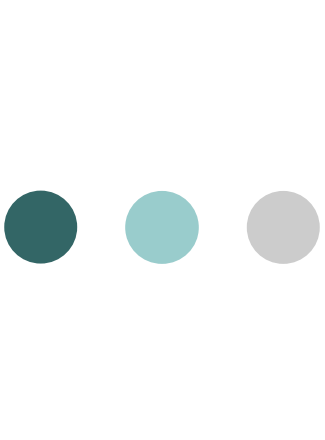


Programación Dinámica

Taller de Programación

Jhonny Felípez Andrade

jrfelizamigo@gmail.com

Contenido

- Consideraciones.
- Números Fibonacci.
- Coeficientes Binomiales.
- Cambio de Monedas

CONSIDERACIONES

Pregunta

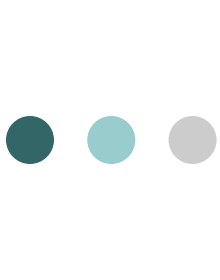
¿Qué es la Programación Dinámica?

Respuesta

- Es un padigma de resolución de problemas que reduce el tiempo de ejecución de un algoritmo.
- Hace uso de las soluciones óptimas de los subproblemas para obtener la solución óptima del problema.
- Es similar al backtracking pero más eficiente.

Programación Dinámica

- Se utiliza principalmente para resolver problemas de optimización y de conteo.
  - “Minimice esto …”
  - “Cuente las maneras de hacerlo…”
- La mayoría de los problemas de PD, solo pide el valor más óptimo.



NÚMEROS DE FIBONACCI

Números de Fibonacci

Tarea: Encuentre un número de la secuencia de Fibonacci

0,1,1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, . . .

DE ARRIBA HACIA ABAJO

Pasos

1. Formule el problema en términos de versiones más pequeñas del problema (subproblemas recursivos).
2. Convierta lo anterior en una función recursiva.
3. Memorice el resultado de la función (guarde los resultados que se calcularon)

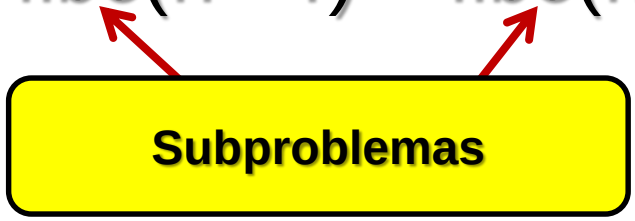
Paso 1

Formule el problema en términos de versiones más pequeñas del problema (subproblemas recursivos).

fibo(0) = 0

fibo(1) = 1

fibo(n) = fibo(n - 1) + fibo(n - 2)

