Programación Dinámica

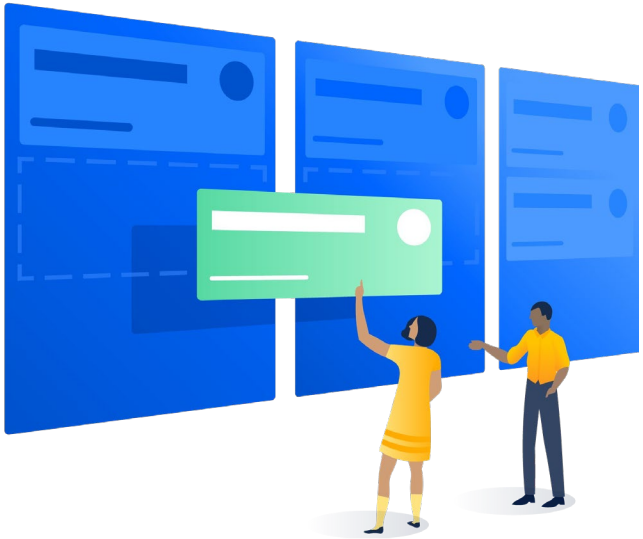
Tema 4

Los contenidos de esta presentación han sido adaptados a partir del material creado por los profesores Javier Junquera Sánchez y Francisco Manuel Sáez de Adana Herrero.

Índice

Universidad de Alcalá

- **Programación dinámica**
  - Fibonacci
- **Ejemplos**
  - Coeficiente binomial
  - Mochila
  - Campeonato mundial
  - Algoritmo de Floyd



Programación dinámica

Universidad de Alcalá

Es un concepto desarrollado por **Richard Bellman**, matemático y economista. Bellman buscaba una forma de resolver problemas complejos de optimización. Estos problemas requieren elegir la mejor solución entre un conjunto de opciones.

Un ejemplo de problema de optimización es el problema del viajante de comercio. El objetivo es encontrar la ruta más corta que permita al vendedor visitar cada ciudad exactamente una vez y regresar a la ciudad de partida.

El planteamiento de Bellman para estos problemas consistía en dividirlos en subproblemas más pequeños y resolver los subproblemas del más pequeño al más grande.

Programación dinámica

Universidad de Alcalá

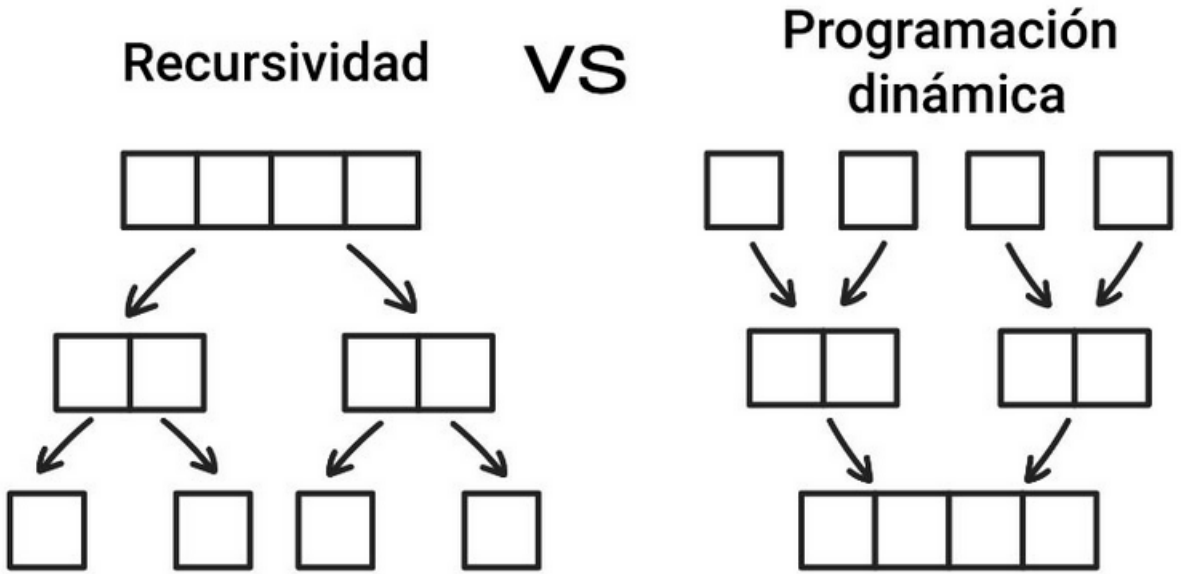
¿Qué es la programación dinámica?

La programación dinámica resuelve problemas de optimización dividiéndolos en subproblemas más pequeños, resolviendo cada subproblema una vez y almacenando sus soluciones para poder reutilizarlas y combinarlas para resolver el problema más grande. Los problemas se resuelven del más pequeño al más grande, lo que permite reutilizar las soluciones.

