Análisis y Diseño de Algoritmos II

Jesús Alexander Aranda Ph.D Robinson Duque, Ph.D
Juan Francisco Díaz, Ph. D

Universidad del Valle

jesus.aranda@correounivalle.edu.co
robinson.duque@correounivalle.edu.co
juanfco.diaz@correounivalle.edu.co
Programa de Ingeniería de Sistemas
Escuela de Ingeniería de Sistemas y Computación

Programación Dinámica

Revisión Elementos de Programación Dinámica

Programación dinámica

Los problemas de optimización donde se aplica programación dinámica se caracterizan por las siguientes características:

- **Subestructura óptima**: la solución (óptima) de un problema se descompone en soluciones (óptimas) de algunos de sus subproblemas.
- **Solapamiento de subproblemas**: Hay subproblemas diferentes que comparten subsubproblemas. El número total de subproblemas es pequeño.

Programación dinámica

Subestructura óptima:

La solución (óptima) de un problema se descompone en soluciones (óptimas) de algunos de sus subproblemas.

La estructura se refiere a la salida o solución del problema, donde la subestructura óptima se refiere a que parte(s) de la solución al problema original es la solución(es) (la mejor) de un subproblema.

Problema LCS

X = ABBC Y = ACBC

Z = ABC (solución)

¿AB será solución de un problema similar? ¿Será la mejor solución?

R/ Sí

X' = ABB Y' = ACB

Z' = AB (solución de un subproblema)

X = ACBB Y = ACBC

Z = ACB (solución)

¿ACB será solución de un problema similar? ¿Será la mejor solución?

R/ Sí

X' = ACBB Y' = ACB o. X' = ACB Y' = ACBC

Z' = ACB (solución de un subproblema)

Programación dinámica

Subestructura óptima:

Problema LCS

X = ABBC Y = ACBB

Z = ABB (solución)

¿ABB será solución de un problema similar? ¿Será la mejor solución?

R/ Sí

X' = ABB Y' = ACBB

Z' = ABB (solución de un subproblema)

X = ABBC Y = ACBCD

Z = ABC (solución)

¿ABC será solución de un problema similar? ¿Será la mejor solución?

R/ Sí

X' = ABBC Y' = ACBC

Z' = ABC (solución de un subproblema)

Programación dinámica

Subestructura óptima:

Problema LCS

$X = x_1 x_2 \ldots x_n$ $Y = y_1 y_2 \ldots y_m$

$Z = z_1 z_2 \ldots z_{k-1} z_k$. (Z es LCS(X,Y))

Si $x_n == y_m$ entonces $Z_{k-1} = LCS(X_{n-1}, Y_{m-1})$

Si $x_n != y_m$ entonces $Z_k = LCS(X_{n-1}, Y)$ o

$Z_k = LCS(X, Y_{m-1})$ (¿Cuál caso usar?)

(Note que $Z_k == Z$ y $Z_{k-1} = z_1 z_2 \ldots z_{k-1}$)

Programación dinámica

Subestructura óptima:

Problema Mochila

Objetos = ((3,3), (4,5), (4,2)) M= 8

Sol=(1,1,0)

¿(1,1) será solución de un problema similar? ¿Será la mejor solución?

R/ Sí

Objetos = ((3,3), (4,5)) M= 8

Sol=(1,1) (solución de un subproblema)

Objetos = ((3,5), (4,1), (4,5)) M= 8

Sol=(1,0,1)

¿(1,0) será solución de un problema similar? ¿Será la mejor solución?

R/ Sí

Objetos = ((3,3), (4,5)) M= 4

Sol=(1,1) (solución de un subproblema)

Objetos = (Ob_1,Ob_2, ..., Ob_n) y M

Sol =(y_1,y_2, ..., y_{n-1} ,y_n) (Sol = Mochila((Ob_1,Ob_2, ..., Ob_n), M))

Si y_n == 0

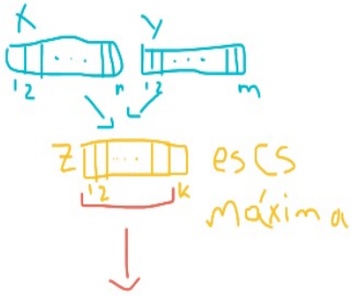
(y_1,y_2, ..., y_{n-1}) = Mochila((Ob_1,Ob_2, ..., Ob_{n-1}), M)

Si y_n == 1

(y_1,y_2, ..., y_{n-1}) = Mochila((Ob_1,Ob_2, ..., Ob_{n-1}), M - $peso_n$)

La subestructura óptima en varios problemas

LCS



Es la mejor para un problema similar?

