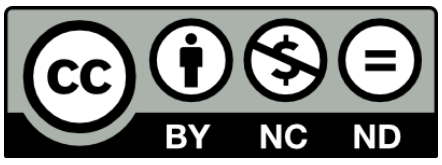
Algorítmica

Los Del DGIIM, `losdeldgiim.github.io`

Doble Grado en Ingeniería Informática y Matemáticas
Universidad de Granada



Algorítmica

Los Del DGIIM, `losdeldgiim.github.io`

José Juan Urrutia Milán
Arturo Olivares Martos

Granada, 2023-2024

Índice general

1. Programación Dinámica

- Para problemas en los que necesitamos estados anteriores (en fibonacci, para calcular fibonacci(n) necesitamos tener fibonacci($n-1$) y fibonacci($n-2$), y para fibonacci($n-1$) necesitamos, ...).
- En el camino aparecen requisitos que se repiten (necesitamos calcular varias veces fibonacci(k)). En vez de calcularlo todas las veces, calcularlo una sola vez. Para calcular fibonacci(6) necesitamos 5 veces fibonacci(2).
- Antes de calcular el subproblema, mira si lo tienes ya resuelto (si lo tiene, lo usa y si no lo tiene, lo calcula).
- Es necesaria una estructura para almacenar las soluciones a los subproblemas, con la finalidad de ahorrar llamadas recursivas.

Ejemplos de dónde usar programación dinámica son:

- Fibonacci.
- Calcular números combinatorios.
- Calcular potencias naturales.
- Cualquier problema con solapamiento de subproblemas (encontramos subproblemas que se repiten).

Una cosa es la programación dinámica y otra es la memorización:

Memorización. Almacenamos en una estructura (como un diccionario) los resultados.

Programación dinámica. Una vez que los hemos almacenado, buscamos un patrón para ver cómo se completan las soluciones de alguna forma más eficiente (quitando sobrecarga por la recursividad). Buscamos una forma de rellenar la estructura de datos.

Cuando aplicar programación dinámica

Normalmente para problemas de optimización (minimizar o maximizar). La solución al problema la tenemos que ver como un proceso de selección de varias etapas.

- Se aplica a problemas que pueden suponer un alto coste computacional que dispone de subestructuras optimales que se solapan (se repiten a lo largo del cálculo de la solución).

La eficiencia del algoritmo suele ser polinomial. Normalmente suele ser $O(n \cdot m)$ donde n es el tamaño de la estructura de datos y m el tiempo para cada casilla.

Comparación con Divide y Vencerás

Divide y vencerás.

- Se aplica a subproblemas independientes.
- La técnica suele ser descendente.

Programación dinámica.

- Se aplica a subproblemas que se solapan (que se resuelven más de una vez).
- La técnica suele ser ascendente.
- Asegura optimilidad pero puede llevar a un algoritmo que no sea eficiente.

Teorema 1.1 (Principio de Optimalidad de Bellman)**.** *Una solución óptima está compuesta de subsoluciones óptimas.*

Cuando se cumpla el principio, se podrá utilizar la programción dinámica.

Ejemplo. Un ejemplo que no cumple el Principio de Optimalidad de Bellman es el problema del camino más largo entre dos nodos en un grafo.

Por tanto, no podemos aplicar programación dinámica.

1.1. Pasos para desarrollar un algoritmo

1. Plantear la solución a un problema como una secuencia de decisiones.
2. Ver que se verifica el principio de optimalidad 1.1.
3. Plantear la solución como una función recursiva y ver la tipología de los subprobelmas.
4. Ver cómo un problema grande se puede calcular a partir de los problemas más pequeños.
5. Tratar de buscar un enfoque ascendente (resolver problemas pequeños y resolver problemas mayores).

Ante un problema del estilo buscar un camino óptimo, es capaz de decir el costo del camino pero no de decir el camino. Para ello:

- O se puede deducir el camino a partir del costo.
- O apuntar en una tabla auxiliar las decisiones tomadas.

2. Relaciones de Problemas