Programación Dinámica

Caracterízacion de la subestructura óptima

Juan Manuel Rabasedas

Programación Dinámica

Es una técnica de diseño de algoritmos como divide y venceras.

Programación Dinámica

Es una técnica de diseño de algoritmos como divide y venceras.

Ejemplo: *Subsecuencia Común Máxima (SCM)*

- Dadas dos secuencias $x[1\,..\,m]$ y $y[1\,..\,n]$, encontrar una subsecuencia común a ambas secuencias de longitud máxima.

Programación Dinámica

Es una técnica de diseño de algoritmos como divide y venceras.

Ejemplo: *Subsecuencia Común Máxima (SCM)*

- Dadas dos secuencias $x[1 \,..\, m]$ y $y[1 \,..\, n]$, encontrar una subsecuencia común a ambas secuencias de longitud máxima.

x: A B C B D A B

y: B D C A B A

Programación Dinámica

Es una técnica de diseño de algoritmos como divide y venceras.

Ejemplo: *Subsecuencia Común Máxima (SCM)*

- Dadas dos secuencias $x[1\,..\,m]$ y $y[1\,..\,n]$, encontrar una subsecuencia común a ambas secuencias de longitud máxima.

x: A B C B D A B

y: B D C A B A

BCBA

¿Qué tan complejo puede ser?

Por fuerza-bruta:

Chequear todas las subsecuencias de $x[1\,.\,.\,m]$ para ver si también es una subsecuencia de $y[1\,.\,.\,n]$.

¿Qué tan complejo puede ser?

Por fuerza-bruta:
Chequear todas las subsecuencias de $x[1 \,..\, m]$ para ver si también es una subsecuencia de $y[1 \,..\, n]$.

¿Cuantas subsecuencias de x tengo?

x:	A	B	C	B	D	A	B
on/off	0	0	0	0	0	0	0

¿Qué tan complejo puede ser?

Por fuerza-bruta:

Chequear todas las subsecuencias de $x[1 \,..\, m]$ para ver si también es una subsecuencia de $y[1 \,..\, n]$.

¿Cuantas subsecuencias de x tengo?

x:	A	B	C	B	D	A	B
on/off	0	1	1	1	0	1	0

¿Qué tan complejo puede ser?

Por fuerza-bruta:

Chequear todas las subsecuencias de $x[1 \,.\,.\, m]$ para ver si también es una subsecuencia de $y[1 \,.\,.\, n]$.

¿Cuantas subsecuencias de x tengo?

x:	A	B	C	B	D	A	B
on/off	0	1	1	1	0	1	0

$\mathbf{2}^m$ subsecuencias de x.

¿Qué tan complejo puede ser?

Por fuerza-bruta:

Chequear todas las subsecuencias de $x[1\,..\,m]$ para ver si también es una subsecuencia de $y[1\,..\,n]$.

¿Cuantas subsecuencias de x tengo?

x:	A	B	C	B	D	A	B
on/off	0	1	1	1	0	1	0

2^m subsecuencias de x.

Dada una secuencia de x.

¿Cuánto me lleva chequear si una subsecuencia está en y ?

¿Qué tan complejo puede ser?

Por fuerza-bruta:

Chequear todas las subsecuencias de $x[1\ ..\ m]$ para ver si también es una subsecuencia de $y[1\ ..\ n]$.

¿Cuantas subsecuencias de x tengo?

x:	A	B	C	B	D	A	B
on/off	0	1	1	1	0	1	0

2^m subsecuencias de x.

Dada una secuencia de x.

¿Cuánto me lleva chequear si una subsecuencia está en y ?

La longitud de y: n

¿Qué tan complejo puede ser?

El tiempo es: $O(n2^m)$ El problema es que tengo muchas subsecuencias.

¿Qué tan complejo puede ser?

El tiempo es: $\mathbf{O}(n2^{m})$ El problema es que tengo muchas subsecuencias.

Simplifiquemos el problema

Calculo la longitud de la subsecuencia común máxima

La longitud es única

Luego busco una SCM de esa longitud

1er Paso para usar Dinámica

Estructurar la solución en termino de problemas más chicos.

