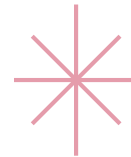
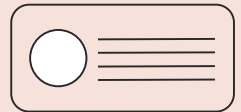
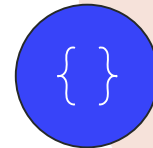
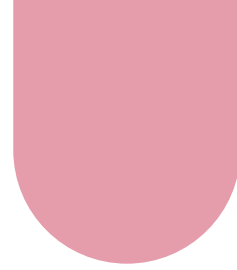


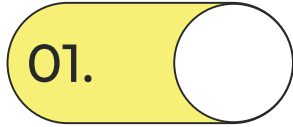


Estructura de datos y algoritmos

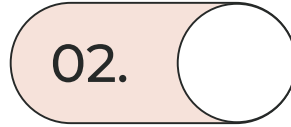
Profesara Responsable: Viviana Gasull
Colaborador: Pablo Poder



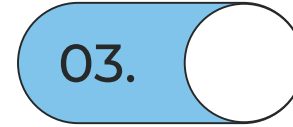
Contenidos de la materia



Algoritmos y
estructuras de datos



Algoritmos de
ordenamiento y
búsqueda



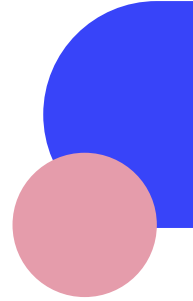
TAD lineales



TAD no lineales



Trabajo
integrador



Algoritmos y estructuras de datos

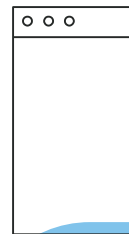


- Introducción
- Estructuras de datos
- Algoritmos
- Análisis de complejidad
- Resumen
- Preguntas

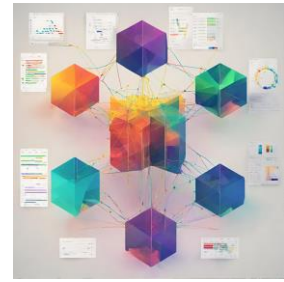


01.

Introducción



Introducción



- ... En el ámbito de la informática, la **representación** y **manipulación** eficiente de la información son aspectos cruciales. Los programas informáticos, en su mayoría, se centran en el almacenamiento y recuperación de datos, así como en la ejecución de cálculos. Para garantizar que estos programas satisfagan los requisitos de almacenamiento y tiempo de ejecución de manera eficiente, es necesario **organizar la información de manera adecuada**. Por esta razón, el estudio de las estructuras de datos y los algoritmos asociados constituye un componente fundamental en ciencias de la computación.

