

# Programación Dinámica: Cálculo de Números Combinatorios

---

Por:

Ibrahim Hernández Jorge

David Pérez Rivero

Nicolás Hernández González

# *Números Combinatorios*

## *(Coeficiente binomial)*

El coeficiente binomial  $\binom{n}{k}$  es el número de subconjuntos de  $k$  elementos escogidos de un conjunto con  $n$  elementos.

$$\binom{n}{k} \equiv \binom{n-1}{k-1} + \binom{n-1}{k}$$

Para  $n, k > 0$

## *Función de recurrencia*

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

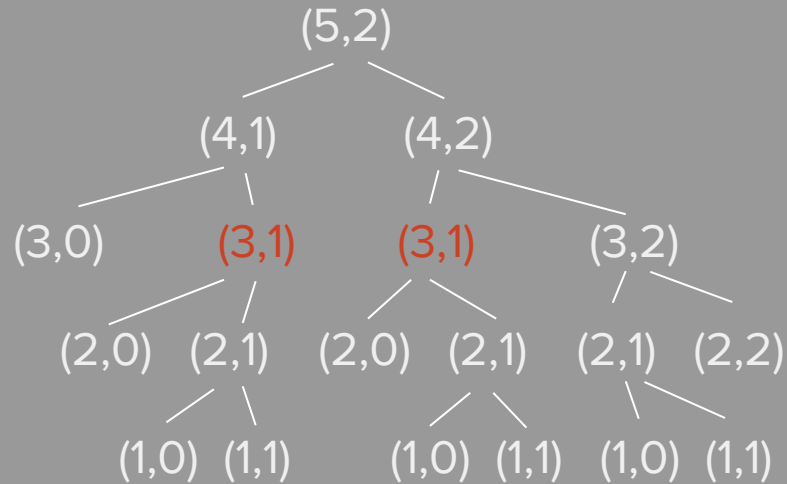
# *Algoritmo recursivo*

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

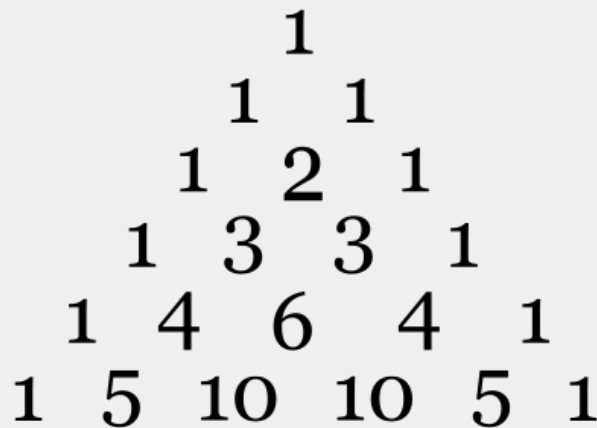
Complejidad:  $\Omega\left(\begin{matrix} n \\ k \end{matrix}\right)$

$O(2^n)$

## *Traza del algoritmo recursivo*



## *Solución con Programación Dinámica*



A Pascal's Triangle with 6 rows. The numbers are arranged in a triangular shape, with each row containing one more number than the previous row. The numbers are: Row 1: 1; Row 2: 1, 1; Row 3: 1, 2, 1; Row 4: 1, 3, 3, 1; Row 5: 1, 4, 6, 4, 1; Row 6: 1, 5, 10, 10, 5, 1.

1					
1	1				
1	2	1			
1	3	3	1		
1	4	6	4	1	
1	5	10	10	5	1

Triángulo de Pascal

Se crea un array bidimensional donde se almacenan los valores intermedios, esto no evita repetir los cálculos

# *Algoritmo Bottom Up*

```
int combinatoriaBottonUp(int n, int k) {  
    int C[][] = new int[n+1][k+1];  
    int i, j;  
    for (i = 0; i <= n; i++)  
    {  
        for (j = 0; j <= min(i, k); j++)  
        {  
            if (j == 0 || j == i)  
                C[i][j] = 1;  
            else  
                C[i][j] = C[i-1][j-1] + C[i-1][j];  
        }  
    }  
    return C[n][k];  
}
```

Complejidad:  $O(n*k)$

Espacio auxiliar:  $n*k$

## *Variante del Algoritmo Bottom Up*

```
int combinatoriaBottonUp2(int n, int k) {  
    int C[] = new int[k+1];  
    for (int i = 1; i <= k; i++)  
        C[i] = 0;  
    C[0] = 1;  
    for (int i = 1; i <= n; i++)  
    {  
        for (int j = min(i, k); j > 0; j--)  
            C[j] = C[j] + C[j-1];  
    }  
    return C[k];  
}
```

Complejidad:  $O(n*k)$

Espacio auxiliar:  $n$



# *Comparativa*

Reducción de complejidad exponencial a polinomial, similar a otros algoritmos de programación dinámica

No es necesario reconstruir la solución a partir de la tabla

El tamaño de la tabla es menor que en otros algoritmos similares, ya que, las dimensiones son de  $n \cdot k$ , donde  $k < n$

# *Cálculo mediante factoriales*

## **Fórmula de número combinatorio**

$$C(n, k) = n! / (n - k)! * k! = [n * (n - 1) * \dots * 1] / [(n - k) * (n - k - 1) * \dots * 1] * (k * (k - 1) * \dots * 1)]$$

## **Simplificando:**

$$C(n, k) = [n * (n - 1) * \dots * (n - k + 1)] / [k * (k - 1) * \dots * 1]$$

$$\binom{n}{k} = \frac{n!}{k! * (n - k)!} = \frac{n * (n - 1) * \dots * (n - k + 1)}{k!}$$

# *Algoritmo Alternativo*

```
int coeficienteBinomialAlt( int n, int k) {  
    int aux= 1;  
    if ( k > n - k )  
        k = n - k;  
    for (int i = 0; i < k; ++i)  
    {  
        aux*= (n - i);  
        aux/= (i + 1);  
    }  
    return aux;  
}
```

Complejidad:  $O(k)$

Espacio auxiliar: 1

*Fin*

Contacto:

- Ibrahim Hernández Jorge: [alu0100884814@ull.edu.es](mailto:alu0100884814@ull.edu.es)
- Nicolás Hernández González: [alu0100898293@ull.edu.es](mailto:alu0100898293@ull.edu.es)
- David Pérez Rivero: [alu0100826166@ull.edu.es](mailto:alu0100826166@ull.edu.es)