

# Programación dinámica

Pedro O. Pérez M., PhD.

Análisis y diseño de algoritmos avanzados  
Tecnológico de Monterrey

*pperezm@tec.mx*

08-2022

## ① Programación dinámica

- Introducción

- Forma general

- Algunos ejemplos

  - Secuencia de Fibonacci

  - Secuencia de suma máxima

  - Cambio de monedas

  - Números feos

  - Cuenta el número de formas posibles

La programación dinámica, al igual que Divide y conquista, resuelve problemas combinando soluciones a subproblemas; pero a diferencia de esta, se aplica cuando los subproblemas se solapan, es decir, cuando comparten problemas más pequeños. Aquí la técnica cobra importancia, ya que calcula cada subproblema una sola vez; esto es, parte del principio de no calcular dos veces la misma información. Por tanto, utiliza estructuras de almacenamiento como vectores, tablas, arreglos, archivos, con el fin de almacenar los resultados parciales a medida que se resuelven los subcasos que contribuyen a la solución definitiva.

- Es una técnica ascendente que, normalmente, empieza por los subcasos más pequeños y más sencillos. Combinando sus soluciones, obtenemos las respuestas para los subcasos cada vez más grandes, hasta que llegamos a la solución del problema original.
- Se aplica muy bien a problemas de optimización. El mayor número de aplicaciones se encuentran en problemas que requieren maximización o minimización, ya que se pueden hallar múltiples soluciones y así evaluar para determinar cuál es la óptima.

La forma general de las soluciones desarrolladas mediante programación dinámica requiere los siguientes pasos:

- 1 Plantear la solución, mediante una serie de decisiones que garanticen que será óptima, es decir, que tendrá la estructura de una solución óptima.
- 2 Encontrar una solución recursiva de la definición.
- 3 Calcular la solución teniendo en cuenta una tabla en la que se almacenen soluciones a problemas parciales para su reutilización, y así evitar un nuevo cálculo.
- 4 Encontrar la solución óptima utilizando la información previamente calcular y almacenada en las tablas.

*Principio de optimalidad de Bellman:* Cualquier subsecuencia de decisiones de una secuencia óptima de decisiones que resuelve un problema también debe ser óptima respecto al subproblema que resuelve.

---

## Procedure 1 FIBONACCI

---

Input:  $n : \text{Integer}$

if  $n < 1$  then

    return  $-1$

else if  $n = 1$  or  $n = 2$  then

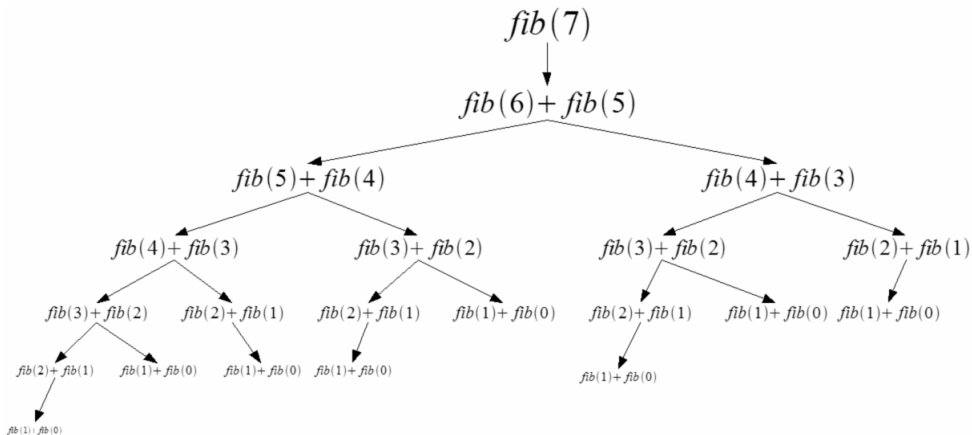
    return  $1$

else

    return  $FIBONACCI(n - 1) + FIBONACCI(n - 2)$

end if

---





---

## Procedure 2 FIBONACCI\_WITH\_MEMORY1

---

Input:  $n$  : Integer,  $A$  : Array

if  $n < 1$  then

return  $-1$

else if  $n = 1$  or  $n = 2$  then

return  $1$

else if  $A[n] \neq -1$  then

return  $A[n]$

else

$A[n] \leftarrow FIBONACCI(n - 1) + FIBONACCI(n - 2)$

return  $A[n]$

end if

---

---

### Procedure 3 FIBONACCI\_WITH\_MEMORY2

---

Input:  $n$  : Integer,  $A$  : Array

if  $n < 1$  then  
    return -1

else

$A[1] \leftarrow 1$

$A[2] \leftarrow 1$

    for  $i \leftarrow 3$  to  $n$  do

$A[i] \leftarrow A[i - 1] + A[i - 2]$

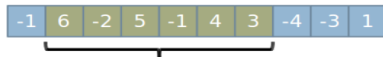
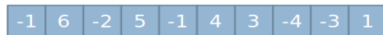
    end for

end if

---

# Secuencia de suma máxima

Dado un arreglo de  $n$  números enteros positivos y negativos, encontrar los  $i$  elementos del arreglo cuya suma se la máxima posible.



La secuencia máxima se encuentra comprendida entre la posición 1 y 6. La suma máxima es 15.