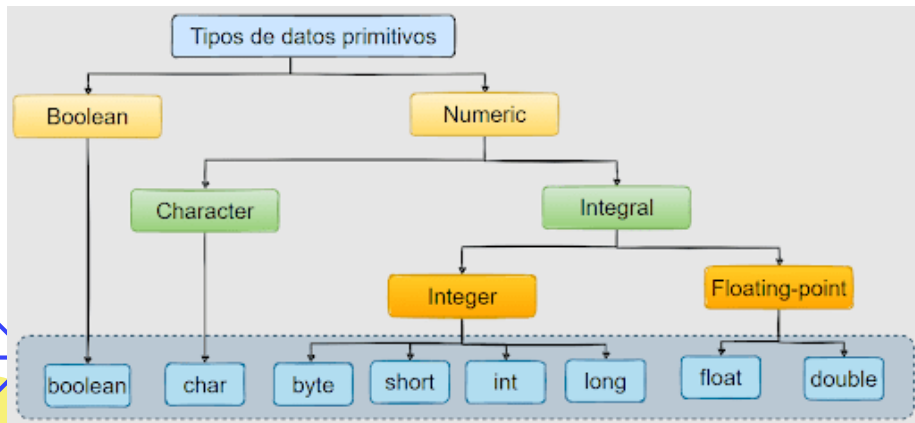


Tipos de datos

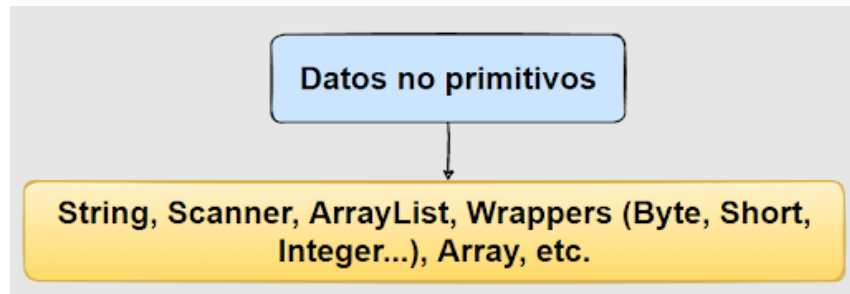
Definición

Un tipo de dato es un conjunto de valores y operaciones asociadas a esos valores.

Tipos simples



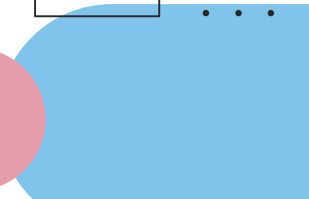
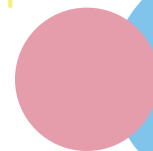
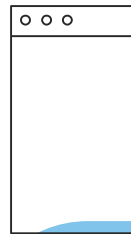
Tipos compuestos





02.

Estructura de datos



Estructuras de datos

Una estructura de datos es una agregación de tipos de datos compuestos y atómicos en un conjunto con relaciones bien definidas.

1. Una combinación de elementos en la que cada uno es o bien un tipo de dato u otra estructura de datos.
2. Un conjunto de asociaciones o relaciones (estructura) que implica a los elementos combinados.

• •
• •
• •
• •
• •
• •
• •



Estructuras de datos

A pesar del poder computacional actual, la eficiencia de los programas sigue siendo una prioridad. Las estructuras de datos ofrecen una organización eficiente de la información y sus operaciones asociadas. La selección adecuada de una estructura de datos puede conducir a programas más eficientes y a una mejor resolución de problemas.

• •
• •
• •
• •
• •
• •
• •
• •



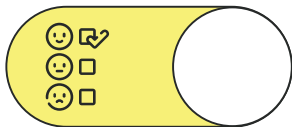
Estructuras de datos

Los pasos a seguir para seleccionar una estructura de datos que resuelva un problema son:

1. **Analizar** el problema para determinar las restricciones de recursos que debe cumplir cada posible solución.
2. **Determinar** las operaciones básicas que se deben soportar y cuantificar las restricciones de recursos para cada una. Ejemplos de operaciones básicas son la inserción de un dato en la estructura de datos, suprimir un dato de la estructura o encontrar un dato determinado en dicha estructura.
3. **Seleccionar** la estructura de datos que cumple mejor los requisitos o requerimientos.

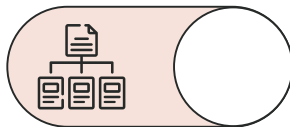
∴
∴
∴
∴
∴
∴
∴

Resumiendo



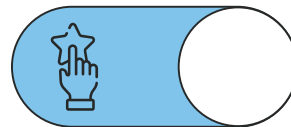
Tipo de dato

Colección de **valores**, por ejemplo Enteros.
Conjunto de **operaciones** que permiten manipular esos datos, para los enteros: suma, resta, producto, módulo, etc



Estructura de datos

Una combinación de elementos en la que cada uno es o bien un tipo de dato u otra estructura de datos.



Ejemplo

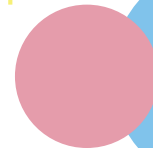
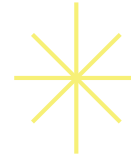
Un registro de una cuenta de X contiene varios campos: Nbre de usuario, mail, teléfono asociado, fecha de nacimiento etc





03.

