PROGRAMACIÓN DINÁMICA

Introducción

- Dentro de los temas dedicados a los esquemas algorítmicos, la **metodología Programación Dinámica** tiene un papel esencial, sobre todo por las posibilidades que ofrece a la hora de optimizar el orden de complejidad de los algoritmos resultantes.

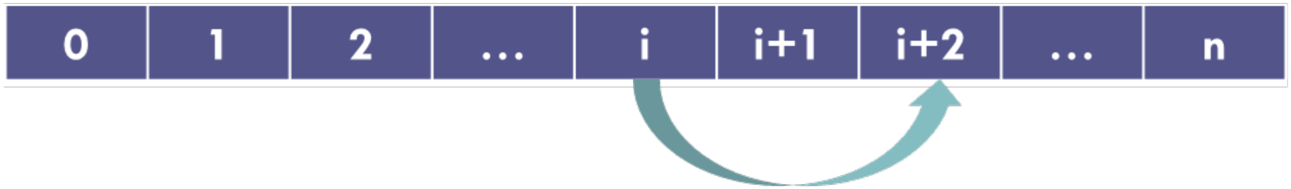
- A veces ocurre que a lo largo del **proceso recursivo** un mismo subproblema es resuelto varias veces, reduciendo la eficiencia, en ocasiones, hasta hacer inviable el algoritmo resultante.

- En cambio, la **Programación Dinámica** nos permite evitar esa posibilidad. Para conseguir este objetivo, se ha de cambiar la metodología de diseño utilizada en el Diseño Recursivo.
- Recordemos que en dicha metodología cada problema se resuelve a partir de la solución de otros problemas de menor tamaño. Se trata, pues, de un **enfoque descendente.**
- En cambio, **Programación Dinámica sigue el proceso inverso,** de tal forma que primero se plantea la solución de los problemas menores y a partir de ellos se ataca la solución de otros mayores.

EJEMPLO 1: RAYUELA

Dados unos cuadros alineados y numerados desde el 0, el juego consiste en ir saltando hasta el cuadro que ocupa la posición n, siendo n>0, permitiéndose para ello dos posibles movimientos:

ó



RAYUELA: ALGORITMO RECURSIVO

Esta solución se abordó en el tema "Diseño de Algoritmos Recursivos".-

```
{ n > 0 }
Funcion Rayuela (n:entero) retorna (f:entero)
    si n=1 ó n=2 entonces retorna n
      si no retorna Rayuela (n-1) + Rayuela (n-2)
    fsi
 ffuncion
```

RAYUELA: ALGORITMO RECURSIVO

Esta solución se abordó en el tema “Diseño de Algoritmos Recursivos”.-

{ n > 0 }

Funcion Rayuela (n:entero) retorna (f:entero)

si n=1 ó n=2 entonces retorna n

si no retorna Rayuela (n-1) + Rayuela (n-2)

fsi

ffuncion

$$T(n) = \begin{cases} c_1 & n \leq 2 \\ T(n-1) + T(n-2) + c_2 & n > 2 \end{cases}$$

RAYUELA: ALGORITMO RECURSIVO

Esta solución se abordó en el tema "Diseño de Algoritmos Recursivos".-

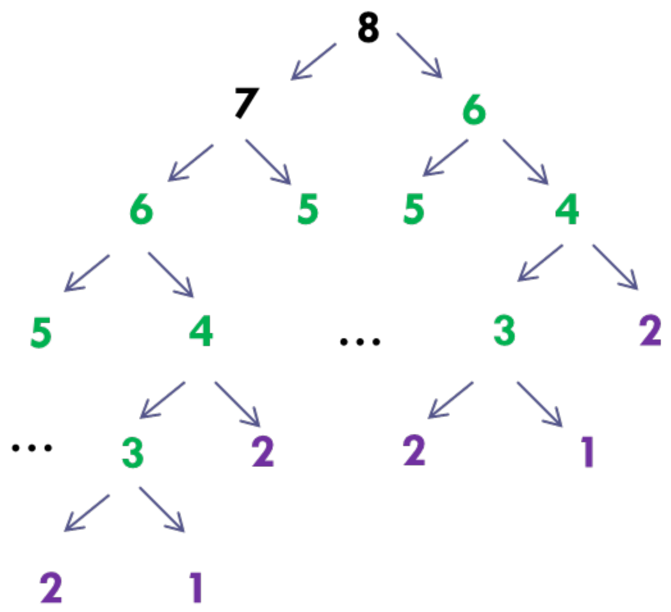
{ n > 0 }

```
Funcion Rayuela (n:entero) retorna (f:entero)
    si n=1 ó n=2 entonces retorna n
      si no retorna Rayuela (n-1) + Rayuela (n-2)
    fsi
 ffuncion
```

$$T(n) = \begin{cases} c_1 & n \leq 2 \\ T(n-1) + T(n-2) + c_2 & n > 2 \end{cases}$$

