PROBLEMAS Y ALGORITMOS

Pasos para resolver un problema

- Analizar y comprender el problema: muy importante
- Construir la solución: se elige y se aplica una estrategia o un conjunto de estrategias combinadas
- Verificar la solución: confrontar los resultados obtenidos con el problema, verificando que la solución sea correcta

Cómo analizar un problema

En todo problema se pueden distinguir los siguientes elementos:

– Los datos.
– La incógnita.
– El conjunto de reglas que los vinculan.

El análisis de un problema comienza con la correcta identificación de cada uno de estos elementos.

Para esto, es una buena idea tener en cuenta las siguientes recomendaciones.

Análisis y comprensión del problema

- Leer con detenimiento todo el enunciado.
- Comprender el significado de cada palabra y frase.
- Atender a los signos de puntuación.
- Identificar la incógnita.
- Identificar datos explícitos (los puede haber relevantes e irrelevantes).
- Identificar datos implícitos (tanto relevantes como irrelevantes) y explicitarlos.
- Eliminar las dobles negaciones.
- Detectar ambigüedades e imprecisiones para luego resolverlas.
- Obtener inferencias a partir de los datos identificados y hacerlas explícitas.
- Construir un enunciado simple y sencillo con los datos considerados relevantes.
- Verificar la equivalencia entre la especificación inicial y la obtenida.

Construir la solución

Sabemos que encontrar la solución a un problema puede no ser una tarea sencilla

Para buscar una solución a un problema, lo que hacemos es plantear un **modelo**

- Un **modelo** es una versión simplificada de la realidad que cuenta sólo con los datos relevantes del problema
- Los modelos se construyen haciendo **Abstracciones**

Abstracciones

- Las abstracciones tratan de simplificar la realidad
- Un mapa es un ejemplo de abstracción que usamos muy a menudo

Formas de describir el problema

- Descripción verbal
- Notación matemática: descripciones algebraicas, ecuaciones, sistemas de ecuaciones
- Gráficos
- Representaciones geométricas

- Diagramas: por ejemplo, de conjuntos
- Notaciones lógicas

Abstracciones y modelos

- Las abstracciones nos permiten construir modelos
- Usaremos los algoritmos como un modelo
- El objetivo es aprender a decirle a la computadora cómo resolver un problema

Algoritmo

Un algoritmo es una secuencia de pasos u operaciones, que al ejecutarlo producirá resultados y terminará luego de una cantidad finita de tiempo

Existen muchos ejemplos de algoritmos en nuestra vida cotidiana, no relacionados con las computadoras

Es importante que en los algoritmos:

- Cada paso debe estar definido sin ambigüedad
- Las operaciones deben ser comprensibles
- Debe haber un único punto de comienzo y al menos un punto final

Ejercicio

1) Algoritmo para hacer un mate.
2) Algoritmo para determinar los alumnos ausentes en el curso.

Los algoritmos tienen datos y acciones

Las acciones modifican los datos de entrada y obtienen a partir de ellos a los datos de salida que son la solución buscada

Algoritmo con datos

```
ALGORITMO PromedioCuatro
DATOS DE ENTRADA: NUM1, NUM2, NUM3, NUM4
DATOS DE SALIDA : PROMEDIO
Promedio ← (num1 + num2 + num3 + num4) / 4
```

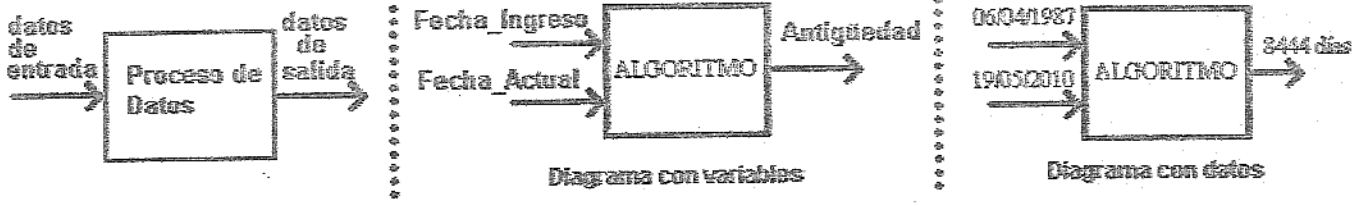
Desarrollo de algoritmos

Los algoritmos se deben escribir en forma clara y prolija.

