

Programación Dinámica



Prof. Alexander J. Benavides

Prof. Marcela Quispe Cruz

Prof. Manuel Loaiza

Escuela Profesional de Ingeniería de Sistemas
Universidad Nacional de San Agustín de Arequipa, Perú

Septiembre, 2024

Advertencia: Tome apuntes. Estas láminas no necesariamente tienen sentido fuera del aula.

Recuerde: Lo más importante es entender los métodos, traducir a C++ es secundario.

Material de estudio

- Hillier, F. S. & Lieberman, G. J. (2010). *Introducción a la Investigación de Operaciones* (Trad. 9^a Ed.). McGraw Hill. Secs. 10.1–10.3
- Wagner, H.M. (1969). *Principles of Operations Research, with Applications to Managerial Decisions*. Prentice-Hall. Sec. 8.2, p. 256–261
<http://archive.org/details/principles-of-operations-research-harvey-m.-wagner/page/256/>
- Levitin, A. (2012). *Introduction to the Design & Analysis of Algorithms* (3rd Ed). Pearson. C.8
- Goodrich, M.T. & Tamassia, R. (2015). *Algorithm Design and Applications*. John Wiley & Sons. C.12
- Cormen, T.H., Leiserson, C.E., Rivest, R.L., & Stein, C. (2022). *Introduction to Algorithms* (4th Ed). MIT Press. C.14

MIT OpenCourseWare:

<http://ocw.mit.edu/6-00F08> Lectures 12, 13 <http://youtu.be/udnyuHzJsjM&list=PL4C4720A6F225E074>

<http://ocw.mit.edu/6-006S20> Lecture 15 <http://youtu.be/r4-cftqTcdI&list=PLUI4u3cNGP63EdVPNLG3ToM6LaEUuStEY>

Objetivos de aprendizaje

Al terminar, el estudiante será capaz de:

1. Entender el método genérico de programación dinámica
2. Listar y describir los componentes básicos del método de programación dinámica: estados, relaciones recursivas, principio de optimalidad, memoización, ...
3. Utilizar el método de programación dinámica para resolver manualmente (en papel) problemas con múltiples estados
4. Diseñar funciones recursivas y de modificarlas para que almacenen y reutilicen los resultados de problemas pequeños en la solución de problemas cada vez más grandes

¿Qué es P.D.?
ooooooooo

P.D. manual
ooooooooo

P.D.: Memo + SRTBOT
oo

Fibonacci
ooooooooo

Mochila 0/1
oooooooooooo

Mochila ilimitada
ooooooooo

Programación Dinámica

¿Qué es programación dinámica?

Solución manual de problemas con programación dinámica

Programación dinámica: Memoización + Recursividad con SRTBOT

Calcular números de Fibonacci sin/con programación dinámica

Resolver el problema de la mochila 0/1 sin/con programación dinámica

Resolver el problema de la mochila ilimitada sin/con programación dinámica

¿Por qué se llama “programación dinámica”?

Bellman quería ocultar los términos "mathematical" y "research" del SecDef

Programación: planificación, calendarización, organización, de recursos, de eventos o de decisiones; *sin computador*

Dinámica: las decisiones o eventos son multi-estado, son sucesivas (varían) a través del tiempo, ...

Técnica matemática para optimizar procesos de decisión con múltiples estados

Bellman, R. (1952). On the theory of dynamic programming. *Proc. Natl. Acad. Sci.*, 38(8), 716–719.

Bellman, R. (1957). *Dynamic programming*. Princeton University Press.

¹⁰ Bellman, R. (1984). Eye of the Hurricane. World Scientific. p.159.

http://en.wikipedia.org/wiki/Dynamic_programming#History

Programación Dinámica [Lev12]

Si el problema es tomar una serie de decisiones, PD consiste en:

1. Formular el problema inicial usando subproblemas recursivos más pequeños
 2. Resolver los subproblemas pequeños (*recursivamente o con otros métodos*)
y guardar estas soluciones parciales MEMOIZACIÓN
 3. Usar las soluciones parciales guardadas junto con la opción más beneficiosa
para resolver subproblemas cada vez más grandes (hasta el problema inicial)

Reduce y Conquista (RECURSIVO) \uparrow MEMOIZACIÓN \uparrow Algoritmos Golosos

Programación dinámica es naturalmente recursivo, de arriba hacia abajo,

pero también puede ser implementado iterativo, de abajo hacia arriba

⊖ Iterativamente calcula todas las soluciones parciales para quizás usarlas

- ⊕ Recursivamente calcula solamente las soluciones parciales necesarias

Principio de Optimalidad [GT15,CLRS22]

PD requiere que la estructura del problema cumpla con el Principio de Optimalidad

Las soluciones óptimas de los subproblemas pequeños:

- ⊕ forman parte de la solución óptima del problema grande (**subestructura óptima**) y
 - ⊕ se pueden encontrar sin necesidad de revisar el problema grande (**independencia**)

Esto permite utilizar recursividad

Subproblemas Superpuestos

PD es eficiente sólo cuando los subproblemas pequeños se calculan repetidas veces

Esto permite utilizar memoización

Componentes básicos en programación dinámica

[HL10, Sec.10.2]

- 1. Etapa:** parte del problema en la que se puede tomar una decisión
Cada etapa tiene un conjunto de estados asociados
 - 2. Estado:** punto de decisión, donde se fija el valor de cada decisión
 - 3. Asociaciones:** relaciones recursivas que llevan de un estado a otro
 - 4. Principio de optimalidad:** (En una secuencia de decisiones...)

“Una política óptima tiene la propiedad de que, sin importar el estado inicial y las decisiones anteriores, las decisiones restantes deben constituir una política óptima con respecto al estado resultante de las primeras decisiones.”

Bellman (1957, Sc.3.3, p.83)

Es decir, **las decisiones restantes (problemas pequeños)** son óptimas e independientes

Ejemplo: Problema de la diligencia

[HL10, Sec.10.1]

Etapa-1

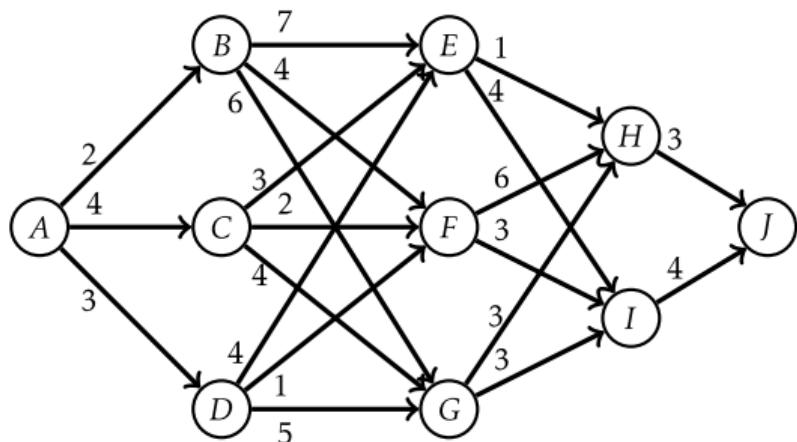
Etapa-2

Etapa-3

Etapa-

Etapa-5

Etapa-1 Etapa-2 Etapa-3 Etapa-4 Etapa-5 Propuesto por [Wag69, Sec.8.2, p.256–261]



	<i>B</i>	<i>C</i>	<i>D</i>
<i>A</i>	2	4	3

<i>E</i>	<i>F</i>	<i>G</i>
7	4	6
3	2	4
4	1	5

H	I
1	4
6	3
3	3

J
3
4

El Sr. Mark Off viajará de NY a SF para buscar fortuna en el salvaje oeste

Como las rutas son *muy* peligrosas
decide comprar seguros de vida

El costo de cada seguro depende de la seguridad de cada ruta

¿Cuáles son las rutas más seguras?

