Análisis de Algoritmos
Lectura: Técnicas de diseño de algoritmos: Programación Dinámica

Rodolfo Conde
rodolfo.conde@itam.mx

Instituto Tecnológico Autónomo de México
Maestría en Ciencias de la Computación

16 de octubre de 2020

Contenido

Diseño y arquitectura de algoritmos

En esta lectura continuamos practicando las técnicas de análisis de algoritmos en general y aprenderemos una nueva técnica de diseño de algoritmos: La Programación Dinámica. Esta técnica es muy parecida a la de Dividir y Conquistar, al dividir el problema en subproblemas y ser de naturaleza recursiva. Sin embargo, los subproblemas en la Programación Dinámica tienden a tener subsubproblemas en común, lo cual da lugar para aplicar otras técnicas y herramientas para optimizar los algoritmos.
Nuestra referencia principal es [Cor+09].

A continuación

Técnica: Programación Dinámica

Ejemplo: Multiplicar cadenas de matrices

Ejemplo: Un problema de Grafos

Notas y ejercicios

Arquitectura y diseño de algoritmos

Como mencionamos en la presentación pasada, en general nadie nos obliga a seguir un método especifico para construir un algoritmo, hay metodologías que nos ayudan a encontrar (o aproximarnos, en su defecto) al algoritmo más eficiente posible. Una de ellas es la Programación Dinámica. De hecho, es muy posible que en una forma indirecta, ya hayamos usado esta técnica para resolver algún problema.

¿Qué es la Programación Dinámica?

La Programación Dinámica es una técnica de diseño de algoritmos que resuelve los problemas al dividirlos en subproblemas (como en DyC), pero

¿Qué es la Programación Dinámica?

La Programación Dinámica es una técnica de diseño de algoritmos que resuelve los problemas al dividirlos en subproblemas (como en DyC), pero

- Los subproblemas no son ajenos, comparte subsubproblemas.

¿Qué es la Programación Dinámica?

La Programación Dinámica es una técnica de diseño de algoritmos que resuelve los problemas al dividirlos en subproblemas (como en DyC), pero

- Los subproblemas no son ajenos, comparte subsubproblemas.
- Para optimizar el algoritmo, se calculan las soluciones de los subsubproblemas y se guardan en una tabla.

¿Qué es la Programación Dinámica?

La Programación Dinámica es una técnica de diseño de algoritmos que resuelve los problemas al dividirlos en subproblemas (como en DyC), pero

- Los subproblemas no son ajenos, comparte subsubproblemas.
- Para optimizar el algoritmo, se calculan las soluciones de los subsubproblemas y se guardan en una tabla.
- La tabla nos ahorra tiempo al no volver a recalcular la solución de un subsubproblema.

¿Qué es la Programación Dinámica?

La Programación Dinámica es una técnica de diseño de algoritmos que resuelve los problemas al dividirlos en subproblemas (como en DyC), pero

- Los subproblemas no son ajenos, comparte subsubproblemas.
- Para optimizar el algoritmo, se calculan las soluciones de los subsubproblemas y se guardan en una tabla.
- La tabla nos ahorra tiempo al no volver a recalcular la solución de un subsubproblema.
- Intercambio Tiempo-Memoria.

Programación Dinámica: Tipos de problemas

La Programación Dinámica se aplica de forma típica a Problemas de Optimización. Estos problemas

- Pueden tener muchas soluciones.
- Se busca la solución más óptima (máximo ó mínimo).
- Puede haber más de una solución óptima.

Programación Dinámica: Pasos básicos

Al desarrollar un algoritmo por el método de Programación Dinámica, es muy común seguir los siguientes pasos:

Programación Dinámica: Pasos básicos

Al desarrollar un algoritmo por el método de Programación Dinámica, es muy común seguir los siguientes pasos:

1. Caracterizar la estructura de una solución óptima.

Programación Dinámica: Pasos básicos

Al desarrollar un algoritmo por el método de Programación Dinámica, es muy común seguir los siguientes pasos:

1. Caracterizar la estructura de una solución óptima.
2. Recursivamente definir el valor de una solución óptima.

Programación Dinámica: Pasos básicos

Al desarrollar un algoritmo por el método de Programación Dinámica, es muy común seguir los siguientes pasos:

1. Caracterizar la estructura de una solución óptima.
2. Recursivamente definir el valor de una solución óptima.
3. Calcular el valor (costo) de una solución óptima.