Algoritmos





Algoritmo

Un algoritmo es un método, un proceso, un conjunto de instrucciones utilizadas para resolver un problema específico. Un problema puede ser resuelto mediante muchos algoritmos. Un algoritmo correcto, resuelve un problema definido y determinado (por ejemplo, encuentra el camino más corto).

“Un conjunto ordenado de pasos o instrucciones ejecutables y no ambiguas”.



Los algoritmos son independientes de los lenguajes de programación, pueden ser escritos en pseudolenguajes.



Propiedades de los Algoritmo



1. **Especificación precisa de la entrada.** Un algoritmo debe dejar claros el número y tipo de valores de entrada y las condiciones iniciales que deben cumplir esos valores de entrada para conseguir que las operaciones tengan éxito.
2. **Especificación precisa de cada instrucción.** Cada etapa de un algoritmo debe ser definida con precisión. Esto significa que no puede haber ambigüedad sobre las acciones que se deban ejecutar en cada momento.
3. **Exactitud, corrección.** Se debe poder demostrar que el algoritmo resuelve el problema. Se debe calcular la función deseada y convertir cada entrada a la salida correcta.
4. **Etapas bien definidas y concretas.** la acción descrita por cada etapa es comprendida por la persona o máquina que debe ejecutar el algoritmo.



Propiedades de los Algoritmo

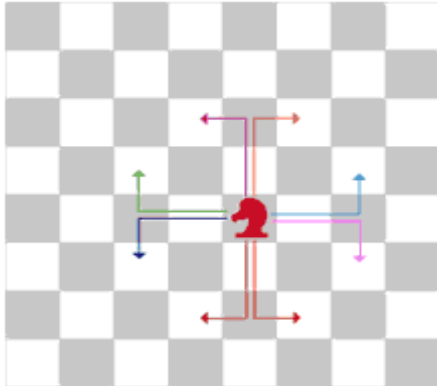


5. **Número finito de pasos.** Un algoritmo se debe componer de un número finito de pasos. La mayoría de los lenguajes que describen algoritmos (español, inglés o pseudocódigo) proporciona un método para ejecutar acciones repetidas, conocidas como iteraciones, que controlan las salidas de bucles o secuencias repetitivas.
 6. **Un algoritmo debe terminar.** En otras palabras, no puede entrar en un bucle infinito.
 7. **Descripción del resultado o efecto.** Se debe mostrar la salida esperada debe estar especificada completamente.
- EJ:** Listar todos los números reales entre 0 y 2, es resoluble mediante un algoritmo?



Ejemplo de Algoritmo

La peregrinación del caballo de ajedrez consiste en su paseo por todas las casillas del tablero sin pasar dos veces por la misma, utilizando sus movimientos: dos casillas horizontales y una vertical o a la inversa. Cuando desde la última casilla podemos pasar a la primera se trata de una "peregrinación cerrada".



1	16	11	6	3
10	5	2	17	12
15	22	19	4	7
20	9	24	13	18
23	14	21	8	25



Programas

Un programa de computadora es una representación concreta de un algoritmo en un lenguaje de programación. Por definición un algoritmo debe proporcionar suficiente detalle para que se pueda convertir en un programa cuando se necesite.

Problema

Un problema es una función o asociación de entradas con salidas.