Todo dato en un algoritmo debe tener un nombre significativo: este nombre podrá constituirse luego en un elemento denominado VARIABLE. La variable será el objeto que contenga el dato (*por ejemplo, la variable* Nombre *puede contener el valor o dato* "Andrea", *y la variable* PI *puede contener el dato 3.14, etc*)

Toda variable de entrada en un algoritmo tiene asociado un valor y toda variable de salida debería devolver un valor.

Es importante distinguir cuáles son los datos (variables) de entrada y de salida. por lo que entonces podríamos comenzar la resolución del problema identificando las variables de entrada y de salida de nuestro algoritmo. Para ello puede ayudarnos el diagrama de "caja negra" donde solo se aprecia la entrada y la salida, sin preocuparnos demasiado qué sucede o cómo es la transformación de los datos de entrada. Podemos usar Datos o Variables en este diagrama, lo importante es que nos ayude en el análisis del problema a resolver.



Lenguaje Pseudocódigo

Los Algoritmos se puede expresar de muchas maneras, aunque en este curso los expresaremos utilizando lenguaje de pseudocódigo. En pseudocódigo la secuencia de instrucciones se representa por medio de frases o proposiciones.

El pseudocódigo está compuesto por proposiciones informales en español que permiten expresar detalladamente las instrucciones que llevan desde un estado inicial (problema) hasta un resultado deseado (solución). Por lo regular, los algoritmos se escriben por refinamiento: se escribe una primera versión que luego se descompone en varios subproblemas (el número depende de la complejidad del problema) independientes entre sí. Si es necesario se va refinando cada vez las instrucciones hasta que las proposiciones generales en español se puedan codificar en el lenguaje seleccionado. Nuestros primeros algoritmos no requerirán de refinamiento, serán algoritmos simples.

Si el algoritmo requiere del uso de elementos de matemática, seguramente requiera del uso de operadores, operandos, y expresiones.

Un operador actúa sobre una, dos o más *variables* (datos) para realizar una determinada operación con un determinado resultado. Ejemplos típicos de operadores son la suma (+), la diferencia (-), el producto (*), etc. Hay varias clases de operadores: Aritméticos, Relacionales, Lógicos, etc.

Aritméticos: + (suma), - (resta), * (producto), / (división), **Mod** (Resto de la división entera)
Relacionales: *(permiten la comparación entre datos del mismo tipo y dan como resultado dos valores posibles: Verdadero o Falso).* = (igual a); < (menor que) ; > (mayor que).
Lógicos: *(permiten la evaluación lógica de dos expresiones de tipo lógico. Dan como resultado uno de dos valores posibles: Verdadero o Falso).* Y (conjunción); O (disyunción) ; NO (negación)
La conjunción devuelve un valor "VERDADERO" sólo si todos sus operandos resultan VERDADEROS, sino, devuelve el valor "FALSO".
La disyunción devuelve un valor FALSO sólo si todos sus operandos resultan con "FALSO".
Miremos como ejemplo la siguiente tabla, donde A y B son variables cuyos valores son lógicos.

CONJUNCIÓN

A	B	A y B
F	F	F
F	V	F
V	F	F
V	V	V

DISYUNCIÓN

A	B	A o B
F	F	F
F	V	V
V	F	V
V	V	V

Una expresión es un conjunto de variables y constantes relacionadas mediante distintos operadores. Un ejemplo de expresión en la que intervienen operadores aritméticos es el siguiente:

3*x - x*x/2

Ejemplo de expresiones con operadores relacionales:

1 < 2 *(FALSO)*
Promedio > 7 (*puede ser VERDADERO o FALSO, de acuerdo al valor de PROMEDIO*)
(Nota1 > 7) Y (Nota2 > 7)

Estructuras

Un Algoritmo está compuesto por instrucciones organizadas secuencialmente, en forma de estructuras de control. De estas estructuras, las más comunes son la estructura Secuencial, la estructura

condicional (estructura de decisión, y estructura de selección) y la estructura Iterativa (estructura de repetición).

Estructura secuencial

La estructura de control secuencial es la más sencilla. También se la conoce como estructura lineal. Se compone de instrucciones que deben ejecutarse en forma consecutiva, una tras otra, siguiendo una línea de flujo. Solamente los problemas muy sencillos pueden resolverse haciendo uso únicamente de esta estructura.