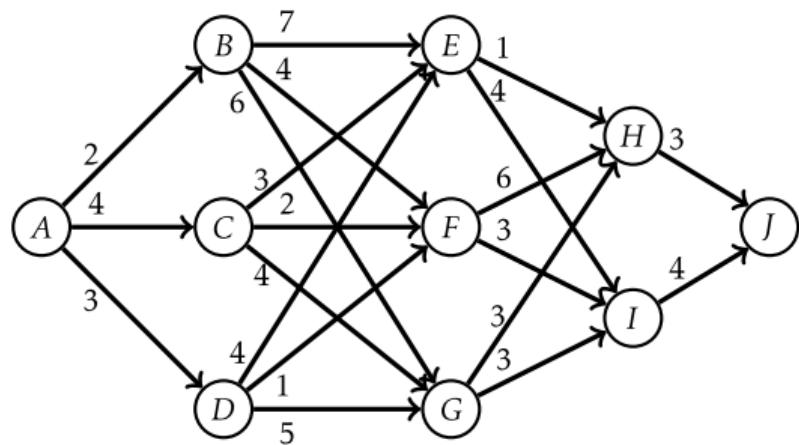
¿Cuál ruta completa minimiza costos?

Podría revisar todas las posibles rutas que hay en el árbol de decisión

Ejemplo: Problema de la diligencia

[HL10, Sec.10.1]

Etapa-1 Etapa-2 Etapa-3 Etapa-4 Etapa-5

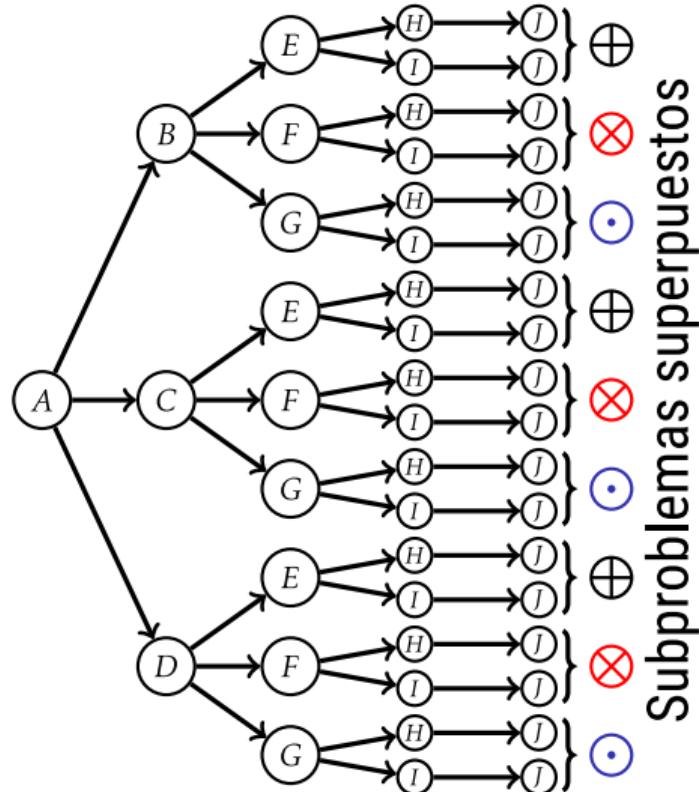


	<i>B</i>	<i>C</i>	<i>D</i>
<i>A</i>	2	4	3

<i>E</i>	<i>F</i>	<i>G</i>
7	4	6
3	2	4
4	1	5

H	I
1	4
6	3
3	3

J



Ejemplo: Problema de la diligencia

[HL10, Sec.10.1]

Etapa-1

Etapa-2

Etapa-3

Etapa-4

Etapa-5

- El árbol de decisión muestra que
- ⊕ Son 18 posibles rutas diferentes
 - ⊕ Hay subproblemas superpuestos

Principio de optimalidad:

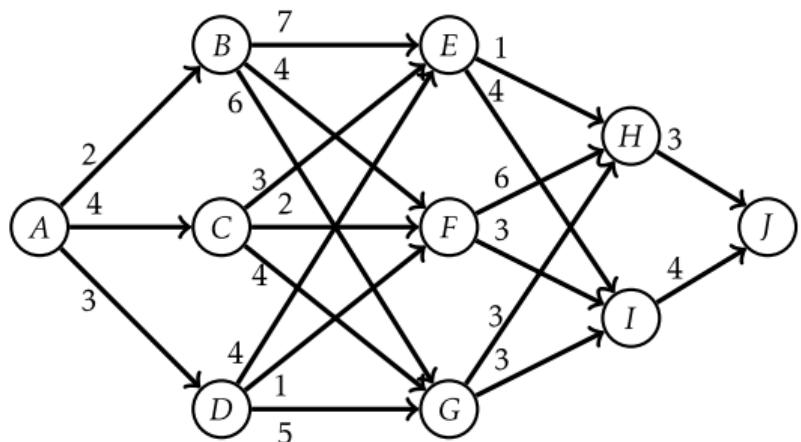
Independencia:

La decisión óptima de F a J
no depende de cómo se llega a F

subestructura óptima:

Con los óptimos de H e I
puedo calcular los óptimos de E , F y G

Si conociera los óptimos de E , F y G ,
podría calcular los óptimos de B , C y D
Y con esos podría calcular toda la ruta



A	B	C	D
2	4	3	

B	E	F	G
7	4	6	
3	2	4	
4	1	5	

E	H	I
1	4	
6	3	
3	3	

J
3

I
4

Ejemplo: Problema de la diligencia

[HL10, Sec.10.1]

Etapa-1

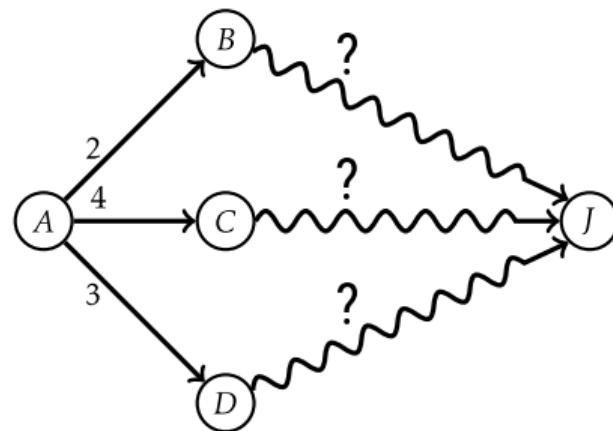
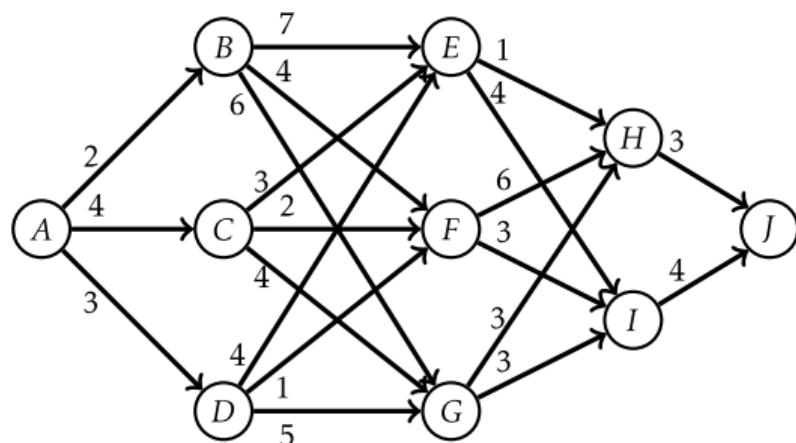
Etapa-2

Etapa-3

Etapa-4

Etapa-5 Etapa-1

Etapa-2 resuelta



A	B	C	D
2	4	3	

B	E	F	G
7	4	6	
3	2	4	
4	1	5	

E	H	I
1	4	
6	3	
3	3	

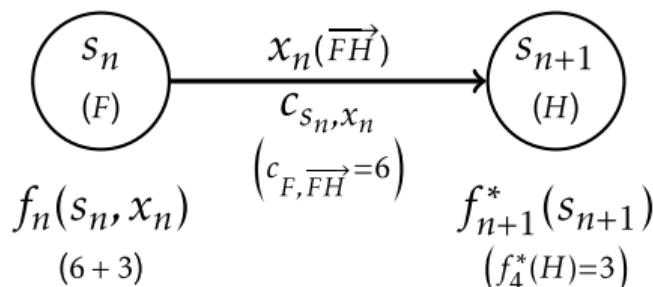
J
3
4

Pasos del método de programación dinámica [HL10]

1. Formular el problema como ecuaciones recursivas.
2. Resolver los subproblemas y guardar resultados.
3. Resolver los problemas mayores reutilizando las soluciones de problemas menores.