Programación Dinámica: Pasos básicos

Al desarrollar un algoritmo por el método de Programación Dinámica, es muy común seguir los siguientes pasos:

1. Caracterizar la estructura de una solución óptima.
2. Recursivamente definir el valor de una solución óptima.
3. Calcular el valor (costo) de una solución óptima.
4. Construir una solución óptima a partir de la información calculada.

Programación Dinámica: Pasos básicos

Ahora veamos unos ejemplos para ver en acción a la Programación Dinámica.

A continuación

Técnica: Programación Dinámica

Ejemplo: Multiplicar cadenas de matrices

Ejemplo: Un problema de Grafos

Notas y ejercicios

Multiplicación de matrices

Como sabemos, multiplicar matrices es una operación fundamental para resolver diversos problemas con algoritmos.

Multiplicación de matrices

Usualmente nos enfrentamos al problema de multiplicar dos matrices Cuadradas $A_1, A_2 \in M_{n\times n}(\mathbb{R})$

$$A_1 \cdot A_2$$

Multiplicación de matrices

Pero a veces también necesitamos calcular la multiplicación de una cantidad arbitraria de matrices $A_1, A_2, \ldots, A_q$ tal que A_i tiene dimensiones $n_i \times m_i$ and $m_i = n_{i+1}$ para $1 \leq i < q$.

$$A_1 \cdot A_2 \cdots A_q$$

Multiplicación de matrices

La multiplicación de matrices es asociativa, entonces en si, no importa en que orden efectuemos la multiplicación, siempre obtendremos el mismo resultado.

$$A_1 \cdot A_2 \cdots A_q$$

Multiplicación de matrices

Pero. . .

$$A_1 \cdot A_2 \cdots A_q$$

Multiplicación de matrices

Pero. . .¿Cambia la cantidad de multiplicaciones escalares que debemos realizar?

$$A_1 \cdot A_2 \cdots A_q$$

Multiplicación de matrices

Ejemplo 1

Consideremos:

$$A_1 \in M_{100\times 20}(\mathbb{R}),$$
$$A_2 \in M_{20\times 20}(\mathbb{R}),$$
$$A_3 \in M_{20\times 200}(\mathbb{R}).$$

Multiplicación de matrices

Ejemplo 1

Consideremos:

$$A_1 \in M_{100\times 20}(\mathbb{R}),$$
$$A_2 \in M_{20\times 20}(\mathbb{R}),$$
$$A_3 \in M_{20\times 200}(\mathbb{R}).$$

Y las posibles formas de asociar la multiplicación:

$$(A_1(A_2A_3)) \quad \text{y} \quad ((A_1A_2)A_3)$$

Multiplicación de matrices

Ejemplo 1

Consideremos:

$$A_1 \in M_{100\times 20}(\mathbb{R}),$$
$$A_2 \in M_{20\times 20}(\mathbb{R}),$$
$$A_3 \in M_{20\times 200}(\mathbb{R}).$$

Para la primera opción:

$$(A_1(A_2A_3))$$

Multiplicación de matrices

Ejemplo 1

Consideremos:

$$A_1 \in M_{100\times 20}(\mathbb{R}),$$
$$A_2 \in M_{20\times 20}(\mathbb{R}),$$
$$A_3 \in M_{20\times 200}(\mathbb{R}).$$

Para la primera opción:

$$(A_1 \quad \underbrace{(A_2 A_3)}_{20\cdot 20\cdot 200=80000} \quad)$$

Multiplicación de matrices

Ejemplo 1

Consideremos:

$$A_1 \in M_{100\times 20}(\mathbb{R}),$$
$$A_2 \in M_{20\times 20}(\mathbb{R}),$$
$$A_3 \in M_{20\times 200}(\mathbb{R}).$$