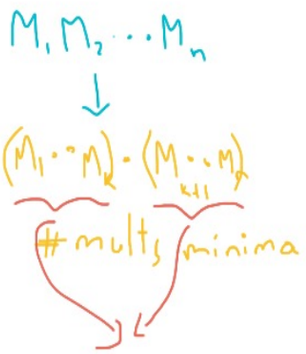
Si

Si x_n == y_m, Z_(k-1) = LCS(X_(n-1), Y_(m-1))

Si x_n != y_m, Z = LCS(X_(n-1), Y) o
Z = LCS(X, Y_(m-1))

Cuál caso usar?

MultMat



Son las mejores para problemas similares?

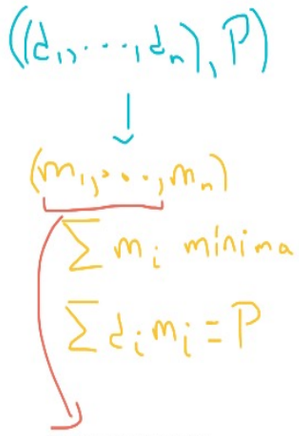
Si

MejorParentiz(M_1 ... M_n) =
(MejorParentiz(M_1 ... M_k))++
(MejorParentiz(M_(k+1) ... M_n))

para ese k de la solución óptima

Quién es k?

CambioMon



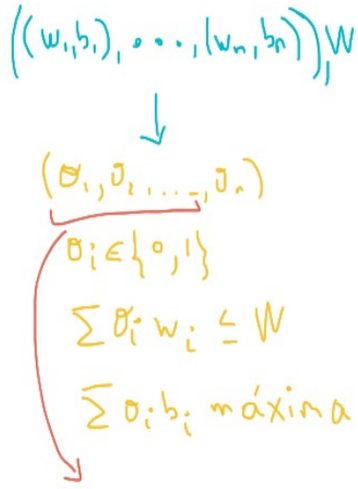
Es la mejor para un problema similar?

Si

(m_1,..., m_(n-1)) es la mejor solución para el problema
((d1,..., d_(n-1)), P-d_n*m_n)

Cuál es el m_n?

Mochila



Es la mejor para un problema similar?

Si

(o_1,..., o_(n-1)) es la mejor solución para el problema
((w_1,b_1), ...,
(w_(n-1),b_(n-1))) , W - o_n*w_n

Cuál es o_n?

Programación dinámica

Subestructura óptima:

La solución (óptima) de un problema se descompone en soluciones (óptimas) de algunos de sus subproblemas.

Aspectos a tener en cuenta al identificar la subestructura óptima:

La estructura se refiere a la salida o solución del problema, donde la subestructura óptima se refiere a que parte(s) de la solución al problema original es la solución(es) (la mejor) de un subproblema.

1. La solución a un problema se construye a partir de hacer una escogencia.
2. Asumir que la escogencia dada, lleva a la solución del problema.
3. A partir de la escogencia, determinar los subproblemas correspondientes y caracterizar el espacio de subproblemas.
4. Mostrar que la solución de los subproblemas (item 3) necesitan ser óptimas.

Programación dinámica

Aspectos Identificar Subestructura óptima:

1. La solución (óptima) a un problema se construye a partir de hacer una escogencia.

Problema MSM (Multiplicación de Sucesión de Matrices)

Problema MSM($A_i, A_{i+1}, \ldots, A_j$)

1. Para determinar cómo parentizar se necesita escoger donde se hará la última multiplicación de matrices.

Programación dinámica

Aspectos Identificar Subestructura óptima:

1. La solución (óptima) a un problema se construye a partir de hacer una escogencia.
2. Asumir que la escogencia dada, lleva a la solución (óptima)del problema.