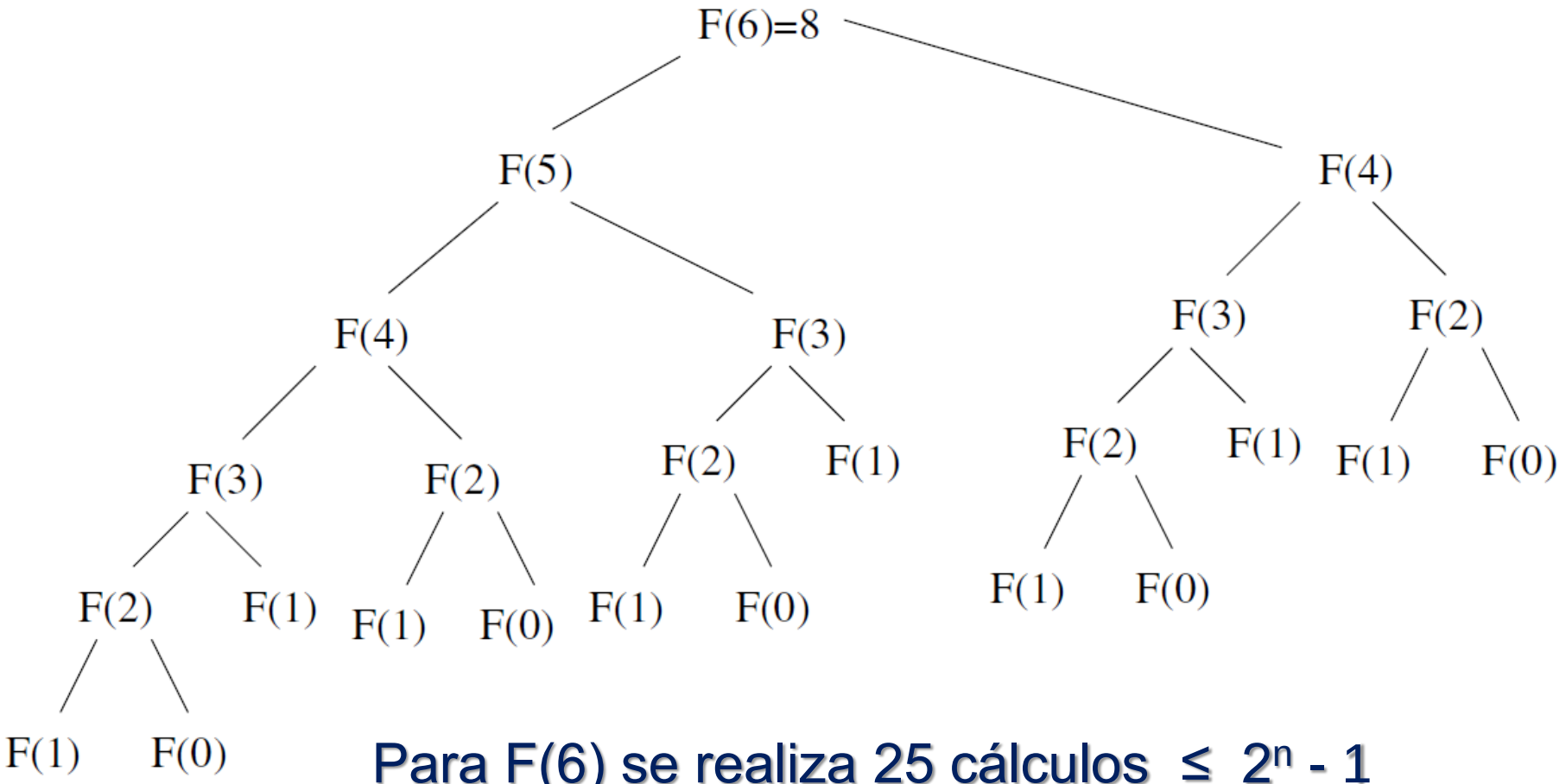


Paso 2

Convierta lo anterior en una función recursiva.

```
public long fibo(int n) {
   if (n == 0) return 0;
   if (n == 1) return 1;
   return fibo(n - 1) + fibo(n - 2);
}
```

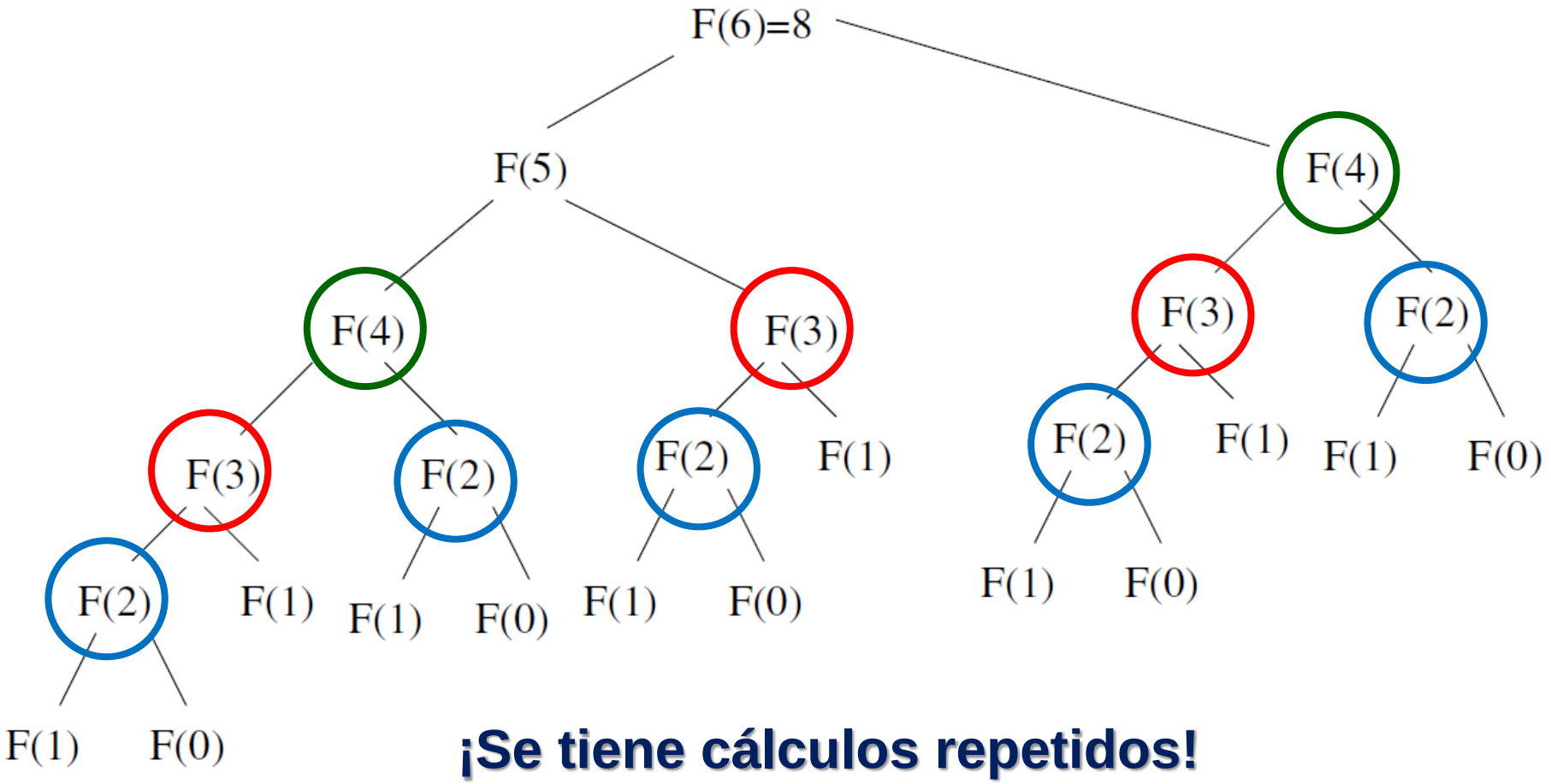
Árbol de recursión para F(6)



Para F(6) se realiza 25 cálculos $\leq 2^n - 1$

$O(2^n)$

Árbol de recursión para F(6)