Para la primera opción:

$$\underbrace{(A_1(A_2A_3))}_{100\cdot 20\cdot 200=400000}$$

Multiplicación de matrices

Ejemplo 1

Consideremos:

$$A_1 \in M_{100\times 20}(\mathbb{R}),$$
$$A_2 \in M_{20\times 20}(\mathbb{R}),$$
$$A_3 \in M_{20\times 200}(\mathbb{R}).$$

Para la primera opción: El total de operaciones es $80000 \cdot 400000 = 32000000000$

$$(A_1(A_2A_3))$$

Multiplicación de matrices

Ejemplo 1

Consideremos:

$$A_1 \in M_{100\times 20}(\mathbb{R}),$$
$$A_2 \in M_{20\times 20}(\mathbb{R}),$$
$$A_3 \in M_{20\times 200}(\mathbb{R}).$$

Un calculo similar para la segunda opción nos da un total de operaciones de $40000 \cdot 400000 = 16000000000$

$$((A_1A_2)A_3)$$

Multiplicación de matrices

Ejemplo 1

Consideremos:

$$A_1 \in M_{100\times 20}(\mathbb{R}),$$
$$A_2 \in M_{20\times 20}(\mathbb{R}),$$
$$A_3 \in M_{20\times 200}(\mathbb{R}).$$

¡¡La segunda opción es mejor!!

$$((A_1 A_2)A_3)$$

Multiplicación de matrices: Formalización

El problema de Multiplicar matrices en cadena consta de los siguientes datos:

Multiplicación de matrices: Formalización

El problema de Multiplicar matrices en cadena consta de los siguientes datos:

Entrada: Una secuencia de enteros $\langle p_0, \ldots, p_n \rangle$ que representan la secuencia de matrices $\langle A_1, \ldots, A_n \rangle$ tal que $A_i \in M_{p_{i-1} \times p_i}(\mathbb{R})$.

Multiplicación de matrices: Formalización

El problema de Multiplicar matrices en cadena consta de los siguientes datos:

Entrada: Una secuencia de enteros $\langle p_0, \ldots, p_n \rangle$ que representan la secuencia de matrices $\langle A_1, \ldots, A_n \rangle$ tal que $A_i \in M_{p_{i-1} \times p_i}(\mathbb{R})$.

Salida: La forma de asociar el producto de todas las matrices, de tal manera que se minimice la cantidad de multiplicaciones escalares necesarias para calcular el producto de todas las matrices.

Multiplicación de matrices: Formalización

El problema de Multiplicar matrices en cadena consta de los siguientes datos:

Entrada: Una secuencia de enteros $\langle p_0, \ldots, p_n \rangle$ que representan la secuencia de matrices $\langle A_1, \ldots, A_n \rangle$ tal que $A_i \in M_{p_{i-1} \times p_i}(\mathbb{R})$.

Salida: La forma de asociar el producto de todas las matrices, de tal manera que se minimice la cantidad de multiplicaciones escalares necesarias para calcular el producto de todas las matrices.

Nota: No se pide multiplicar las matrices.

Multiplicación de matrices: Primer análisis

La primera opción es verificar por busqueda exaustiva. ¿De cuantas formas se pueden poner los parantesis para una secuencia de n matrices? Sea $P(n)$ el número de formas posibles.

Multiplicación de matrices: Primer análisis

Si $n = 1$, solo hay una forma. $P(1) = 1$

Multiplicación de matrices: Primer análisis

$n > 1$
Cuando hemos seleccionado donde poner los primeros parentesis, encerrando k matrices $A_i \cdots A_{i+k}$, debemos escoger una forma de poner parentesis para las otras $n - k$ matrices.

Multiplicación de matrices: Primer análisis

$n > 1$
Cuando hemos seleccionado donde poner los primeros parentesis, encerrando k matrices $A_i \cdots A_{i+k}$, debemos escoger una forma de poner parentesis para las otras $n - k$ matrices.
Y esto lo hacemos para todo $k = 1, \ldots, n - 1$.

