Tecnologías en Educación Matemática

MODULO 9

Dpto. de Ciencias e Ingeniería de la Computación
UNIVERSIDAD NACIONAL DEL SUR
Año 2019

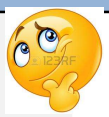
1

Algoritmos - Primitivas

Dada una secuencia de números ingresada por teclado, calcular la suma de aquellos cuya cantidad de dígitos es PAR. Inicialmente contamos con la longitud de la secuencia.

```
ALGORITMO SumaSecuencia
DE: long, Números ingresados por teclado
DS: suma
Aux: N
COMIENZO
suma ← 0
Repetir long veces
        Leer(N)
        {calcular la cant de dígitos de N}
        Si esa cantidad es PAR
        entonces
                suma ← suma + N
FIN
```

Primitiva que resuelve el sub-problema



Algoritmos - Primitivas

Dada una secuencia de números ingresada por teclado, calcular la suma de aquellos cuya cantidad de dígitos es PAR. Inicialmente contamos con la longitud de la secuencia.

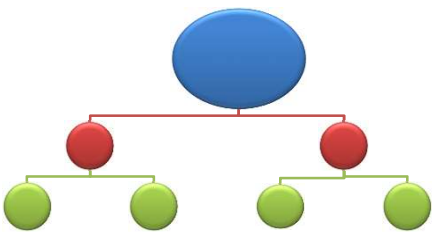
```
ALGORITMO SumaSecuencia
DE: long, Números ingresados por teclado
DS: suma
Aux: N
COMIENZO
suma ← 0
Repetir long veces
        Leer(N)
        Si CantDIG(N) mod 2 = 0
        entonces
                suma ← suma + N
FIN
```

Algoritmos – Primitivas

Descomposición de Problemas en Sub-Problemas:

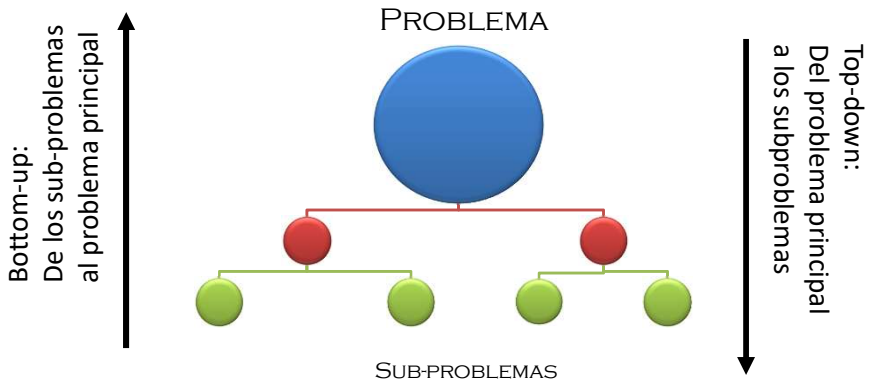
Cuando la complejidad de los problemas aumenta, la tarea de hallar una solución se torna más difícil.

Una metodología para reducir la complejidad consiste en plantear la solución del problema a partir de la solución de una serie de sub-problemas más sencillos que forman parte del problema original.



Algoritmos – Primitivas

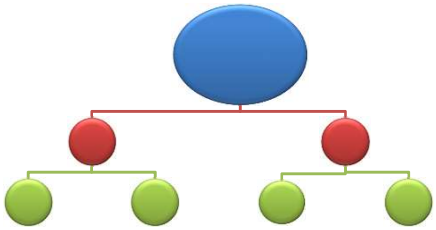
"Bottom-up vs. Top-down"



Algoritmos – Primitivas

"Bottom-up vs. Top-down"

- La técnica *bottom-up* consiste en ir resolviendo en primer lugar los problemas más elementales, usando esas soluciones para resolver problemas mayores.
- La técnica *top-down* resuelve primeramente el problema mayor, dejando pendiente la solución de los problemas más elementales para más adelante.



Algoritmos – Primitivas

Problema: Dados dos números N1 y N2, ver cuántos dígitos en común tienen.