Problema MSM (Multiplicación de Sucesión de Matrices)

Problema MSM(A_i,A_{i+1},..., A_j)

1. Para determinar cómo parentizar se necesita escoger donde se hará la última multiplicación de matrices.
2. Asumir que se cuenta con el valor k, que indica dónde se hará la última multiplicación en la solución óptima del problema.

Programación dinámica

Aspectos Identificar Subestructura óptima:

1. La solución (óptima) a un problema se construye a partir de hacer una escogencia.
2. Asumir que la escogencia dada, lleva a la solución (óptima)del problema.
3. A partir de la escogencia, determinar los subproblemas correspondientes y caracterizar el espacio de subproblemas.

Problema MSM (Multiplicación de Sucesión de Matrices)

Problema MSM(A_i,A_{i+1},..., A_j)

1. Para determinar cómo parentizar se necesita escoger donde se hará la última multiplicación de matrices.
2. Asumir que se cuenta con el valor k, que indica dónde se hará la última multiplicación en la solución óptima del problema.
3. A partir del valor k, se puede identificar los siguientes subproblemas partiendo del problema original MSM(A_i,A_{i+1},..., A_j): MSM(A_i,A_{i+1},..., A_k) y MSM(A_{k+1},A_{k+2},..., A_j). El número de subproblemas sería $\theta\left((j-i+1)^2\right)$ (Si i=1 y j =n entonces el número de subproblemas sería $\theta\ (n^2)$).

Programación dinámica

Aspectos Identificar Subestructura óptima:

1. La solución (óptima) a un problema se construye a partir de hacer una escogencia.
2. Asumir que la escogencia dada, lleva a la solución (óptima)del problema.
3. A partir de la escogencia, determinar los subproblemas correspondientes y caracterizar el espacio de subproblemas.
4. Mostrar que las soluciones de los subproblemas (item 3) necesitan ser óptimas.

Problema MSM (Multiplicación de Sucesión de Matrices)

Problema MSM(A_i,A_{i+1},…, A_j)

1. Para determinar cómo parentizar se necesita escoger donde se hará la última multiplicación de matrices.
2. Asumir que se cuenta con el valor k, que indica dónde se hará la última multiplicación en la solución óptima del problema.
3. A partir del valor k, se puede identificar los siguientes subproblemas partiendo del problema original MSM(A_i,A_{i+1},…, A_j): MSM(A_i,A_{i+1},…, A_k) y MSM(A_{k+1},A_{k+2},…, A_j). El número de subproblemas sería $\theta\left((j-i+1)^2\right)$ (Si i=1 y j =n entonces el número de subproblemas sería $\theta\ (n^2)$).
4. Si las soluciones de MSM(A_i,A_{i+1},…, A_k) y MSM(A_{k+1},A_{k+2},…, A_j) no fueran óptimas, entonces habría una mejor manera de parentizar desde A_i hasta A_k y desde A_{k+1} hasta A_j y por consiguiente una mejor forma desde A_i hasta A_j. Lo cual es contradictorio. Por lo tanto, las soluciones de MSM(A_i,A_{i+1},…, A_k) y MSM(A_{k+1},A_{k+2},…, A_j) deben ser las mejores.

Programación dinámica

Observación I sobre Caracterización de la estructura de una solución óptima (subestructura óptima)

La subestructura óptima se cumple para todo problema, no solo el original

(Subestructura óptima en el problema original MSM($A_1,A_2,\ldots, A_n$))

- **Una solución (óptima) de** MSM($A_1,A_2,\ldots, A_n$)) **se divide en una solución (óptima) de** MSM($A_1,A_2,\ldots, A_k$) y en una solución **(óptima) de** MSM($A_{k+1},A_{k+2},\ldots, A_n$)