1er Paso para usar Dinámica

Estructurar la solución en termino de problemas más chicos.

Es la parte más difícil

1er Paso para usar Dinámica

Estructurar la solución en termino de problemas más chicos.

Es la parte más difícil

En nuestro ejemplo:
Vamos a considerar los prefijos entre x e y.

1er Paso para usar Dinámica

Estructurar la solución en termino de problemas más chicos.

Es la parte más difícil

En nuestro ejemplo:
Vamos a considerar los prefijos entre x e y.

Sea:

$$c[i,j] = |\ \mathrm{SCM}(x[1..i], y[1..j])\ |$$

1er Paso para usar Dinámica

Estructurar la solución en termino de problemas más chicos.

Es la parte más difícil

En nuestro ejemplo:
Vamos a considerar los prefijos entre x e y.

Sea:

$$c[i,j] = |\, \mathrm{SCM}(x[1..i], y[1..j]) \,|$$

luego

$$c[m,n] = |\, \mathrm{SCM}(x, y) \,|$$

Estructuremos el problema

¿Cómo vamos a expresar c[i,j] en términos de otros?

Estructuremos el problema

¿Cómo vamos a expresar c[i,j] en términos de otros?

Usemos recursión

Estructuremos el problema

¿Cómo vamos a expresar c[i,j] en términos de otros?

Usemos recursión

$$c[i,j] = \begin{cases} 0 & \text{si } i=0 \text{ or } j=0, \\ c[i-1, j-1] + 1 & \text{si } i,j>0 \text{ and } x[i] = y[j], \\ \max\{c[i-1, j], c[i, j-1]\} & \text{en otro caso} \end{cases}$$

Estructuremos el problema

¿Cómo vamos a expresar c[i,j] en términos de otros?

Usemos recursión

$$
c[i,j] = \begin{cases} 0 & \text{si } i=0 \text{ or } j=0, \\ c[i-1, j-1] + 1 & \text{si } i,j>0 \text{ and } x[i] = y[j], \\ \max\{c[i-1, j], c[i, j-1]\} & \text{en otro caso} \end{cases}
$$

los términos son menores a los que quiero calcular

Hemos planteado nuestro problema en términos de subproblemas más chicos

Probemos

Debemos probar que nuestro recursión nos lleva a una solución óptima.

Probemos

Debemos probar que nuestro recursión nos lleva a una solución óptima.

Caso $x[i]=y[j]$

Sea $r[1..k]$ = SCM ($x[1..i]$, $y[1..j]$) con $c[i,j]=k$

$r[k]$ = ?

Probemos

Debemos probar que nuestro recursión nos lleva a una solución óptima.

Caso $x[i]=y[j]$

Sea $r[1..k]$ = SCM ($x[1..i]$, $y[1..j]$) con $c[i,j]=k$

$r[k] = x[i]$ $(=y[j])$ ¿Por qué?

Si $r[1..k]$ no incluye a $x[i]$ ($y[j]$) como su último elemento podríamos extender r ya que $x[i]=y[j]$

Probemos

Debemos probar que nuestro recursión nos lleva a una solución óptima.

Caso $x[i]=y[j]$

Sea $r[1..k]$ = SCM ($x[1..i]$, $y[1..j]$) con $c[i,j]=k$

$r[k] = x[i]\ (=y[j])$ ¿Por qué?

Si $r[1..k]$ no incluye a $x[i]$ ($y[j]$) como su último elemento podríamos extender r ya que $x[i] = y[j]$

Luego $r[1..k-1]$ es una SC de $x[1..i-1]$ e $y[1..j-1]$

Probemos

Debemos probar que nuestro recursión nos lleva a una solución óptima.

Caso $x[i]=y[j]$

Sea $r[1..\mathrm{k}]$ = SCM ($x[1..i]$, $y[1..j]$) con $\mathrm{c}[i,j]=\mathrm{k}$

$r[\mathrm{k}]$ = $x[i]$ ($=y[j]$) ¿Por qué?

