Programación Dinámica (Dynamic Programming)

Programación Dinámica (Dynamic Programming)
Objetivos

- Comprender la finalidad de la programación dinámica.
- Comprender el concepto de programación dinámica.
- Comprender el principio de optimalidad de Bellman.
- Saber aplicar programación dinámica.

Programación Dinámica (Dynamic Programming)
Comprender la finalidad de la programación dinámica

La **programación dinámica** es una técnica aplicada a problemas que cumplan una de estas dos condiciones:

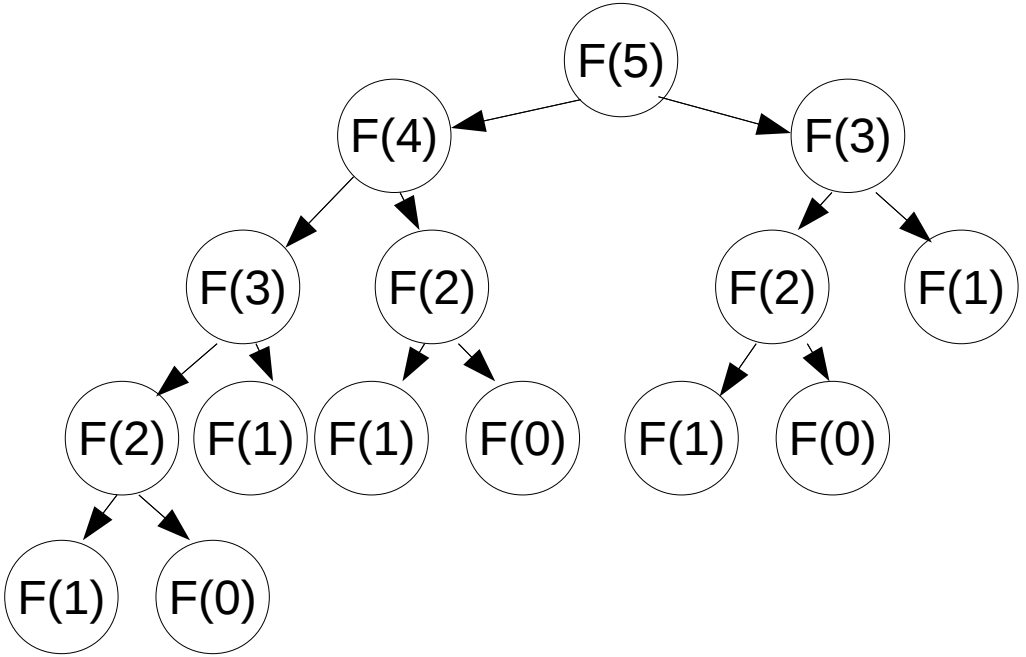
- Que puedan descomponerse en **subproblemas solapados**.
- Que puedan descomponerse en **subestructuras optimales**.

Se aplica principalmente a problemas de **optimización** que se resuelven como una secuencia de decisiones.

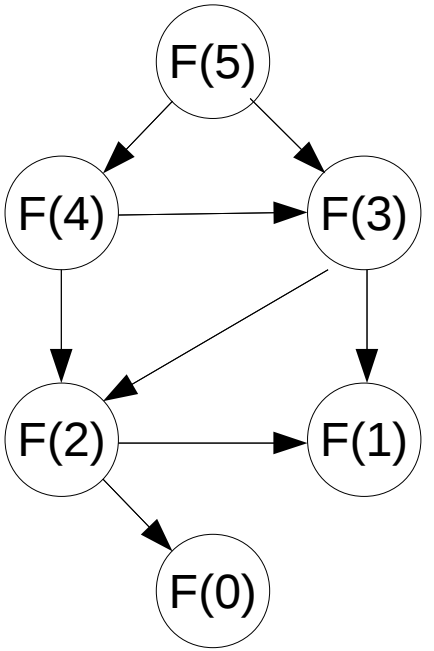
Programación Dinámica (Dynamic Programming)
Comprender la finalidad de la programación dinámica

Podemos observar el problema de los subproblemas solapados

Fibonacci: F(n) = F(n-1)+F(n-2) si n>= 2,n si n<2



Solución por función recursiva.
$O(2^n)$



Solución Programación Dinámica.
O(n)

Programación Dinámica (Dynamic Programming)
Comprender el concepto de programación dinámica

La **programación dinámica** se diferencia de los algoritmos greedy en que se consideran **varias secuencias de decisiones** y sólo al final se sabe cuál es la mejor de ellas.

Estas decisiones se almacenan en una tabla cuyo estudio posterior proporciona el itinerario de decisiones tomadas.

Tal y como comentamos en greedy, si el problema se puede descomponer en **subestructuras optimales**, basta con resolver **el primer paso de forma óptima** y luego resolver **el resto de pasos de forma óptima** (**recursivamente**).

Obsérvese que justamente por esto no es lo mismo que resolver por fuerza bruta, que supondría explorar todas las combinaciones posibles.

