Programación dinámica

Pedro O. Pérez M., PhD.

Análisis y diseño de algoritmos
Tecnológico de Monterrey

pperezm@tec.mx

06-2020

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 encuentran en problemas que requieren maximización o minimización, ya que se pueden hallar múltiples soluciones y así evaluar para determinar cuál es la óptima.

Forma general

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

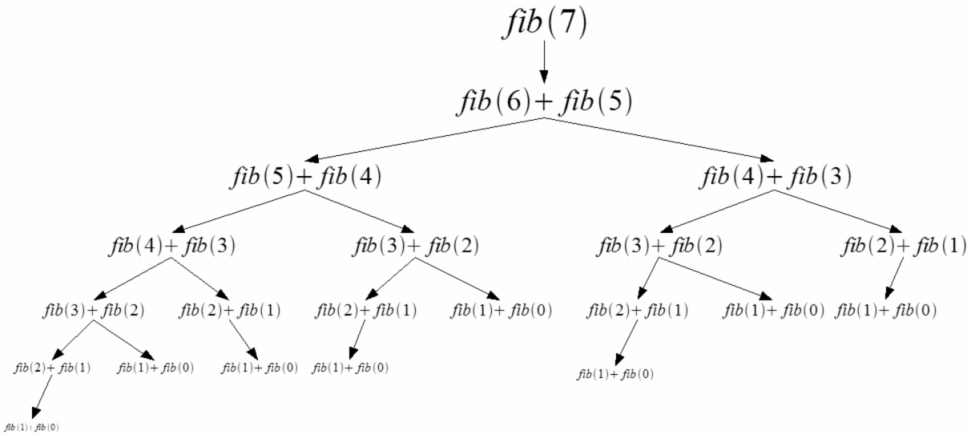
1. Plantear la solución, mediante una serie de decisiones que garanticen que será óptima, es decir, que tendrá la estructura de una solución óptima.
2. Encontrar una solución recursiva de la definición.
3. Calcular la solución teniendo en cuenta una tabla en la que se almacenen soluciones a problemas 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)

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
```

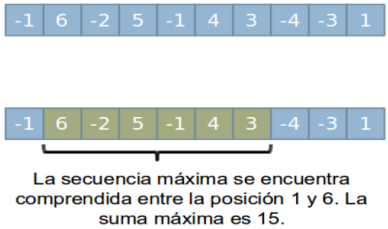
Procedure 3 FIBONACCI_WITH_MEMORY2

Input: n : *Integer*, A : *Array*

```
if n < 1 then
    return −1
else
    A[1] ← 1
    A[2] ← 1
    for i ← 3 to n do
        A[i] ← 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)

Input: $n : Integer, A : Array$
  $sum \leftarrow 0$
  $ans \leftarrow 0$
  for $i \leftarrow 1$ to $A.length$ **do**
    $sum \leftarrow sum + A[i]$
    $ans \leftarrow MAX(ans, sum)$
    if $sum < 0$ **then**
      $sum \leftarrow 0$
    end if
  end for
  return ans

The jackpot [1]

Como Manuel quiere hacerse rico rápidamente y sin demasiado trabajo, decidió hacer una carrera en el juego. Inicialmente, planea estudiar las ganancias y pérdidas de los jugadores, para poder identificar patrones de victorias consecutivas y elaborar una estrategia de ganar-ganar. Pero Manuel, tan inteligente como cree que es, no sabe programar computadoras. Entonces te contrató para escribir programas que lo ayudarán a elaborar su estrategia.
Tu primera tarea es escribir un programa que identifique la ganancia máxima posible de una secuencia de apuestas. Una apuesta es una cantidad de dinero y está ganando (y esto se registra como un valor positivo) o perdiendo (y esto se registra como un valor negativo).

[1] https://onlinejudge.org/external/106/10684.pdf

Entrada

La primera línea consta de un número positivo N <= 10000, que da la longitud de la secuencia, seguido de N enteros. Cada apuesta es un número entero mayor que 0 y menor que 1000.