Solución(MSM($A_1,A_2,\ldots, A_n$)) =(Solución(MSM($A_1,A_2,\ldots, A_k$)) . Solución(MSM($A_{k+1},A_{k+2},\ldots, A_n$)))

para algún k, $1 \leq k < n$

(Subestructura óptima para el problema MSM en general)

- **Una solución (óptima) de** MSM($A_i,A_{i+1},\ldots, A_j$)) **se divide en una solución (óptima) de** MSM($A_i,A_{i+1},\ldots, A_k$) y en una solución **(óptima) de** MSM($A_{k+1},A_{k+2},\ldots, A_j$)

Solución(MSM($A_i,A_{i+1},\ldots, A_j$)) =(Solución(MSM($A_i,A_{i+1},\ldots, A_k$)) . Solución(MSM($A_{k+1},A_{k+2},\ldots, A_j$)))

para algún k, $i \leq k < j$

Programación dinámica

Observación II sobre Caracterización de la estructura de una solución óptima (subestructura óptima)

La subestructura óptima puede variar según el problema de dos formas:

- ¿Cuántos subproblemas directos se necesitan resolver para determinar una solución óptima al problema original?
- ¿Cuántos alternativas se tienen para poder determinar cuales subproblemas se necesitan para determinar una solución óptima al problema original?

Programación dinámica

Observación II sobre Caracterización de la estructura de una solución óptima (subestructura óptima)

La subestructura óptima puede variar según el problema de dos formas:

- ¿Cuántos subproblemas directos se necesitan resolver para determinar una solución óptima al problema original?

R/ Considerando el problema MSM, para la construcción de la solución óptima se necesitan la solución de dos subproblemas

$$\text{Solución}(\text{MSM}(A_i, A_{i+1}, \ldots, A_j)) = (\text{Solución}(\text{MSM}(A_i, A_{i+1}, \ldots, A_k)) . \text{Solución}(\text{MSM}(A_{k+1}, A_{k+2}, \ldots, A_j)))$$

para algún k, $i \leq k < j$

Programación dinámica

Observación II sobre Caracterización de la estructura de una solución óptima (subestructura óptima)

La subestructura óptima puede variar según el problema de dos formas:

- ¿Cuántos subproblemas directos se necesitan resolver para determinar una solución óptima al problema original?

R/ Considerando el problema MSM($A_i, A_{i+1}, \ldots, A_j$), para la construcción de la solución óptima se necesitan la solución de dos subproblemas

$$\text{Solución}(\text{MSM}(A_i, A_{i+1}, \ldots, A_j)) = (\text{Solución}(\text{MSM}(A_i, A_{i+1}, \ldots, A_k)) . \text{Solución}(\text{MSM}(A_{k+1}, A_{k+2}, \ldots, A_j)))$$

para algún k, $i \leq k < j$

- ¿Cuántos alternativas se tienen para poder determinar cuales subproblemas se necesitan para determinar una solución óptima al problema original?

R/ Considerando el problema MSM($A_i, A_{i+1}, \ldots, A_j$), para la construcción de la solución óptima se contemplan (j–i) alternativas

$$\text{Solución}(\text{MSM}(A_i, A_{i+1}, \ldots, A_j)) = (\text{Solución}(\text{MSM}(A_i, A_{i+1}, \ldots, A_k)) . \text{Solución}(\text{MSM}(A_{k+1}, A_{k+2}, \ldots, A_j)))$$

para algún k, $i \leq k < j$

Programación dinámica

Observación III sobre Caracterización de la estructura de una solución óptima (subestructura óptima)

Al contar con la caracterización de la estructura de una solución óptima de forma recursiva del problema original se pueden determinar cuántos subproblemas son necesarios en total.

(Subestructura óptima en el problema original MSM(A_1,A_2,…, A_n))

- **Una solución (óptima) de MSM(A_1,A_2,…, A_n)) se divide en una solución (óptima) de MSM(A_1,A_2,…, A_k) y en una solución (óptima) de MSM(A_{k+1},A_{k+2},…, A_n)**

Solución(MSM(A_1,A_2,…, A_n)) =(Solución(MSM(A_1,A_2,…, A_k)) !Solución(MSM(A_{k+1},A_{k+2},…, A_n)))

En este caso, todo subproblema MSM(A_i,A_{i+1},…, A_j) debe cumplir con que $i \leq j$ donde $1 \leq i \leq n$ y

$1 \leq j \leq n$. El número total de subproblemas sería $\binom{n}{2} + n = \frac{(n-1).n}{2} + n = \theta(n^2)$

¡El número de subproblemas es polinomial!