Multiplicación de matrices: Primer análisis

$$P(n) = \begin{cases} 1 & n = 1 \\ \sum_{k=1}^{n-1} P(k)P(n-k) & n > 1 \end{cases} \tag{1}$$

Multiplicación de matrices: Primer análisis

$$P(n) = \begin{cases} 1 & n = 1 \\ \sum_{k=1}^{n-1} P(k)P(n-k) & n > 1 \end{cases} \tag{1}$$

Una recurrencia.

Multiplicación de matrices: De cuantas formas

No es difícil ver que una solución a la recurrencia

$$P(n) = \begin{cases} 1 & n = 1 \\ \sum_{k=1}^{n-1} P(k)P(n-k) & n > 1 \end{cases} \tag{2}$$

Multiplicación de matrices: De cuantas formas

No es difícil ver que una solución a la recurrencia

$$P(n) = \begin{cases} 1 & n = 1 \\ \sum_{k=1}^{n-1} P(k)P(n-k) & n > 1 \end{cases} \tag{2}$$

es (en pizarron) $P(n) = \Omega(2^n)$. (¡¡Exponencial!!)

Multiplicación de matrices: ¿Qué tal DyC?

¿DyC?

Multiplicación de matrices: ¿Qué tal DyC?

¿DyC? Tampoco es buena opción, pues estariamos recorriendo todo el arbol de formas posibles de asociar las matrices, lo cual esta relacionado con $P(n)$.

Multiplicación de matrices: Formalización

Vamos a intentar resolverlo con Programación Dinámica (PD).

PD Paso 1

El primer paso es caracterizar la estructura de una solución óptima. Es decir, si tenemos una solución optima para multiplicar $A_{i \ldots j}$ a partir de poner parentesis entre A_k y A_{k+1} para k tal que $i \leq k < j$

PD Paso 1

El primer paso es caracterizar la estructura de una solución óptima. Es decir, si tenemos una solución optima para multiplicar $A_{i \dots j}$ a partir de poner parentesis entre A_k y A_{k+1} para k tal que $i \leq k < j$ ¿Las formas en las que hemos colocado parentesis para multiplicar $A_{i \dots k}$ y $A_{k+1 \dots j}$ son óptimas?

PD Paso 1

El primer paso es caracterizar la estructura de una solución óptima. Es decir, si tenemos una solución optima para multiplicar $A_{i \cdots j}$ a partir de poner parentesis entre A_k y A_{k+1} para k tal que $i \leq k < j$ ¿Las formas en las que hemos colocado parentesis para multiplicar $A_{i \cdots k}$ y $A_{k+1 \cdots j}$ son óptimas?
(Vamonos al pizarrón).

PD Paso 2

El siguiente paso es encontrar una ecuación recursiva que define el valor de una solución óptima. Esta ecuación será:

PD Paso 2

El siguiente paso es encontrar una ecuación recursiva que define el valor de una solución óptima. Esta ecuación será:

$$m[i,j] = \begin{cases} 0 & i = j, \\ m[i,k] + m[k+1,j] + p_{i-1}p_kp_j & i < j. \end{cases} \tag{3}$$

PD Paso 2

El siguiente paso es encontrar una ecuación recursiva que define el valor de una solución óptima. Esta ecuación será:

$$\underbrace{m\,[i,j]}_{\text{Óptimo de } A_{i\ldots j}} = \begin{cases} 0 & i = j, \\ m\,[i,k] + m\,[k+1,j] + p_{i-1}p_kp_j & i < j. \end{cases} \tag{3}$$

PD Paso 2

El siguiente paso es encontrar una ecuación recursiva que define el valor de una solución óptima. Esta ecuación será:

$$m[i,j] = \begin{cases} 0 & i = j, \\ \underbrace{m[i,k]}_{\text{Óptimo de } A_{i \ldots k}} + m[k+1,j] + p_{i-1}p_k p_j & i < j. \end{cases} \quad (3)$$