Salida

La salida muestra la solución correspondiente. Si no hay posibilidad de ganar dinero, la salida es el mensaje Racha perdedora".

Ejemplo de entrada

5
12 -4
-10 4
9

Ejemplo de salida

The maximum winning streak is 13.

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: $S : Array, c : Integer$

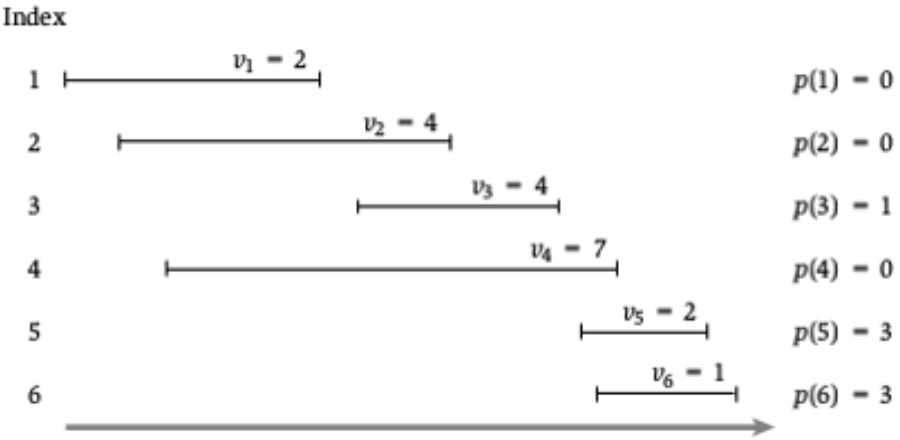
$Aux : Array[0, c]$
$INIT\_ARRAY(A, \infty)$
$Aux[0] \leftarrow 0$
for $i \leftarrow 1$ to $S.length$ **do**
  for $j \leftarrow S[i]$ to c **do**
    $Aux[j] \leftarrow MIN(1 + Aux[j - S[i]], Aux[j])$
  end for
end for
return Aux[c]

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. [2]
Ejemplo:

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

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


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

Procedure 6 ACTIVITIES_SELECTION

Input: $A : Activity\_Array, Opt : Array$

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

$Aux[0] \leftarrow 0$

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

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

end for

Números feos

Los números feos son números cuyos únicos factores primos son 2, 3 o 5 (o combinación de ellos). La secuencia 1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, ... muestra los primeros 11 números feos. Por convención, se incluye 1.

Como vemos en la diapositiva anterior, la secuencia de los primeros números feos es 1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, ... Es más fácil de determinar la forma en que se genera está secuencia si la dividimos entre sus tres factores:

- 1x2, 2x2, 3x2, 4x2, 5x2...
- 1x3, 2x3, 3x3, 4x3, 5x3...
- 1x5, 2x5, 3x5, 4x5, 5x5...

¿Cuál se imprime primero?

Procedure 7 UGLY_NUMBER

Input: n : *Integer*

A : *Array*$[n]$
$A[1] = 1$
$i2 \leftarrow 1$
$i3 \leftarrow 1$
$i5 \leftarrow 1$
$ugly\_number \leftarrow 0$
for $i \leftarrow 2$ to n **do**
  next slide
end for
return *ugly_number*

$ugly\_number = MIN((A[i2] * 2), (A[i3] * 3), (A[i5] * 5))$
$A[i] \leftarrow ugly\_number$
if $ugly\_number == (A[i2] * 2)$ **then**
  $i2 \leftarrow i2 + 1$
end if
if $ugly\_number == (A[i3] * 3)$ **then**
  $i3 \leftarrow i3 + 1$
end if
if $ugly\_number == (A[i5] * 5)$ **then**
  $i5 \leftarrow i5 + 1$
end if

¿Qué es una subsecuencia?

Considera una cadena A = [a, b, c, d]. Una subsecuencia(también llamada subcadena) se obtiene eliminando 0 o más símbolos (no necesariamente consecutivos) de A.
Por ejemplo:

- Los ejemplos [a, b, c, d] , [a], [b], [c], [d], [a, d], [a, c], [b, d] son subsecuencias de A.
- Pero [d, b], [d, a], [d, a, c] no son subsecuencias de A.