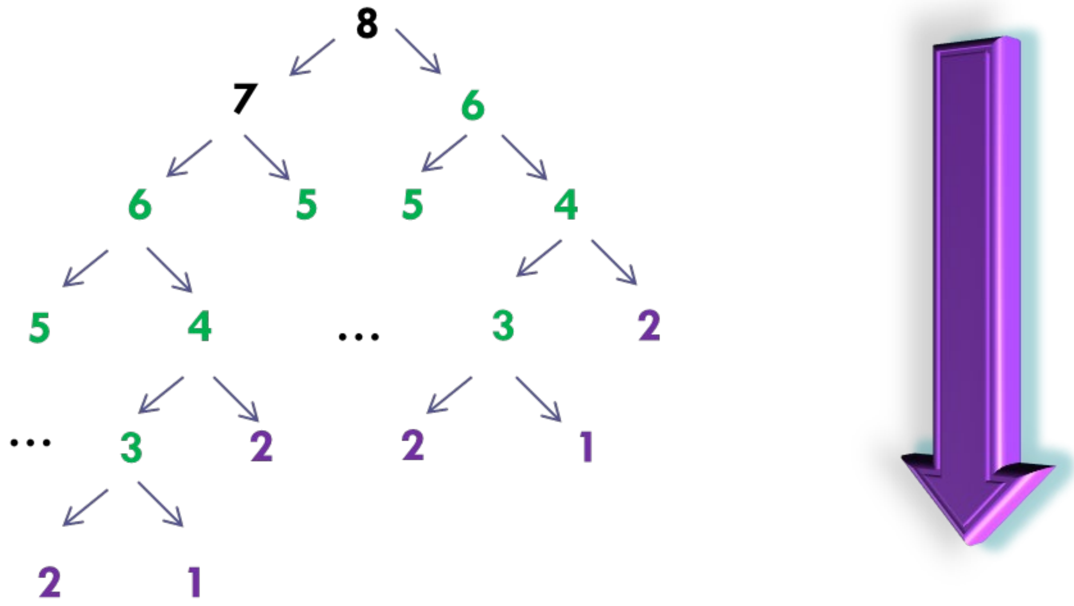
En consecuencia, $T(n) \in \Omega(2^{n/2})$ y $T(n) \in O(2^n)$

RAYUELA RECURSIVA: ÁRBOL DE LLAMADAS

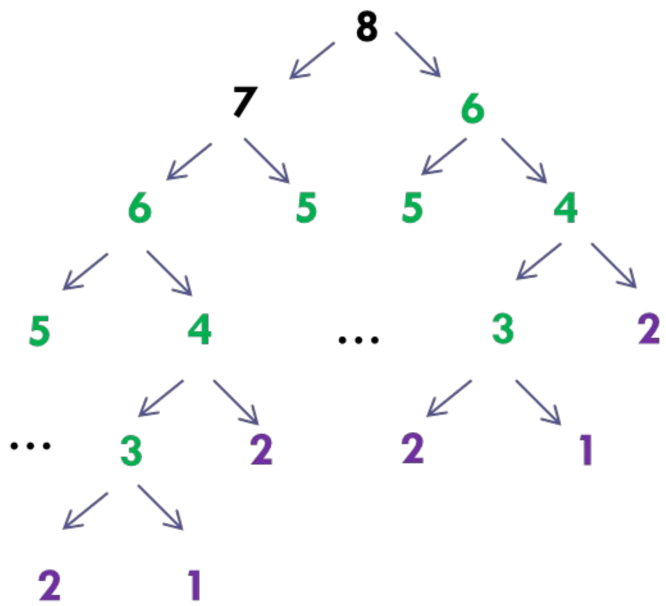


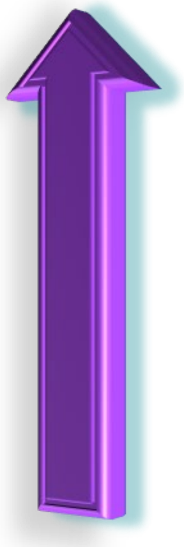
¡¡ REPETICIÓN DE SUBPROBLEMAS !!