Por ejemplo, N1=2461 y N2=652 tienen 2 dígitos en común.

Algoritmos – Primitivas

Sub-problema: ¿Cómo determino si un dígito Dig está presente en el número N?

```
Esta ← falso
Mientras (Esta = falso) Y (N>0) hacer
        SI (Dig = (N mod 10))
           ENTONCES Esta ← verdadero
           SINO N  ← N div 10
```

Algoritmos – Primitivas

Problema: Dados dos números N1 y N2, ver cuántos dígitos en común tienen.
Sub-problema: ¿Cómo determino si un dígito Dig está presente en el número N?

```
Algoritmo CuantosComunes
DE: N1, N2
DS: cuantos
Daux: dig
cuantos ← 0
Para dig desde 0 hasta 9
        si esta(dig, N1) y esta(dig, N2)
        entonces
                cuantos ← cuantos + 1
Fin Para
Fin Algoritmo
```

Primitiva: esta
DE: digito, numero
DS: esta (v o f)
Se invoca 2 veces, nos ahorramos de repetir el código

Algoritmos – Primitivas

Algoritmos como Primitivas

Un algoritmo se identifica por su **nombre**, sus **datos de entrada** y sus **datos de salida**.

Un algoritmo A puede usar otro algoritmo B como primitiva, y para ello debe indicar:

- el nombre de B,
- los datos de A que serán los datos de entrada para B,
- en qué datos el algoritmo A recibirá los datos de salida del algoritmo B (NO SIEMPRE).

Al usar un algoritmo como primitiva, decimos que lo estamos "invocando" o "llamando".

Algoritmos – Primitivas

Invocación

La invocación de un algoritmo debe coincidir con la definición del algoritmo en los siguientes aspectos:

- el nombre,
- la cantidad de datos de entrada,
- la cantidad de datos de salida,
- el orden de los datos de entrada,
- el orden de los datos de salida y
- el tipo de los datos (numérico o lógico).

Algoritmos – Primitivas

Invocación a FUNCION

Cuando un algoritmo tiene **un sólo dato de salida**, puede ser invocado **desde una expresión**.

En estos casos, la invocación sólo contendrá el nombre de la primitiva y los datos de entrada.

Nombre_Funcion (dato-ent1, dato-ent2,...,)

En cada llamada, los datos de entrada deben tener un valor ya asignado, y ese valor es enviado al algoritmo llamado. El dato de salida será un único valor devuelto al lugar donde se realizó la invocación.

Se usa cuando tenemos 1 UNICO DATO DE SALIDA

Algoritmos – Primitivas

Ejemplos de invocación a funcion

CantidadDigitos: El dato de salida es único y de tipo entero

EsPrimo: El dato de salida es único y de tipo lógico

```
Valor ← (CantidadDígitos(Num)*2) +1
        SI EsPrimo(Número) = verdadero
        ENTONCES
                M ← Máximo3Números(Num1,Num2,Num3)
        ....
```

Algoritmos – Primitivas

Problema Propuesto

Calcular cual es el dígito que más veces se repite en un número. Si son varios los dígitos que se repiten más veces, entonces el algoritmo debe devolver el más chico de todos.

- Asuma que cuenta con la primitiva ContarVeces vista previamente.

Algoritmos – Primitivas

Problema

Sucesiones

Escriba un algoritmo para calcular el i-ésimo término de la siguiente sucesión infinita:

S = 2/1, 4/2, 8/6, 16/24, 32/120, 64/720…

Término general: $2^N/N!$

Algoritmos – Primitivas

¿Cómo resolver el problema?

- Descomponer el problema en sub-problemas:
  - Calcular Potencia
  - Calcular Factorial
  - Calcular el i-ésimo Término usando *Potencia* y *Factorial*.

