Programación dinámica

Pedro O. Pérez M., MTI

Análisis y diseño de algoritmos
Tecnológico de Monterrey

pperezm@tec.mx

03-2019

Contenido

Definición

La programación dinámica, al aigual que dividir y conquistar, resuelve problemas combinanco 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 hallas la óptima.

Forma general

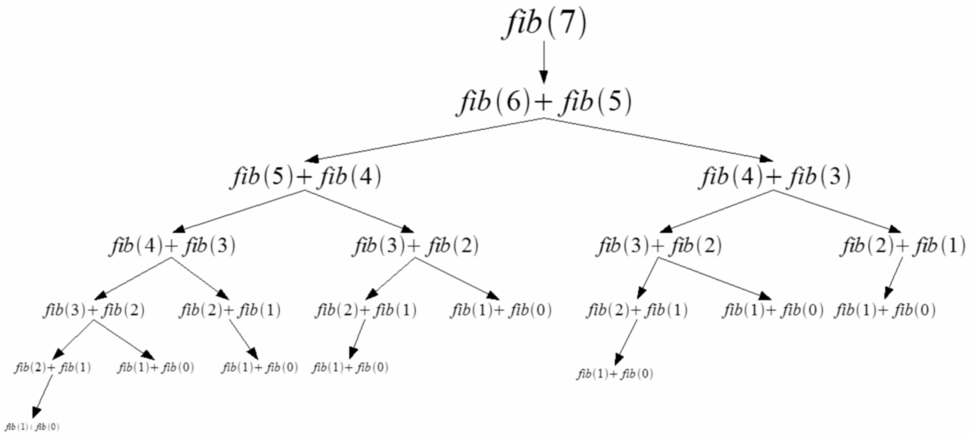
La forma general de las soluciones desarrolladas mediante programación dinámica requiere los siguientes pasos:

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 : 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
```


fib(7)
fib(6)+ fib(5)
fib(5)+ fib(4)
fib(4)+ fib(3)
fib(4)+ fib(3)
fib(3)+ fib(2)
fib(3)+ fib(2)
fib(2)+ fib(1)
fib(3)+ fib(2)
fib(2)+ fib(1)
fib(2)+ fib(1)
fib(1)+ fib(0)
fib(2)+ fib(1)
fib(1)+ fib(0)
fib(1)+ fib(0)
fib(2)+ fib(1)
fib(1)+ fib(0)
fib(1)+ fib(0)
fib(1)+ fib(0)
fib(1)+ fib(0)

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] ← FIBONACCI(n − 1) + FIBONACCI(n − 2)
    return A[n]
end if
```

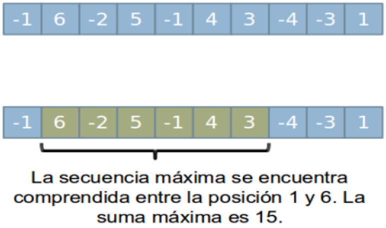
Procedure 3 FIBONACCI_WITH_MEMORY2

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

Secuencia de suma máxima

Dado un arreglo de n números enteros positivos y negativos, encontrar los i elementos del arreglo cuya suma se la máxima posible.



Procedure 4 MAX_SUM(A:Array, n:Integer)

Input: $n : Integer, A : Array$

$sum \leftarrow 0$

$ans \leftarrow 0$

for $i \leftarrow 1$ to n **do**

  $sum \leftarrow sum + A[i]$

  $ans \leftarrow MAX(ans, sum)$

  if $sum < 0$ **then**

    $sum \leftarrow 0$

  end if

end for

Cambio de monedas

Dado un sistema monetario S con N monedas de diferentes denominaciones y una cantidad de cambio C, calcular el menor número de monedas del sistema monetario S equivalente a C.

Ejemplos:

- **Input** : s[] = 1, 3, 4 c = 6
  Output : 2
  Explanation : The change will be (3 + 3) = 2

$$C[j] = \begin{cases} \infty & \text{if } j < 0, \\ 0 & \text{if } j = 0, \\ 1 + \min\limits_{1 \le i \le k} \{C[j - d_i]\} & \text{if } j \ge 1 \end{cases}$$

Procedure 5 COIN_CHANGE

Input: $S : Array, c : Integer$

```
Aux : Array[0, c]
Aux[0] ← 0
for j ← 1 to S.length do
    for j ← S[i] to c do
        Aux[j] ← MIN(1 + Aux[j − S[i]], Aux[j])
    end for
end for
return Aux[c]
```

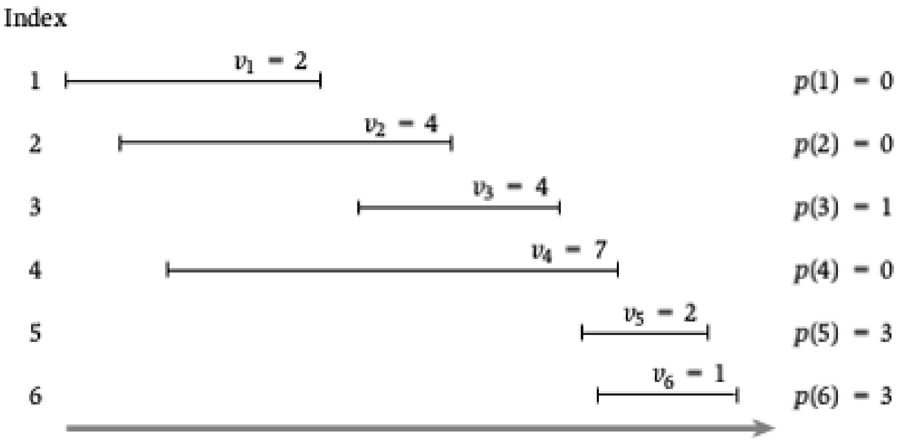
Programación de actividades

Te dan N actividades con sus tiempos de inicio (S_i) y finalización (F_i). Selecciona el número máximo de actividades que puede realizar una sola persona, asumiendo que una persona solo puede trabajar en una sola actividad a la vez. [1]

Ejemplo:

- **Input** :
  start[] = (1, 3, 0, 5, 8, 5)
  finish[] = (2, 4, 6, 7, 9, 9)
  Output : 4

[1] `https://goo.gl/1RG25M`


Index
1
$v_1 = 2$
$p(1) = 0$
2
$v_2 = 4$
$p(2) = 0$
3
$v_3 = 4$
$p(3) = 1$
4
$v_4 = 7$
$p(4) = 0$
5
$v_5 = 2$
$p(5) = 3$
6
$v_6 = 1$
$p(6) = 3$

Procedure 6 ACTIVITIES_SELECTION

Input: $A : Activity\_Array$

$Aux : Array[0..A.length]$

$Aux[0] \leftarrow 0$

for $i \leftarrow 1$ to $A.length$ **do**

$Aux[i] \leftarrow MAX(A[i] + Aux[opt(i)], Aux[j-1])$

end for