Formular como ecuaciones recursivas [HL10, Sec.10.2]

Etapa n
(3, actual)



$$f_n(s_n, x_n) = c_{s_n, x_n} + f_{n+1}^*(s_{n+1})$$

$$\begin{aligned} f_n^*(s_n) &= \min_{x_n} \{f_n(s_n, x_n)\} \\ &= f_n(s_n, x_n^*) \end{aligned}$$

N : número de etapas,

$n \in \{1, 2, \dots, N\}$ etapa actual,

$n + 1$: etapa ya resuelta,

s_n : un estado de la etapa actual n

s_{n+1} : estado en la etapa resuelta $n+1$

x_n : decisión que asocia dos estados

c_{s_n, x_n} : ganancia directa en el estado s_n cuando se toma la decisión x_n

$f_n(s_n, x_n)$: ganancia total en el estado s_n cuando se toma la decisión x_n

$f_n^*(s_n)$: valor óptimo para el estado s_n

x_n^* : decisión óptima para la etapa n

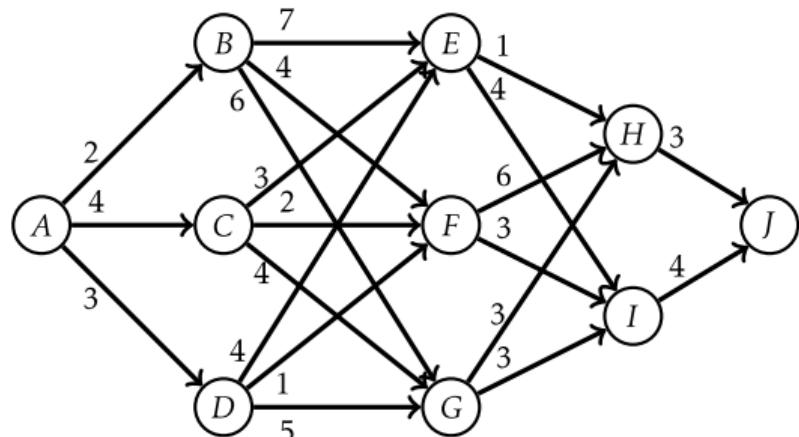
Tabla de decisión para solución manual [HL10, Sec.10.1]

Etapa-1

Etapa-2

Etapa-3

Etapa-4 Etapa-5



$$f_n(s_n, x_n) = c_{s_n, x_n} + f^*_{n+1}(s_{n+1})$$

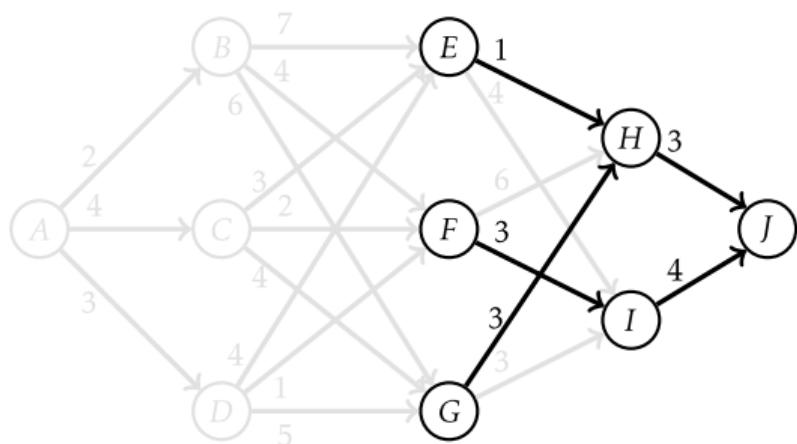
x_n	$f_n(s_n, x_n)$						
s_n	x_1	x_2	.	.	.	$f^*_n(s_n)$	x_n^*
s_1							
s_2							
\vdots							
s_n							

	B	C	D		E	F	G		H	I		J
A	2	4	3		7	4	6		1	4		
	3	2	4		6	3						
D	4	1	5		3	3						

Tabla de decisión para solución manual [HL10, Sec.10.1]

Etapa-3

Etapa-4



	<i>H</i>	<i>I</i>
<i>E</i>	1	4
<i>F</i>	6	3
<i>G</i>	3	3

$$f_n(s_n, x_n) = \textcolor{blue}{c}_{s_n, x_n} + f_{n+1}^*(s_{n+1})$$

x_3	$f_n(s_n, x_n)$			
s_3	H	I	$f_3^*(s_3)$	x_3^*
E	$1 + 3$	$4 + 4$	4	H
F	$6 + 3$	$3 + 4$	7	I
G	$3 + 3$	$3 + 4$	6	H

Etapa-4 resuelta: $f_4^*(H) = 3$

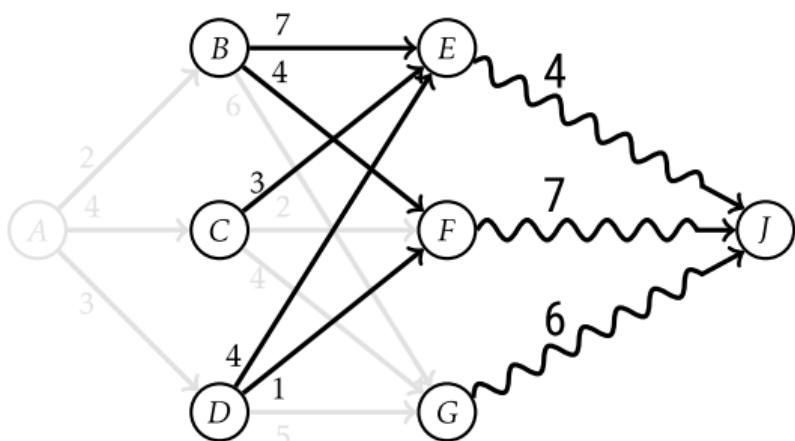
$$f_4^*(I) = 4$$

Tabla de decisión para solución manual [HL10, Sec.10.1]

Etapa-1

Etapa-2

Etapa-3



	B	C	D
A	2	4	3

	E	F	G
B	7	4	6
C	3	2	4
D	4	1	5

	H	I	J
E	1	4	
F	6	3	
G	3	3	

$$f_n(s_n, x_n) = c_{s_n, x_n} + f_{n+1}^*(s_{n+1})$$

x_2	$f_n(s_n, x_n)$			$f_{n+1}^*(s_{n+1})$	x_{n+1}^*
	E	F	G		
s_2					
B	$7 + 4$	$4 + 7$	$6 + 6$	11	$E \circ F$
C	$3 + 4$	$2 + 7$	$4 + 6$	7	E
D	$4 + 4$	$1 + 7$	$5 + 6$	8	$E \circ F$