Si $r[1..k]$ no incluye a $x[\mathrm{i}]$ ($y[\mathrm{j}]$) como su último elemento podríamos extender r ya que $x[\mathrm{i}] = y[\mathrm{j}]$

Luego $r[1..\mathrm{k}-1]$ es una SC de $x[1..i-1]$ e $y[1..j-1]$

¿Qué me gustaría que pase?

Me gustaría que:

$r[1..k\text{-}1]$ sea SCM de $x[1..i\text{-}1]$ e $y[1..j\text{-}1]$

Me gustaría que:

$r[1..k\text{-}1]$ sea SCM de $x[1..i\text{-}1]$ e $y[1..j\text{-}1]$

Supongamos que w una SC más larga de $x[1..i\text{-}1]$ e $y[1..j\text{-}1]$

Es decir que $|w| > k\text{-}1$

Me gustaría que:

$r[1..k\text{-}1]$ sea SCM de $x[1..i\text{-}1]$ e $y[1..j\text{-}1]$

Supongamos que w una SC más larga de $x[1..i\text{-}1]$ e $y[1..j\text{-}1]$

Es decir que $|w| > k\text{-}1$

Luego concatenando w con $r[k]$,

$$w{+}{+}r[k]$$

obtenemos una SC de $x[1..i]$ e $y[1..j]$

con $|w{+}{+}r[k]|>k$

Me gustaría que:

r[1..k-1] sea SCM de $x[1..i\text{-}1]$ e $y[1..j\text{-}1]$

Supongamos que w una SC más larga de $x[1..i\text{-}1]$ e $y[1..j\text{-}1]$

Es decir que $|w| > k\text{-}1$

Luego concatenando w con $r[k]$,

$$w{+}{+}r[k]$$

obtenemos una SC de $x[1..i]$ e $y[1..j]$

con $|w{+}{+}r[k]|>k$ Contradicción!

Me gustaría que:

r[1..k-1] sea SCM de x[1..i-1] e y[1..j-1]

Supongamos que w una SC más larga de x[1..i-1] e y[1..j-1]

Es decir que $|w| > k-1$

Luego concatenando w con $r[k]$,

$$w{+}{+}r[k]$$

obtenemos una SC de x[1..i] e y[1..j]

con $|w{+}{+}r[k]|>k$ Contradicción!

Entonces, $c[i-1,j-1]= k-1$ lo que implica que

$c[i,j]= c[i-1, j-1] +1$ en el caso $x[i]=y[j]$

Los otros casos son similares.

Caracterización de la sub estructura óptima

Sub Estructura Óptima

Una solución óptima de un problema contiene una solución óptima para los subproblemas

Caracterización de la sub estructura óptima

Sub Estructura Óptima

Una solución óptima de un problema contiene una solución óptima para los subproblemas

Cuando veamos este tipo de estructura es un
INDICIO
que podemos aplicar Programación Dinámica

Caracterización de la sub estructura óptima

Sub Estructura Óptima

Una solución óptima de un problema contiene una solución óptima para los subproblemas

Cuando veamos este tipo de estructura es un

INDICIO

que podemos aplicar Programación Dinámica

En nuestro problema:

Si r = SCM(x,y), entonces cualquier prefijo de r es una SCM de un prefijo de x e y.

Implementación

SCM(x, y, i, j)

  if $x[i] = y[j]$

    then $c[i, j] \leftarrow$ SCM $(x, y, i\text{–}1, j\text{–}1) + 1$

    else $c[i, j] \leftarrow \max\{$ SCM$(x, y, i\text{–}1, j)$,
                SCM$(x, y, i, j\text{–}1)\}$

  return $c[i,j]$

Implementación

SCM(x, y, i, j)