Subsecuencia creciente más larga

Dado un arreglo de n símbolos comparables, dar la subsecuencia creciente más larga. Por ejemplo, A = [-7, 10, 9, 2, 3, 8, 8, 1], la subsecuencia creciente más larga es [-7, 2, 3, 8].

Procedure 8 LIS

Input: $A : Array$

```
Aux : Array
Aux[1] ← 1
for i ← 2 to n do
    count ← 0
    for j ← 1 to (i − 1) do
        if A[j] < A[i] then
            count ← MAX(count, Aux[j])
        end if
    end for
    Aux[i] ← count + 1
end for
```

```
count ← 0
for i ← 1 to n do
  count ← MAX(count, Aux[i])
end for
return count
```

Testing the CATCHER [3]

Un contratista militar del Departamento de Defensa acaba de completar una serie de pruebas preliminares para un nuevo misil defensivo llamado FREEDOM, que es capaz de interceptar múltiples misiles ofensivos entrantes. Se supone que el FREEDOM es un notable misil defensivo. Puede moverse hacia adelante, lateralmente y hacia abajo a velocidades muy rápidas, y puede interceptar un misil ofensivo sin sufrir daños. Pero tiene un defecto importante. Aunque puede dispararse para alcanzar cualquier elevación inicial, no tiene poder para moverse más alto que el último misil que ha interceptado.
Las pruebas que completó el contratista fueron simulaciones por computadora de campo de batalla y condiciones de ataque hostil. Como solo eran preliminares, las simulaciones solo probaron la capacidad de movimiento vertical del FREEDOM. En cada simulación, el FREEDOM fue disparado contra una secuencia de misiles ofensivos que llegaban a intervalos de tiempo fijos. La única información disponible para el FREEDOM para cada misil entrante era su altura en el punto donde podía ser interceptada y donde aparecía en la secuencia de misiles. Cada misil entrante para una ejecución de prueba se representa en la secuencia solo una vez.

[3] https://onlinejudge.org/external/2/231.pdf

El resultado de cada prueba se informa como la secuencia de misiles entrantes y el número total de esos misiles que son interceptados por el FREEDOM en esa prueba.
La Oficina de Contabilidad General quiere asegurarse de que los resultados de la prueba de simulación presentados por el contratista militar sean alcanzables, dadas las limitaciones del FREEDOM. Escribir una solución que tome datos de entrada que representen el patrón de misiles entrantes para una prueba y genere la cantidad máxima de misiles que el FREEDOM puede interceptar para esa prueba. Para cualquier misil entrante en la prueba, el FREEDOM puede interceptarlo si y solo si cumple una de estas dos condiciones:
1. El misil entrante es el primer misil que se intercepta en esta prueba.
-o-
2. El misil fue disparado después del último misil que fue interceptado y no es más alto que el último misil que fue interceptado.

Entrada

Los datos de entrada para cualquier prueba consisten en una secuencia de uno o más enteros no negativos, todos los cuales son menores o iguales a 32,767, que representan las alturas de los misiles entrantes (el patrón de prueba). El último número en cada secuencia es -1, lo que significa el final de los datos para esa prueba.

Salida

La salida consta del número máximo de misiles entrantes que el FREEDOM posiblemente podría interceptar para la prueba. Nota: El número de misiles para cualquier prueba dada no está limitado. Si la solución se basa en un algoritmo ineficiente, es posible que no se ejecute en el tiempo asignado.

