ESTRATEGIAS DE PROGRAMACIÓN

Algoritmos y Programación

Javier Miranda

Escuela de Ingeniería Informática
Universidad de Las Palmas de Gran Canaria

25 de octubre de 2023

Estrategias

- Fuerza bruta (*brute force*)
- Vuelta atrás (*backtracking*)
- Voráz (*greedy*)

- Divide y vencerás
  - Reduce y vencerás

Técnica

- Programación Dinámica

Programación Dinámica

- Es una técnica inventada por el matemático norteamericano Richard Bellman en los años 50 para resolver problemas de optimización.

- ¿ Cuando debemos utilizarla ?

  Cuando el problema tiene subproblemas que se solapan, ya que en este caso la estrategia **Divide y Vencerás** genera algoritmos **poco eficientes**.

https://en.wikipedia.org/wiki/Richard_E._Bellman
https://en.wikipedia.org/wiki/Dynamic_programming

Programación Dinámica

Richard Bellman
(1920-1984)

"I spent the Fall quarter (of 1950) at RAND. My first task was to find a name for ***multistage decision processes****.*

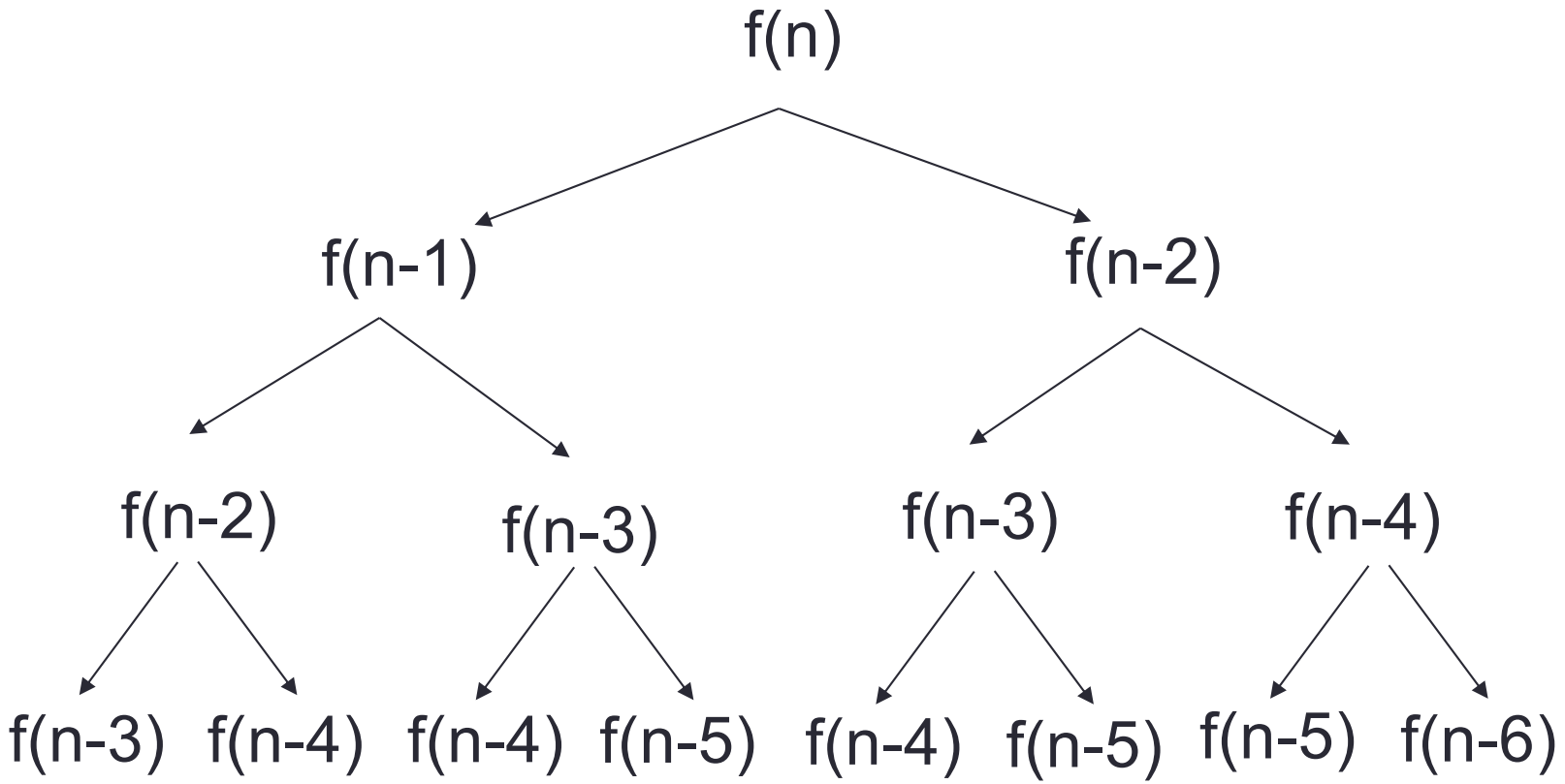
"An interesting question is, 'Where did the name, ***dynamic programming****, come from?' The 1950s were not good years for mathematical research. We had a very interesting gentleman in Washington named Wilson. He was Secretary of Defense, and he actually had a pathological fear and hatred of the word, research ⋯You can imagine how he felt, then, about the term, mathematical ⋯*