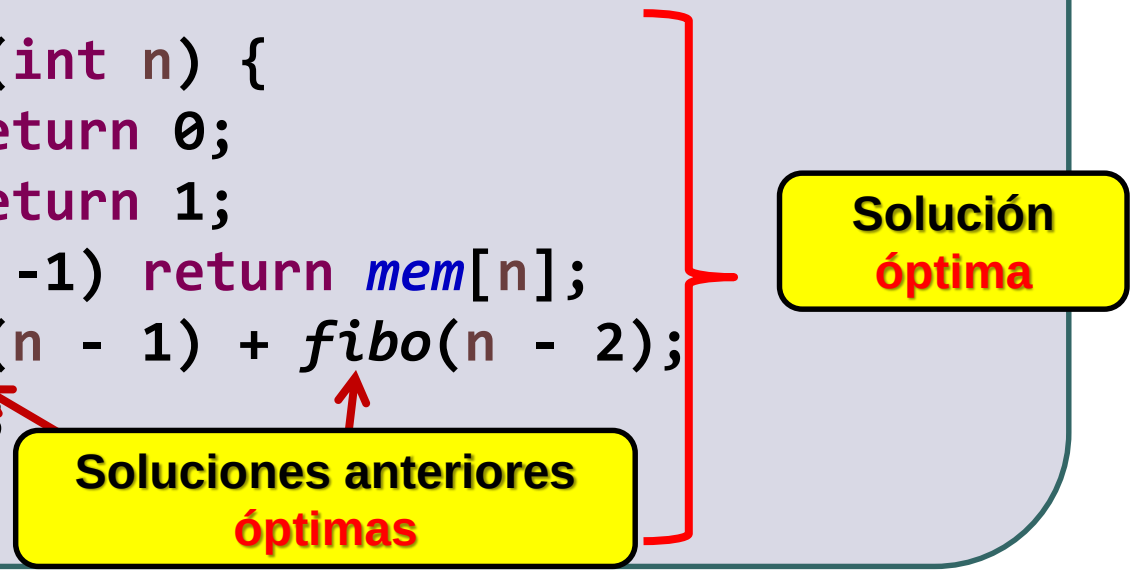
¡Se tiene cálculos repetidos!