Etapa-3 resuelta: $f_3^*(E) = 4$

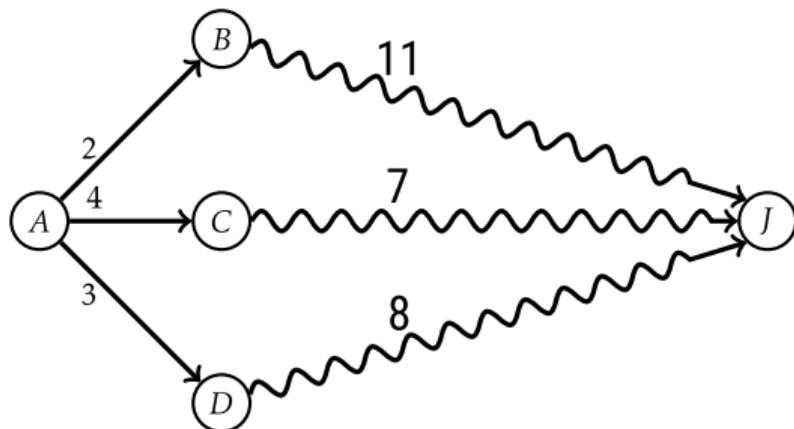
$$f_3^*(F) = 7$$

$$f_3^*(G) = 6$$

Tabla de decisión para solución manual [HL10, Sec.10.1]

Etapa-1

Etapa-2



	B	C	D	E	F	G	H	I	J
A	2	4	3	7	4	6	1	4	3
	3	2	4	6	3	3			
	4	1	5	3	3				

$$f_n(s_n, x_n) = c_{s_n, x_n} + f_{n+1}^*(s_{n+1})$$

x_1	$f_n(s_n, x_n)$				
s_1	B	C	D	$f_1^*(s_1)$	x_1^*
A	$2 + 11$	$4 + 7$	$3 + 8$	11	C o D

Etapa-2 resuelta: $f_2^*(B) = 11$

$$f_2^*(C) = 7$$

$$f_2^*(D) = 8$$

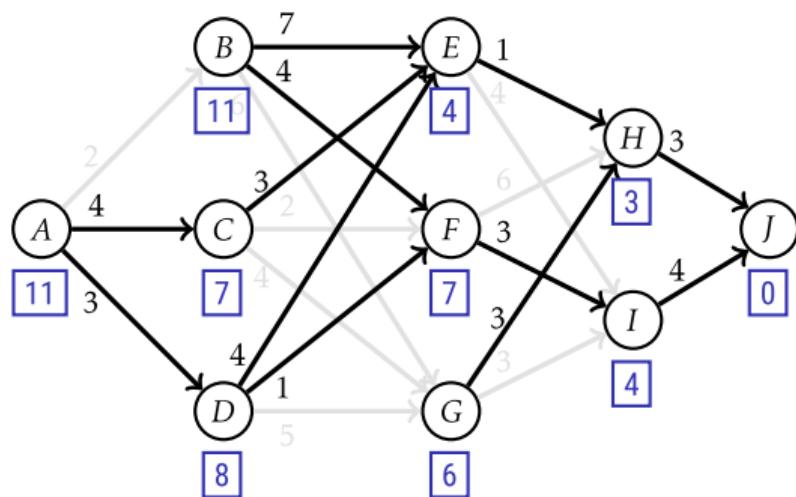
Tabla de decisión para solución manual [HL10, Sec.10.1]

Etapa-1

Etapa-2

Etapa-3

Etapa-4 Etapa-5



Rutas óptimas:

- (A, C, E, H, J)
- (A, D, E, H, J)
- (A, D, F, I, J)

s_4	$f_4^*(s_4)$	x_4^*
H	3	J
I	4	J

s_2	$f_2^*(s_2)$	x_2^*
B	11	E o F
C	7	E
D	8	E o F

s_3	$f_3^*(s_3)$	x_3^*
E	4	H
F	7	I
G	6	H

s_1	$f_1^*(s_1)$	x_1^*
A	11	C o D

Tabla de decisión para solución manual [HL10, Sec.10.1]

Etapa-1

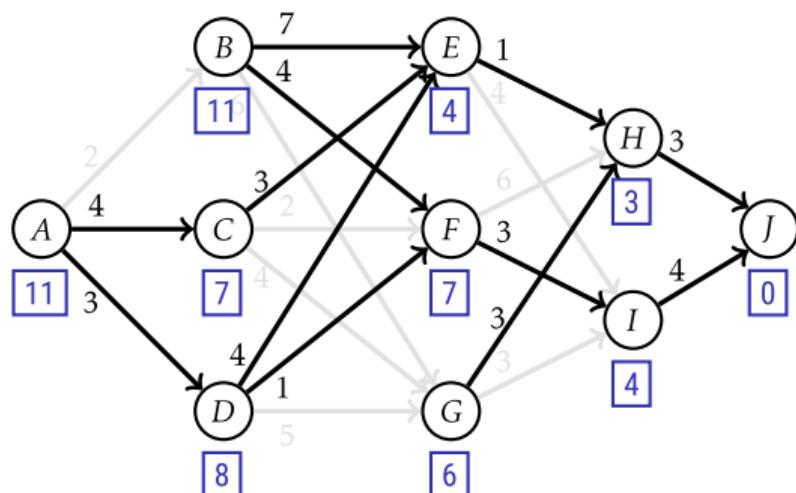
Etapa-2

Etapa-3

Etapa-4

Etapa-5

Cuestionario de [Wag69, Sec.8.2, p.261]



Rutas óptimas: (A, C, E, H, J)
 (A, D, E, H, J)
 (A, D, F, I, J)

Cuando estudie un nuevo problema pregúntese:

¿Cuáles son las variables de decisión?

¿Cuál es el criterio o la función objetivo para determinar una decisión óptima?

¿Cuáles son las etapas independientes que dividen adecuadamente el problema?

¿Cuáles son los estados dentro de cada etapa que representan al problema?

¿Cómo influencian las restricciones a los estados del problema y a los valores factibles?

Memoización: Guardar para ser recordado

Latín *memorandum* → Inglés *memo* → *memoize* → *memoization*

Michie, D. (1968). "Memo" functions and machine learning. *Nature*, 218(5136), 19–22.

Algorithm factorial(n)

Input: $n \in \mathbb{Z}_{\geq 0}$

Output: $n!$

```
1: if  $n = 0$  then
2:   | return 1
3: else
4:   | return  $n * \text{factorial}(n-1)$ 
```

Cuánto cuesta calcular $6!$, $8!$, $10!$

Algorithm factorialMEMO(n)

Input: $n \in \mathbb{Z}_{\geq 0}$

Output: $n!$

```
1: if  $n = 0$  then
2:   | return 1
3: else
4:   | if memo[ $n$ ] is undefined then
5:     |   | memo[ $n$ ] =  $n * \text{factorialMEMO}(n-1)$ 
6:   | return memo[ $n$ ]
```

Ambos algoritmos son $\mathcal{O}(n)$, pero memoizar amortiza el tiempo
memoizar es *muy eficiente* cuando hay subproblemas superpuestos

SRTBOT: Diseño de algoritmos recursivos

Subproblema: Definir los subproblemas en que se divide el problema (vértices)

Relacionar: Definir las relaciones recursivas entre los subproblemas (enlaces)

Topología: Garantizar que es un **Grafo Dirigido Acíclico**
Verificar la **subestructura óptima** y los **subproblemas superpuestos**

Básico: Establecer la solución de los subproblemas mínimos (**casos base**)

Original: Resolver subproblemas cada vez más grandes,
sin o con memoización (guardando soluciones para reutilizarlas)
hasta resolver el problema original

Tiempo: Analizar el tiempo de ejecución para resolver el problema original

MIT OCW: <http://youtu.be/r4-cftqTcdI>

Números de Fibonacci

Fibonacci (1202) *Liber abaci [Libro de cálculos]*.

Sigler, L. (2002). *Fibonacci's Liber Abaci: a translation into modern English of Leonardo Pisano's book of calculation*. Springer.

Introduce 1...9 hindúes, 0, álgebra, algoritmos, ...

