

## Ejercicios de diagrama de flujo y pseudolenguaje

### **Búsqueda de un elemento en un conjunto:**

- se sabe que el elemento está
- no se sabe si el elemento está

### **Problema:** Buscar una carta

Se tiene un mazo de cartas inglesas, el cual no está completo, pero se sabe que tiene "n" cartas, y está desordenado. Se desea determinar si una carta determinada está o no en el mazo, y si está, en qué posición

En este caso no sabemos si la carta está o no en el mazo.  
Solución usando bandera para señalar que la carta está:

```
Inicio
Entrada: n % total de cartas en el mazo
Entrada: buscado % carta que quiero buscar
contador ← 1
bandera ← 0
entrada: carta % valor de la carta
mientras (contador < n)
  {entrada: carta
   contador ← contador +1
   si (carta = buscado)
     { bandera ← 1
      pos_buscado ← contador
     }
  }
si ( bandera = 0)
  { salida: "la carta no está en el mazo"
  }
sino
  { salida: "la carta está en la posición:", contador
  }
fin
```

Otro algoritmo:

```
Inicio
Entrada: n % total de cartas en el mazo
Entrada: buscado % carta que quiero buscar
contador ← 1
entrada: carta % valor de la carta
mientras (buscado != carta & contador < n)
  {entrada: carta
   contador ← contador +1
  }
si ( buscado = carta)
```

```

    { salida: "la carta está en la posición:", contador
  }
sino
  { salida: "la carta no está en el mazo"
  }
fin

```

**Problema:**

Si sabemos que la carta está en el mazo ¿Cómo puede variar los algoritmos anteriores? ¿Es mejor el algoritmo en este caso? ¿Por qué?

**Problema:**

Problema anterior, pero que despliegue un mensaje “encontrado” cada vez que encuentre la carta, y la cantidad de veces que la ha encontrado hasta ese momento.

**Problema:**

Como puede variar el problema si los números se ingresan ordenados.

**Problema:**

Buscar la carta mayor de un mazo no completo, que tiene “n” cartas.  
Indicar la posición en que se encuentra

En este caso, la carta buscada está en el mazo, pero no sabemos cual es. Debe de todas maneras mirar el mazo hasta la última carta

```

Inicio
Entrada: n
Entrada: carta
mayor ← carta
pos_mayor ← 1
contador ← 1
mientras (contador <n)
  {entrada: carta
  si (carta > mayor)
    {mayor ← carta
    pos_mayor ← contador +1
    }
  contador ← contador +1
  }
salida: "La carta mayor es",mayor
salida: "la posición del mayor es:",pos_mayor

```

fin

**Problema:**

Buscar la menor, y la cantidad de veces que esta carta se repite

**Problema:**

Dada como entrada una hora en formato hh:mm, [24], genere como salida la misma hora pero en formato hh:mm [am/pm]

Para ello considere:

0 hrs --> 12 pm  
1 --> 1 am  
12 --> 12 am  
13 --> 1 pm  
23 --> 11 pm

**Problema: factorial**

Dado un entero positivo, "n", calcule el factorial de "n"

```
Inicicion
Entrada: n
factorial ← 1
mientras (n > 1)
{
    factorial ← factorial * n
    n ← n -1
}
salida: factorial
fin
```

**Problema: serie de ulam**

Dado un valor inicial "x", se pide generar los términos de la serie de Ulam.

Se dice que cualquier número menor que 1800 converge a 1 si se calcula cada término siguiendo las reglas de la serie de Ulam.

La regla para calcular el término siguiente a "x"es:

- Si x par, el siguiente término se obtiene dividiendo x por 2
- Si x es impar, el siguiente término se obtiene multiplicando x por 3 y sumando 1 al resultado.

Por ejemplo:

```
x = 45
x_1 = (45*3)+1 = 136
x_2 = (136/2) = 68
x_3 = (68 /2) =34
x_4=(34/2) = 17
x_5 = (17*3)+1 =52
...
26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1
```

```
Inicio
Entrada: x
salida: x
mientras (x>1)
{ si (x % 2 =0)
  { x ← x/2
    salida: x
  }
  sino
  { x ← x*3+1
    salida: x
  }
}
fin
```

Problema: caída libre

La ecuación que rige el movimiento uniformemente acelerado de una partícula está dado por la siguiente fórmula:  $x(t) = x_0 + v_0 \cdot t + \frac{1}{2} \cdot a \cdot t^2$ , donde  $x_0$  es la posición inicial de la partícula,  $v_0$  es la velocidad inicial y  $a$ , la aceleración.

El movimiento de caída libre se rige por la misma ecuación anterior, salvo que la velocidad inicial es cero, (se deja caer el objeto del reposo), y la aceleración es la constante  $g$ , que se puede aproximar por 10 con signo negativo.

Construya un algoritmo que reciba como entrada una altura  $h$ , desde la cual se deja caer un objeto, y entregue como salida la posición de la partícula, y el tiempo transcurrido en cada segundo desde que es soltado hasta que llega al suelo.

Indicación. Considere posición inicial  $h$ , y posición final 0 (suelo)

```
Inicio
Entrada: h
```

```

xt ← h
t ← 0
g ← 10
Mientras (h > 0)
{
    Salida: "tiempo", t, "altura", xt
    t ← t+1
    xt ← h - ½ * g * t * t
}
Fin

```

### **Problema: máquina de sencillar**

Se tiene una máquina que permite de sencillar dinero. La máquina está diseñada para que, dado un monto cualquiera, indique cuantos billetes de 1000 y monedas de 100, 10 y 1 debe entregar, a cambio de ese monto. Una característica de esta máquina es que siempre intentará dar la menor cantidad de monedas.

Construya el algoritmo que permita que la máquina funcione de la forma descrita. Es decir, un algoritmo que reciba como entrada un monto de dinero, entero, positivo y entregue como salida la cantidad de billetes y de monedas de las distintas denominaciones que equivalen a dicho monto.

```

Inicio
Entrada: monto
b1000 ← 0
m100 ← 0
m10 ← 0
m1 ← 0
Mientras (monto >= 1000)
{
    b1000 ← b1000 +1
    monto ← monto - 1000
}
Salida: "de 1000:", b1000
Mientras (monto >= 100)
{
    m100 ← m100 +1
    monto ← monto - 100
}
Salida: "de 100:", m100
Mientras (monto >= 10)
{
    m10 ← m10 +1
    monto ← monto - 10
}
Salida: "de 10:", m10
Salida: "de 1:", monto
Fin

```

### **Otra solución:**

```

Inicio
Entrada: monto
total ← 0
denominación ← 1000

```

```
Mientras (monto >= 10)
{
  Mientras (monto >= denominacion)
  {
    total ← total + 1
    monto ← monto - denominacion
  }
  Salida: "de", denominacion , total
  total ← 0
  denominacion ← denominacion / 10
}
Salida: "de 1:", monto
Fin
```