Programación dinámica

Los problemas de optimización donde se aplica programación dinámica se caracterizan por las siguientes características:

- **Subestructura óptima**: la solución (óptima) de un problema se descompone en soluciones (óptimas) de algunos de sus subproblemas.
- **Solapamiento de subproblemas**: Hay subproblemas diferentes que comparten subsubproblemas. El número total de subproblemas es pequeño.

Programación dinámica

Solapamiento de subproblemas:

Hay subproblemas diferentes que comparten subsubproblemas. El número total de subproblemas es pequeño.

Considerando el problema original MSM(A_1,A_2,…, A_n))

- **Una solución (óptima) de** MSM(A_1,A_2,…, A_n)) **se divide en una solución (óptima) de** MSM(A_1,A_2,…, A_k) y en una solución **(óptima) de** MSM(A_{k+1},A_{k+2},…, A_n)

Solución(MSM(A_1,A_2,…, A_n)) =(Solución(MSM(A_1,A_2,…, A_k)) !Solución(MSM(A_{k+1},A_{k+2},…, A_n)))

En este caso, todo subproblema MSM(A_i,A_{i+1},…, A_j) debe cumplir con que i ≤ j donde 1 ≤ i ≤ n y

1 ≤ j ≤ n. El número total de subproblemas sería $\binom{n}{2} + n = \frac{(n-1).n}{2} + n = \theta(n^2)$

¡El número de subproblemas es polinomial!

Programación dinámica

Solapamiento de subproblemas:

Hay subproblemas diferentes que comparten subsubproblemas. El número total de subproblemas es pequeño.

Un algoritmo recursivo necesitaría resolver el mismo problema varias veces.

Dado que estamos abordando problemas de optimización, se calcularía el valor de la solución del problema.

En el caso de MSM

```
RECURSIVE-MATRIX-CHAIN(p,i,j)
  if  i ==  j
    then  return 0
  m[i,j] = ∞
  for  k = i to j -1
        q = RECURSIVE-MATRIX-CHAIN(p,i,k) + RECURSIVE-MATRIX-CHAIN(p,k+1,j) + p_{i-1} p_k p_j
       if q < m[i,j]
          m[i,j] = q
  return m[i,j]
```

Llamado inicial RECURSIVE-MATRIX-CHAIN(p,1,n)

Programación dinámica

Solapamiento de subproblemas:

En el caso de MSM, a continuación, un algoritmo recursivo que calcula el valor de la solución del problema original.

$$m[i,j]: \begin{cases} 0 & \text{si } i=j \\ \min_{i'<=k<j}\{m[i,k]+m[k+1,j]+p_{i-1}p_k p_j\} & \text{si } i<j \end{cases}$$

```
RECURSIVE-MATRIX-CHAIN(p,i,j)
 if  i == j
   then  return 0
 m[i,j] = ∞
 for  k = i to j -1
       q = RECURSIVE-MATRIX-CHAIN(p,i,k) + RECURSIVE-MATRIX-CHAIN(p,k+1,j) + p_{i-1}p_k p_j
      if q < m[i,j]
         m[i,j] = q
 return m[i,j]
```

Llamado inicial RECURSIVE-MATRIX-CHAIN(p,1,n)

Costo computacional: $\Omega(2^n)$ Exponencial

Número de subproblemas diferentes: $\Theta(n^2)$

¡¡ Los subproblemas se repiten muchas veces!!

¡¡ Hay solapamiento de subproblemas!!