Algoritmos – Primitivas

Resuelvo el sub-problema Potencia

```
ALGORITMO Potencia
DATOS DE ENTRADA: base, Exponente {naturales}
DATOS DE SALIDA: Potencia {natural}
DATOS AUXILIARES:
COMIENZO
    Potencia ← 1
    REPETIR Exponente VECES
        Potencia ← Potencia*base
FIN ALGORITMO
```

Algoritmos – Primitivas

Resuelvo el sub-problema Factorial

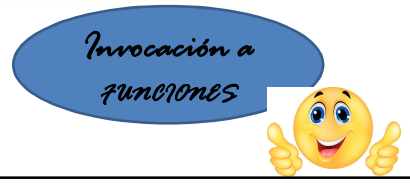
```
ALGORITMO Factorial
DATOS DE ENTRADA: N {naturales}
DATOS DE SALIDA: Factorial {natural}
COMIENZO
    Factorial ← 1
    para i desde 1 hasta N hacer
        Factorial ← Factorial*i

FIN ALGORITMO
```

Algoritmos – Primitivas

Ahora utilizo los algoritmos anteriores para resolver el problema de calcular el i-ésimo término.

```
ALGORITMO Término-i-ésimo
DATOS DE ENTRADA: i {natural}
DATOS DE SALIDA: Término-i-ésimo {real}
COMIENZO
    Término-i-ésimo ← Potencia(2,i)/Factorial(i)
 FIN ALGORITMO
```



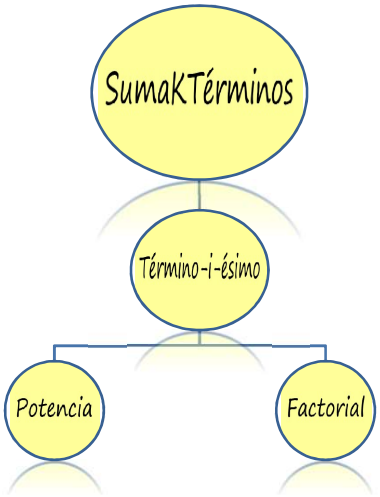
Algoritmos – Primitivas

Si ahora quisiera escribir un algoritmo para calcular la sumatoria de los primeros K términos de la sucesión anterior...

```
ALGORITMO SumaKTérminos
DATOS DE ENTRADA: K {natural}
DATOS DE SALIDA: Sumatoria {real}
DATOS AUXILIARES: i {natural}
COMIENZO
    Sumatoria ← 0
    PARA i Desde 1 hasta K hacer
          Sumatoria  ← Sumatoria + Término-i-ésimo(i)
   Fin PARA
FIN ALGORITMO
```

Algoritmos – Primitivas

Descomposición del Problema



Algoritmos – Primitivas

Invocación a PROCEDIMIENTO

Se escribe el nombre seguido de los datos de entrada y los datos de salida subrayados; todos los datos van encerrados entre paréntesis.

Nombre(dato-ent1, dato-ent2,..., dato-sal1, dato-sal2, ...)

Al momento de realizar la llamada, los datos de entrada deben tener un valor ya asignado, y ese valor es enviado al algoritmo llamado. Los datos de salida toman el valor devuelto por el algoritmo que fue llamado.

Se usa cuando tenemos CERO o MAS DE 1 DATO DE SALIDA

Algoritmos – Primitivas

Ejemplo: A partir de un número M >0, mostrar el resultado de calcular cada dígito de M elevado a la cantidad de veces que el dígito lo indica (asuma que ningún dígito de M es 0).

```
ALGORITMO MostrarNumeroEnPartes
DATOS DE ENTRADA: M
DATOS DE SALIDA: -
DATOS AUXILIARES:
COMIENZO
Mientras (M>0)
    mostrarPotDigito(M mod 10)
    M ← M div 10

FIN ALGORITMO
```

Algoritmos – Primitivas

OJO!! Para que la solución esté completa falta especificar el procedimiento MostrarPotDigito

```
ALGORITMO MostrarPotDigito
DATOS DE ENTRADA: d
DATOS DE SALIDA: -
DATOS AUXILIARES: i {natural}
COMIENZO
    pot ← 1
    PARA i Desde 1 hasta d hacer
        pot ← pot * d
   Fin PARA
   mostrar(pot)
FIN ALGORITMO
```

Tecnologías en Educación Matemática

FIN MODULO 9

Dpto. de Ciencias e Ingeniería de la Computación
UNIVERSIDAD NACIONAL DEL SUR
Año 2019

25