Secuencia Fibonacci modela reproducción de conejos

$$f(0) = 0$$

$$f(1) = 1$$

$$f(n) = f(n - 1) + f(n - 2)$$

Algorithm $F(n)$

Input: $n \in \mathbb{Z}_{\geq 0}$

Output: n^{th} Fibonacci number

```

1: if  $n = 0$  or  $n = 1$  then
2:   return  $n$ 
3: else
4:   return  $F(n-1)+F(n-2)$ 

```

$f(0)$	$f(1)$	$f(2)$	$f(3)$	$f(4)$	$f(5)$	$f(6)$	$f(7)$	$f(8)$	$f(9)$	$f(10)$	$f(11)$	$f(12)$	$f(13)$	$f(14)$	$f(15)$...
0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	...

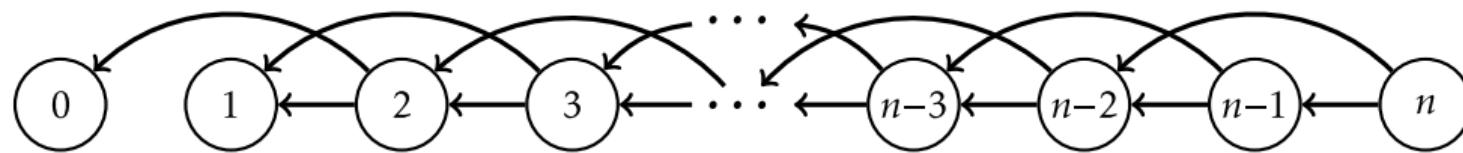
Por favor calcule $f(46)$ y $f(92)$

Fibonacci con SRTBOT

Subproblema: Encuentre $f(i)$ para $i \in \mathbb{Z}_{\geq 0}$

Relacionar: $f(i) = f(i - 1) + f(i - 2)$

Topología:

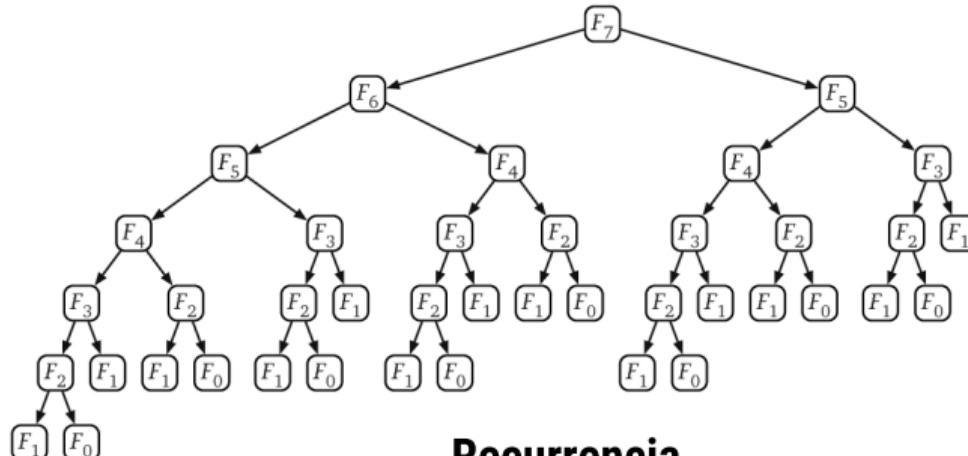


Básico: $f(i) = i$ para $i \in \{0, 1\}$

Original: Implementar primero sin memoización y después con memoización

Tiempo:

Fibonacci recursivo sin memoización



Recurrencia

Fibonacci: $f(n) = f(n - 1) + f(n - 2) \implies T_f(n) = T_f(n - 1) + T_f(n - 2) + c$

$$\begin{array}{lll} f(0) = 0 & f(1) = 1 & T_f(0) = T_f(1) = c_0 \end{array}$$

Límite Inferior: $li(n) = 2li(n - 2) \implies T_i(n) = 2T_i(n - 2) + c_i$

Límite Superior: $ls(n) = 2ls(n - 1) \implies T_s(n) = 2T_s(n - 1) + c_s$

Algorithm $F(n)$

Input: $n \in \mathbb{Z}_{\geq 0}$

Output: n^{th} Fibonacci number

```

1: if  $n = 0$  or  $n = 1$  then
2:   return  $n$ 
3: else
4:   return  $F(n-1)+F(n-2)$ 
  
```

Tiempo

Límite Inferior

$$\begin{aligned}
 T_i(0) &= T_i(1) = c_{i_0} \\
 T_i(n) &= 2T_i(n-2) + c_i \\
 &= 2(2T_i(n-4) + c_i) + c_i \\
 &= 2^2 T_i(n-4) + 2c_i \\
 &= 2^2(2T_i(n-6) + c_i) + 2c_i \\
 &= 2^3 T_i(n-6) + 3c_i \\
 &\quad \vdots \\
 &= 2^k T_i(n-2k) + kc_i \\
 n - 2k &= 0 \implies k = \frac{n}{2} \\
 &= 2^{\frac{n}{2}} T_i(0) + \binom{n}{2} c_i
 \end{aligned}$$

$$T_i(n) = c_{i_0} \left(\sqrt{2}\right)^n + \left(\frac{c_i}{2}\right)n$$

$$T_i(n) \in \Omega\left(\left(\sqrt{2}\right)^n\right)$$

Fibonacci: $T_f(n) \in \Theta(\varphi^n)$ con $\sqrt{2} \leq \varphi \leq 2$

Límite Superior

$$\begin{aligned}
 T_s(0) &= c_{s_0} \\
 T_s(n) &= 2T_s(n-1) + c_s \\
 &= 2(2T_s(n-2) + c_s) + c_s \\
 &= 2^2 T_s(n-2) + 2c_s \\
 &= 2^2(2T_s(n-3) + c_s) + 2c_s \\
 &= 2^3 T_s(n-3) + 3c_s \\
 &\quad \vdots \\
 &= 2^k T_s(n-k) + kc_s \\
 n - k &= 0 \implies k = n \\
 &= 2^n T_s(0) + nc_s
 \end{aligned}$$

$$T_s(n) = c_{s_0} 2^n + c_s n$$

$$T_s(n) \in \mathcal{O}(2^n)$$

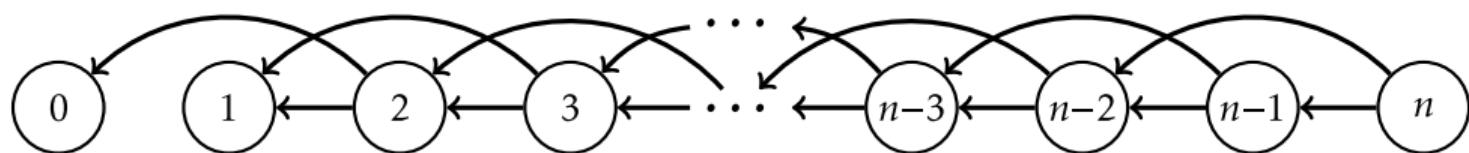
golden ratio: $\varphi = (1+\sqrt{5})/2 \approx 1.618$

Fibonacci con SRTBOT

Subproblema: Encuentre $f(i)$ para $i \in \mathbb{Z}_{\geq 0}$

Relacionar: $f(i) = f(i - 1) + f(i - 2)$

Topología:

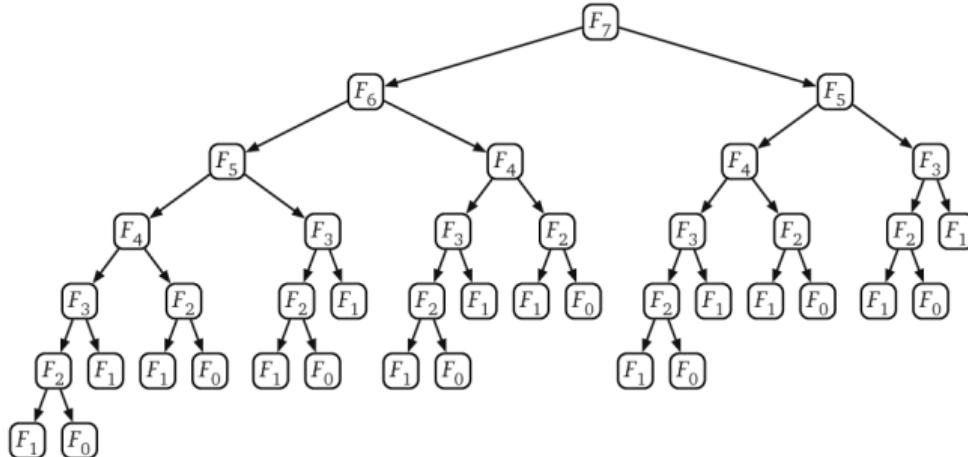


Básico: $f(i) = i$ para $i \in \{0, 1\}$

Original: Implementar primero sin memoización y después con memoización

Tiempo: sin memoización $T_f(n) \in \Theta(\varphi^n)$
con memoización $T_f(n) \in \Theta(n)$

Fibonacci recursivo sin memoización



Algorithm $F(n)$

Input: $n \in \mathbb{Z}_{\geq 0}$

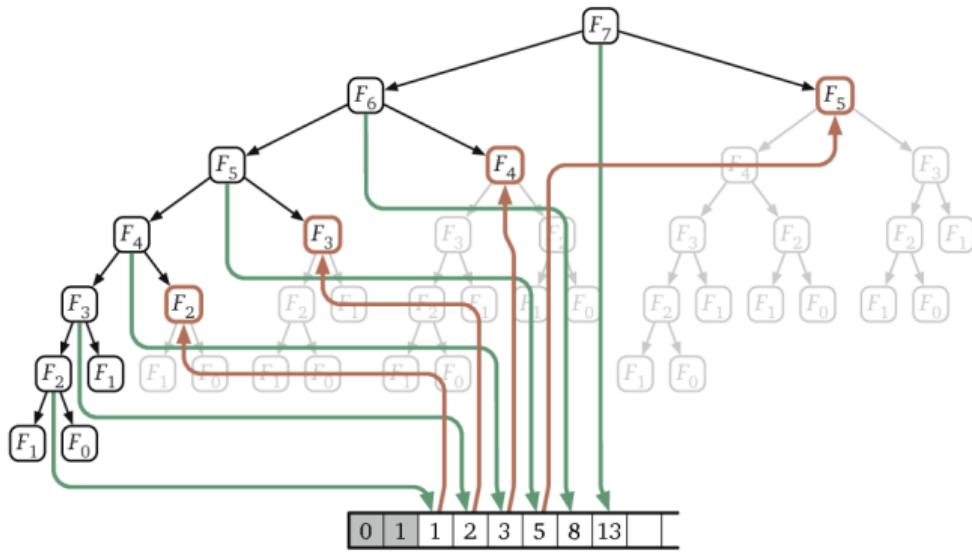
Output: n^{th} Fib.

```
1: if  $n = 0$  or  $n = 1$  then
2:   return  $n$ 
3: else
4:   return  $F(n-1)+F(n-2)$ 
5:
6:
```

$$T_f(n) \in \Theta(\varphi^n)$$

$$\varphi = (1+\sqrt{5})/2 \approx 1.618$$

Fibonacci recursivo con memoización



Algorithm $FM(n)$

Input: $n \in \mathbb{Z}_{\geq 0}$ **Output:** n^{th} Fib.

```
1: if  $n = 0$  or  $n = 1$  then
2:   return  $n$ 
3: else
4:   if  $M[n]$  is undefined then
5:      $M[n] = FM(n-1) + FM(n-2)$ 
6:   return  $M[n]$ 
```

$$T_f(n) \in \Theta(n)$$

Problema de la mochila: 0/1 knapsack problem

Un ladrón encuentra en una casa

Objeto	Costo(C)	Peso(W)
1. Ajedrez	5	3
2. Barco	7	3
3. Campana	9	2
4. Jarrón	14	5

Mochila con peso límite $wl = 5$

$$\begin{aligned} & \text{Maximizar } \sum x_i c_i, \\ & \text{sujeto a } \sum x_i w_i \leq wl, \\ & \quad x_i \in \{0, 1\} \end{aligned}$$

¿Qué debería llevar el ladrón?

Ladrón rápido (algoritmo goloso):

Lleva lo más caro

Ladrón minucioso (búsq. exhaustiva):

Evalua todas las posibles soluciones para encontrar la óptima

¿Cuántas posibles soluciones debemos revisar si tenemos n objetos para decidir?

Problema de la mochila 0/1: Espacio de soluciones

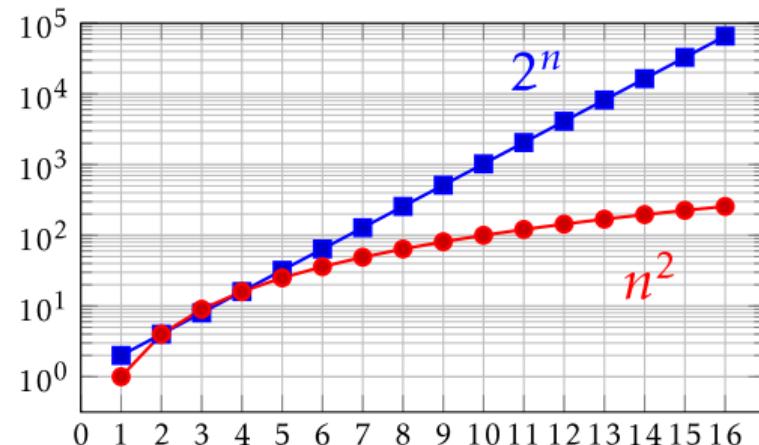
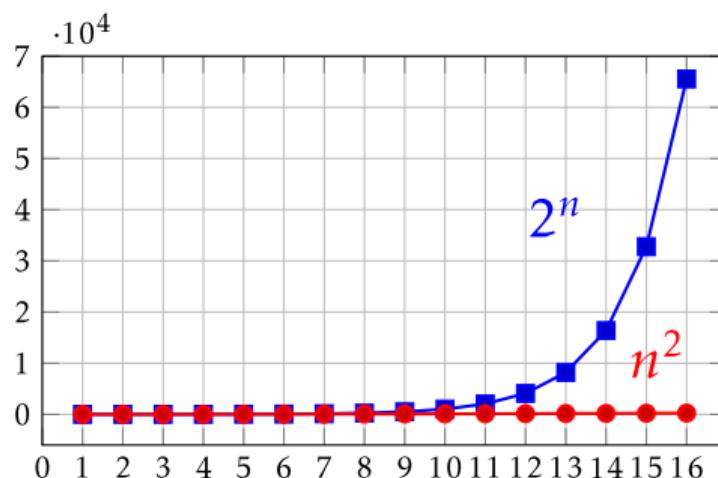
Para 2 objetos: $2 * 2$

Para 3 objetos: $2 * 2 * 2$

Para 4 objetos: $2 * 2 * 2 * 2$

Para n objetos: 2^n

Búsqueda exhaustiva $\in \mathcal{O}(2^n)$



Problema de la mochila 0/1: solución manual

Preparación

Objeto	Costo(C)	Peso(W)
1. Ajedrez	5	3
2. Barco	7	3
3. Campana	9	2
4. Jarrón	14	5

Mochila con peso límite $wl = 5$

$$f_n(s_n, x_n) = c_{s_n, x_n} + f_{n+1}^*(s_{n+1})$$

x_n	$f_n(s_n, x_n)$		$f_n^*(s_n)$	x_n^*
s_n	0	1		
0				
1				
2				
3				
4				
5				

Problema de la mochila 0/1: solución manual

Etapa 4

Objeto	Costo(C)	Peso(W)
1. Ajedrez	5	3
2. Barco	7	3
3. Campana	9	2
4. Jarrón	14	5