if $x[i] = y[j]$

then $c[i, j] \leftarrow$ SCM $(x, y, i-1, j-1) + 1$

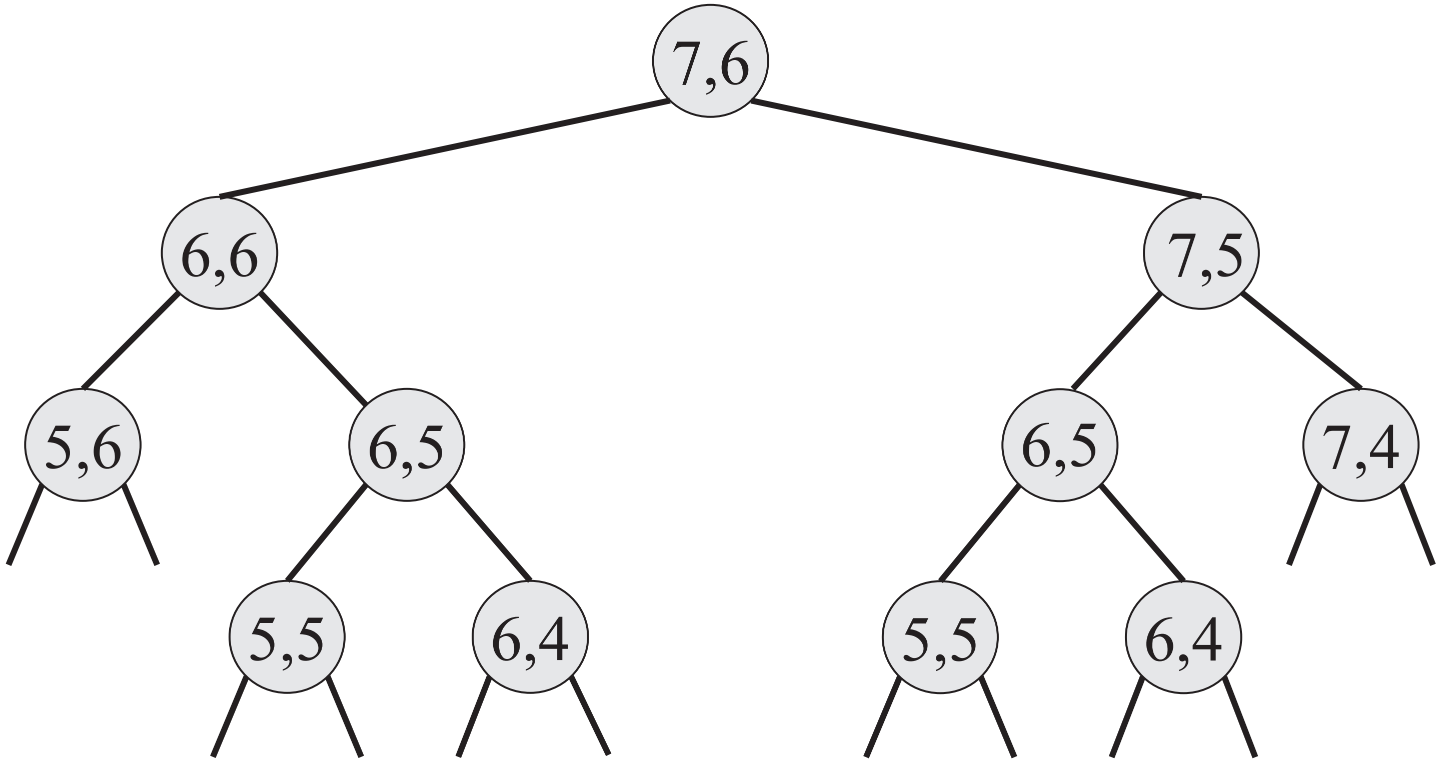
else $c[i, j] \leftarrow \max\{$ SCM$(x, y, i-1, j)$,