Programación dinámica

Universidad de Alcalá

El trabajo de la programación dinámica es muy similar a la recursión, pero almacenando las soluciones intermedias.



Programación dinámica

Universidad de Alcalá

La resolución de un problema mediante programación dinámica implica los siguientes pasos:

- **Definir los subproblemas:** Un problema grande se divide en pequeños subproblemas.
- **Resolver los subproblemas:** Esto implica resolver el subproblema identificado, lo que puede hacerse utilizando la recursividad o la iteración.
- **Almacenar las soluciones:** Las soluciones a los subproblemas se almacenan para poder reutilizarlas.
- **Construir la solución al problema original:** La solución al problema grande se construye a partir de los subproblemas que ya se han calculado.

Programación dinámica

Universidad de Alcalá

Enfoque descendente (recursivo):

El enfoque descendente implica recursión y almacenamiento en caché. La recursividad implica que una función se llame a sí misma con versiones más simples del problema como argumento. La recursividad se utiliza para descomponer el problema en subproblemas más pequeños y resolver los subproblemas.

Una vez resuelto un subproblema, su resultado **se almacena en caché y se reutiliza** cada vez que se presenta un problema similar. El método descendente es fácil de entender e implementar y sólo resuelve un subproblema una vez. Sin embargo, su inconveniente es que ocupa mucha memoria debido a la recursividad. Esto puede provocar un error de desbordamiento de pila.

Programación dinámica

Universidad de Alcalá

Enfoque ascendente (iterativo):

Los subproblemas más pequeños se resuelven primero y sus resultados se almacenan en memoria, normalmente usando una matriz o tabla.

Con los resultados almacenados se resuelven cada vez problemas mayores que dependen de los subproblemas y se almacenan sus resultados. El resultado del problema original se calcula resolviendo los subproblemas más grandes, utilizando los valores calculados previamente.

Este enfoque tiene la ventaja de ser eficiente en memoria y tiempo al prescindir de la recursividad.

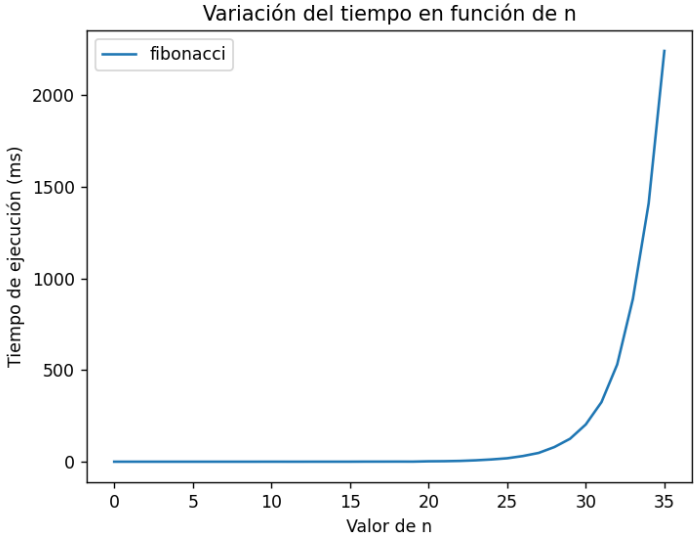
Fibonacci

Universidad de Alcalá

Algoritmo recursivo: $O(2^n)$

```python
def fibonacci(n: int) -> int:
    """ n-ésimo término de fibonacci """

    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)
```



¿Cómo podemos almacenar los resultados intermedios?

Fibonacci

Universidad de Alcalá

Enfoque descendente (recursivo): $O(n)$

```python
def fibonacci(n: int) -> int:

    def _fibonacci_cache(n: int, cache: dict) -> int:
        if n == 0:
            return 0
        elif n == 1:
            return 1
        elif n in cache:
            return cache[n]
        else:
            resultado = _fibonacci_cache(n-1, cache) + _fibonacci_cache(n-2, cache)
            cache[n] = resultado
            return resultado

    cache = {}
    return _fibonacci_cache(n, cache)
```

Fibonacci

Universidad de Alcalá

Enfoque descendente (optimizando la memoria):

Solo quiero almacenar los dos valores que necesito.

```
else:
    resultado = _fibonacci_cache...
    cache[n] = resultado
    if len(cache) > 2:
        del cache[n - 2]
     return resultado
```

Sigo teniendo el problema de la pila de llamadas.

