

Programación dinámica

Pedro O. Pérez M., MTI

Análisis y diseño de algoritmos
Tecnológico de Monterrey

pperezm@tec.mx

03-2019

Contenido

Introducción

Un ejemplo para empezar

Definición

La programación dinámica, al igual que dividir y conquistar, resuelve problemas combinando soluciones a subproblemas; pero a diferencia de esta, se aplica cuando los subproblemas se solapan, es decir, cuando comparten problemas más pequeños. Aquí la técnica cobra importancia, ya que calcula cada subproblema una sola vez; esto es, parte del principio de no calcular dos veces la misma información. Por tanto, utiliza estructuras de almacenamiento como vectores, tablas, arreglos, archivos, con el fin de almacenar los resultados parciales a medida que se resuelven los subcasos que contribuyen a la solución definitiva.

- ▶ Es una técnica ascendente que, normalmente, empieza por los subcasos más pequeños y más sencillos. Combinando sus soluciones, obtenemos las respuestas para los subcasos cada vez más grandes, hasta que llegamos a la solución del problema original.
- ▶ Se aplica muy bien a problemas de optimización. El mayor número de aplicaciones se encuentra en problemas que requieren maximización o minimización, ya que se pueden hallar múltiples soluciones y así evaluar para hallar la óptima.

Forma general

La forma general de las soluciones desarrolladas mediante programación dinámica requiere los siguientes pasos:

1. Plantear la solución, mediante una serie de decisiones que garanticen que será óptima, es decir, que tendrá la estructura de una solución óptima.
2. Encontrar una solución recursiva de la definición.
3. Calcular la solución teniendo en cuenta una tabla en la que se almacenen soluciones a problemas parciales para su reutilización, y así evitar un nuevo cálculo.
4. Encontrar la solución óptima utilizando la información previamente calcular y almacenada en las tablas.

Principio de optimalidad de Bellman: Cualquier subsecuencia de decisiones de una secuencia óptima de decisiones que resuelve un problema también debe ser óptima respecto al subproblema que resuelve.

Procedure 1 FIBONACCI

Input: $n : \text{Integer}$

if $n < 1$ **then**

return -1

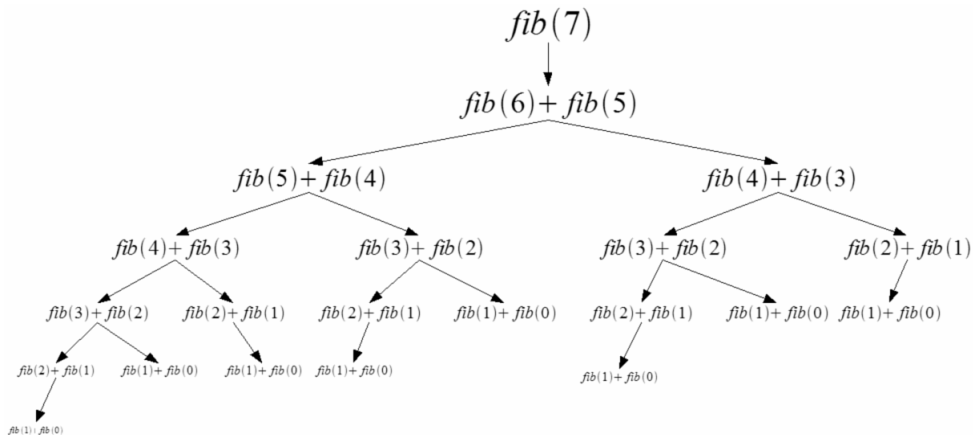
else if $n = 1$ **or** $n = 2$ **then**

return 1

else

return $FIBONACCI(n - 1) + FIBONACCI(n - 2)$

end if



Procedure 2 FIBONACCI_WITH_MEMORY1

Input: n : Integer, A : Array

if $n < 1$ then

return -1

else if $n = 1$ or $n = 2$ then

return 1

else if $A[n] <> -1$ then

return $A[n]$

else

$A[n] \leftarrow FIBONACCI(n - 1) + FIBONACCI(n - 2)$

return $A[n]$

end if

Procedure 3 FIBONACCI_WITH_MEMORY2

Input: n : Integer, A : Array

if $n < 1$ **then**

return -1

else

$A[1] \leftarrow 1$

$A[2] \leftarrow 1$

for $i \leftarrow 3$ **to** n **do**

$A[i] \leftarrow A[i - 1] + A[i - 2]$

end for

end if