SCM$(x, y, i, j-1)\}$

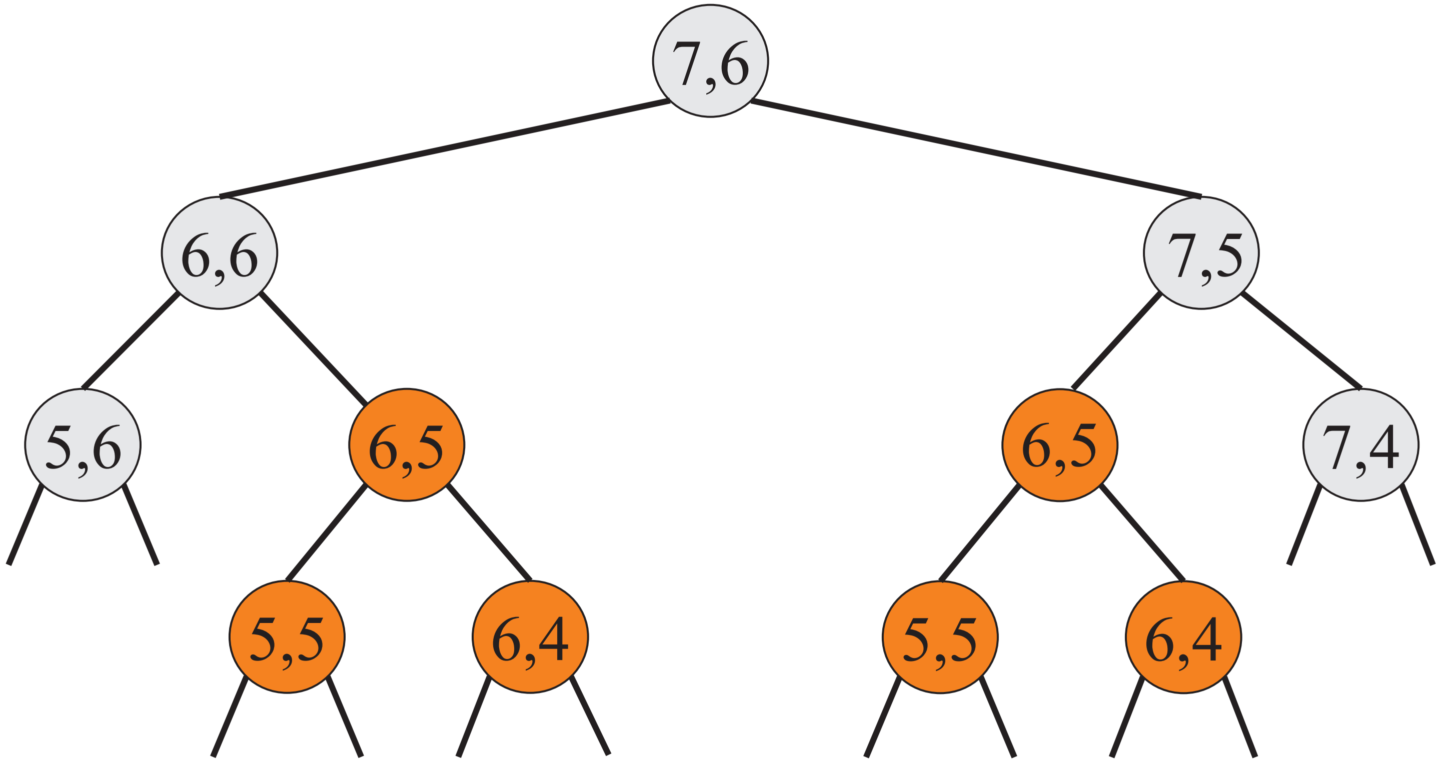
return $c[i,j]$

Peor caso: $x[i] \neq y[j]$ en este caso el algoritmo evalurá 2 subproblemas, cada uno con un sólo decremento

Ejemplo peor caso: *m*=7 n=6



Ejemplo peor caso: m=7 n=6



Los subproblemas se solapan
Estamos repitiendo trabajo

Problemas solapados

Cuando una solución recursiva contienen subproblemas que se repiten varias veces.

Es otra característica de los problemas de Programación Dinámica

Problemas solapados

Cuando una solución recursiva contienen subproblemas que se repiten varias veces.

Es otra característica de los problemas de Programación Dinámica

En nuestro problema, ¿cuál es el número de problemas distintos para cadenas de tamaño m y n?

Problemas solapados

Cuando una solución recursiva contienen subproblemas que se repiten varias veces.

Es otra característica de los problemas de Programación Dinámica

En nuestro problema, ¿cuál es el número de problemas distintos para cadenas de tamaño m y n? *Sólo el producto mn*

Memorizamos la solución: Top-Down

Después de calcular la solución de un subproblema la guardamos.