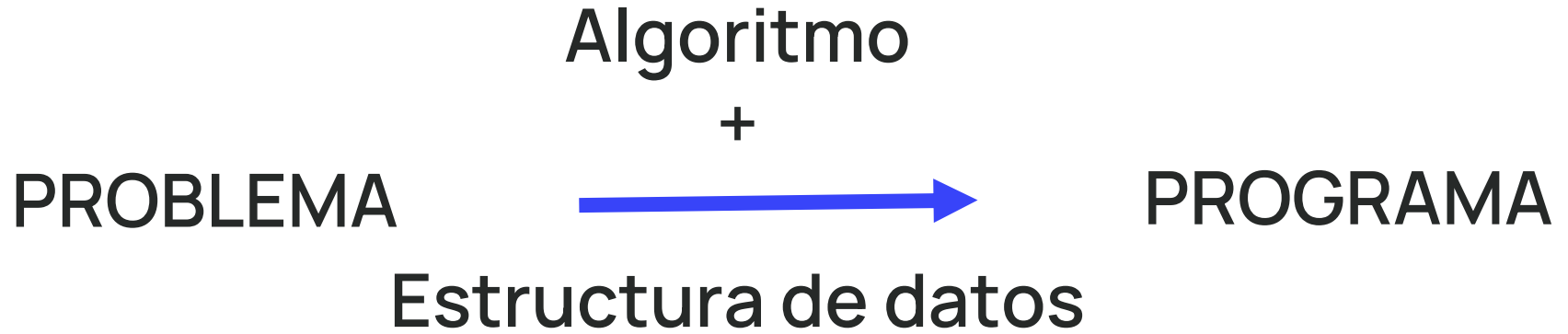
Algoritmo

Un algoritmo es una receta para resolver un problema cuyas etapas son concretas y no ambiguas. Además debe ser correcto y finito, y debe terminar para todas las entradas.

Programa

Un programa es una traducción de un algoritmo en un lenguaje de programación





Problema: Conjunto de hechos o circunstancias que dificultan la consecución de algún fin.

Algoritmo: Conjunto de reglas finito y no ambigua.

Estructura de datos: Disposición en memoria de la información.

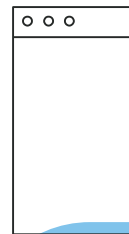
Programa: Algoritmos escrito en lenguaje de programación + Estructuras de datos.





04.

Análisis de complejidad





Análisis de algoritmos:

Consiste en determinar los recursos (tiempo, espacio) que consume un algoritmo en la resolución de un problema.

Porque es importante:

- Permite comparar algoritmos con criterios cuantitativos
- Facilita la elección de un algoritmo entre varios para resolver un problema

• •
• •
• •
• •
• •
• •
• •
• •





Eficiencia de un algoritmo

La **eficiencia** de un algoritmo es la propiedad mediante la cual un algoritmo debe alcanzar la **solución al problema** en el **tiempo** más corto posible o utilizando la cantidad más pequeña posible de **recursos** físicos y que sea compatible con su exactitud o corrección.

Un buen programador buscará el algoritmo más eficiente dentro del conjunto de aquellos que resuelven con exactitud un problema dado.

• •
• •
• •
• •
• •
• •
• •



Eficiencia y exactitud

La eficiencia se refiere a la capacidad de un algoritmo para alcanzar la solución en el menor tiempo posible o utilizando la menor cantidad de recursos físicos, manteniendo su exactitud o corrección.

Selección de algoritmos: Se plantean dos objetivos en el diseño

- a. hacer algoritmos fáciles de entender, codificar y depurar
- b. hacer algoritmos que utilicen eficientemente los recursos de la computadora.

Idealmente, se busca un equilibrio entre estos objetivos para lograr programas "elegantes".

Métodos para medir la eficiencia

La medida del rendimiento de un programa se consigue mediante la complejidad del espacio y del tiempo de un programa.

La complejidad del **espacio** de un programa es la cantidad de memoria que se necesita para ejecutarlo hasta la compleción (terminación). El avance tecnológico proporciona hoy en día memoria abundante; por esa razón, el análisis de algoritmos se centra fundamentalmente en el tiempo de ejecución.

La complejidad del **tiempo** de un programa es la cantidad de tiempo de computadora que se necesita para ejecutarlo. Se utiliza una función, $T(n)$, para representar el número de unidades de tiempo tomadas por un programa o algoritmo para cualquier entrada de tamaño n . Si la función $T(n)$ de un programa es $T(n) = c * n$, entonces el tiempo de ejecución es linealmente proporcional al tamaño de la entrada sobre la que se ejecuta. Tal programa se dice que es de tiempo lineal o, simplemente lineal.