PD Paso 2

El siguiente paso es encontrar una ecuación recursiva que define el valor de una solución óptima. Esta ecuación será:

$$m[i,j] = \begin{cases} 0 & i = j, \\ m[i,k] + \underbrace{m[k+1,j]}_{\text{Óptimo de } A_{k+1\cdots j}} + p_{i-1}p_k p_j & i < j. \end{cases} \tag{3}$$

PD Paso 2

El siguiente paso es encontrar una ecuación recursiva que define el valor de una solución óptima. Esta ecuación será:

$$m[i,j] = \begin{cases} 0 & i = j, \\ m[i,k] + m[k+1,j] + \underbrace{p_{i-1}p_kp_j}_{\text{Costo de } A_{i\dots k}\cdot A_{k+1\dots j}} & i < j. \end{cases} \tag{3}$$

PD Paso 2

El siguiente paso es encontrar una ecuación recursiva que define el valor de una solución óptima. Esta ecuación será:

$$m[i,j] = \begin{cases} 0 & i = j, \\ m[i,k] + m[k+1,j] + p_{i-1}p_kp_j & i < j. \end{cases} \tag{3}$$

¡¡ Suponiendo que conocemos k !!

PD Paso 2

El siguiente paso es encontrar una ecuación recursiva que define el valor de una solución óptima. Esta ecuación será:

$$m[i,j] = \begin{cases} 0 & i = j, \\ \min_{i \leq k < j}\{m[i,k] + m[k+1,j] + p_{i-1}p_kp_j\} & i < j. \end{cases} \tag{3}$$

- ¿Cómo encontrar k para cada subproblema?

PD Paso 2

El siguiente paso es encontrar una ecuación recursiva que define el valor de una solución óptima. Esta ecuación será:

$$m[i,j] = \begin{cases} 0 & i = j, \\ \text{mín}_{i \le k < j}\{m[i,k] + m[k+1,j] + p_{i-1}p_k p_j\} & i < j. \end{cases} \tag{3}$$

- ¿Cómo encontrar k para cada subproblema?
- ¿Cuántas posibilidades hay en total?

PD Paso 2

El siguiente paso es encontrar una ecuación recursiva que define el valor de una solución óptima. Esta ecuación será:

$$m[i,j] = \begin{cases} 0 & i = j, \\ \text{mín}_{i \le k < j}\{m[i,k] + m[k+1,j] + p_{i-1}p_k p_j\} & i < j. \end{cases} \tag{3}$$

- ¿Cómo encontrar k para cada subproblema?
- ¿Cuántas posibilidades hay en total?
- $\binom{n}{2} + n = \Theta(n^2)$. (Polinomial).

PD Paso 3

El siguiente paso es calcular el costo de una solución óptima.

PD Paso 3

El siguiente paso es calcular el costo de una solución óptima.

- Solución recursiva $\rightarrow$ Exponencial

PD Paso 3

El siguiente paso es calcular el costo de una solución óptima.

- Solución recursiva $\rightarrow$ Exponencial
- Recordemos soluciones calculadas

PD Paso 3

El siguiente paso es calcular el costo de una solución óptima.

- Solución recursiva $\rightarrow$ Exponencial
- Recordemos soluciones calculadas
- Estrategía de abajo hacia arriba[1].

[1] bottom-up

PD Paso 3: Algoritmo I

function $\textsc{Matrix-Chain-Order}(\mathrm{p})$ $\quad // \ p = \langle p_0, \ldots p_n \rangle$
    $n = p.lenght - 1$
    Let $m\,[1 \cdots n, 1 \cdots n]$ be a new table

    for $i = 1$ **to** n **do**
        $m\,[i, i] = 0$ $\quad //$ Costo de la cadena A_i
    end for

    for $l = 2$ **to** n **do**
        for $i = 1$ **to** $n - l + 1$ **do**
            $j = i + l - 1$
            $m\,[i, j] = \infty$

            for $k = i$ **to** $j - 1$ **do**
                $q = m\,[i, k] + m\,[k + 1, j] + p_{i-1} p_k p_j$

PD Paso 3: Algoritmo II

```
                if q < m[i, j] then
                    m[i, j] = q
                end if
            end for
        end for
    end for
    return m
end function
```