Ejemplo de entrada

389
207
155
300
299
170
158
65
-1

Ejemplo de salida

6

Cuenta el número de formas posibles

Queremos dar un cambio de C pesos y tenemos un suministro infinito de monedas de valor S = [S_1, S_2, ..., S_n] pesos, ¿de cuántas maneras podemos dar el cambio? Por ejemplo para C = 4, S = [1, 2, 3], existen cuatro formas de dar el cambio: [1, 1, 1, 1], [1, 1, 2], [2, 2], [1, 3]. Para C = 10, S = [2, 5, 3, 6], existen cinco soluciones: [2, 2, 2, 2, 2], [2, 2, 3, 3], [2, 2, 6], [2, 3, 5], [5, 5].

$$COUNT(type, value) = \begin{cases} 0, & \text{si } value < 0, \\ 1, & \text{si } value = 0, \\ COUNT(type + 1, value) & \\ +COUNT(type, value - coins[type]), & \text{si } value > type. \end{cases}$$

Procedure 9 COUNT

Input: S : *Array*, c : *Integer*

```
table : Matrix[0...c][S.length]
for i ← 0 to c do
    table[0][i] ← 1
end for
for i ← 1 to c do
    for j ← 1 to S.length do
        NEXT SLIDE
    end for
end for
return table[c][n]
```

```
acum ← 0
/* Can the currency S [j] be used to give the change? */
if (i − S[j]) ≥ 0 then
    acum ← table[(i − S[j])][j]
end if
/* If we do not use S[j] */
if j ≥ 1 then
    acum ← acum + table[i][j − 1]
end if
table[i][j] ← acum
```

Let Me Count The Ways [4]

Después de realizar una compra en una gran tienda por departamentos, el cambio de Mel fue de 17 centavos. Recibió 1 centavo, 1 níquel y 2 centavos. Más tarde ese día, estaba comprando en una tienda de conveniencia. Nuevamente su cambio fue de 17 centavos. Esta vez recibió 2 monedas de 5 centavos y 7 centavos. Él comenzó a preguntarse "¿En cuántas tiendas puedo comprar y recibir un cambio de 17 centavos en una configuración diferente de monedas? Después de una lucha mental adecuada, decidió que la respuesta era 6. Luego lo retó a considerar el problema general.
Escriba un programa que determine la cantidad de combinaciones diferentes de monedas estadounidenses (centavo: 1c, níquel: 5c, dime: 10c, trimestre: 25c, medio dólar: 50c) que se pueden usar para producir una cantidad de dinero determinada.

[4] https://onlinejudge.org/external/3/357.pdf

Entrada

La entrada consistirá en un conjunto de números entre 0 y 30000 inclusive, uno por línea en el archivo de entrada.

Salida

La salida consistirá en la declaración apropiada de la selección a continuación en una sola línea en el archivo de salida para cada valor de entrada. El número m es el número que calcula su programa, n es el valor de entrada.

Hay muchas formas de producir n centavos de cambio.

Solo hay 1 forma de producir n centavos de cambio.

Ejemplo de entrada

17
11
4

Ejemplo de salida

There are 6 ways to produce 17 cents change.
There are 4 ways to produce 11 cents change.
There is only 1 way to produce 4 cents change.

Mochila 0/1

Sean N objetos no fraccionables de pesos w_i, y beneficios v_i, y sea C el peso máximo que puede llevar la mochila. El problema consiste en llenar la mochila con objetos, tal que se maximice el beneficio. Por ejemplo, W = [10, 4, 6, 12], V = [100, 70, 50, 10], y C = 12. ¿Cuál es la beneficio máximo que podemos obtener?

Procedure 10 KNAPSACK

Input: $W, V : Array, C : Integer$

```
M : Matrix[0...S.length][0..C]
n ← W.length
INIT(M, 0)
for i ← 1 to n do
    for j ← 1 to C do
        /* There is not space in the container */
        if j < W[i] then
            M[i][j] ← M[i − 1][j]
        else
            M[i][j] ← MAX(V[i] + M[i − 1][j − W[i]], M[i − 1][j])
        end if
    end for
end for
return M[n][C]
```

SuperSale [5]

Hay una Superventa en un SuperHiperMarket. Cada persona puede llevar solo un objeto de cada tipo, es decir, un televisor, una zanahoria, pero a un precio muy bajo. Vamos con toda una familia a ese SuperHiperMarket.
Cada persona puede tomar tantos objetos como pueda llevar a cabo desde la Superventa. Hemos dado una lista de objetos con precios y su peso. También sabemos cuál es el peso máximo que toda persona puede soportar. ¿Cuál es el valor máximo de los objetos que podemos comprar en SuperSale?