*What title, what name, could I choose? In the first place I was interested in planning, in decision making, in thinking. But planning is not a good word for various reasons. I decided therefore to use the word, '****programming****.' I wanted to get across the idea that this was dynamic, this was multistage, this was time varying—I thought, let's kill two birds with one stone. Let's take a word that has an absolutely precise meaning, namely* ***dynamic****, in the classical physical sense ⋯*

Documento completo disponible en el Campus Virtual

Ejemplo 1: Fibonacci

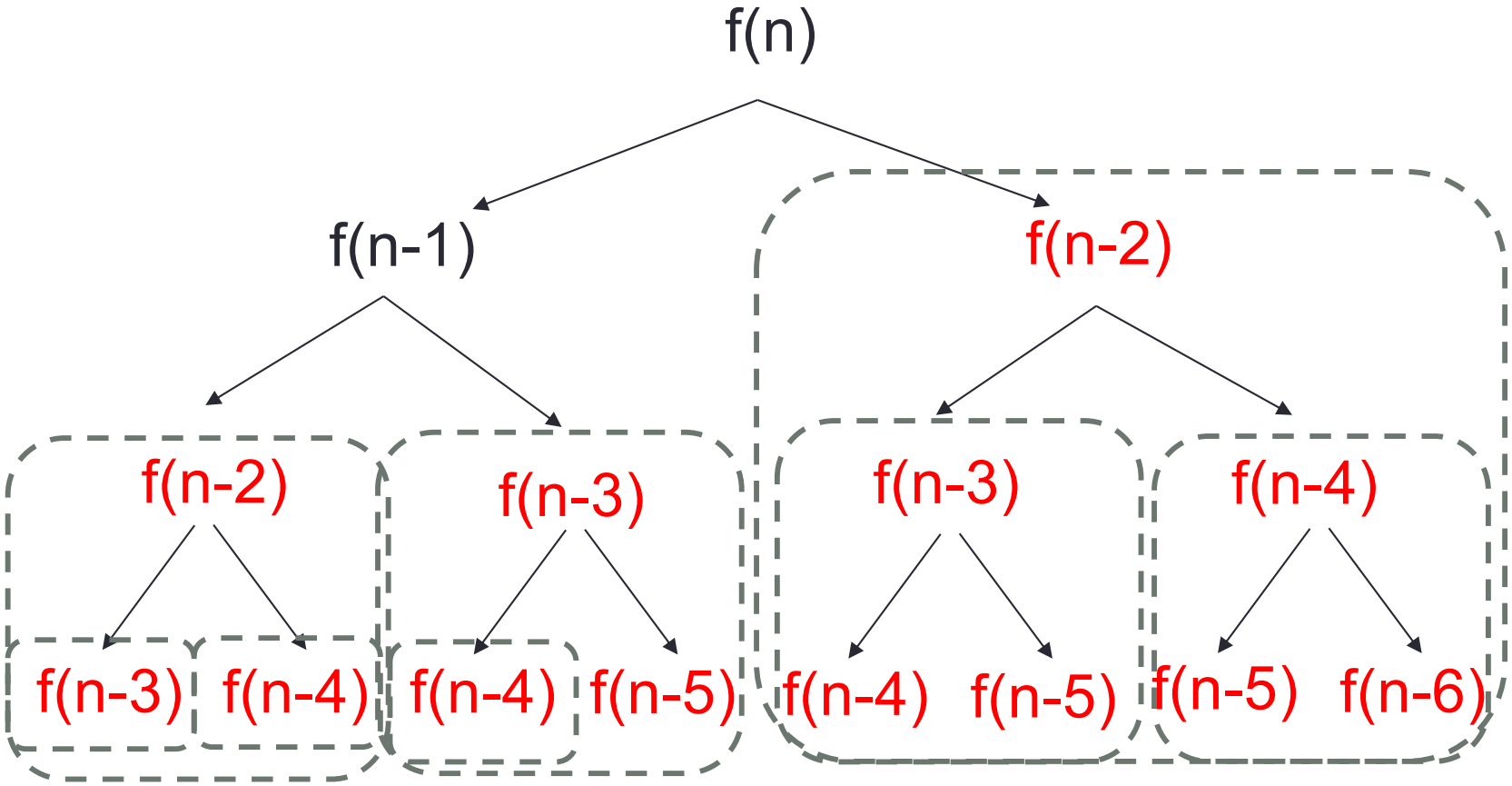
$$f(0) = 0$$
$$f(1) = 1$$
$$f(n) = f(n\text{-}1) + f(n\text{-}2)$$