RECURSIÓN: TÉCNICA DESCENDENTE



PROGRAMACIÓN DINÁMICA: TÉCNICA ASCENDENTE





Para la construcción del algoritmo se precisa:

- Elegir una estructura de almacenamiento adecuada
- Rellenar dicha estructura en sentido ascendente

RAYUELA: ESTRUCTURA DE ALMACENAMIENTO

{ n > 0 }

```
Función RAYUELA (n: entero) retorna (f:entero)
    si n = 1 ó n = 2 entonces retorna  n
    sino retorna RAYUELA (n-1) + RAYUELA (n-2)
    fsi
ffuncion
```

Estructura de Almacenamiento: **un vector V de dimensión n**

RAYUELA: ESTRUCTURA DE ALMACENAMIENTO

{ n > 0 }

```
Función RAYUELA (n: entero) retorna (f:entero)
    si n = 1 ó n = 2 entonces retorna  n
    sino retorna RAYUELA (n-1) + RAYUELA (n-2)
    fsi
ffuncion
```

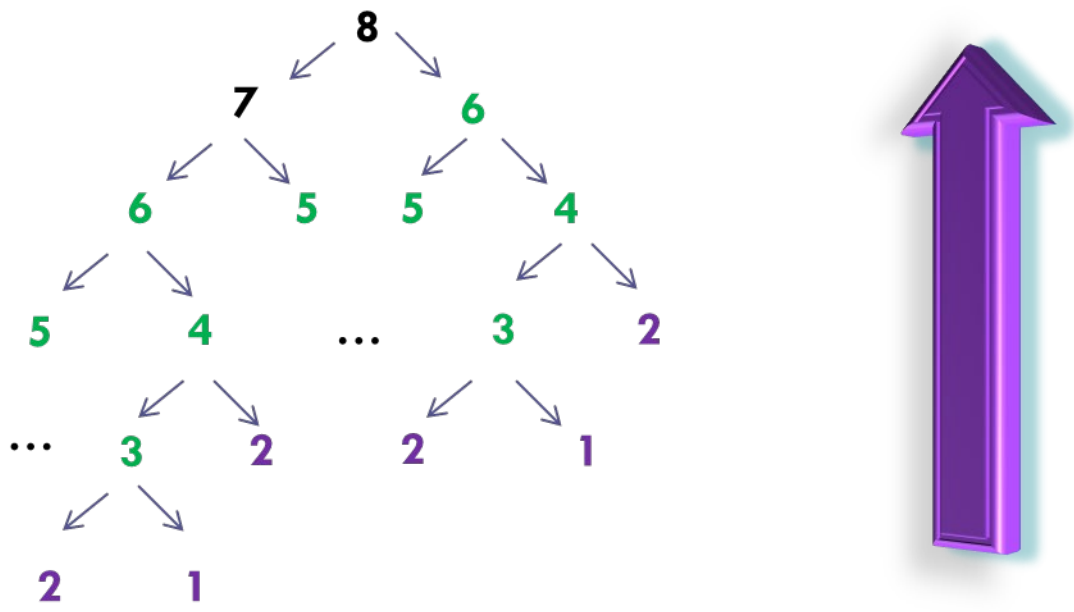
Estructura de Almacenamiento: **un vector V de dimensión n**

RAYUELA(1) → V[1]

...

RAYUELA(n) → V[n]

RAYUELA: RELLENADO DE LA ESTRUCTURA



Rellenado de la estructura ⟷ Dependencia de los subproblemas

RAYUELA: RELLENADO DE LA ESTRUCTURA

/* PROBLEMAS TRIVIALES */

V[1]=1

V[2]=2

/* RESTO DE PROBLEMAS EN ORDEN ASCENDENTE */

V[3]=V[2]+V[1]

V[4]=V[3]+V[2]

…

	1	2	3	4	…
V	1	2	3	5	…

RAYUELA: RELLENADO DE LA ESTRUCTURA

/* PROBLEMAS TRIVIALES */

V[1]=1

V[2]=2

/* RESTO DE PROBLEMAS EN ORDEN ASCENDENTE */

V[3]=V[2]+V[1]

V[4]=V[3]+V[2]

…

V[i]=V[i-1] + V[i-2]

	1	2	3	4		i-2	i-1	i
V	1	2	3	5	…	x	y	¿?