Ejemplo de estructura secuencial: algoritmo para calcular el promedio de 4 notas

Para resolver este algoritmo, usaremos instrucciones en pseudocódigo para señalar el *Comienzo* y *Fin* del algoritmo, la entrada de datos (*Leer*) y la salida de datos (*Escribir*), la asignación del dato a una variable (←), y expresiones con operadores aritméticos. Observá que el algoritmo también tiene asignado un nombre (Prom_4) y un comentario, que comienza con el símbolo apóstrofe (')

```
ALGORITMO Prom_4
'Obtiene el Promedio de Cuatro valores
Comienzo
  Leer(Nota1)
  Leer(Nota2)
  Leer(Nota3)
  Leer(Nota4)
  Promedio ← (Nota1 + Nota2 + Nota3 + Nota4) / 4
  Escribir("El Promedio es:")
  Escribir(Promedio)
Fin.
```

Leer(Nota1) es la instrucción en pseudocódigo que usaremos para asignar a la variable "Nota1" un valor ingresado desde el teclado por el usuario.

Escribir(Promedio) es la instrucción en pseudocódigo que usaremos para que el valor de la variable "Promedio" sea mostrada en el monitor.

Estructura condicional

La estructura condicional (*o estructura de selección, estructura selectiva o estructura de decisión*) se utiliza para indicarle al computador que debe evaluar una condición y, a partir del resultado, ejecutar el bloque de instrucciones correspondiente. La forma más común está compuesta por una proposición (condición) que se evalúa y dos bloques de instrucciones que se ejecutan, uno cuando la condición es verdadera (condicional simple y doble) y otro cuando ésta es falsa (condicional doble).

Para que una proposición sea válida, debe poder afirmarse que es verdadera o falsa.

La estructura condicional tiene tres variantes: simple, doble, y múltiple.

Las estructuras condicionales simple y doble evalúan una proposición (condición) que devuelve como resultado únicamente dos valores posibles y excluyentes: verdadero o falso. En cambio, la estructura condicional de selección múltiple permite que la condición devuelva más de un valor posible y que para cada uno de esos valores se ejecute el bloque de instrucciones correspondiente.

condición simple

```
Si <condicion>
  Entonces
    < bloque de instrucciones
      que se ejecutan
      si la condición
      es VERDADERA >
FinSi
```

condición doble

```
Si <condición>
  Entonces
    < bloque de instrucciones
      que se ejecutan
      si la condición
      es VERDADERA >
  Si no
    < bloque de instrucciones
      que se ejecutan
      si la condición
      es FALSA >
FinSi
```

condición múltiple

```
En Caso <expresión> sea
  S1:<instrucción o bloque de
      instrucciones que se
      ejecutan si la expresión
      es igual a S1>
  S2:
  S3:
  . . .
  Sn:
  Si no
    < bloque que se ejecuta
      si la expresión no coincide
      con ningún caso >
Fin CASO
```

condición simple

```
Si Monto > $ 100
  Entonces
    Descuento ←— 5%
FinSi
```

condición doble

```
Si Sexo='M'
  Entonces
    Escribir('Masculino')
  Si no
    Escribir('Femenino')
FinSi
```

condición múltiple

```
En Caso Nota sea
  7: Escribir('Buena')
  8: Escribir('Muy Buena')
  9: Escribir('Felicitaciones')
 10: Escribir('Excelente')
Si no
  Escribir('¡¡¡la próxima vez será!!!)
FinCaso
```

Estructuras condicionales anidadas

Hay situaciones que requieren el uso de estructuras condicionales anidadas. En estas, el resultado de la primera proposición implica evaluar a continuación una segunda proposición y esta a su vez requiere que se evalúe una tercera proposición, y así sucesivamente, hasta agotar todas las condiciones. Se pueden anidar las tres variantes citadas.

Estructura iterativa o de repetición.

En un algoritmo puede darse el caso de operaciones de repetición, como por ejemplo, supongamos el caso donde tenemos que ingresar 10 valores numéricos y escribir la suma, etc

La estructura iterativa o de repetición permite ejecutar una o varias instrucciones, un número determinado de veces (*como en el ejemplo citado*), o, indefinidamente, mientras se cumpla una condición (por ejemplo, *supongamos que deseamos ingresar valores numéricos mientras que la suma de ellos no supere a 41*, etc).

En programación existen al menos dos tipos de estructuras repetitivas, las cuales a su vez tienen variantes en los diferentes lenguajes de programación. La característica común es que ambos tipos permiten ejecutar una o varias instrucciones: un número determinado de veces mientras se cumpla una condición.

Si sabemos la cantidad de repeticiones a realizar, podemos utilizar la estructura "Para…".