https://es.wikipedia.org/wiki/Sucesi%C3%B3n_de_Fibonacci

Ejemplo 1: Fibonacci

$$f(0) = 0$$
$$f(1) = 1$$
$$f(n) = f(n\text{-}1) + f(n\text{-}2)$$

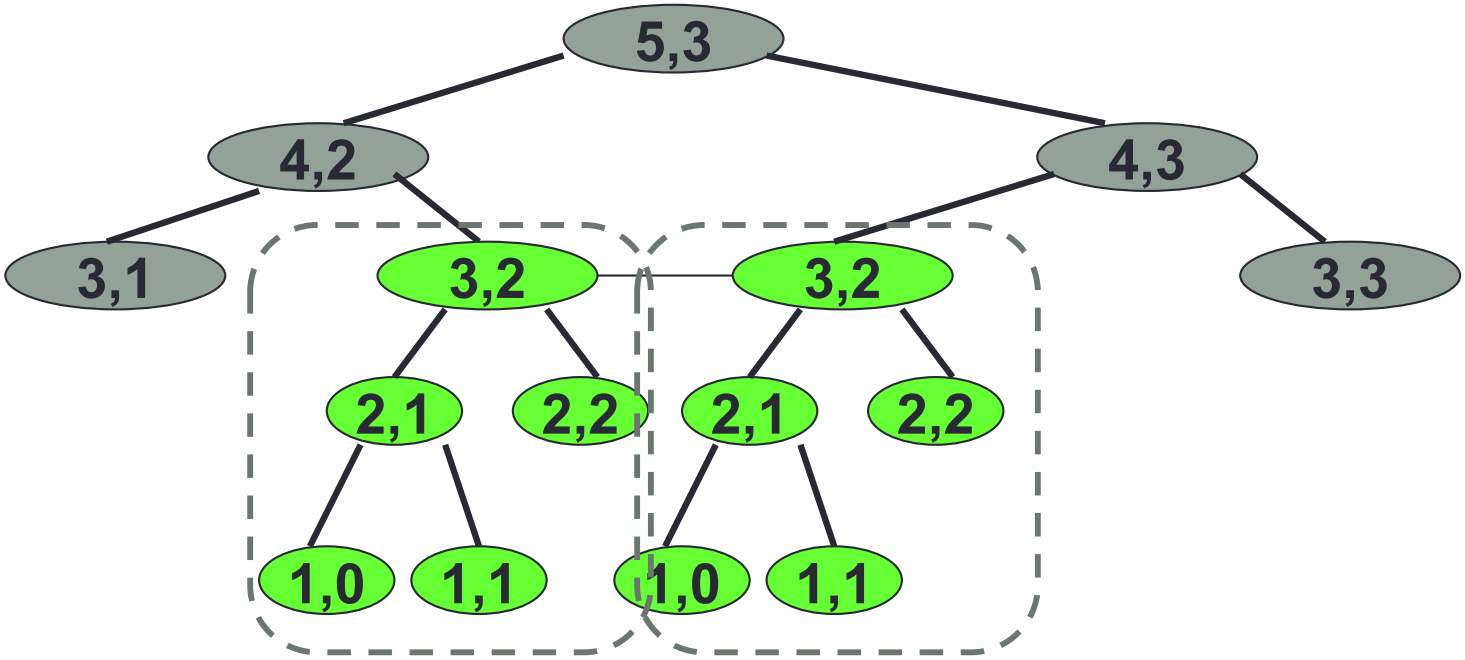


Calculando Fibonacci recursivamente repetimos muchos cálculos.
La programación dinámica evita repetirlos!.