Si tenemos que resolver un caso ya resuelto buscamos la solución almacenada.

Memorizamos la solución: Top-Down

Después de calcular la solución de un subproblema la guardamos.

Si tenemos que resolver un caso ya resuelto buscamos la solución almacenada.

SCM(x, y, i, j)
    if $c[i, j] = \mathrm{NIL}$
        then if $x[i] = y[j]$
            then $c[i, j] \leftarrow$ SCM$(x, y, i-1, j-1) + 1$
            else $c[i, j] \leftarrow \max\{SCM(x, y, i-1, j)$,
                                SCM$(x, y, i, j-1)\}$
    return $c[i,j]$

Memorizamos la solución: Top-Down

Después de calcular la solución de un subproblema la guardamos.

Si tenemos que resolver un caso ya resuelto buscamos la solución almacenada.

SCM(x, y, i, j)
    if $c[i, j] = \text{NIL}$ ⟵ *me fijo si no está calculado*
        then if $x[i] = y[j]$
            then $c[i, j] \leftarrow \text{SCM}(x, y, i-1, j-1) + 1$
            else $c[i, j] \leftarrow \max\{\text{SCM}(x, y, i-1, j), \text{SCM}(x, y, i, j-1)\}$
    return $c[i,j]$

Memorizamos la solución: Top-Down

Después de calcular la solución de un subproblema la guardamos.

Si tenemos que resolver un caso ya resuelto buscamos la solución almacenada.

SCM(x, y, i, j)

  if $c[i, j] = \text{NIL}$ ← *me fijo si no está calculado*

    then if $x[i] = y[j]$

      then $c[i, j] \leftarrow$ SCM$(x, y, i-1, j-1) + 1$

      else $c[i, j] \leftarrow \max\{SCM(x, y, i-1, j)$, SCM$(x, y, i, j-1)\}$

  return $c[i,j]$

Solución Top - Down

Programación Dinámica: Bottom-up

Podemos resolver primero los problemas más chicos y combinar esas soluciones para resolver los problemas más grandes. Recorremos la tabla de abajo hacia arriba.

	ε	A	B	C	B	D	A	B
ε	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4

Programación Dinámica: Bottom-up

Podemos resolver primero los problemas más chicos y combinar esas soluciones para resolver los problemas más grandes. Recorremos la tabla de abajo hacia arriba.

	ε	A	B	C	B	D	A	B
ε	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4

$c[m,n]$

Programación Dinámica: Bottom-up

Podemos resolver primero los problemas más chicos y combinar esas soluciones para resolver los problemas más grandes. Recorremos la tabla de abajo hacia arriba.

	ε	A	B	C	B	D	A	B
ε	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4

$c[m,n]$

Complejidad: $\Theta(nm)$

Programación Dinámica: Bottom-up

Podemos resolver primero los problemas más chicos y combinar esas soluciones para resolver los problemas más grandes. Recorremos la tabla de abajo hacia arriba.

	ε	A	B	C	B	D	A	B
ε	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	**4**

Complejidad: $\Theta(nm)$

¿Cómo consigo una SCM?

$c[m,n]$

Programación Dinámica: Bottom-up

Podemos resolver primero los problemas más chicos y combinar esas soluciones para resolver los problemas más grandes. Recorremos la tabla de abajo hacia arriba.

	ε	A	B	C	B	D	A	B
ε	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4

$c[m,n]$

Complejidad: $\Theta(nm)$

¿Cómo consigo una SCM?

Volviendo para atras

Resumen

Hoy estudiamos una técnica de diseño de algoritmos, conocida como Programación Dinámica a través de un ejemplo SCM:

- Vimos la complejidad del problema y lo simplificamos.
- Estructuramos la solución en termino de problemas más chicos.
- Definimos recursivamente la solución
- Probamos subestructura óptima.
- Encontramos subproblemas que se solapan.
- Usamos memorización para no recalcular los problemas: algoritmo Top - Down.
- Vimos como resolver el problema con Programación Dinámica: Bottom-Up

Bibliografía

Introduction to Algorithms:
Thomas H. Cormen,Charles E. Leiserson,
Ronald L. Rivest,Clifford.