Si la cantidad de repeticiones depende de alguna condición, podemos usar la estructura "Mientras <condición>…Hacer", o "Repetir…Hasta <condición>".

Estructura PARA. *CASO 1: incrementos de a 1.*

La sintaxis la podemos interpretar así "Para un *Valor_a_Contar desde Valor_Inicial* Hasta *Valor_Final* incrementando de a 1 hacer… "

ESTRUCTURA "PARA"

```
Para VC = VI hasta VF
  <bloque de instrucciones>
Fin Para
```

```
ALGORITMO Suma10
'Leer 10 valores, ir sumándolos,
'y mostrar el resultado obtenido
Comienzo
  Para i = 1 hasta 10
    Leer(Valor)
    Acumulado ←— Acumulado + Valor
  Fin Para
  Escribir('La suma de los 10 valores ingresados es:')
  Escribir( Acumulado )
Fin
```

En el ejemplo vemos que se utilizó una variable iteradora **i** cuyo valor inicial es "1", y se irá incrementando de a 1 hasta que su valor llegue a "10". El bucle será entonces de 10 iteraciones, esto es, el bloque de instrucciones dentro del **Para…FinPara** se ejecutará 10 veces.

Estructura MIENTRAS (While).

Las instrucciones dentro del bucle se ejecutarán mientras se cumpla una condición evaluada antes de cada iteración, es decir, la condición deberá ser VERDADERA para que se continúe iterando.

El numero de repeticiones oscila entre 0 e infinito, dependiendo de la evaluación de la condición, cuyos argumentos en los casos de repetición, al menos una vez, deberían modificarse dentro del bucle, pues de no ser así el numero de repeticiones seria infinito y nos encontraremos en un bucle sin salida.

ESTRUCTURA ITERATIVA "MIENTRAS"

```
MIENTRAS Expresión hacer
  < Sentencia o bloque de sentencias >
FIN Mientras
```

```
ALGORITMO MontoTotal
 // Ingreso de montos, suma y cantidad ingresada.
 COMIENZO
  Continuar ← 'S'
  Mientras Continuar = 'S' hacer
   Leer(Monto)
   Cuenta ← Cuenta + 1
   S ← S + Monto
   Escribir('Ingresa otro Monto? S / N')
   Leer(Continuar)
  Fin Mientras
  Escribir('Monto Total: ', S, ' cantidad ingresada: ',Cuenta)
 FIN
```

Expresión debe retornar un valor lógico (Verdadero o Falso). Si se evalúa como VERDADERO, continúa el bucle, y si se evalúa como FALSO la ejecución del bucle finaliza.

Estructura Repetir (Repeat)

En esta estructura se repite una acción hasta que se cumpla la condición que controla el bucle, la cual se evalúa después de cada ejecución del mismo.

El número de repeticiones oscila entre 1 e infinito, dependiendo de la evaluación de la condición, cuyos argumentos en los casos de repetición, al menos dos veces, deberán modificarse dentro del bucle, pues de no ser así el número de repeticiones será infinito y nos encontraremos en un bucle sin salida.

ESTRUCTURA ITERATIVA "REPETIR"

```
REPETIR
  <sentencias o bloque de sentencias>
HASTA Expresión
```

```
ALGORITMO MontoTotal
 // Ingreso de montos, suma y cantidad ingresada.
 COMIENZO
  REPETIR
   Leer(Monto)
   Cuenta ← Cuenta + 1
   S ← S + Monto
   Escribir('Ingresa otro Monto? S / N')
   Leer(Continuar)
  HASTA Continuar='N'
  Escribir('Monto Total: ', S, ' cantidad ingresada: ',Cuenta)
 FIN
```

Expresión debe devolver un valor lógico: si devuelve Verdadero, se finaliza la ejecución del bucle. Por lo tanto, el bucle seguirá ejecutándose mientras la condición sea Falsa.

ATENCIÓN!: el algoritmo "MontoTotal" fue implementado con las estructuras "Mientras" y "Repetir". Observemos que la expresión lógica de la que depende la finalización del bucle es diferente en cada caso. En la estructura "Mientras" fue necesario inicializar la variable "Continuar" para que pueda ingresarse al bucle al menos una vez, lo que no fue necesario en el caso de la estructura "Repetir". Tal vez para el ejemplo se adapte mejor el uso de la estructura "Repetir" que el de la estructura "Mientras". Para cada caso, debemos analizar cuál sería la estructura más cómoda. La elección de cuál estructura usar quedará a criterio del programador.