RAYUELA: RELLENADO DE LA ESTRUCTURA

/* PROBLEMAS TRIVIALES */

V[1]=1

V[2]=2

/* RESTO DE PROBLEMAS EN ORDEN ASCENDENTE */

V[3]=V[2]+V[1]

V[4]=V[3]+V[2]

…

V[i]=V[i-1]+V[i-2]

…

/* SOLUCIÓN */

retorna V[n]

RAYUELA: ALGORITMO


```
{ n > 0 }
Función RAYUELA_Programacion_Dinamica (n : entero) retorna (f : entero)
  var V[1..n] : vector de enteros, i : entero fvar

    V[1]=1
    V[2]=2

    para i=3 hasta n hacer
            V[i]=V[i-1]+V[i-2]
    fpara

retorna V[n]

ffuncion
```

RAYUELA: ALGORITMO

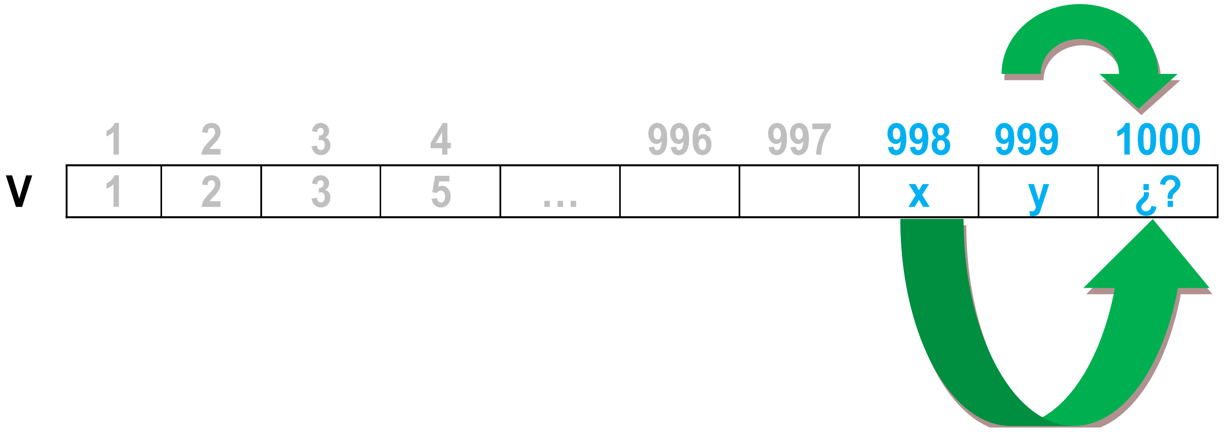
$\{ n > 0 \}$

```
Función RAYUELA_Programacion_Dinamica (n : entero) retorna (f : entero)
  var V[1..n] : vector de enteros, i : entero fvar

     V[1]=1
     V[2]=2

     para i=3 hasta n hacer
              V[i]=V[i-1]+V[i-2]
     fpara

retorna V[n]

ffuncion
```

Complejidad Temporal: $T(n) \in \theta(n)$

Complejidad Espacial: $T(n) \in \theta(n)$

RAYUELA: ¿MEJORA DEL ALGORITMO?



RAYUELA: ALGORITMO CON MEJORA EN COSTE ESPACIAL


```
{ n > 0 }
Función RAYUELA_Programacion_Dinámica_mejorado (n:entero) retorna (f:entero)
    var a, b : entero fvar
    a=1
    b=2
    para i = 3 hasta n hacer                /* ó para i=1 hasta n-2 hacer */
        b = a + b
        a = b - a
    fpara
    si n = 1 entonces retorna a sino retorna b fsi
ffuncion
```

RAYUELA: ALGORITMO CON MEJORA EN COSTE ESPACIAL

{ n > 0 }

```
Función RAYUELA_Programacion_Dinámica_mejorado (n:entero) retorna (f:entero)
    var a, b : entero fvar
    a=1
    b=2
    para i = 3 hasta n hacer                /* ó para i=1 hasta n-2 hacer */
        b = a + b
        a = b - a
    fpara
    si n = 1 entonces retorna a sino retorna b fsi
ffuncion
```

Complejidad Temporal: $T(n) \in \theta(n)$

Complejidad Espacial: $T(n) \in \theta(1)$

EJEMPLO 2: COMPETICIÓN INTERNACIONAL

Dos equipos A y B juegan partidas sucesivas entre sí, en las que no cabe el empate y en las que el resultado de cada una es independiente de los resultados habidos en las demás.