Subproblemas sobrecargados

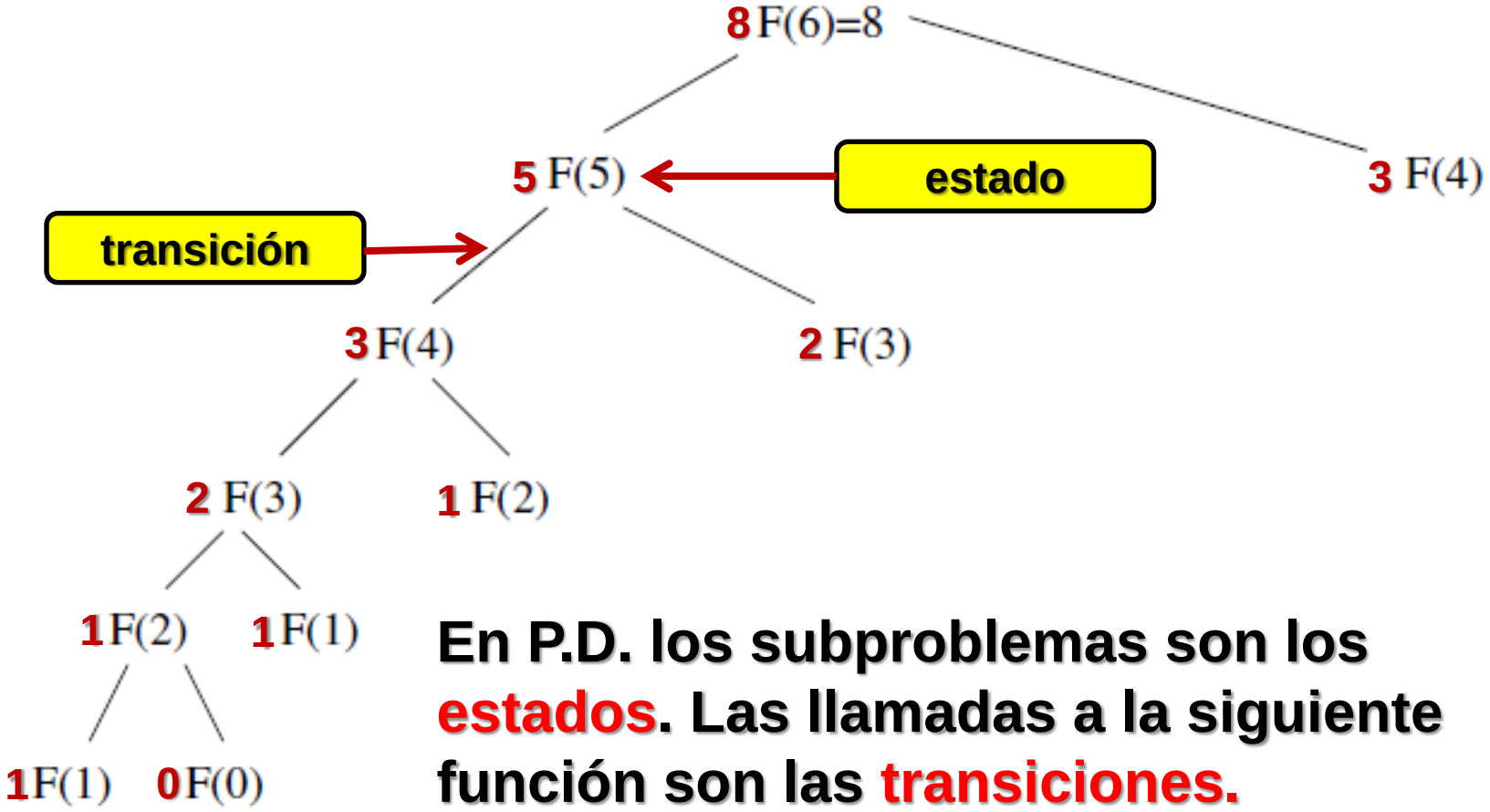
Paso 3

Memorice el resultado de la función (guarde los resultados que se calcularon)

```
long mem[] = new long[1000];
for (int i = 0; i< 1000; i++)
    mem[i] = -1;
public long fibo(int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
    if (mem[n] != -1) return mem[n];
    mem[n] = fibo(n - 1) + fibo(n - 2);
    return mem[n];
}
```



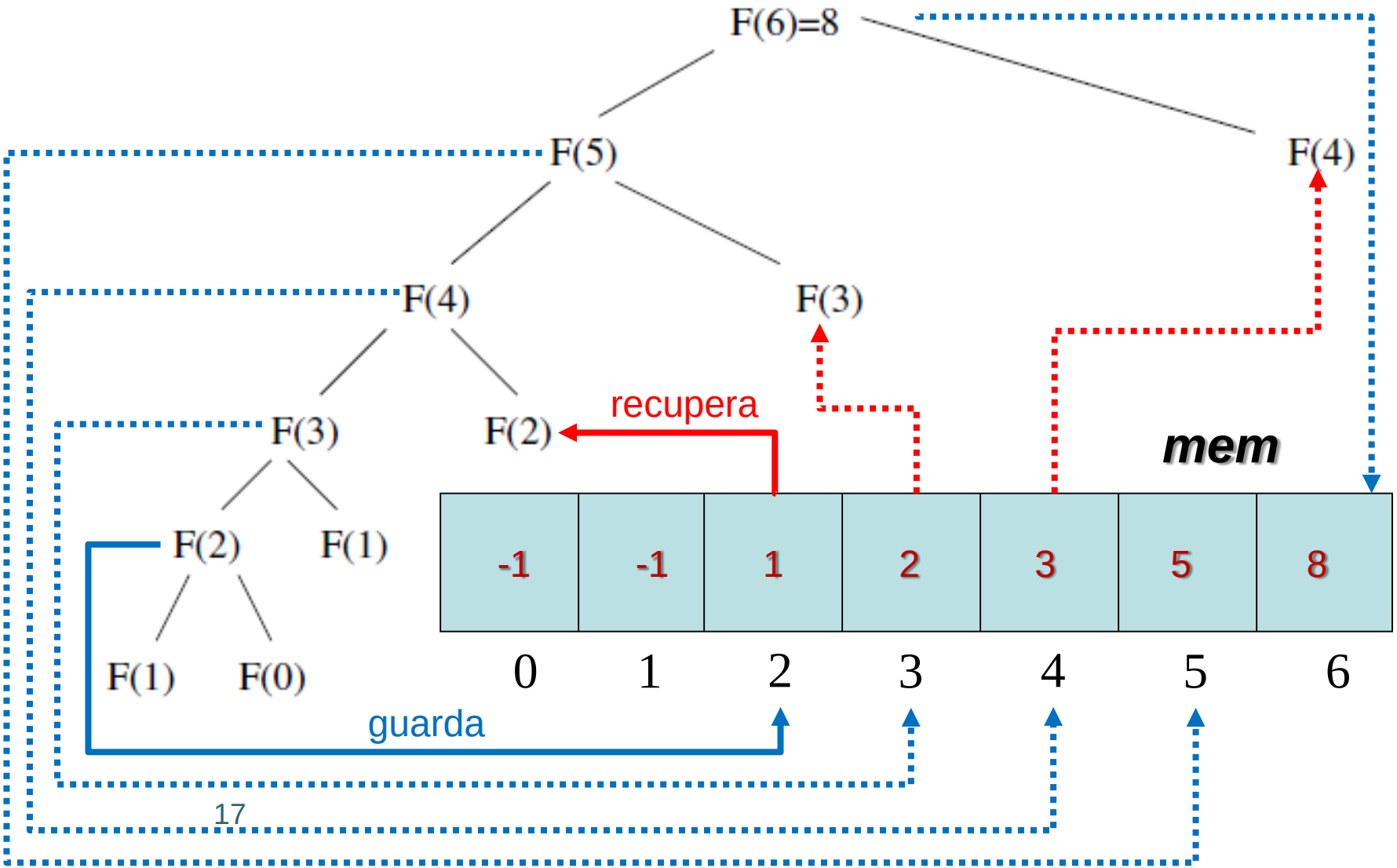
Árbol de recursión para F(6)



En P.D. los subproblemas son los estados. Las llamadas a la siguiente función son las transiciones.

Para F(6) se realiza 11 cálculos

Árbol de recursión para F(6)



¿Cuál será la complejidad?

- La función se llama para n+1 entradas posibles: 0, 1, 2,. . ., n.
- Cada entrada debe ser calculado y el resultado será guardado o recuperado de la memoria.
- Cada entrada debe ser calculado como máximo una vez

¿Cuál será la complejidad?