• •
• •
• •
• •
• •
• •
• •
• •
• •
• •

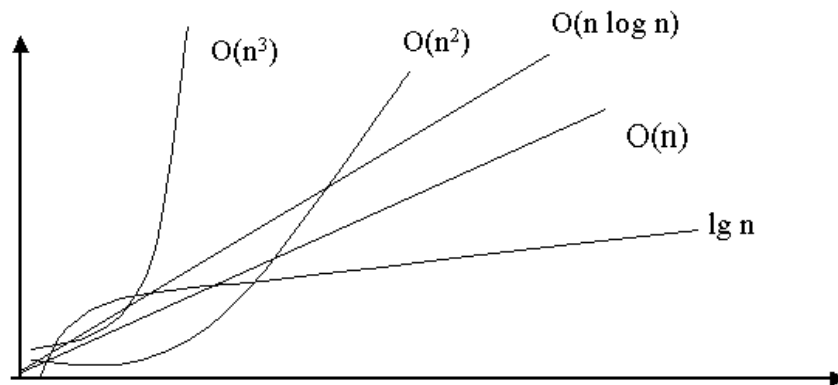
Análisis de algoritmos:



Tiempo de ejecución:

Hay 4 funciones típicas que describen el tiempo de ejecución de los algoritmos:

- Lineal
- Logarítmica
- Cuadrática
- Cúbica





Análisis de algoritmos - Tiempo de ejecución:



Los algoritmos de complejidad $O(n)$ y $O(n \log n)$ son los que muestran un comportamiento más "natural": prácticamente a doble de tiempo, doble de datos procesables o a doble cantidad de datos doble de tiempo.

Los algoritmos de complejidad logarítmica son muy buenos, dado que en el doble de tiempo permiten atacar problemas notablemente mayores, y para resolver un problema el doble de grande sólo hace falta un poco más de tiempo.








Análisis de algoritmos - Tiempo de ejecución:

Dentro de las complejidades polinómicas, las de orden $O(n^2)$ y $O(n^3)$ suelen ser efectivamente abordables, mientras que prácticamente nadie aceptaría algoritmos de orden $O(n^{100})$, aunque sean de orden polinómico.

Cualquier algoritmo por encima de una complejidad polinómica se dice "intratable" y sólo será aplicable a problemas muy pequeños.

A la vista de lo anterior se comprende que los programadores busquen algoritmos de complejidad lineal. Es un muy buen encontrar algo de complejidad logarítmica. Si se encuentran soluciones polinomiales, pueden ser aceptables; mientras que soluciones de complejidad exponencial, deben llevar a seguir buscando.



Calculo de $T(n)$

Función que calcula una serie de n términos.

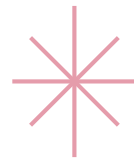
```
double serie(double x, int n){  
    Double s;  
    int i;  
    s = 0.0; // tiempo t1  
    for (i = 1; i <= n; i++) // tiempo t2  
    {  
        s += i*x; // tiempo t3  
    }  
    return s; // tiempo t4
```

La función $T(n)$ del método es:

$$T(n) = t1 + n*t2 + n*t3 + t4$$

El tiempo crece a medida que lo hace n , por lo que es preferible expresar el tiempo de ejecución de tal forma que indique el comportamiento que va a tener la función con respecto al valor de n .

Notación O grande



En el contexto de la alta velocidad de las computadoras actuales, la medida exacta de la eficiencia de un algoritmo no es crítica, pero sí lo es el orden de magnitud de la misma. Para evaluar las diferencias entre algoritmos, se utiliza el factor de eficiencia, expresado mediante la notación "O grande" ($O(n)$), que indica la cota superior del tiempo de ejecución de un algoritmo en relación con el tamaño de la entrada.

Que importa:

- Si el análisis de dos algoritmos muestra que uno ejecuta 25 iteraciones mientras otro ejecuta 40, la práctica muestra que ambos son muy rápidos;
- Si un algoritmo realiza 25 iteraciones y otro 2.500 iteraciones, entonces debemos estar preocupados por la rapidez de uno o la lentitud de otro.



