usm.cl
gonzalo.fernandezc@sansano.usm.cl
tomas.guttman@sansano.usm.cl
hector.larranaga@sansano.usm.cl
martin.salinass@sansano.usm.cl

4 de mayo de 2020

Índice

1 TDA

- Sobre la abstracción
- Características de los TDA
- Privacidad

2 Complejidad

- Ejercicio
- Motivación
- Análisis asintótico
- Ejercicio

Índice

1 TDA

- Sobre la abstracción
- Características de los TDA
- Privacidad

2 Complejidad

- Ejercicio
- Motivación
- Análisis asintótico
- Ejercicio

Sobre abstracción

La abstracción es uno de los conceptos esenciales en la informática. No siempre necesitamos saber como funciona cada parte más chica del computador (circuitos análogos, digitales, física por detrás) para usarlo correctamente.

La misma lógica se aplica a los TDA - no necesitamos saber todas las interacciones por detrás en el TDA, queremos solamente usarlo a través de sus funciones definidas.

Sobre abstracción

Un ejemplo claro de los TDA que ya conocemos es la lista de Python. Esta tiene las funciones propias definidas (como las `append()`, `length()`, etc) , y nosotros **nunca** manejamos las listas de Python de manera directa - siempre se usan las **funciones asociadas**.

```
>>> lst = [1, 2]
>>> lst.
```

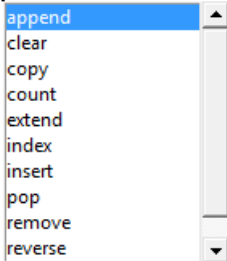


Figura 1: Operaciones de TDA lista de Python

Características de los TDA

De lo dicho, se puede concluir que los TDA se caracterizan por:

- Encapsulamiento - se oculta la implementación del usuario
- Operaciones disponibles para usar el TDA - su **interfaz**
- El comportamiento de cada función es determinado por sus salidas y entradas

Uso correcto de los TDA

Como queremos utilizar **solamente** las funciones para interactuar con nuestro TDA, ¡**no** podemos acceder a sus campos directamente!

Digamos que tenemos un TDA Biblioteca que nos permite manejar una biblioteca. Este cuenta con una cierta estructura, que nos permite almacenar libros, revistas, cantidad de libros, etc.

Si un día llega un nuevo libro y queremos aumentar la cantidad total de libros en la biblioteca, ¿cuál es la manera correcta de hacerlo?

Uso correcto de los TDA

Debemos utilizar una función que nos permite aumentar la cantidad de libros (un cierto campo de la estructura, que llamaremos `lib_total`.)

Es **incorrecto** hacer aumento directamente: `bibli.lib_total++`! Debemos utilizar una función para hacerlo, como, por ejemplo, `aumentar_cant(bibli)`.

Números complejos

A causa de las clases online y el millon de trabajos, el semestre de *Eleusterio* está *complejo*. Realice un **TDA** que le permita representar un número de este tipo.

Su **TDA** debe implementar las siguientes funciones:

- Constructor
- Valor Absoluto: Retorna double
- Conjugado: Retorna complejo
- Suma: Retorna complejo
- Parte Real y Parte Imaginaria: Retornan int
- Print: Imprime el complejo de la forma $a + bi$. Retorna void.



Compleja la situación xd

Figura 2: Un meme fome

Índice

1 TDA

- Sobre la abstracción
- Características de los TDA
- Privacidad

2 Complejidad

- Ejercicio
- Motivación
- Análisis asintótico
- Ejercicio

Motivación

La eficiencia de un algoritmo, muchas veces, es un factor determinante a la hora de escribir una solución del problema. ¿Por qué?

Si su computador tiene la capacidad de procesar 10.000 [operación/seg], un algoritmo $\mathcal{O}(n^4)$ va a tomar 1 segundo para procesar 10 datos, pero **3 años** para procesar 10.000 elementos.

Notación Big-O

La notación Big-O se utiliza para representar el peor caso - el límite superior temporal de un algoritmo.

Esta notación se ocupa para comparar los algoritmos entre sí, dado que la notación da entender con claridad que tanto escala el algoritmo al aumentar el input. Este concepto se conoce como la velocidad de crecimiento.

Cotas y Limites

- Regla de la suma: si f_1 es de orden $\mathcal{O}(g)$ y f_2 es de orden $\mathcal{O}(h)$, entonces su suma es $f_1 + f_2 \in \mathcal{O}(\max(g, h))$

Por ejemplo, si f_1 es $\mathcal{O}(n)$ y f_2 es $\mathcal{O}(n^2)$, el algoritmo total se encuentra acotado por $\mathcal{O}(n^2)$



Figura 4: Representación gráfica

Cotas y Limites

- Regla del producto: si f_1 es de orden $\mathcal{O}(g)$ y f_2 es de orden $\mathcal{O}(h)$, entonces su producto es $f_1 \cdot f_2 \in \mathcal{O}(g \cdot h)$

Por ejemplo, si f_1 es $\mathcal{O}(n)$ y f_2 es $\mathcal{O}(n^2)$, el algoritmo total se encuentra acotado por $\mathcal{O}(n^3)$

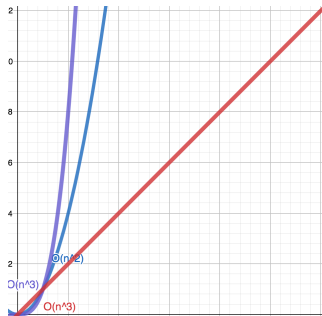


Figura 5: Representación gráfica

Cotas y Limites

■ Regla del limite: $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = k$

1. Si $k = 0$, entonces
 $f \in \mathcal{O}(g)$, $g \notin \mathcal{O}(f)$
2. Si $k \neq 0, \neq \infty$, entonces
 $\mathcal{O}(f) = \mathcal{O}(g)$

Por ejemplo, si $f(n) = n^2$ y $g(n) = n^3$, el límite da como resultado $k = 0$, por lo que se puede decir que f está acotado por el orden $\mathcal{O}(n^3)$ pero no al revés.

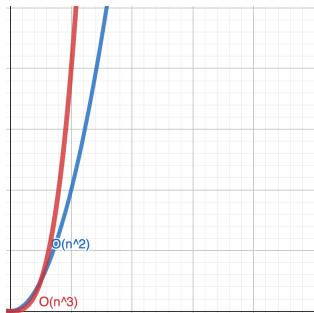


Figura 6: Representación gráfica

Ejercicio

Determine la complejidad de las siguientes funciones en notación \mathcal{O} :

```
1 int func1(int n){  
2     if(n == 1) return 1;  
3     return n*func1(n-1);  
4 }  
5  
6 void func2(int n){  
7     for(int i = 0; i < n; i++){  
8         for(int j = i; j < n; j++){  
9             printf("%d ", j);  
10            printf("\n");  
11        }  
12    }
```

Ejercicio

```
1 int func3(int n){
2     int count = 0;
3     for(int i = 1; i < n; i*=2)
4         count++;
5     return count;
6 }
7
8 void func4(int n) {
9     for(int i=0; i<n; i++)
10         func4(n-1);
11     printf("%d ", n);
12 }
```

Ejercicio — Solucion

- func1: $\mathcal{O}(n)$
- func2: $\mathcal{O}(n^2)$
- func3: $\mathcal{O}(\log n)$
- func4: $\mathcal{O}(n!)$