- La complejidad es O(n*f), donde f es la complejidad del tiempo para calcular una entrada.
- Si asumimos que las llamadas recursivas retornan directamente de la memoria entonces esto será O(1), o sea que f será constante.
- Por lo tanto la complejidad total del tiempo es **O(n)**

Ventajas

- Es una transformación natural de un programa recursivo de búsqueda completa.
- Calcula los sub-problemas sólo cuando es necesario (a veces esto es más rápido).

Desventaja

- Más lento si se vuelven a examinar muchos problemas secundarios debido a la sobrecarga de la función.
- Si hay M estados se requiere un tamaño de la tabla M. Para problemas que requieren mucha memoria esto es una limitación (excepto si se utiliza un árbol binario balanceado)

DE ABAJO HACIA ARRIBA

De Abajo hacia Arriba

- Se construye una tabla de abajo hacia arriba y se devuelve la última entrada de la tabla. Por ejemplo, para el mismo número de Fibonacci, primero calculamos fibo(0) luego fibo(1) luego fibo(2) luego fib (3) y así sucesivamente.

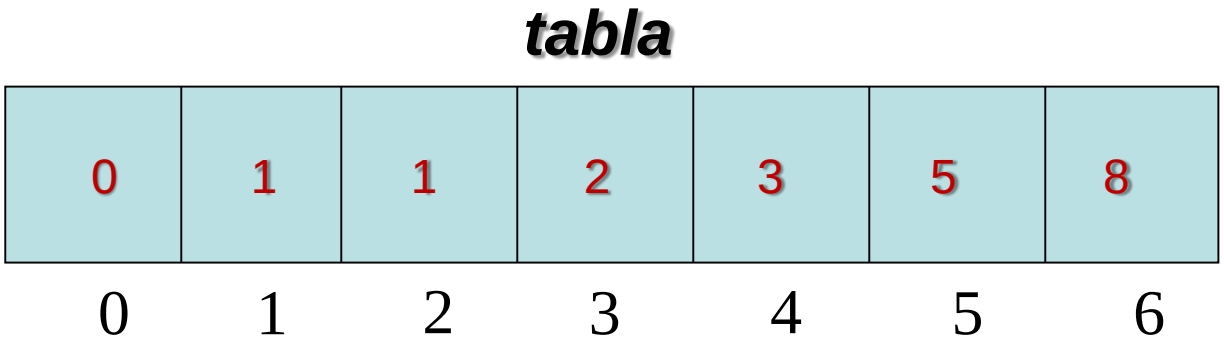
De Abajo hacia Arriba

- Así que literalmente, estamos construyendo las soluciones de los subproblemas de abajo hacia arriba.

De Abajo hacia Arriba

```
public static long fib(int n) {
    long tabla[] = new long[n + 1];
    tabla[0] = 0;
    tabla[1] = 1;
    for (int i = 2; i <= n; i++)
        tabla[i] =
            tabla[i - 1] + tabla[i - 2];
    return tabla[n];
}
```

Tabla



O(n)

Ventajas

- Mas rápido si se vuelven a examinar muchos sub-problemas, ya que no se tiene que sobrecargar de llamadas recursivas.

Desventaja

- Para los programadores que les guste programar utilizando la recursividad, este estilo puede no ser intuitivo.
- Si se tiene M estados, se visita y llena todos los M estados.

COEFICIENTES BINOMIALES

Coeficientes Binomiales

Tarea: Encuentre las combinaciones de n tomados de k elementos.