Ejemplo 2: Coeficiente Binomial

$$comb(n,m) = \binom{n}{m} = \binom{n-1}{m-1} + \binom{n-1}{m}, \quad m \le n$$

$$\text{with } comb(n,0) = comb(n,n) = 1$$



https://en.wikipedia.org/wiki/Binomial_coefficient

Programación Dinámica

- **¿ Qué ?**
  - Técnica que combina soluciones de subproblemas para resolver problemas mayores de forma eficiente
- **¿ Cómo ?**
  - Guardando las soluciones de los subproblemas y reutilizándolas para evitar repetir cálculos al resolver problemas mayores

Podemos hacer el recorrido bottom-up o top-down

Implementación de Programación Dinámica

Memoization

- Se utiliza cuando el problema se resuelve recursivamente (***top-down***)

Tabulation

- Se utiliza cuando el problema se resuelve comenzando por los sub-problemas (***bottom-up***)

El objetivo de ambas técnicas es el mismo: almacenar y reutilizar las soluciones de los subproblemas

https://en.wikipedia.org/wiki/Memoization

Ejemplo Fibonacci

$$t_n = f(t_{n-1}, t_{n-2}) = t_{n-1} + t_{n-2} \qquad n \geq 2$$
$$t_0 = 0 \quad t_1 = 1$$

Ecuación de recurrencia de Fibonacci

```
def Fib(n) {
    if (n < 2)
        return n
    else
        return Fib(n-2) + Fib(n-1)
}
```

Versión Recursiva

Ejemplo Fibonacci

$$t_n = f(t_{n-1}, t_{n-2}) = t_{n-1} + t_{n-2} \qquad n \geq 2$$
$$t_0 = 0 \quad t_1 = 1$$

```
def Fib(n) {
    if (n < 2)
        return n
    else
        return Fib(n-2) + Fib(n-1)
}
```

Versión Recursiva

```
def Fib(n):
    mem = {}          # Diccionario
    def memFib(n):
        key = n
        if key not in mem:
            if n<2:
                r = n
            else:
                r = memFib(n-1)+memFib(n-2)

            mem[key] = r
        return mem[key]

    return memFib(n)
```

Versión implementada en Python con MEMOIZATION

Ejemplo Fibonacci

$$t_n = f(t_{n-1}, t_{n-2}) = t_{n-1} + t_{n-2} \qquad n \geq 2$$
$$t_0 = 0 \quad t_1 = 1$$

```
def Fib(n) {
    if (n < 2)
        return n
    else
        return Fib(n-2) + Fib(n-1)
}
```

Versión Recursiva

```
def Fib(n):
    if n < 2:
        return n
    else:
        table = [ ]              # Lista o array

        table.append(0)
        table.append(1)

        for j in range(2,n+1):
            table.append(table[j-2] + table[j-1])
        return table[n]
```

Versión implementada en Python con TABULATION

Como sabes del curso anterior, esta versión se puede optimizar para que sólo utilice 2 variables (en vez de una tabla)

¡Cuidado!

- Lo que identifica a una técnica como memoization o como tabulation es el **tipo de recorrido** (recursivo o **top-down**, frente a iterativo o **bottom-up**), no el tipo de memoria utilizada en la programación

```
def Fib(n):
    mem = {}            # Diccionario
    def memFib(n):
        key = n
        if key not in mem:
            if n<2:
                r = n
            else:
                r = memFib(n-1)+memFib(n-2)
                # … llamadas recursivas!
            mem[key] = r
        return mem[key]

    return memFib(n)
```

Memoization: Top-down (recursivo)

```
def Fib(n):
    if n < 2:
        return n
    else:
        table = []          # Lista o array

        table.append(0)
        table.append(1)

        for j in range(2,n+1):
            table.append(table[j-2] + table[j-1])

        return table[n]
```

Tabulation: Bottom-Up (iterativo)