Programación Dinámica (Dynamic Programming)
Comprender el principio de optimalidad de Bellman

El **principio de optimalidad de Bellman** recoge esta propiedad:

“Cualquier subsecuencia de decisiones de una secuencia óptima de decisiones que resuelve un problema también debe ser óptima respecto al subproblema que resuelve.”

Aunque parece evidente, este principio no se cumple para todos los problemas (por ejemplo, el viajante de comercio), por lo que para aplicar programación dinámica hay que, en primer lugar, demostrar que sí lo hace tal y como lo hacíamos en greedy.

Programación Dinámica (Dynamic Programming)
Saber aplicar programación dinámica

Paso 1: **Planteamiento** de la solución como una sucesión de decisiones y verificación de que esta cumple el principio de óptimo.

Paso 2: Definición **recursiva** de la solución.

Paso 3: Cálculo del valor de la solución óptima mediante una **tabla** en donde se almacenan **soluciones a problemas parciales** para reutilizar los cálculos.

Paso 4: **Construcción de la solución óptima** haciendo uso de la información contenida en la tabla anterior.

Programación Dinámica (Dynamic Programming)
¿Qué hemos aprendido?

- **Finalidad de la programación dinámica.**
- **Concepto de programación dinámica.**
- **Principio de optimalidad de Bellman.**
- **Aplicar programación dinámica.**

Programación Dinámica (Dynamic Programming)
Ejemplos

Ejemplo: Mochila indivisible con peso y beneficio.

Objetivo: Escoger elementos que maximicen beneficio sin pasarse del peso máximo de la mochila.

Programación Dinámica (Dynamic Programming)
Ejemplos

Ejemplo: Mochila indivisible con peso y beneficio.

$La\ función\ mochila(k, l, P_{max}), donde:$

$k = desde\ dónde\ resolver\ el\ problema\ (0 = principio)$

$l = hasta\ dónde\ resolver\ el\ problema\ (n = final)$

$P_{max} = peso\ máximo$

$x_i = si\ hemos\ tomado\ (1)\ o\ no\ (0)\ el\ objeto\ i$

$b_i = beneficio\ de\ tomar\ el\ objeto\ i$

$p_i = peso\ del\ objeto\ i$

$$\text{Maximizar} \sum_{i=k}^{l} b_i x_i$$

$$Sujeto\ a \sum_{i=k}^{l} p_i x_i \leq P_{max}$$

$$Con\ x_i \{0, 1\}, k \leq i \leq l$$

Programación Dinámica (Dynamic Programming)
Ejemplos

Ejemplo: Mochila indivisible con peso y beneficio.

$mochila(k, l, P_{max})$.

$Si\ y_1, \ldots, y_n\ es\ una\ solución\ óptima\ del\ problema\ mochila(1, n, C),$

$entonces\ para\ todo\ j, 1 \leq j \leq n:$

$y_1, \ldots, y_j\ es\ solución\ óptima\ de$

$$mochila(1, j, \sum_{i=1}^{j} p_i x_i)$$

$y_{j+1}, \ldots, y_n\ es\ solución\ óptima\ de$

$$mochila(j+1, n, C - \sum_{i=1}^{j} p_i x_i)$$

Programación Dinámica (Dynamic Programming)
Ejemplos

Ejemplo: Mochila indivisible con peso y beneficio.

$mochila(k, l, P_{max})$.

Recurrencia hacia atrás:

El beneficio o ganancia total g sub{j}(C) de una solución óptima de mochila(1,j,C) es:

$$g_j = max\{ g_{j-1}(C), g_{j-1}(C - p_j) + b_j \}$$

Dependiendo de que el objeto j-ésimo entre o no en la solución (sólo puede entrar si $c - p\,j \geq 0$)

Ojo, el cálculo en este caso se haría hacia delante

Programación Dinámica (Dynamic Programming)
Ejemplos

Ejemplo: Mochila indivisible con peso y beneficio.

$mochila(k, l, P_{max})$.

Aunque también podríamos definir una recurrencia hacia delante:
El beneficio o ganancia total g sub{j}(C) de una solución óptima de mochila(j,n,C) es:

$$g_j = max\{ g_{j+1}(C), g_{j+1}(C - p_j) + b_j \}$$

Dependiendo de que el objeto j-ésimo entre o no en la solución
(sólo puede entrar si $c - p\,j \geq 0$)

Ojo, el cálculo en este caso se haría hacia atrás

Programación Dinámica (Dynamic Programming)
Ejemplos

Ejemplo: Mochila indivisible con peso y beneficio.

Hacemos una tabla con las variables del problema (los pesos de los objetos y la capacidad máxima), la rellenamos de izquierda a derecha y de arriba abajo.

$p_1 = 9$ $p_2 = 6$ $p_3 = 5$ $C = 15$

$b_1 = 38$ $b_2 = 40$ $b_3 = 24$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$p_1 = 9$	0	0	0	0	0	0	0	0	0	38	38	38	38	38	38	38
$p_2 = 6$	0	0	0	0	0	0	40	40	40	40	40	40	40	40	40	78
$p_3 = 5$	0	0	0	0	0	24	40	40	40	40	40	64	64	64	64	78

Obtenemos el valor óptimo, desde ahí navegando hacia atrás vemos qué decisiones se han tomado.