PD Paso 3: Análisis del algoritmo

```
function M-C-O(p)
    n = p.lenght − 1
    Let m[1···n, 1···n] be a new table
    for i = 1 to n do
        m[i, i] = 0
    end for
    for l = 2 to n do
        for i = 1 to n − l + 1 do
            j = i + l − 1
            m[i, j] = ∞
            for k = i to j − 1 do
                q = m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
                if q < m[i, j] then
                    m[i, j] = q
                end if
            end for
        end for
    end for
    return m
end function
```

Vamos a analizar como funciona este código (Pizarrón)

PD Paso 3: Análisis del algoritmo

function M-C-O(p)
  $n = p.lenght - 1$
  Let $m[1 \cdots n, 1 \cdots n]$ be a new table
  for $i = 1$ **to** n **do**
    $m[i,i] = 0$
  end for
  for $l = 2$ **to** n **do**
    for $i = 1$ **to** $n - l + 1$ **do**
      $j = i + l - 1$
      $m[i,j] = \infty$
      for $k = i$ **to** $j - 1$ **do**
        $q = m[i,k] + m[k+1,j] + p_{i-1}p_kp_j$
        if $q < m[i,j]$ **then**
          $m[i,j] = q$
        end if
      end for
    end for
  end for
  return m
end function

¿Cual es la complejidad de tiempo del algoritmo?

PD Paso 3: Análisis del algoritmo

function M-C-O(p)
  $n = p.lenght - 1$
  Let $m[1 \cdots n, 1 \cdots n]$ be a new table
  for $i = 1$ **to** n **do**
    $m[i, i] = 0$
  end for
  for $l = 2$ **to** n **do**
    for $i = 1$ **to** $n - l + 1$ **do**
      $j = i + l - 1$
      $m[i, j] = \infty$
      for $k = i$ **to** $j - 1$ **do**
        $q = m[i, k] + m[k + 1, j] + p_{i-1} p_k p_j$
        if $q < m[i, j]$ **then**
          $m[i, j] = q$
        end if
      end for
    end for
  end for
  return m
end function

$O(n^3)$.

PD Paso 3: Análisis del algoritmo

```
function M-C-O(p)
    n = p.lenght − 1
    Let m[1···n, 1···n] be a new table
    for i = 1 to n do
        m[i, i] = 0
    end for
    for l = 2 to n do
        for i = 1 to n − l + 1 do
            j = i + l − 1
            m[i, j] = ∞
            for k = i to j − 1 do
                q = m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
                if q < m[i, j] then
                    m[i, j] = q
                end if
            end for
        end for
    end for
    return m
end function
```

Se puede probar que de hecho el algoritmo corre en tiempo $\Omega(n^3)$

PD Paso 3: Análisis del algoritmo

function M-C-O(p)
  $n = p.lenght - 1$
  Let $m[1 \cdots n, 1 \cdots n]$ be a new table
  for $i = 1$ **to** n **do**
    $m[i, i] = 0$
  end for
  for $l = 2$ **to** n **do**
    for $i = 1$ **to** $n - l + 1$ **do**
      $j = i + l - 1$
      $m[i, j] = \infty$
      for $k = i$ **to** $j - 1$ **do**
        $q = m[i, k] + m[k+1, j] + p_{i-1}p_kp_j$
        if $q < m[i, j]$ **then**
          $m[i, j] = q$
        end if
      end for
    end for
  end for
  return m
end function

¿Cual es la complejidad de espacio del algoritmo?

PD Paso 3: Análisis del algoritmo

function M-C-O(p)
    $n = p.lenght - 1$
    Let $m[1 \cdots n, 1 \cdots n]$ be a new table
    for $i = 1$ **to** n **do**
        $m[i,i] = 0$
    end for
    for $l = 2$ **to** n **do**
        for $i = 1$ **to** $n - l + 1$ **do**
            $j = i + l - 1$
            $m[i,j] = \infty$
            for $k = i$ **to** $j - 1$ **do**
                $q = m[i,k] + m[k+1,j] + p_{i-1}p_k p_j$
                if $q < m[i,j]$ **then**
                    $m[i,j] = q$
                end if
            end for
        end for
    end for
    return m
end function

$\Theta(n^2)$

PD Paso 4: Calcular la solución óptima

El último paso en nuestra solución con PD es calcular una solución optima que nos da el costo óptimo calculado en el paso 3. En el caso del problema de Multiplicar matrices en cadena, la solución óptima es una forma de acomodar los parentesis para efectuar la multiplicación de la cadena de matrices, de tal forma que el número de operaciones escalares sea de costo óptimo.