Requisitos para Programación Dinámica

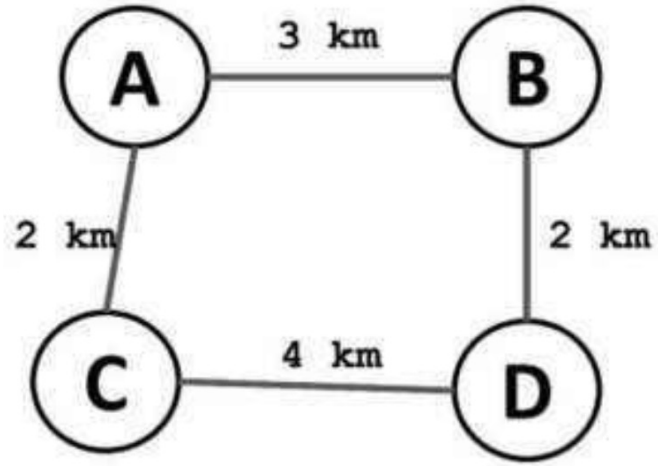
1. **Subproblemas Solapados**
   - Las soluciones de los subproblemas se reutilizan varias veces para resolver problemas mayores

2. **Subestructura Optima**
   - La solución optima del problema se puede construir a partir de soluciones óptimas de los subproblemas

Requisitos para Programación Dinámica

Subestructura Optima

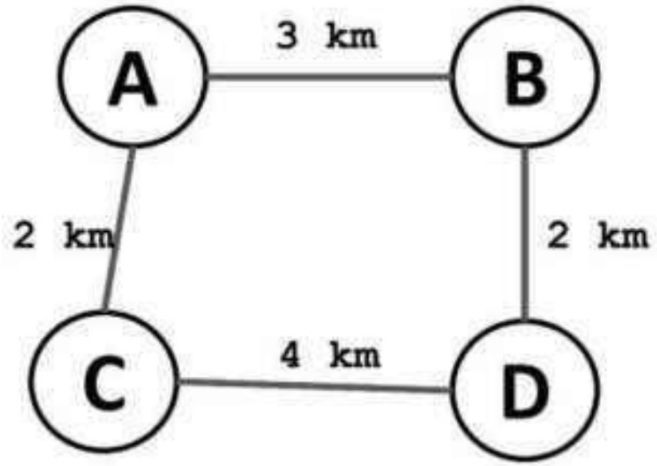
- La solución optima del problema se puede construir a partir de soluciones óptimas de los subproblemas
- Ejemplo: Camino más corto



Sin embargo, el camino más largo no cumple la propiedad de subestructura óptima (no se resuelve con programación dinámica)

Requisitos para Programación Dinámica

- **Ejemplo**: El problema de calcular el camino más largo no cumple la propiedad de subestructura óptima



La distancia más larga desde A hasta D es 6 km a través de la ciudad C, pero este camino no es la combinación de la distancia más larga de A a C (9 km) y de C a D (7 km)

Rendimiento

- **En problemas pequeños tabulation es más eficiente que memoization**
  - Porque no tiene llamadas recursivas

n =	2	3	4	5	10	20	40
Recursive	1	3	5	9	109	13529	204668309
Iterative	1	1	1	1	1	1	1
Memo	1	3	5	7	17	37	77