Mochila con peso límite $wl = 5$

Etapa-5 resuelta: $f_5^*(s_5) = 0$

$$f_n(s_n, x_n) = c_{s_n, x_n} + f_{n+1}^*(s_{n+1})$$

x_4	$f_n(s_n, x_n)$		$f_n^*(s_n)$	x_n^*
s_4	0	1		
0	0 + 0	-	0	0
1	0 + 0	-	0	0
2	0 + 0	-	0	0
3	0 + 0	-	0	0
4	0 + 0	-	0	0
5	0 + 0	14 + 0	14	1

Problema de la mochila 0/1: solución manual

Etapa 3

Objeto	Costo(C)	Peso(W)
1. Ajedrez	5	3
2. Barco	7	3
3. Campana	9	2
4. Jarrón	14	5

Mochila con peso límite $wl = 5$

Etapa-4 resuelta: $f_4^*(0, \dots, 4) = 0$

$$f_4^*(5) = 14$$

$$f_n(s_n, x_n) = c_{s_n, x_n} + f_{n+1}^*(s_{n+1})$$

x_3	$f_n(s_n, x_n)$		$f_n^*(s_n)$	x_n^*
	0	1		
s_3	0	1		
0	$0 + 0$	-	0	0
1	$0 + 0$	-	0	0
2	$0 + 0$	$9 + 0$	9	1
3	$0 + 0$	$9 + 0$	9	1
4	$0 + 0$	$9 + 0$	9	1
5	$0 + 14$	$9 + 0$	14	0

Problema de la mochila 0/1: solución manual

Etapa 2

Objeto	Costo(C)	Peso(W)
1. Ajedrez	5	3
2. Barco	7	3
3. Campana	9	2
4. Jarrón	14	5

Mochila con peso límite $wl = 5$

Etapa-3 resuelta: $f_3^*(0, 1) = 0$

$$f_3^*(2, 3, 4) = 9$$

$$f_3^*(5) = 14$$

$$f_n(s_n, x_n) = c_{s_n, x_n} + f_{n+1}^*(s_{n+1})$$

x_2	$f_n(s_n, x_n)$		$f_n^*(s_n)$	x_n^*
	0	1		
s_2	0	1		
0	$0 + 0$	-	0	0
1	$0 + 0$	-	0	0
2	$0 + 9$	-	9	0
3	$0 + 9$	$7 + 0$	9	0
4	$0 + 9$	$7 + 0$	9	0
5	$0 + 14$	$7 + 9$	16	1

Problema de la mochila 0/1: solución manual

Etapa 1

Objeto	Costo(C)	Peso(W)
1. Ajedrez	5	3
2. Barco	7	3
3. Campana	9	2
4. Jarrón	14	5

$$f_n(s_n, x_n) = c_{s_n, x_n} + f_{n+1}^*(s_{n+1})$$

x_1	$f_n(s_n, x_n)$			
s_1	0	1	$f_n^*(s_n)$	x_n^*
5	$0 + 16$	$5 + 9$	16	0

Mochila con peso límite $wl = 5$

Etapa-2 resuelta: $f_2^*(0, 1) = 0$

$$f_2^*(2, 3, 4) = 9$$

$$f_2^*(5) = 16$$

Problema de la mochila 0/1: solución manual

Interpretando

Objeto	Costo(C)	Peso(W)
1. Ajedrez	5	3
2. Barco	7	3
3. Campana	9	2
4. Jarrón	14	5

Mochila con peso límite $wl = 5$

$$f_n(s_n, x_n) = c_{s_n, x_n} + f^*_{n+1}(s_{n+1})$$

	$f_4^*(s_4)$	x_4^*	$f_3^*(s_3)$	x_3^*	$f_2^*(s_2)$	x_2^*	$f_1^*(s_1)$	x_1^*
0	0	0	0	0	0	0		
1	0	0	0	0	0	0		
2	0	0	9	1	9	0		
3	0	0	9	1	9	0		
4	0	0	9	1	9	0		
5	14	1	14	0	16	1	16	0

¿Y si la mochila tuviera un límite de 7?

Problema de la mochila 0/1 con SRTBOT

Subproblema: Encuentre $F(i, w)$ para $0 \leq i \leq n, 0 \leq w \leq wl$;

Relacionar: $F(i, w) = F(i-1, w)$ si $w < w_i$

$$F(i, w) = \max (c_i + F(i-1, w-w_i), F(i-1, w)) \text{ si } w \geq w_i$$

Topología: Grafo Dirigido Acíclico (grafíquelo)

Básico: $F(\emptyset, w) = 0$ $F(i, 0) = 0$

Original: Implementar primero sin memoización y después con memoización

Tiempo: con memoización $T(n, wl) \in \mathcal{O}(n * wl)$

Problema de la mochila 0/1 con SRTBOT

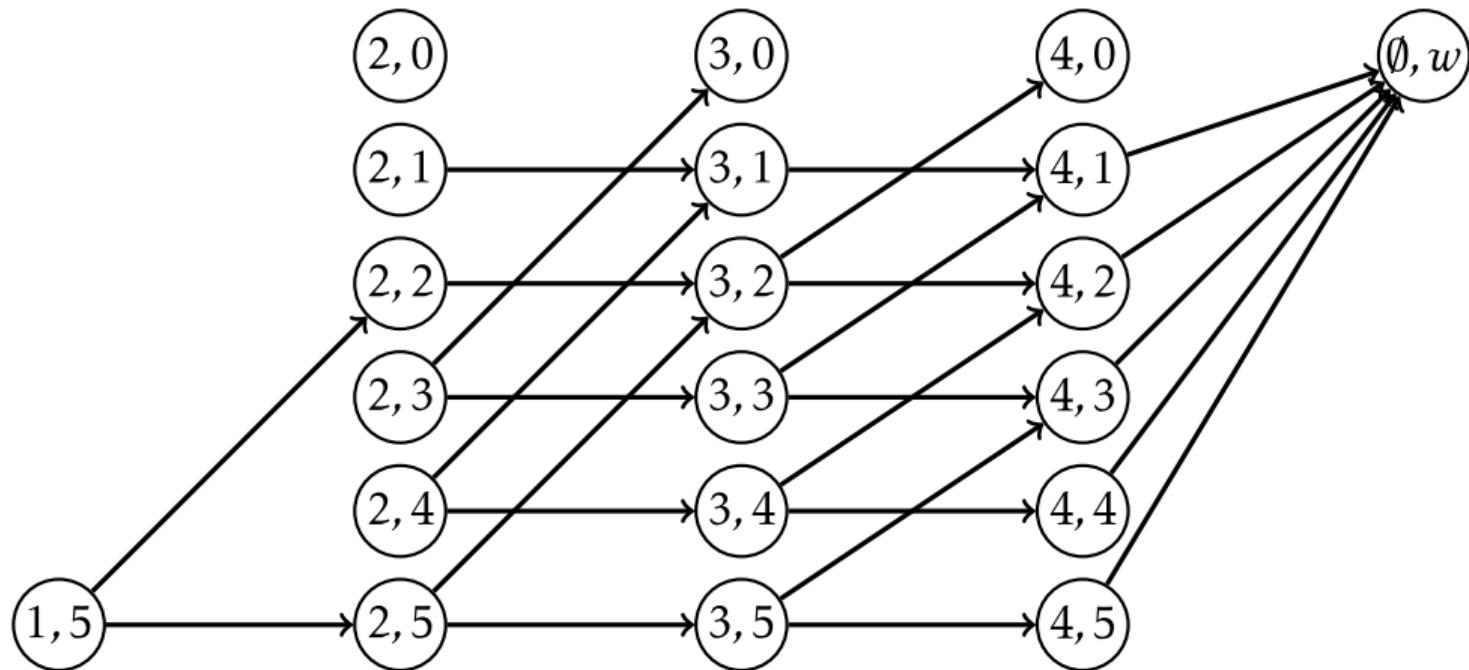
Etapa-1

Etapa-2

Etapa-3

Etapa-4

Etapa-5



Problema de la mochila 0/1 sin memoización

Algorithm $ks(i, w)$

Input: object i , weight limit w **Output:** maximum cost

```
1: if  $w = 0$  or  $i = 0$  then
2:   | return 0
3: else
4:   | if  $w < W[i]$  then
5:     |   | return  $ks(i - 1, w)$ 
6:   | else
7:     |   | return  $\max(ks(i - 1, w), C[i] + ks(i - 1, w - W[i]))$ 
```

