3. PROGRAMACIÓN DINÁMICA

La programación dinámica es una técnica poderosa para implementar de manera eficiente algoritmos recursivos almacenando resultados parciales y reutilizándolos cuando sea necesario.

La programación dinámica funciona de manera eficiente cuando:

- Subproblemas: Debe haber una manera de dividir el problema de optimización global en subproblemas, cada uno con una estructura similar al problema original, pero de menor tamaño.
- Subestructura óptima: Una solución óptima a un problema debe ser una composición de soluciones óptimas de subproblemas, utilizando una operación de combinación relativamente simple.
- Subproblemas repetidos: el algoritmo recursivo resuelve una pequeña cantidad de subproblemas distintos, pero se resuelven repetidamente muchas veces.

Esta última propiedad nos permite aprovechar la memorización, almacenar valores intermedios utilizando la estructura de datos diccionario adecuada y reutilizarlos cuando sea necesario.

Guia para implementar la programación dinámica

1. Caracterizar la estructura de los subproblemas: asegurarse de que el espacio de los subproblemas no sea exponencial. Definir variables.
2. Definir recursivamente el valor de una solución óptima: encontrar la recurrencia correcta, con solución a un problema mayor en función de las soluciones de subproblemas.
3. Calcular, memorizar / ascender, el costo de una solución: usando la fórmula recursiva, tabular soluciones a problemas menores, hasta llegar al valor del problema completo.
4. Construir una solución óptima: calcular información adicional para llegar desde la solución óptima hasta el valor óptimo.

Distancias de grafos

Un ciclo es un camino que comienza y finaliza en el mismo vértice.

Sea G = (V, E, w) un digrafo ponderado. Se puede definir una distancia entre todos los pares de vértices u, v ∈ V (G) si y sólo si G no tiene ciclos de peso negativos.

Si un grafo G no es dirigido, consideramos cada arista como doblemente dirigida y asignamos el mismo peso a ambas direcciones.

Si el grafo o digrafo no está ponderado, asignamos a cada arista un peso de 1. En este caso, el peso de un camino coincide con su longitud.

Sea G = (V, E, w) tal que, para u, v ∈ V, δ(u, v) se pueda definir. Para u, v, z ∈ V(G), δ(u, v) ≤ δ(u, z) + δ(z, v).

Single Source Shortest Path (SSSP): Dado G = (V, E, w) y s ∈ V, encontrar el camino más corto desde s hasta cada otro vértice en G, si existe.

ALGORITMO 10: DIJKSTRA'S ALGORITHM

- Definición: encuentra el camino más corto desde s hasta otro vértice en G, sólo funciona para pesos positivos.
- Coste: O(m lg n)

ALGORITMO 11: BELLMAN-FORD'S ALGORITHM

- Definición: Funciona para pesos generales y detecta si se puede definir la distancia, es decir, detecta la existencia de ciclos negativos.
- Coste: O(n*m) n = |V| y m = |E|

(*) topological sort -> O(n + m)

All Pairs Shortest Paths (APSP): Dado G = (V, E, w) sin ciclos de peso negativos, para cada u, v ∈ V(G), encontrar el camino más corto de u a v si existe.

ALGORITMO 12: FLOYD-WARSHALL'S ALGORITHM

- Definición: Utiliza la programación dinámica y toma como entrada la matriz de distancia de peso de G.
- Coste: O(n^3)

ALGORITMO 13: JOHNSON'S ALGORITHM

- Definición: un algoritmo eficiente para grafos dispersos.
- Coste: O(nm + n^2 * lg n)

SSSP no negative weight cycles accessible form s.

	Dijkstra	BF
$w \geq 0$	$O(m + n \lg n)$	$O(nm)$
$w \in \mathbb{Z}$	NO	$O(nm)$

APSP no negative weight cycles.

	Dijkstra	BF	FW	Johnson
$w \geq 0$	$O(nm + n^2 \lg n)$	$O(n^2 m)$	$O(n^3)$	$O(nm + n^2 \lg n)$
$w \in \mathbb{R}$	NO	$O(n^2 m)$	$O(n^3)$	$O(nm + n^2 \lg n)$