Programación dinámica

Solapamiento de subproblemas:

En el caso de MSM, a continuación, un algoritmo recursivo que calcula el valor de la solución del problema original.

```
RECURSIVE-MATRIX-CHAIN(p,i,j)
 if  i == j
   then  return 0
 m[i,j] = ∞
 for  k = i to j -1
       q = RECURSIVE-MATRIX-CHAIN(p,i,k) + RECURSIVE-MATRIX-CHAIN(p,k+1,j) + p_{i-1} p_k p_j
      if q < m[i,j]
         m[i,j] = q
 return m[i,j]
```

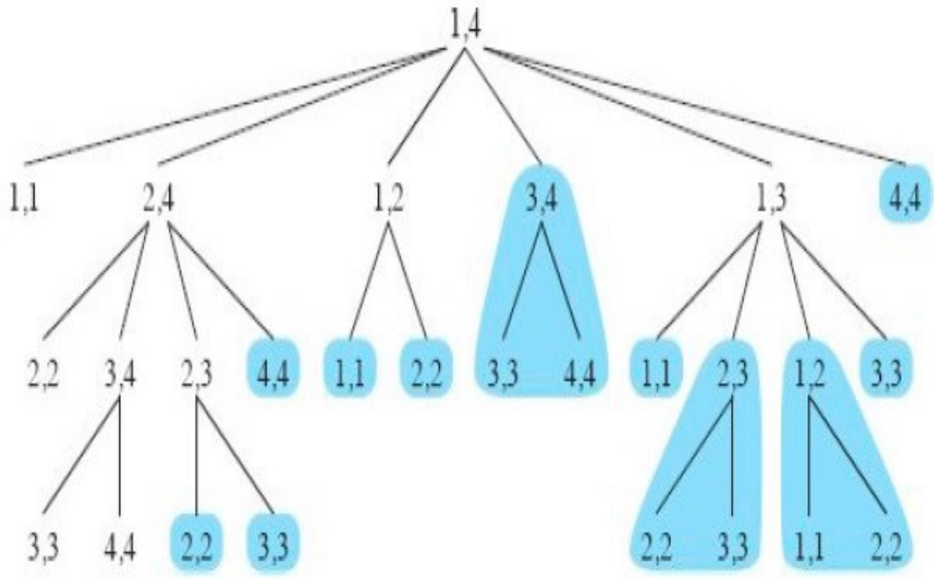
Llamado inicial RECURSIVE-MATRIX-CHAIN(p,1,n)

Costo computacional: $\Omega(2^n)$ Exponencial

Número de subproblemas diferentes: $\Theta(n^2)$

ii Los subproblemas se repiten muchas veces!!

ii Hay solapamiento de subproblemas!!



Tomado de [1]

Programación dinámica

Solapamiento de subproblemas (Alternativa Memoization):

Se plantea un algoritmo similar al recursivo (estrategia top-down) pero guarda los valores que va calculando asociado a cada subproblema para luego recuperarlo.

```
RECURSIVE-MATRIX-CHAIN(p,i,j)
 if  i ==  j
   then  return 0
 m[i,j] = ∞
 for  k = i to j -1
       q = RECURSIVE-MATRIX-CHAIN(p,i,k) + RECURSIVE-MATRIX-CHAIN(p,k+1,j) + p_{i-1} p_k p_j
      if q < m[i,j]
         m[i,j] = q
 return m[i,j]
```

Llamado inicial RECURSIVE-MATRIX-CHAIN(p,1,n)

```
MEMOIZED-MATRIX-CHAIN(p, n)

 let m[1 : n, 1 : n] be a new table
 for i = 1 to n
   for j = i to n
       m[i, j] = ∞
 return LOOKUP-CHAIN(m, p, 1, n)
```

```
LOOKUP-CHAIN(m, p, i, j)
 if m[i, j] < ∞
    return m[i, j]
 if i == j
    m[i, j] = 0
 else for k = i to j – 1
           q = LOOKUP-CHAIN(m, p, i, k)
               + LOOKUP-CHAIN(m, p, k + 1, j)
                + p_{i-1} p_k p_j
           if q < m[i, j]
              m[i, j] = q
 return m[i, j]
```

Llamado inicial MEMOIZED-MATRIX-CHAIN(p, n)