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}$$

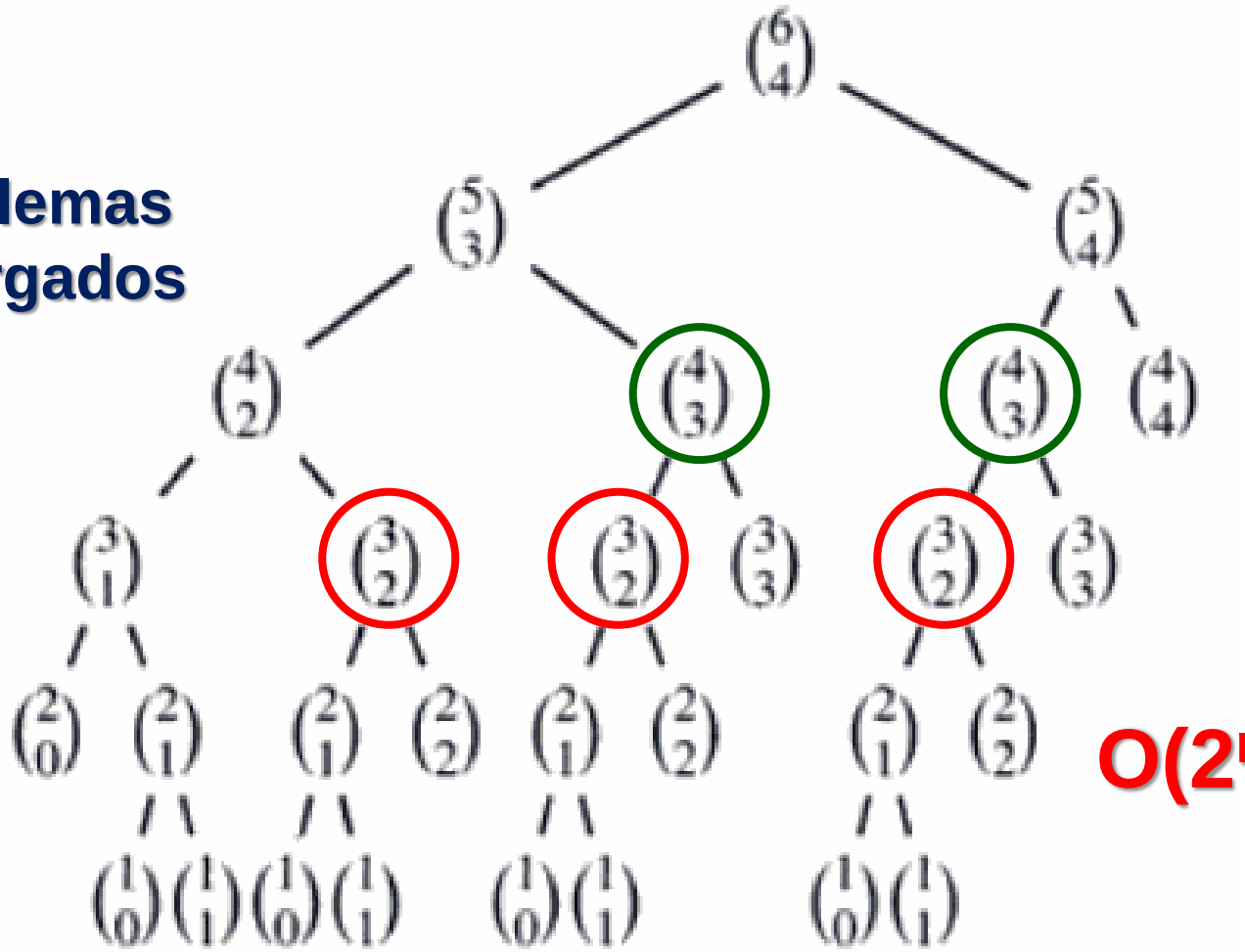
Fuerza Bruta

$$\binom{n}{k} = \begin{cases} 1 & si\ k = 0\ o\ k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & si\ 0 < k < n \\ 0 & en\ otro\ caso \end{cases}$$

```
long cb(int n, int k) {
    if (k == 0 || k == n)
        return 1;
    else
        return cb(n-1, k-1) + cb(n-1, k);
}
```

Árbol de recursión para C(6,4)