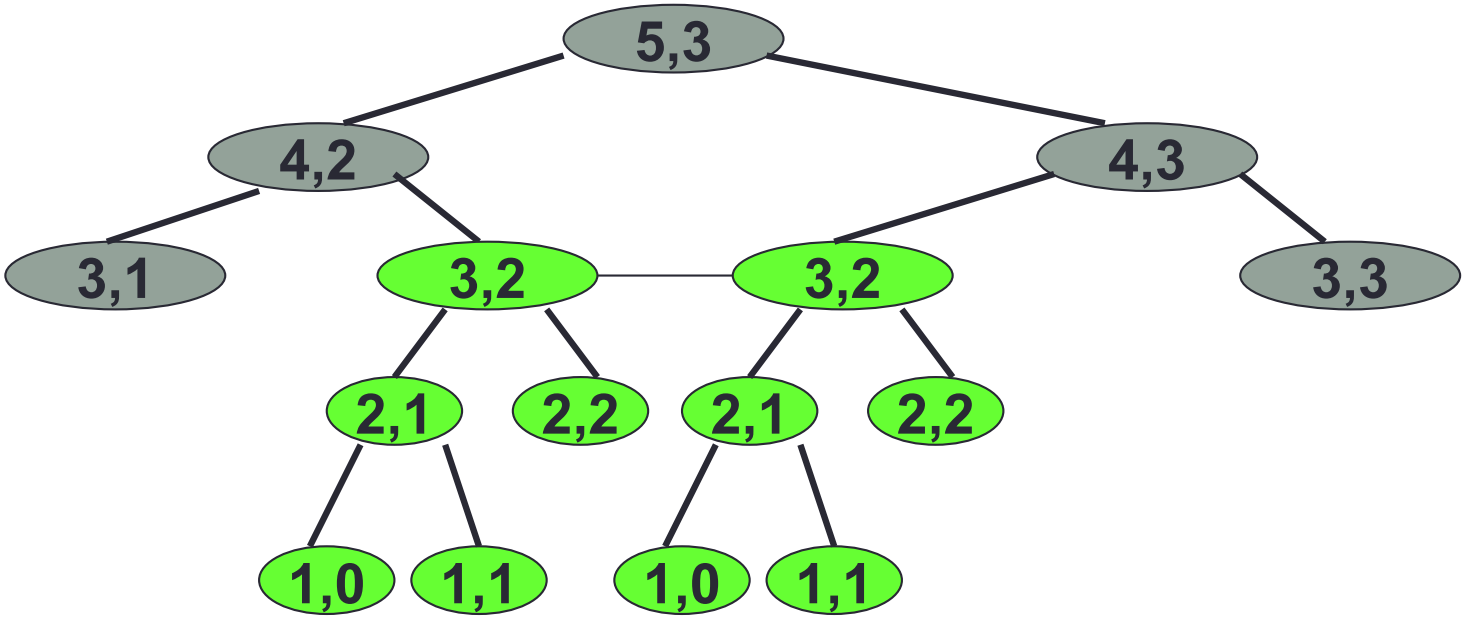
Número de llamadas a la función para calcular Fibonacci

- **Pero en problemas grandes puede ser mejor memoization**
  - Porque no necesita calcular TODOS los elementos
  - Porque consume menos memoria

Ejemplo: Coeficiente Binomial

$$comb(n,m) = \binom{n}{m} = \binom{n-1}{m-1} + \binom{n-1}{m}, \quad m \le n$$

$$\text{with } comb(n,0) = comb(n,n) = 1$$

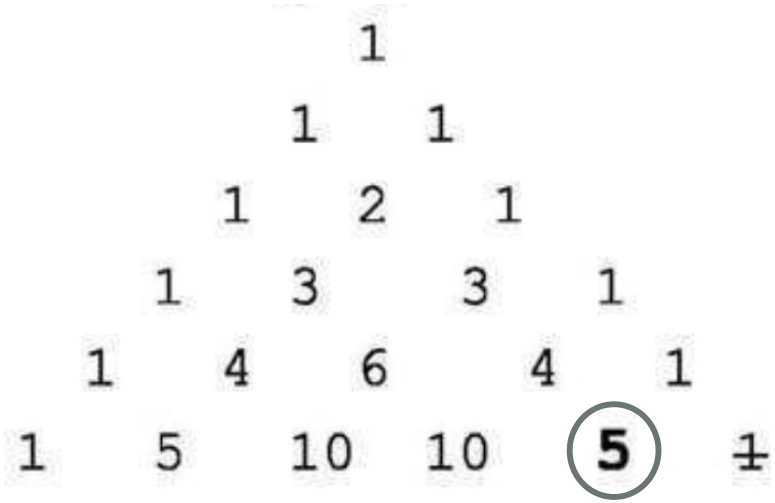


https://en.wikipedia.org/wiki/Binomial_coefficient

Ejemplo: Coeficiente Binomial

$$comb(n,m) = \binom{n}{m} = \binom{n-1}{m-1} + \binom{n-1}{m}, \quad m \leq n$$

$$\text{with } comb(n,0) = comb(n,n) = 1$$



Ejemplo: Cálculo de comb (5,4)