Cómo se calcula $O(n)$



La notación O grande se puede obtener de $f(n)$ utilizando los siguientes pasos:

1. En cada término, establecer el coeficiente del término en 1.
2. Mantener el término mayor de la función y descartar los restantes. Los términos se ordenan de menor a mayor:

$\log_2 n$ n $n \log_2 n$ n^2 n^3 ... n^k 2^n
Ejemplo: Calcular la función O grande de eficiencia de la función $f(n) = 1/2n(n+1)$

$$f(n) = 1/2n(n+1) = 1/2n^2 + 1/2n$$

1. Se eliminan todos los coeficientes y se obtiene
 $n^2 + n$
2. Se eliminan los factores más pequeños
 n^2
3. La notación O correspondiente es $O(f(n)) = O(n^2)$



Complejidad de las instrucciones en Java

Sentencias de asignación:

$O(1)$

Sentencias de selección:

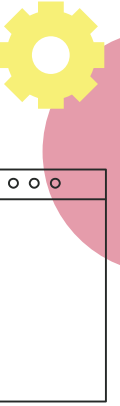
Máximo de las complejidades entre then y else

Bucle for

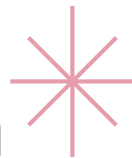
Máxima cantidad de iteraciones por la complejidad de las instrucciones del cuerpo del bucle

Complejidad de un bloque

La suma de las complejidades de cada sentencia del bloque.



Ejemplo de cálculo de O grande en Java



```
void escribeVector(double[] x, int n){
    int j;
    for (j = 0; j < n; j++)
    {
        System.out.println(x[j]);
    }
}
```

El método consta de un bucle que se ejecuta n veces, $O(n)$. El cuerpo del bucle es la llamada a `println()`, complejidad constante $O(1)$. Como conclusión, la complejidad del método es $O(n)*O(1) = O(n)$.

```
void traspuesta(float[][] d, int n){
    int i, j;
    float t;
    for (i = n - 2; i > 0; i--){
        for (j = i + 1; j < n; j++){
            t = d[i][j]; d[i][j] = d[j][i]; d[j][i] = t;
        }
    }
```

El método consta de dos bucles for anidados. El bucle interno está formado por tres sentencias de complejidad constante, $O(1)$. El bucle externo siempre realiza $n-1$ veces el bucle interno. A su vez, el bucle interno hace k veces su bloque de sentencias, k varía de 1 a $n-1$, de modo que el número total de iteraciones es:

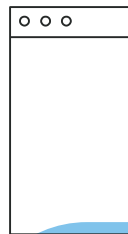
$$(n-1) + (n-2) + \dots + 1 = \frac{n*(n-1)}{2} \quad \begin{matrix} \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{matrix}$$

$O(n^2)$



05.

Resumen



Resumen



Una de las herramientas típicas más utilizadas para definir algoritmos es el pseudocódigo, representación en español u otro idioma del código requerido para un algoritmo.

...

...

...

...

...

...

...

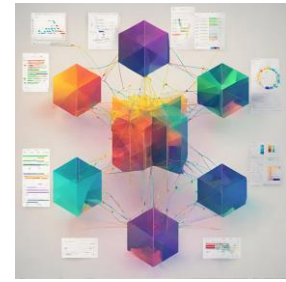
...

...

Los datos atómicos son datos simples que no se pueden descomponer. Un tipo de dato atómico es un conjunto de datos atómicos con propiedades idénticas. Este tipo de datos se definen por un conjunto de valores y un conjunto de operaciones que actúa sobre esos valores.



Resumen



Una estructura de datos es un agregado de datos atómicos y datos compuestos en un conjunto con relaciones bien definidas.

... La eficiencia de un algoritmo se define, generalmente, en función del número de elementos a procesar y el tipo de bucle que se va a utilizar.





¿Preguntas?