Subproblemas sobrecargados



$O(2^n/\sqrt{n})$

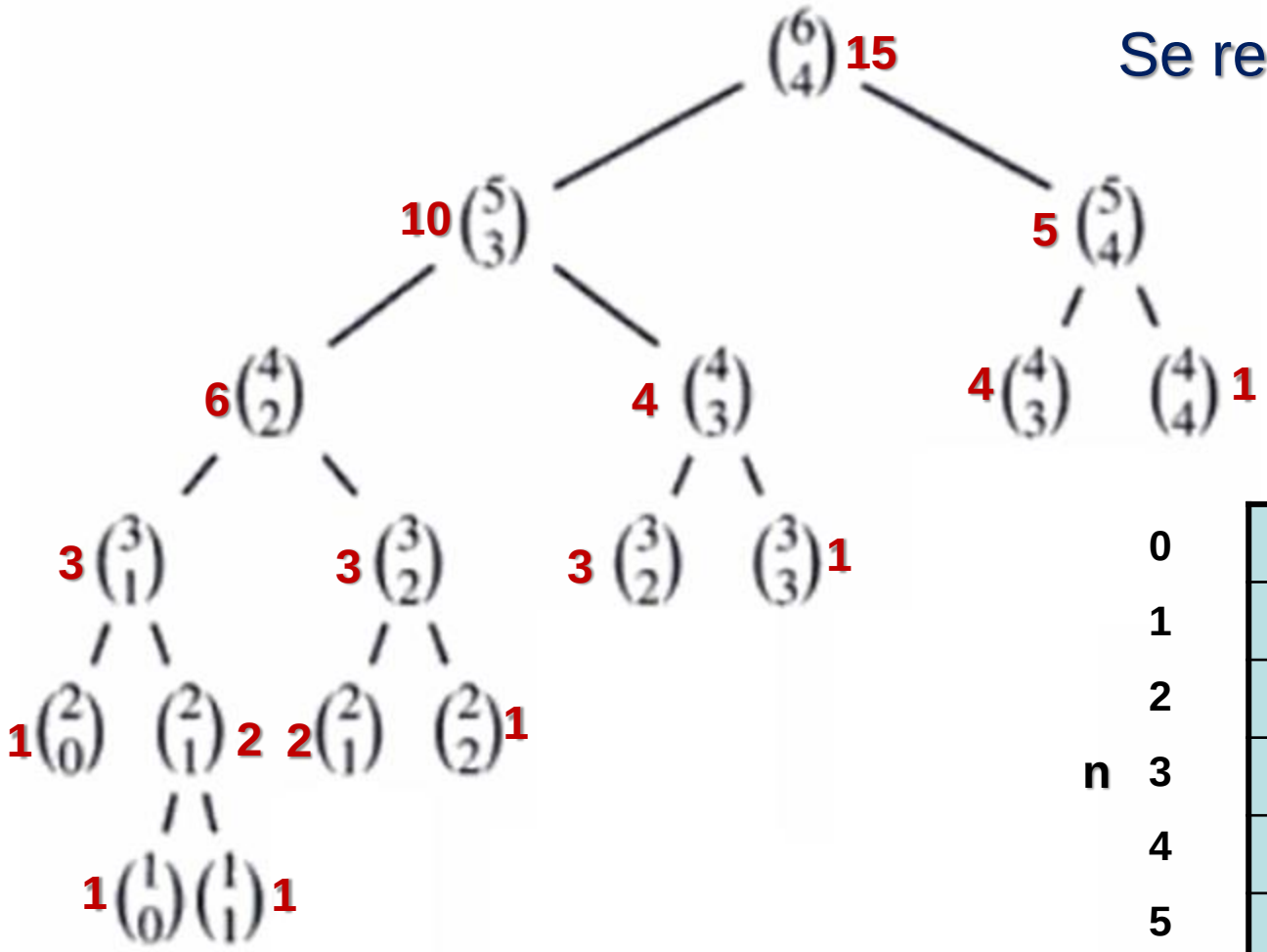
Se realiza 29 cálculos

DE ARRIBA HACIA ABAJO

P.D. Arriba hacia Abajo

```
long mem[][] = new long[MAX][MAX];
for (int i = 0; i < MAX; i++)
    for (int j = 0; j < MAX; j++)
        mem[i][j] = 0;
long cb(int n, int k) {
    if (k == 0 || n == k) return 1;
    if (mem[n][k] > 0)
        return mem[n][k];
    mem[n][k] = cb(n-1, k-1) + cb(n-1, k);
    return mem[n][k];
}
```

Árbol de recursión para C(6,4)



Se realiza 17 cálculos

O(nk)

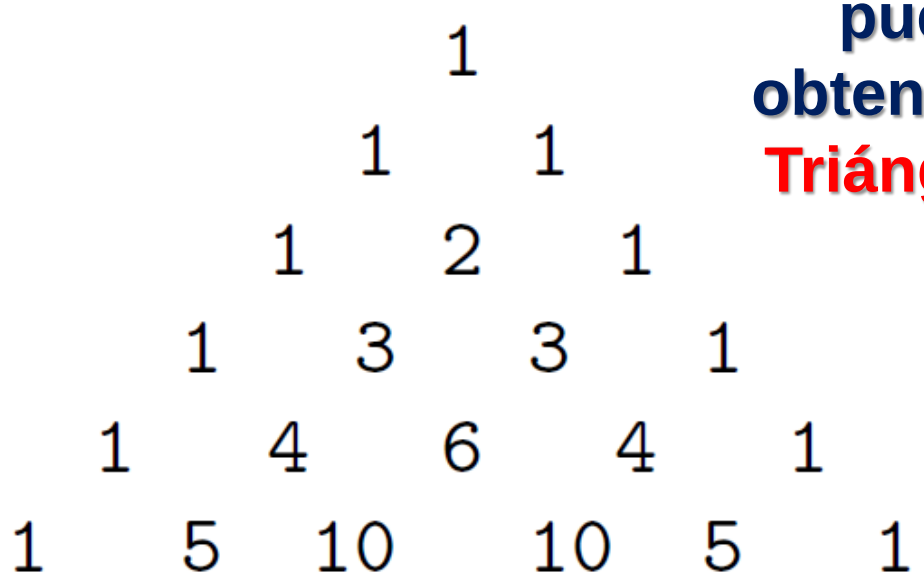
n \ k	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	0	2	0	0	0
3	0	3	3	0	0
4	0	0	6	4	0
5	0	0	0	10	5
6	0	0	0	0	15

DE ABAJO HACIA ARRIBA

Coeficientes Binomiales

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}$$

Las combinaciones se puede también obtener mediante el:
Triángulo de Pascal



Representación Matricial del Triángulo de Pascal

n \ k	0	1	2	3	4	5
0	1					
1	1	1				
2	1	2	1			
3	1	3	3	1		
4	1	4	6	4	1	
5	1	5	10	10	5	1

P.D. Abajo hacia Arriba

```
long coeficientes_binomiales(int n, int k) {
   long tabla[][] = new long[MAX][MAX];
   for (int i = 0; i <= n; i++)
      tabla[i][0] = 1;
   for (int j = 0; j <= n; j++)
      tabla[j][j] = 1;
   for (int i = 1; i <= n; i++)
      for (int j = 1; j < i; j++)
         tabla[i][j] =
            tabla[i-1][j-1] + tabla[i-1][j];
   return (tabla[n][k]);
}
```

Tabla

4

	0	1	2	3	4
0	1	0	0	0	0
1	1	1	0	0	0
2	1	2	1	0	0
3	1	3	3	1	0
4	1	4	6	4	1
5	1	5	10	10	5
6	1	6	15	20	15

6

tabla

O(nk)

CAMBIO DE MONEDAS

Problema

- **¿Cuál es el mínimo número de monedas que se tiene en 3 bolivianos?**

1 moneda de

1 moneda de

2 monedas

0 monedas de

Problema

- **Se tiene que dar N monedas de cambio, usando la menor cantidad entre monedas de denominaciones d_1, d_2, d_3, …, d_n. Se supone que se tiene una cantidad ilimitada de monedas de cada denominación**

Función de Recurrencia

$$c[i][j] = \begin{cases} 0 & si\ \ j = 0 \\ \infty & si\ i = 1\ y\ 0 < j < d_i \\ 1 + c[i][j - d_i] & si\ i = 1\ y\ \ j \geq d_i \\ c[i-1][j] & si\ i > 1\ y\ \ j < d_i \\ \min(c[i-1][j], 1 + c[i][j - d_i]) & si\ i > 1\ y\ \ j \geq d_i \end{cases}$$