Programación dinámica

Solapamiento de subproblemas (Alternativa Memoization Top-down):

Se plantea un algoritmo similar al recursivo (estrategia top-down) pero guarda los valores que va calculando asociado a cada subproblema para luego recuperarlo.

```
MEMOIZED-MATRIX-CHAIN(p, n)

 let m[1 : n, 1 : n] be a new table
 for i = 1 to n
    for j = i to n
        m[i, j] = ∞
 return LOOKUP-CHAIN(m, p, 1, n)


LOOKUP-CHAIN(m, p, i, j)
  if m[i, j] < ∞
     return m[i, j]
  if i == j
     m[i, j] = 0
  else for k = i to j – 1
             q = LOOKUP-CHAIN(m, p, i, k)
                 + LOOKUP-CHAIN(m, p, k + 1, j)
                  + p_{i-1} p_k p_j
              if q < m[i, j]
                   m[i, j] = q
 return m[i, j]
```

Llamado inicial MEMOIZED-MATRIX-CHAIN(p, n)

Costo Computacional Temporal

Inicializar toda la matriz $O(n^2)$

Calcular el valor asociado a una posición de la matriz $O(n)$

Calcular los valores en la matriz $O(n^3)$

Costo Computacional Espacial

Tamaño de la matriz $\theta(n^2)$

Programación dinámica

Solapamiento de subproblemas (Alternativa Bottom-up):

Se plantea un algoritmo similar al recursivo (estrategia top-down) pero guarda los valores que va calculando asociado a cada subproblema para luego recuperarlo.

Algoritmo Top-down (Memoization)

```
MEMOIZED-MATRIX-CHAIN(p, n)

 let m[1 : n, 1 : n] be a new table
 for i = 1 to n
   for j = i to n
       m[i, j] = ∞
 return LOOKUP-CHAIN(m, p, 1, n)


LOOKUP-CHAIN(m, p, i, j)
 if m[i, j] < ∞
    return m[i, j]
  if i == j
    m[i, j] = 0
  else for k = i to j – 1
            q = LOOKUP-CHAIN(m, p, i, k)
                + LOOKUP-CHAIN(m, p, k + 1, j)
                 + p_{i-1} p_k p_j
             if q < m[i, j]
                  m[i, j] = q
 return m[i, j]
```

Llamado inicial MEMOIZED-MATRIX-CHAIN(p, n)

Algoritmo Bottom-up

```
MATRIX-CHAIN-ORDER(p,n)
 let m[1 : n, 1 : n] be a new table
 for i  = 1  to n
      m[i,i]= ∞
 for l = 2  to n
    for i =  1  to n-l+1
          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

 return m
```

Llamado inicial MATRIX-CHAIN-ORDER(p, n)

Costo Computacional Temporal $O(n^3)$

Costo Computacional Espacial $\theta(n^2)$

Programación dinámica

Solapamiento de subproblemas

(Alternativas Top Down (Memoization) y Bottom-up)

¿ En qué etapa de la programación dinámica se aplica la alternativa Top Down o Bottom-up con el fin de plantear una solución algorítmica eficiente?

1) Caracterizar la estructura de la solución óptima (subestructura óptima).
2) Definir recursivamente el valor de la solución óptima.
3) **Calcular el valor de la solución óptima (algoritmo).**
4) Construir la solución óptima (algoritmo).

Programación dinámica

- Dos características fundamentales para poder aplicar programación dinámica para solucionar un problema de optimización son:
    - Subestructura Óptima
    - Solapamiento de problemas

- Al existir una subestructura óptima, es posible plantear una estrategia recursiva que resuelva el problema, sin embargo, su costo es exponencial.

- Al existir solapamiento de problemas, es posible proponer estrategias más eficientes que la recursiva tal que calculen la solución de los subproblemas solo una vez para que posteriormente puedan ser recuperadas.

- Se plantean dos formas de calcular las soluciones (valores) de los subproblemas con el fin de hacerlo solo una vez para que luego puedan ser recuperadas:

    - Estrategia Top-down con Memoization
    - Estrategia Bottom-up

  El costo computacional de estas estrategias es similar.