---

## Procedure 4 MAX\_SUM( $A$ :Array)

---

**Input:**  $n$  : Integer,  $A$  : Array

$sum \leftarrow 0$

$ans \leftarrow 0$

**for**  $i \leftarrow 1$  to  $A.length$  **do**

$sum \leftarrow sum + A[i]$

$ans \leftarrow MAX(ans, sum)$

**if**  $sum < 0$  **then**

$sum \leftarrow 0$

**end if**

**end for**

**return**  $ans$

---

Dado un sistema monetario  $S$  con  $N$  monedas de diferentes denominaciones y una cantidad de cambio  $C$ , calcular el menor número de monedas del sistema monetario  $S$  equivalente a  $C$ .

Ejemplos:

- **Input** :  $s[] = 1, 3, 4$   $c = 6$

**Output** : 2

**Explanation** : The change will be  $( 3 + 3 ) = 2$

$$C[j] = \begin{cases} \infty & \text{if } j < 0, \\ 0 & \text{if } j = 0, \\ 1 + \min_{1 \leq i \leq k} \{C[j - d_i]\} & \text{if } j \geq 1 \end{cases}$$

---

## Procedure 5 COIN\_CHANGE

---

**Input:**  $S$  : Array,  $C$  : Integer

$M$  : Matrix[0... $S.length$ ][0.. $C$ ]

$n \leftarrow S.length$

**for**  $j \leftarrow 1$  to  $n$  **do**

$M[0][j] \leftarrow 1$

**end for**

**for**  $i \leftarrow 1$  to  $n$  **do**

**for**  $j \leftarrow 1$  to  $C$  **do**

        /\* Can i use currency of this denomination to this change (j)? \*/

**if**  $j < S[i]$  **then**

$M[i][j] \leftarrow M[i-1][j]$

**else**

$M[i][j] \leftarrow \text{MIN}(1 + M[i-1][j - S[i]], M[i-1][j])$

**end if**

**end for**

**end for**

**return**  $M[n][C]$

---

Los números feos son números cuyos únicos factores primos son 2, 3 o 5 (o combinación de ellos). La secuencia 1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, ... muestra los primeros 11 números feos. Por convención, se incluye 1.



Como vemos en la diapositiva anterior, la secuencia de los primeros números feos es 1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, ... Es más fácil de determinar la forma en que se genera esta secuencia si la dividimos entre sus tres factores:

- $1 \times 2, 2 \times 2, 3 \times 2, 4 \times 2, 5 \times 2 \dots$
- $1 \times 3, 2 \times 3, 3 \times 3, 4 \times 3, 5 \times 3 \dots$
- $1 \times 5, 2 \times 5, 3 \times 5, 4 \times 5, 5 \times 5 \dots$

¿Cuál se imprime primero?

---

## Procedure 6 UGLY\_NUMBER

---

**Input:**  $n$  : *Integer*

$A$  : *Array*[ $n$ ]

$A[1] = 1$

$i_2 \leftarrow 1$

$i_3 \leftarrow 1$

$i_5 \leftarrow 1$

$ugly\_number \leftarrow 0$

**for**  $i \leftarrow 2$  **to**  $n$  **do**

    next slide

**end for**

**return**  $ugly\_number$

---

$$ugly\_number = MIN((A[i2] * 2), (A[i3] * 3), (A[i5] * 5))$$
$$A[i] \leftarrow ugly\_number$$

```
if ugly_number == (A[i2] * 2) then
```

$$i2 \leftarrow i2 + 1$$

end if

```
if ugly number == (A[i3] * 3) then
```

$$i3 \leftarrow i3 + 1$$

end if

```
if ugly number == (A[i5] * 5) then
```

$$i5 \leftarrow i5 + 1$$

end if

# Cuenta el número de formas posibles

Queremos dar un cambio de  $C$  pesos y tenemos un suministro infinito de monedas de valor  $S = [S_1, S_2, \dots, S_n]$  pesos, ¿de cuántas maneras podemos dar el cambio? Por ejemplo para  $C = 4$ ,  $S = [1, 2, 3]$ , existen cuatro formas de dar el cambio:  $[1, 1, 1, 1]$ ,  $[1, 1, 2]$ ,  $[2, 2]$ ,  $[1, 3]$ . Para  $C = 10$ ,  $S = [2, 5, 3, 6]$ , existen cinco soluciones:  $[2, 2, 2, 2, 2]$ ,  $[2, 2, 3, 3]$ ,  $[2, 2, 6]$ ,  $[2, 3, 5]$ ,  $[5, 5]$ .

$$COUNT(type, value) = \begin{cases} 0, & \text{si } value < 0, \\ 1, & \text{si } value = 0, \\ COUNT(type + 1, value) \\ + COUNT(type, value - coins[type]), & \text{si } value > type. \end{cases}$$

---

## Procedure 7 COUNT

---

**Input:**  $S$  : Array,  $c$  : Integer

$table$  : Matrix[0.. $c$ ][ $S.length$ ]

**for**  $j \leftarrow 1$  to  $S.length$  **do**

$table[0][j] \leftarrow 1$

**end for**

NEXT SLIDE

**return**  $table[c][S.length]$

---

---

```
for  $i \leftarrow 2$  to  $c$  do
  for  $j \leftarrow 1$  to  $S.length$  do
     $x \leftarrow 0$ 
    if  $i - S[j] \geq 0$  then
       $x \leftarrow table[i - S[j]][j]$ 
    end if
     $y \leftarrow 0$ 
    if  $j \geq 1$  then
       $x \leftarrow table[i][j - 1]$ 
    end if
     $table[i][j] \leftarrow x + y$ 
  end for
end for
```

---