[5] https://onlinejudge.org/external/101/10130.pdf

Entrada

La entrada consta de casos de prueba T. El número de ellos (1 <= T <= 1000) se da en la primera línea del archivo de entrada. Cada caso de prueba comienza con una línea que contiene un solo número entero N que indica el número de objetos (1 <= N <= 1000). Luego sigue N líneas, cada una con dos enteros: P y W. El primer entero (1 <= P <= 100) corresponde al precio del objeto. El segundo entero (1 <= W <= 30) corresponde al peso del objeto. La siguiente línea contiene un número entero (1 <= G <= 100) es el número de personas en nuestro grupo. Las siguientes líneas G contienen el peso máximo (1 <= MW <= 30) que puede soportar esta i-ésima persona de nuestra familia (1 <= i <= G).

Salida

Para cada caso de prueba, su programa tiene que determinar un número entero. Imprima el valor máximo de los bienes que podemos comprar con esa familia.

Ejemplo de entrada

```
6
64 26
85 22
52 4
99 18
39 13
54 9
4
23
20
20
26
```

Ejemplo de salida

```
514
```

Subsecuencia común más larga

Dado dos cadenas, A y B, determinar la subsecuencia común más larga de ambas cadenas. Por ejemplo, si A = [c, d, c, c, f, g, e] y B = [e, c, c, e, g, f, e], ¿cuál la subsecuencia común más larga?

Ahora veamos como se plantea el problema: considere dos secuencias A = $[a_1, a_2, ..., a_m]$ y B = $[b_1, b_2, ..., b_n]$. Para resolver el problema nos fijaremos en los últimos dos símbolos: a_m y b_n . Como podemos ver hay dos posibilidades:

- Caso 1: $a_m = b_n$. En este caso, la subsecuencia común más larga debe contener a_m . Simplemente basta encontrar la subsecuencia común más larga de $[a_1, a_2, ..., a_{m-1}]$ y $[b_1, b_2, ..., b_{n-1}]$.
- Caso 2: $a_m \neq b_n$. En este caso, puede hacerse corresponder $[a_1, a_2, ..., a_m]$ con $[b_1, b_2, ..., b_{n-1}]$ y también $[a_1, a_2, ..., a_{m-1}]$ con $[b_1, b_2, ..., b_n]$, y nos quedamos con el mayor de los dos resultados.

Procedure 11 LCS

Input: $A, B : Array$
  $M : Matrix[0..A.length][0..B.length]$
  $m \leftarrow A.length$
  $n \leftarrow B.length$
  $INIT(M, 0)$
  for $i \leftarrow 1$ to m **do**
    for $j \leftarrow 1$ to n **do**
      if $A[i] = B[j]$ **then**
        $M[i][j] \leftarrow 1 + M[i-1][j-1]$
      else
        $M[][] \leftarrow MAX(M[i-1][j], M[i][j-1])$
      end if
    end for
  end for
  return $M[m][n]$

All-Pairs Shortest Path

El algoritmo Floyd – Warshall permite encontrar las rutas más cortas en un grfo ponderado con costos positivos o negativos (pero sin ciclos negativos). Una sola ejecución del algoritmo encontrará los costos de las rutas más cortas entre **todos los pares de vértices**. Aunque no devuelve detalles de las rutas en sí, es posible reconstruir las rutas con simples modificaciones al algoritmo. Las versiones del algoritmo también se pueden usar para encontrar el cierre transitivo de una relación R.

Procedure 12 VERSION1

Input: M : *Adjacent_Matrix*

```
for k ← 1 to M.length do
    for i ← 1 to M.length do
        for j ← 1 to M.length do
            M[i][j] ← M[i][k] and M[k][j]
        end for
    end for
end for
```

Procedure 13 VERSION2

Input: M : *Adjacent_Matrix*

```
for k ← 1 to M.length do
    for i ← 1 to M.length do
        for j ← 1 to M.length do
            M[i][j] ← MIN(M[i][j], M[i][k] + M[k][j])
        end for
    end for
end for
```