¿Y si no tengo diccionarios?

https://docs.python.org/3/library/functools.html#functools.lru_cache

Fibonacci

Universidad de Alcalá

Enfoque ascendente (iterativo): $O(n)$

```python
def fibonacci_dinamico(n: int) -> int:

    cache = [0, 1]
    for i in range(2, n + 1):
        cache.append(cache[i - 1] + cache[i - 2])

    return cache[n]
```

```python
def fibonacci_dinamico(n: int) -> int:

    cache = [0, 1]
    for _ in range(2, n + 1):
        aux = cache[0]
        cache[0] = cache[1]
        cache[1] += aux

    return cache[1]
```

Programación dinámica

Universidad de Alcalá

Enfoque ascendente (iterativo):

¿Qué se necesita para resolver un problema de manera dinámica, además de sus datos iniciales? Solo tres cosas:

- Una tabla en la que se almacenarán los resultados intermedios. Uno de ellos se seleccionará al final del algoritmo como la respuesta.
- Algunas reglas simples para llenar las celdas vacías de la tabla, basadas en los valores de las celdas ya llenas. No hay una receta universal y cada problema requiere su propio enfoque.
- Una regla para seleccionar la solución final después de llenar la tabla.

Programación dinámica

Universidad de Alcalá

Enfoque ascendente (iterativo):

- En los casos más simples, esta tabla consistirá en una sola fila, similar a una matriz regular. Estos casos se llaman programación dinámica unidimensional y requieren $O(n)$ memoria.
- En los casos más comunes, esta tabla tendrá filas y columnas, similar a una tabla de Excel. Esto se llama programación dinámica bidimensional y requiere $O(n^2)$ memoria para n filas y n columnas. Por ejemplo, una tabla cuadrada de 10 filas y 10 columnas contendrá 100 celdas. A continuación, se analizará en detalle un problema de este tipo.

Coeficiente Binomial

Universidad de Alcalá

El coeficiente binomial n sobre k es el número de subconjuntos de k elementos escogidos de un conjunto con n elementos.

$$\binom{n}{k}$$

```
coef_binom(n, k) = n! / ( k! * ( n! - k! ) )
```

Coeficiente Binomial

Universidad de Alcalá

Algoritmo recursivo:

```python
def binomial(n: int, k: int) -> int:

    if k == 0:
        return 1
    if n == k:
        return 1

    return binomial(n-1, k-1) + binomial(n-1, k)
```

$$coef\_binom(n, k) = coef\_binom(n-1, k-1) + coef\_binom(n-1, k)$$

$$coef\_binom(n, 0) = coef\_binom(n, n) = 1$$

Coeficiente Binomial

Universidad de Alcalá

Programación dinámica:

$$
\begin{array}{ll}
n=1 & \binom{1}{0}\ \binom{1}{1} \\
n=2 & \binom{2}{0}\ \binom{2}{1}\ \binom{2}{2} \\
n=3 & \binom{3}{0}\ \binom{3}{1}\ \binom{3}{2}\ \binom{3}{3} \\
n=4 & \binom{4}{0}\ \binom{4}{1}\ \binom{4}{2}\ \binom{4}{3}\ \binom{4}{4} \\
n=5 & \binom{5}{0}\ \binom{5}{1}\ \binom{5}{2}\ \binom{5}{3}\ \binom{5}{4}\ \binom{5}{5}
\end{array}
$$

$$
\begin{array}{ll}
n=1 & 1\ \ 1 \\
n=2 & 1\ \ 2\ \ 1 \\
n=3 & 1\ \ 3\ \ 3\ \ 1 \\
n=4 & 1\ \ 4\ \ 6\ \ 4\ \ 1 \\
n=5 & 1\ \ 5\ \ 10\ \ 10\ \ 5\ \ 1
\end{array}
$$

Coeficiente Binomial

Universidad de Alcalá

Programación dinámica:

n \ k	1	2	3	4	5	6	7	8
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		
6	1	6	15	20	15	6	1	
7	1	7	21	35	35	21	7	1

Coeficiente Binomial

Universidad de Alcalá

Programación dinámica:

```python
def binomial_dinamico(n: int, k: int) -> int:

    def _matriz_binomial(n: int, k: int) -> list[list[int]]:
        """
            _matriz_binomial(n, k) es la matriz de orden (n+1)x(k+1) tal que el valor en
            la posición (i,j) (con j <= i) es el coeficiente binomial i sobre j.
        """
        matriz = [[0 for i in range(k + 1)] for j in range(n + 1)]
        for i in range(n + 1):
            for j in range(min(i, k) + 1):
                if j == 0:
                    matriz[i][j] = 1
                elif i == j:
                    matriz[i][j] = 1
                else:
                    matriz[i][j] = matriz[i - 1][j - 1] + matriz[i - 1][j]
        return matriz

    return _matriz_binomial(n, k)[n][k]
```

Universidad de Alcalá

Decisión analítica

Hay que determinar si el problema es abordable mediante programación dinámica.

(e.g., los cálculos deben poder introducirse en la memoria)