La competición continúa hasta que uno de los equipos consigue **n victorias**. Esto implica que el número máximo de partidas que pueden llegar a disputarse es 2n-1.

Partimos del supuesto de que las probabilidades de que A y B ganen una partida son conocidas y suman 1, al no existir la posibilidad de empate. Sea **p la probabilidad de A gane una partida** y por tanto, **1-p, la probabilidad de que gane B**.

Nuestro objetivo es **diseñar un algoritmo que nos permita calcular la probabilidad de que A gane la competición**.

Inicialmente, razonaremos de forma recursiva. Para ello planteamos una situación intermedia.

COMPETICIÓN INTERNACIONAL

Llamaremos Competicion(i, j) a la probabilidad de que el equipo A gane la competición cuando nos encontramos en un momento en el que:

al equipo A le faltan i victorias para ganar la competición y al equipo B le faltan j

Por tanto, $0 \leq i, j \leq n$ y el problema inicial es: Competición(n, n)

Supongamos que se juega una nueva partida. El problema (i, j) se transformará en uno de los subproblemas siguientes:

- (i-1, j) si esa partida la gana A (lo que sucede con probabilidad p) ó
- (i, j -1) si la gana B (que sucede con probabilidad 1-p).

En resumen,

$$\text{Competicion}(i, j) = p * \text{Competicion}(i-1, j) + (1-p) * \text{Competicion}(i, j-1)$$

COMPETICIÓN INTERNACIONAL

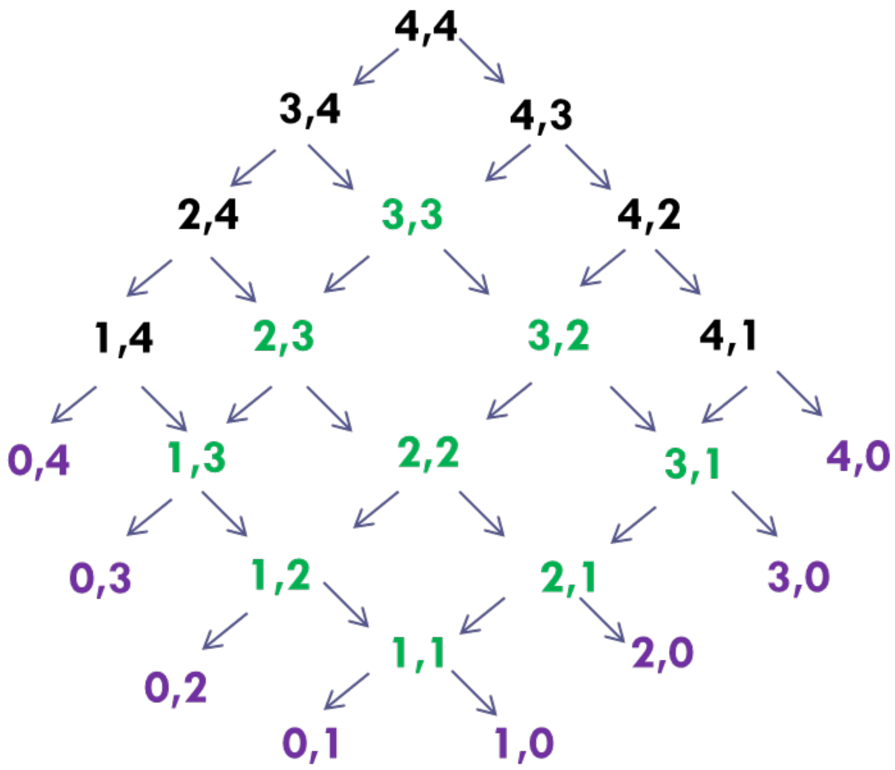
De acuerdo con el preorden establecido entre los subproblemas, los casos triviales corresponden a los problemas de la forma (0, j) y (i,0), cuyas soluciones respectivas son 1.00 y 0.00

El algoritmo recursivo es:

```
{ 0 ≤ i, j ≤ n}
Función Competicion ( i, j : entero; p : real ) retorna ( s : real )
 si i = 0 entonces retorna 1.00
  sino    si j = 0 entonces retorna 0.00
             sino retorna p * Competicion( i-1, j, p ) + (1-p) * Competicion( i, j-1, p)
          fsi
 fsi
ffuncion
```

La llamada inicial a la función sería: **Competicion(n, n, p)**

COMPETICIÓN INTERNACIONAL: ÁRBOL DE LLAMADAS



¡¡¡ Repetición de subproblemas !!!