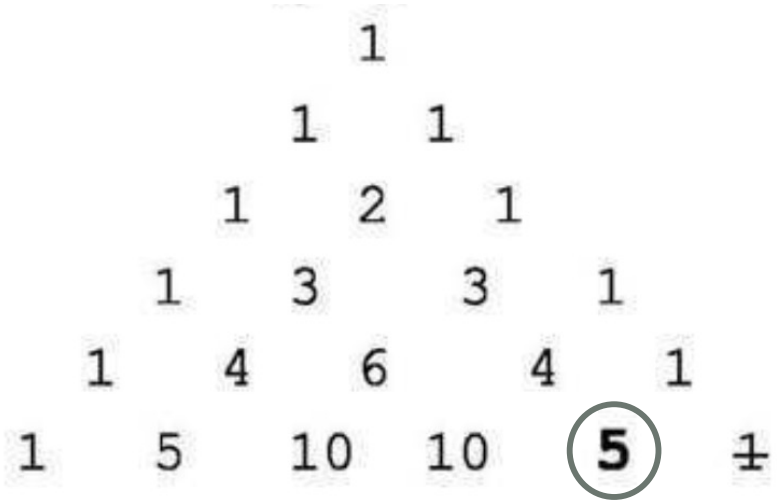
Resultado = 5

https://en.wikipedia.org/wiki/Binomial_coefficient

Ejemplo: Coeficiente Binomial

$$comb(n,m) = \binom{n}{m} = \binom{n-1}{m-1} + \binom{n-1}{m}, \quad m \le n$$

$$\text{with } comb(n,0) = comb(n,n) = 1$$



n \ k	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	⋯
⋮	⋮	⋮	⋮	⋮	⋮	⋱

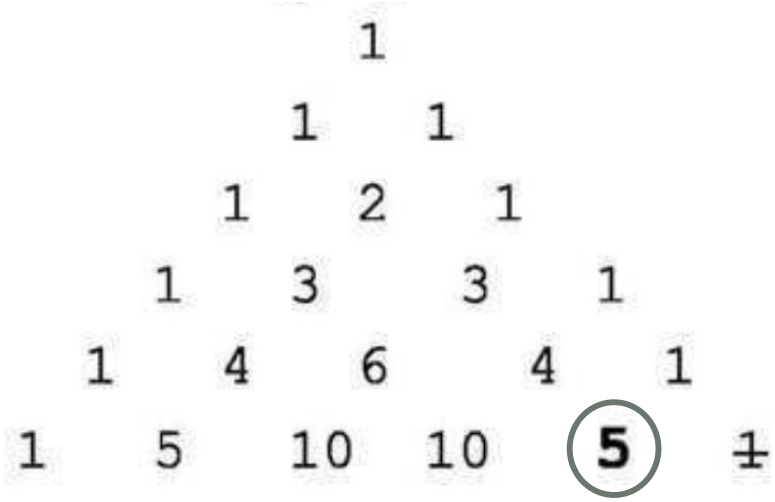
Tabulation calcula todos los elementos del triángulo de Pascal hasta llegar al objetivo