Problema de la mochila 0/1 con memoización

Algorithm $ksM(i, w)$

Input: object i , weight limit w **Output:** maximum cost

```
1: if  $w = 0$  or  $i = 0$  then
2:   | return 0
3: else
4:   | if  $M[i, w]$  is undefined then
5:     |   | if  $w < W[i]$  then
6:       |     |    $M[i, w] = ksM(i - 1, w)$ 
7:     |   | else
8:       |     |    $M[i, w] = \max(ksM(i - 1, w), C[i] + ksM(i - 1, w - W[i]))$ 
9:   | return  $M[i, w]$ 
```

Problema de la mochila ilimitada: unlimited knapsack problem

Un ladrón encuentra en una tienda

Objeto	Costo(C)	Peso(W)
1. Ajedrez	5	3
2. Barco	7	3
3. Campana	9	2
4. Jarrón	14	5

Mochila con peso límite $wl = 7$

Maximizar $\sum x_i c_i$,

sujeto a $\sum x_i w_i \leq wl$,

$x_i \in \mathbb{Z}_{\geq 0}$

Ahora es una tienda

Hay un número ilimitado de cada objeto

El ladrón puede recoger
varios duplicados del mismo objeto

¿Qué tendría que cambiar?

¿en la solución manual?

¿en el modelo **SRTBOT**?

¿en el pseudocódigo?

Problema de la mochila ilimitada: solución manual

Etapa 4

Objeto	Costo(C)	Peso(W)
1. Ajedrez	5	3
2. Barco	7	3
3. Campana	9	2
4. Jarrón	14	5

Mochila con peso límite $wl = 7$

Etapa-5 resuelta: $f_5^*(s_5) = 0$

$$f_n(s_n, x_n) = c_{s_n, x_n} + f_{n+1}^*(s_{n+1})$$

x_4	$f_n(s_n, x_n)$		$f_4^*(s_4)$	x_4^*
	0	1		
s_4	0	1		
0	$0 + 0$	-	0	0
1	$0 + 0$	-	0	0
2	$0 + 0$	-	0	0
3	$0 + 0$	-	0	0
4	$0 + 0$	-	0	0
5	$0 + 0$	$14 + 0$	14	1
6	$0 + 0$	$14 + 0$	14	1
7	$0 + 0$	$14 + 0$	14	1

Problema de la mochila ilimitada: solución manual

Etapa 3

$$f_n(s_n, x_n) = \textcolor{blue}{c}_{s_n, x_n} + f_{n+1}^*(s_{n+1})$$

Objeto	Costo(C)	Peso(W)
1. Ajedrez	5	3
2. Barco	7	3
3. Campana	9	2
4. Jarrón	14	5

Mochila con peso límite $wl = 7$

Etapa-4 resuelta: $f_4^*(0, \dots, 4) = 0$

$$f_4^*(5,6,7) = 14$$

x_3	$f_n(s_n, x_n)$				$f_3^*(s_3)$	x_3^*
s_3	0	1	2	3		
0	0+0	-	-	-	0	0
1	0+0	-	-	-	0	0
2	0+0	9+0	-	-	9	1
3	0+0	9+0	-	-	9	1
4	0+0	9+0	18+0	-	18	2
5	0+14	9+0	18+0	-	18	2
6	0+14	9+0	18+0	27+0	27	3
7	0+14	9+14	18+0	27+0	27	3

Problema de la mochila ilimitada: solución manual

Etapa 2

Objeto	Costo(C)	Peso(W)
1. Ajedrez	5	3
2. Barco	7	3
3. Campana	9	2
4. Jarrón	14	5

Mochila con peso límite $wl = 7$

Etapa-3 resuelta: $f_3^*(0, 1) = 0$

$$f_3^*(2, 3) = 9$$

$$f_3^*(4, 5) = 18$$

$$f_3^*(6, 7) = 27$$

$$f_n(s_n, x_n) = c_{s_n, x_n} + f_{n+1}^*(s_{n+1})$$

x_2	$f_n(s_n, x_n)$			$f_3^*(s_3)$	x_3^*
	0	1	2		
s_2					
0	$0 + 0$	-	-	0	0
1	$0 + 0$	-	-	0	0
2	$0 + 9$	-	-	9	0
3	$0 + 9$	$7 + 0$	-	9	0
4	$0 + 18$	$7 + 0$	-	18	0
5	$0 + 18$	$7 + 9$	-	18	0
6	$0 + 27$	$7 + 9$	$14 + 0$	27	0
7	$0 + 27$	$7 + 18$	$14 + 0$	27	0

Problema de la mochila ilimitada: solución manual

Etapa 1

Objeto	Costo(C)	Peso(W)
1. Ajedrez	5	3
2. Barco	7	3
3. Campana	9	2
4. Jarrón	14	5

$$f_n(s_n, x_n) = c_{s_n, x_n} + f_{n+1}^*(s_{n+1})$$

		$f_n(s_n, x_n)$				
		0	1	2	$f_1^*(s_1)$	x_1^*
s_1	x_1					
7		0 + 27	5 + 18	10 + 0	27	0

Mochila con peso límite $wl = 7$

Etapa-2 resuelta: $f_2^*(0, 1) = 0$

$$f_2^*(2, 3) = 9$$

$$f_2^*(4, 5) = 18$$

$$f_2^*(6, 7) = 27$$

Problema de la mochila ilimitada: solución manual

Interpretando

Objeto	Costo(C)	Peso(W)
1. Ajedrez	5	3
2. Barco	7	3
3. Campana	9	2
4. Jarrón	14	5

Mochila con peso límite $wl = 7$

$$f_n(s_n, x_n) = c_{s_n, x_n} + f^*_{n+1}(s_{n+1})$$

	$f_4^*(s_4)$	x_4^*	$f_3^*(s_3)$	x_3^*	$f_2^*(s_2)$	x_2^*	$f_1^*(s_1)$	x_1^*
0	0	0	0	0	0	0		
1	0	0	0	0	0	0		
2	0	0	9	1	9	0		
3	0	0	9	1	9	0		
4	0	0	18	2	18	0		
5	14	1	18	2	18	0		
6	14	1	27	3	27	0		
7	14	1	27	3	27	0	27	0

¿Y si la mochila tuviera un límite de 5?

Problema de la mochila ilimitada con SRTBOT

Subproblema: Encuentre $F(i, w)$ para $0 \leq i \leq n, 0 \leq w \leq wl$;

Relacionar: $F(i, w) = F(i-1, w)$ si $w < w_i$

$$F(i, w) = \max (c_i + F(i-1, w-w_i), F(i-1, w)) \text{ si } w \geq w_i$$

Topología: Grafo Dirigido Acíclico (grafíquelo)

Básico: $F(0, w) = 0$ $F(i, 0) = 0$

Original: Implementar primero sin memoización y después con memoización

Tiempo: con memoización $T(n, wl) \in \mathcal{O}(n * wl)$

Problema de la mochila ilimitada con memoización

Algorithm $uksM(i, w)$

Input: object i , weight limit w **Output:** maximum cost

```
1: if  $w = 0$  or  $i = 0$  then
2:   | return 0
3: else
4:   | if  $M[i, w]$  is undefined then
5:     |   | if  $w < W[i]$  then
6:       |     |    $M[i, w] = uksM(i - 1, w)$ 
7:     |   | else
8:       |     |    $M[i, w] = \max(uksM(i - 1, w), C[i] + uksM(i - 1, w - W[i]))$ 
9:   | return  $M[i, w]$ 
```