COMPETICIÓN INTERNACIONAL: ESTRUCTURA DE ALMACENAMIENTO

{ $0 \leq i, j \leq n$}

```
Función Competicion ( i, j : entero; p : real ) retorna ( s : real )
  si i = 0 entonces retorna 1.0
   sino    si j = 0 entonces retorna 0.0
              sino retorna p * Competicion( i-1, j, p ) + (1-p) * Competicion( i, j-1, p )
            fsi
  fsi
ffuncion
```

Llamada inicial a la función: **Competicion(n, n, p)**

Estructura de Almacenamiento: **una matriz M de dimensión (n+1) filas x (n+1) columnas**

COMPETICIÓN INTERNACIONAL: ESTRUCTURA DE ALMACENAMIENTO

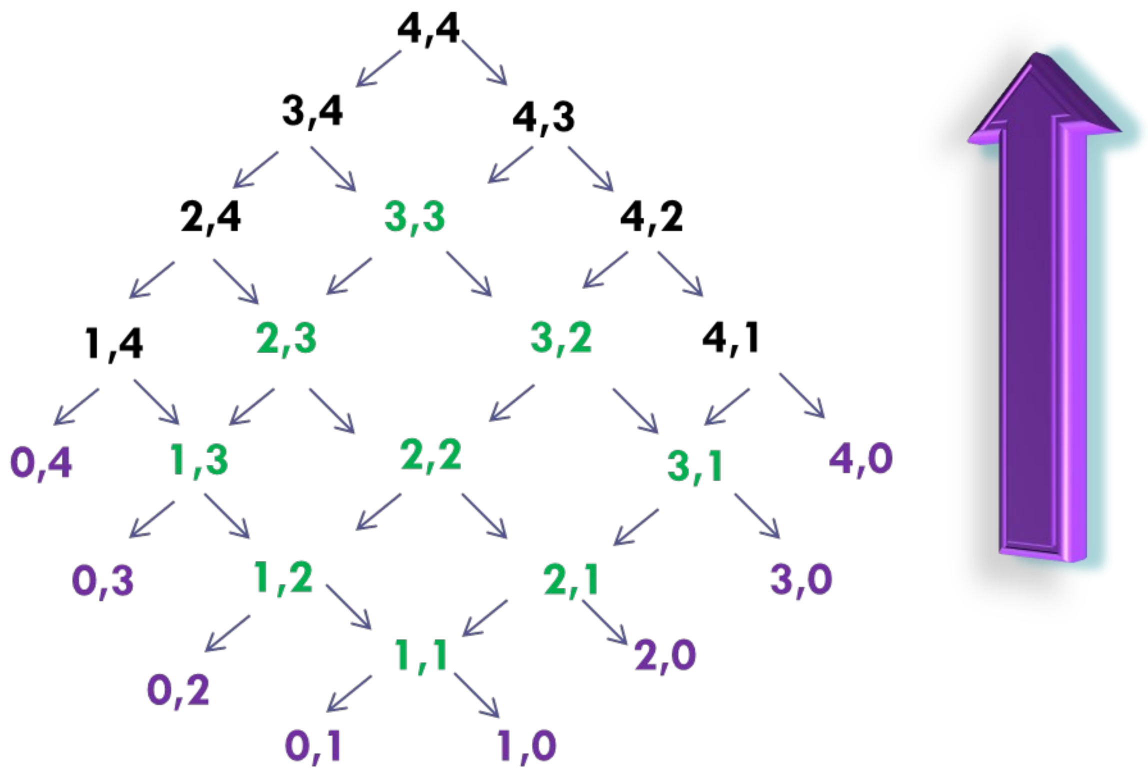
{ $0 \leq i, j \leq n$}

```
Función Competicion ( i, j : entero; p : real ) retorna ( s : real )
  si i = 0 entonces retorna 1.0
   sino    si j = 0 entonces retorna 0.0
               sino retorna p * Competicion( i-1, j, p ) + (1-p) * Competicion( i, j-1, p )
           fsi
  fsi
ffuncion
```

Llamada inicial a la función: **Competicion(n, n, p)**

Estructura de Almacenamiento: **una matriz M de dimensión (n+1) filas x (n+1) columnas**

de tal modo que **Competicion (i, j, p) $\rightarrow$ M[i][j]**



Rellenado de la estructura ⟺ Dependencia de los subproblemas

COMPETICIÓN INTERNACIONAL: RELLENADO DE LA ESTRUCTURA

/* PROBLEMAS TRIVIALES */

¿?