A continuación

Técnica: Programación Dinámica

Ejemplo: Multiplicar cadenas de matrices

Ejemplo: Un problema de Grafos

Notas y ejercicios

Un problema común en Grafos

El camino más largo simple dirigido entre un par de nodos en un grafo dirigido acíclico.

Un problema común en Grafos

Entrada: Un grafo dirigido acíclico $D = (N, A)$ con pesos en los arcos $(p \colon A \to \mathbb{R})$ y dos nodos distinguidos $s, t \in N$.

Salida: Un camino simple de s a t que maximice la suma de pesos de los arcos.

Camino simple más largo: PD Paso 1

Primero debemos mostrar que las instancias optimas de este problema muestran subestructura optima (Pizarrón)

Camino simple más largo: PD Paso 2

El siguiente paso es construir una ecuación recursiva para calcular el costo de una solución óptima (Pizarrón).

Camino simple más largo: PD Paso 3

Ahora debemos construir un algoritmo para calcular el costo óptimo de una solución óptima.

Camino simple más largo: PD Paso 3

Require: $D = (N, A)$ un grafo dirigido acíclico

function LONGEST-SIMPLE-PATH(D, p, s, t)
    $q = -\infty$

    for all $v \in$ outNeigbors(s) **do**
        $w_1 = p(\overrightarrow{sv}) +$ LONGEST-SIMPLE-PATH(D, p, v, t)

        if $q < w_1$ **then**
            $q = w_1$
        end if
    end for

    return q
end function

Camino simple más largo: PD Paso 3

Require: $D = (N, A)$ un grafo dirigido acíclico

// De arriba hacia abajo con memoización (top-down, memoization)

```
function LONGEST-SIMPLE-PATH-PD(D, p, s, t)
    n = |N|
    Let w[1···n, 1···n] be a new table

    for all i, j ∈ N do
        if ij→ ∈ A then
            w[i, j] = p(ij→)
        else if i == j then
            w[i, j] = 0
        else
            w[i, j] = −∞
        end if
    end for

    return LONGEST-SIMPLE-PATH-PD-AUX(D, p, s, t, w)
end function
```

Camino simple más largo: PD Paso 3

```
function LONGEST-SIMPLE-PATH-PD-AUX(D, p, s, t, w)
    if w[s, t] ≥ 0 then
        return w[s, t]
    else
        q = −∞

        for all v ∈ outNeigbors(s) do
            w1 = w[s, v] + LONGEST-SIMPLE-PATH-PD-AUX(D, p, v, t, w)

            if q < w1 then
                q = w1
            end if
        end for

        w[s, t] = q
    end if

    return q
end function
```

A continuación

Técnica: Programación Dinámica

Ejemplo: Multiplicar cadenas de matrices

Ejemplo: Un problema de Grafos

Notas y ejercicios

Ejercicios del capítulo I

EJERCICIO 1

Construye en pseudocódigo un algoritmo para multiplicar dos matrices de dimensiones arbitrarias y compatibles. Tu programa debe estar preparado para el caso en que las matrices no sean compatibles y mandar un mensaje de error. ¿Cuál es la complejidad de tiempo de tu algoritmo?

EJERCICIO 2

Considera una variante del problema de Multiplicar matrices en cadena, en la cual en lugar de minimizar la cantidad de multiplicaciones escalares, se busca maximizar esta cantidad. ¿Este problema exibe subestructura óptima? Justifica.

Bibliografía I

Thomas H. Cormen y col. *Introduction to Algorithms, Third Edition*. 3rd. The MIT Press, 2009. ISBN: 0262033844, 9780262033848.