https://en.wikipedia.org/wiki/Binomial_coefficient

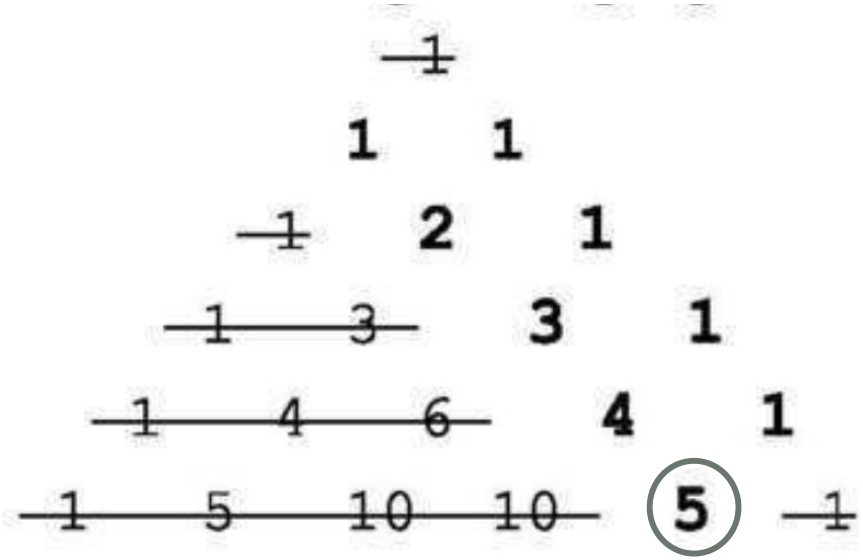
Ejemplo: Coeficiente Binomial

$$comb(n,m) = \binom{n}{m} = \binom{n-1}{m-1} + \binom{n-1}{m}, \quad m \le n$$

$$\text{with } comb(n,0) = comb(n,n) = 1$$



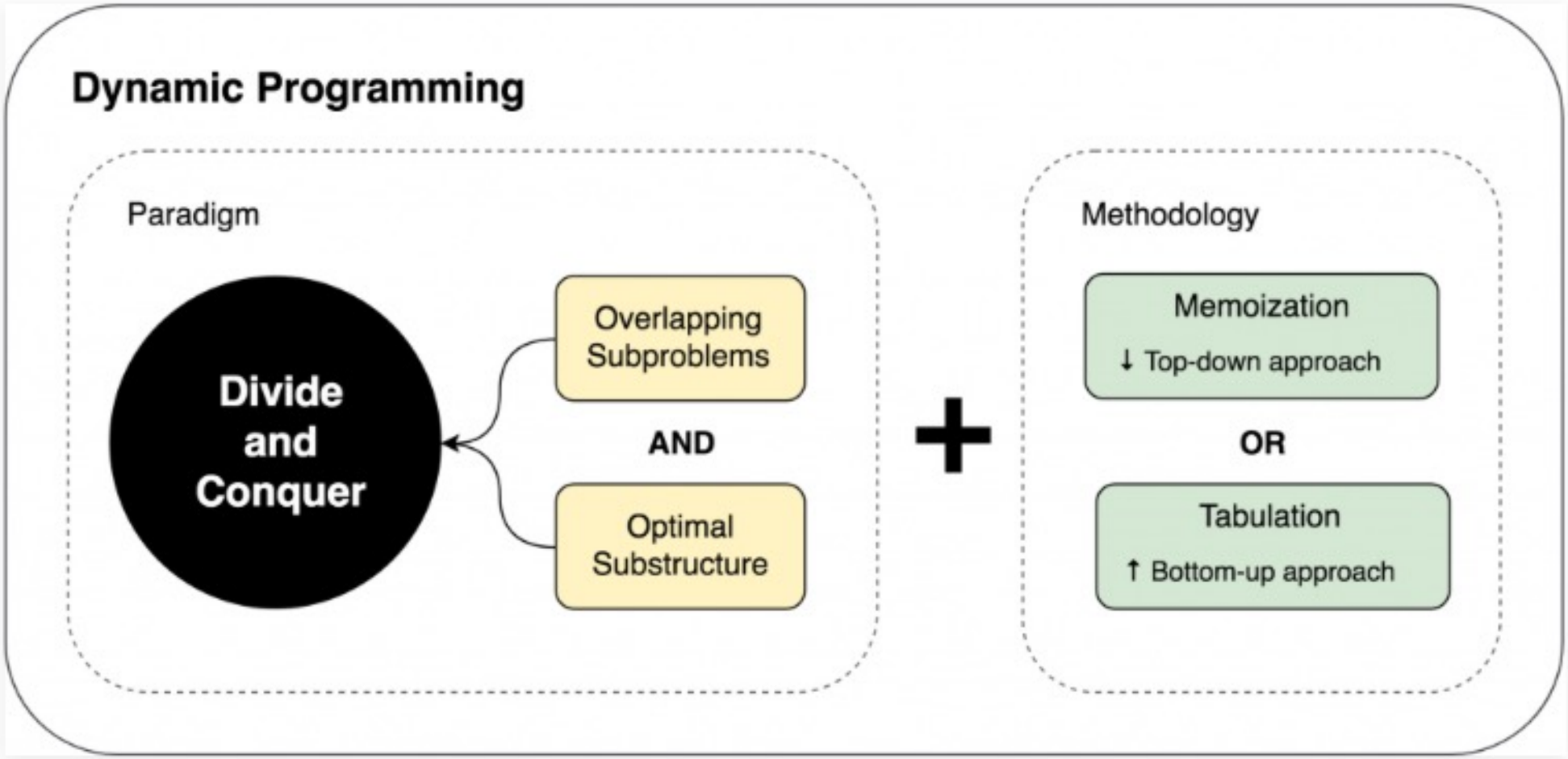
Tabulation calcula todos los elementos del triángulo de Pascal



Memoization sólo calcula los elementos que necesita

https://en.wikipedia.org/wiki/Binomial_coefficient

Resumen


Dynamic Programming
Paradigm
Divide and Conquer
Overlapping Subproblems
AND
Optimal Substructure
Methodology
Memoization
↓ Top-down approach
OR
Tabulation
↑ Bottom-up approach