/* RESTO DE PROBLEMAS EN ORDEN ASCENDENTE */

¿?

/* SOLUCIÓN */

¿?

COMPETICIÓN INTERNACIONAL: INICIALIZACIÓN DE LA MATRIZ

Datos del problema: **$n = 4$ y $p = 0.45$**

Problemas Triviales: $i = 0$ ó $j = 0$

↓ i	j → 0	1	2	3	4
0		1.00	1.00	1.00	1.00
1	0.00				
2	0.00				
3	0.00				
4	0.00				

COMPETICIÓN INTERNACIONAL: RELLENADO DE LA MATRIZ

Datos del problema: $n = 4$ y $p = 0.45$

Resto de los problemas en sentido creciente → rellenado por filas o por columnas

↓ i	j → 0	1	2	3	4
0	M	1.00	1.00	1.00	1.00
1	0.00	0.45	0.70	0.83	0.91
2	0.00	0.20	0.43	0.61	0.74
3	0.00	0.09	0.24		
4	0.00				

M[3][2] = 0.45 * M[2][2] + (1-0.45) * M[3][1] = 0.24

COMPETICIÓN INTERNACIONAL: RESULTADO FINAL

Datos del problema: **n = 4 y p = 0.45**

Solución en M[4][4]

↓ i	j → 0	1	2	3	4
0	M	1.00	1.00	1.00	1.00
1	0.00	0.45	0.70	0.83	0.91
2	0.00	0.20	0.43	0.61	0.74
3	0.00	0.09	0.24	0.41	0.56
4	0.00	0.04	0.13	0.26	0.39

COMPETICIÓN INTERNACIONAL: ALGORITMO


```
Función Competición_Programacion_Dinamica ( n : entero; p : real ) retorna ( s : real )
var M[0..n][0..n]: matriz de reales; i, j : entero fvar
/* PROBLEMAS TRIVIALES */
para  i=1 hasta n hacer
      M[ i ][ 0 ] = 0.00
      M[ 0 ][ i ] = 1.00
fpara
/* RESTO DE PROBLEMAS EN ORDEN ASCENDENTE */
para  i = 1 hasta n hacer
   para  j = 1 hasta n hacer
        M[ i ][ j ] = p * M[ i - 1 ][ j ] + ( 1-p ) * M[ i ][ j - 1 ]
   fpara
fpara
/* SOLUCIÓN */
retorna M[n][n]
ffuncion
```

COMPETICIÓN INTERNACIONAL: ALGORITMO


```
Función Competición_Programacion_Dinamica ( n : entero; p : real ) retorna ( s : real )
var M[0..n][0..n]: matriz de reales; i, j : entero fvar
/* PROBLEMAS TRIVIALES */
para  i=1 hasta n hacer
      M[ i ][ 0 ] = 0.00
      M[ 0 ][ i ] = 1.00
fpara
/* RESTO DE PROBLEMAS EN ORDEN ASCENDENTE */
para  i = 1 hasta n hacer
   para  j = 1 hasta n hacer
        M[ i ][ j ] = p * M[ i - 1 ][ j ] + ( 1-p ) * M[ i ][ j - 1 ]
   fpara
fpara
/* SOLUCIÓN */
retorna M[n][n]
ffuncion
```

Complejidad Temporal: $T(n) \in \theta(n^2)$

Complejidad Espacial: $T(n) \in \theta(n^2)$

COMPETICIÓN INTERNACIONAL: ¿MEJORA DEL ALGORITMO?

A la hora de rellenar la fila i-ésima sólo se precisa la información almacenada en la misma fila i y en la fila i-1de la matriz.

	0	1	...	n-1	n
0	M	1.00	1.00	1.00	1.00
1	0.00				
...	0.00				
i-1	0.00				
i	0.00				

COMPETICIÓN INTERNACIONAL: ¿MEJORA DEL ALGORITMO?