Función de recurrencia

- **La solución optima se obtiene en C[n][total].**

Monedas	0	1	2	3
 $d_1 = 1$	0	0	0	0
 $d_1 = 2$	0	0	0	0

Caso 1

- **¿Cuál es el número de monedas que se tiene en cero bolivianos?**

0 monedas de 

$d_1 = 1$

0 monedas de 

$d_2 = 2$

`C[i][j] = 0` `j=0`

Caso 2

- **¿Cuál es el número de monedas que se tiene de 1 boliviano si solo se tiene monedas de 2 bolivianos?**

$d_1 = 2$ 

0 monedas de 

$C[i][j] = \infty \qquad i=1 \qquad 0<j<d_i$

Caso 3

- **¿Cuál es el número de monedas que se tiene en 6 bolivianos si solo se cuenta con monedas de 2 bolivianos?**

3 monedas de

$d_1 = 2$

2 monedas

$C[i][j] = 1+C[i][j-d_i] \quad i=1 \; j>=d_i$

Caso 4

- **¿Cuál es el número de monedas que se tiene de 1 boliviano si se tiene monedas de 1 y de 2 bolivianos?**

1 moneda de 

$d_1 = 1$

0 moneda de 

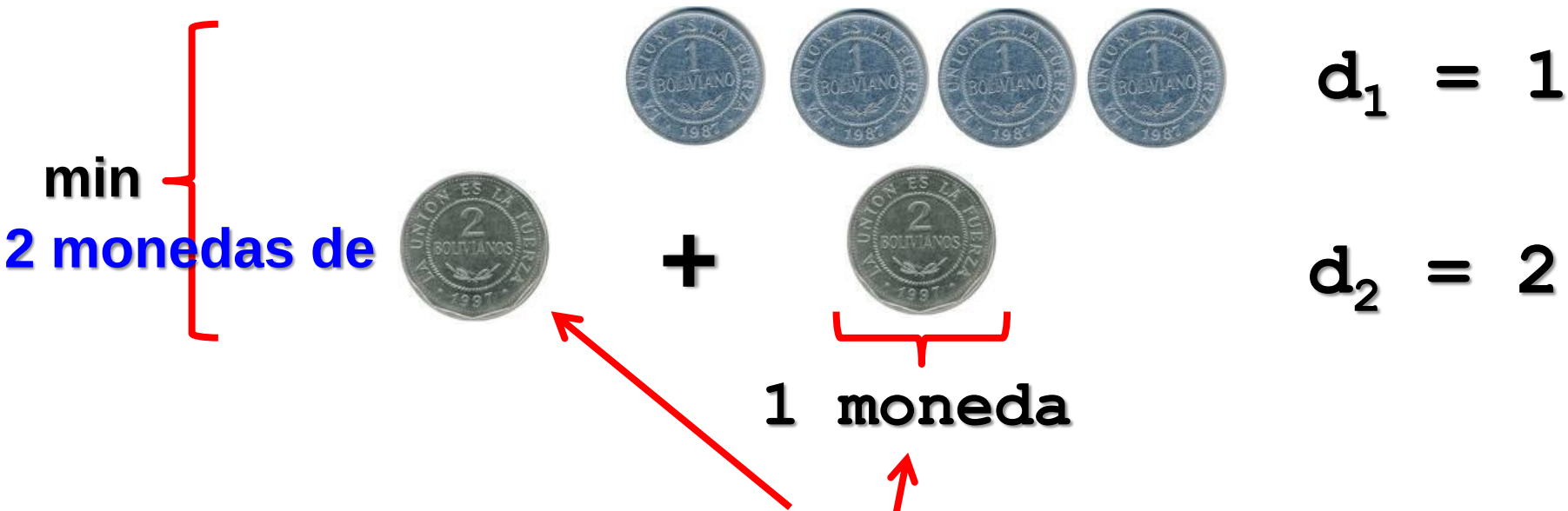
$d_2 = 2$

$C[i][j] = C[i-1][j]$ $\quad i>1 \quad j<d_i$

1 moneda

Caso 5

- **¿Cuál es el mínimo número de monedas que se tiene en 4 bolivianos si se tiene monedas de 1 y 2 bolivianos?**



$C[i][j] = \min(C[i-1][j],\ 1+C[i][j-d_i])\quad i>1\ j>=d_i$

Algoritmo

```
static int cambio(int d[], int n, int total) {
    int c[][] = new int[n + 1][total + 1];
    for (int i = 1; i <= n; i++) {
        c[i][0] = 0;
        for (int j = 1; j <= total; j++) {
            if (i == 1 && j < d[i])
                c[i][j] = 100;
            else if (i == 1 && j >= d[i])
                c[i][j] = 1 + c[i][j - d[i]];
            else if (i > 1 && j < d[i])
                c[i][j] = c[i - 1][j];
            else if (i > 1 && j >= d[i])
                c[i][j] = Math.min(c[i - 1][j], 1 + c[i][j - d[i]]);
        }
    }
    return c[n][total];
}
```

O(n * total)

Para total = 3 con $d_1 = 1$, $d_2 = 2$

Monedas	0	1	2	3
$d_1 = 1$ 	0			
$d_1 = 2$ 	0			

Ejercicio

¿Cómo se modificaría el programa si se dispone de una cantidad limitada de monedas de cada denominación?

Ejercicios

- 369
- 485
- 507
- 787
- 10755

Bibliografía

- The Algorithm Design Manual, Steven S. Skenia. Segunda edición, Springer, 2010.
- Competitive Programming 3rd Ed., Steven Halim & Felix Halim, 2013.

FIN