A la hora de rellenar la fila i-ésima sólo se precisa la información almacenada en la misma fila i y en la fila i-1de la matriz.

	0	1	…	n-1	n
0	M	1.00	1.00	1.00	1.00
1	0.00				
…	0.00				
i-1	0.00				
i	0.00				

Por lo que podremos hacer uso únicamente de un vector de (n+1) elementos

COMPETICIÓN INTERNACIONAL: INICIALIZACIÓN DEL VECTOR V

$\downarrow$ i	j $\rightarrow$ 0	1	2	3	4
0	M	1.00	1.00	1.00	1.00
1	0.00	0.45	0.70	0.83	0.91
2	0.00	0.20	0.43	0.61	0.74
3	0.00	0.09	0.24	0.41	0.56
4	0.00	0.04	0.13	0.26	0.39

	0	1	2	3	4
V	0.00	1.00	1.00	1.00	1.00

COMPETICIÓN INTERNACIONAL: RELLENADO DE V

$\downarrow$ i	j $\rightarrow$ 0	1	2	3	4
0	M	1.00	1.00	1.00	1.00
1	0.00	0.45	0.70	0.83	0.91
2	0.00	0.20	0.43	0.61	0.74
3	0.00	0.09	0.24	0.41	0.56
4	0.00	0.04	0.13	0.26	0.39

V[1]=0.45*V[1]+0.55*V[0]=0.45*1.00+0.55*0.00=0.45

	0	**1**	**2**	**3**	**4**
V	0.00	~~1.00~~ 0.45	1.00	1.00	1.00

COMPETICIÓN INTERNACIONAL: RELLENADO DE V

↓ i	j → 0	1	2	3	4
0	M	1.00	1.00	1.00	1.00
1	0.00	0.45	0.70	0.83	0.91
2	0.00	0.20	0.43	0.61	0.74
3	0.00	0.09	0.24	0.41	0.56
4	0.00	0.04	0.13	0.26	0.39

V[2]= 0.45*V[2]+0.55*V[1]=0.45*1.00+0.55*0.45=0.70

…

	0	1	2	3	4
V	0.00	0.45	~~1.00~~ 0.70	1.00	1.00

COMPETICIÓN INTERNACIONAL: RELLENADO DE V

↓ i	j → 0	1	2	3	4
0	M	1.00	1.00	1.00	1.00
1	0.00	0.45	0.70	0.83	0.91
2	0.00	0.20	0.43	0.61	0.74
3	0.00	0.09	0.24	0.41	0.56
4	0.00	0.04	0.13	0.26	0.39

En este punto, el contenido del vector V es equivalente a la fila 1 de la matriz

	0	**1**	**2**	**3**	**4**
V	0.00	0.45	0.70	0.83	0.91

COMPETICIÓN INTERNACIONAL: ALGORITMO MEJORA COSTE ESPACIAL


```
Función Competición_Programacion_Dinamica ( n : entero; p : real ) retorna ( s : real )
var V[0..n]: vector de reales; i, j : entero fvar
/* PROBLEMAS TRIVIALES */
para  i = 1 hasta n hacer
      V[ i ] = 1.00
fpara
V[ 0 ] = 0.00
/* RESTO DE PROBLEMAS EN ORDEN ASCENDENTE */
para  i = 1 hasta n hacer
   para  j = 1 hasta n hacer
       V[ j ] = p * V[ j ] + ( 1-p ) * V[ j-1 ]
   fpara
fpara
/* SOLUCIÓN */
retorna V[n]
ffuncion
```

Complejidad Temporal: $T(n) \in \theta(n^2)$
Complejidad Espacial: $T(n) \in \theta(n)$

EJERCICIO PROPUESTO: CÁLCULO DE NÚMEROS COMBINATORIOS

Dada la función que calcula el número combinatorio:

$\{ 0 \leq k \leq n \}$

```
Función Combinatorio ( n, k : entero ) retorna ( s : entero )
   si k = 0 ó k = n  entonces retorna 1
    sino retorna  Combinatorio ( n-1, k-1 ) + Combinatorio ( n-1, k )
   fsi
ffuncion
```

Se trata de:

- **Comprobar la repetición del cálculo de subproblemas.**
- **Elegir una estructura de almacenamiento adecuada.**
- **Rellenar dicha estructura en sentido ascendente**