

# Algoritmos y estructuras de datos

## Programación Dinámica

CEIS

Escuela Colombiana de Ingeniería

2024-1

# Agenda

## ① Rod-Cutting Problem

Formulación

Diseño

Análisis

Programación dinámica

Diseño

## ② Aspectos finales

Ejercicios

# Agenda

## ① Rod-Cutting Problem

Formulación

Diseño

Análisis

Programación dinámica

Diseño

## ② Aspectos finales

Ejercicios

# Formulación

Dado una varilla de longitud  $n$  y una tabla de precios  $p_i$  para  $i = 1, 2, \dots, n$ . Determinar la ganancia máxima  $r_n$  obtenible de cortar la varilla y vender las piezas.

length $i$	1	2	3	4	5	6	7	8	9	10
price $p_i$	1	5	8	9	10	17	17	20	24	30

# Diseño

Se puede cortar la varilla de  $2^{n-1}$  formas diferentes

$n=4$



(a)



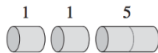
(b)



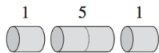
(c)



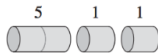
(d)



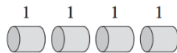
(e)



(f)



(g)



(h)

length $i$	1	2	3	4	5	6	7	8	9	10
price $p_i$	1	5	8	9	10	17	17	20	24	30

## Dividir y conquistar


$r_n$  : La máxima ganancia para una varilla de longitud  $n$   
considerando los precios  $p[1..N]$ ,  $n \leq N$

## Dividir y conquistar


$r_n$  : La máxima ganancia para una varilla de longitud  $n$   
considerando los precios  $p[1..N]$ ,  $n \leq N$

$$r_n \quad 0 \\ n = 0$$

$$r_n \\ n \geq 1$$

$$p_1 + \quad r_{n-1}$$


1

$$p_2 + \quad r_{n-2}$$


2

$$\max_{1 \leq i \leq n} (p_i + r_{n-i})$$

$$p_n + \quad r_0$$


n

## Dividir y conquistar

$r_n$  : La máxima ganancia para una varilla de longitud  $n$  considerando los precios  $p[1..N]$ ,  $n \leq N$

$$r_n = \begin{cases} 0 & n = 0 \\ \max_{1 \leq i \leq n} (p_i + r_{n-i}) & n \geq 1 \end{cases}$$

$n$  llamadas recursivas



## Dividir y conquistar

CUT-ROD( $p, n$ ) es la máxima ganancia para una varilla de longitud  $n$  considerando los precios  $p[1..N]$ ,  $n \leq N$

$$r_n = \begin{cases} 0 & n = 0 \\ \max_{1 \leq i \leq n} (p_i + r_{n-i}) & n \geq 1 \end{cases}$$

CUT-ROD( $p, n$ )

```
1  if  $n == 0$ 
2      return 0
3   $q = -\infty$ 
4  for  $i = 1$  to  $n$ 
5       $q = \max(q, p[i] + \text{CUT-ROD}(p, n - i))$ 
6  return  $q$ 
```

# Análisis

CUT-ROD( $p, n$ )

```
1  if  $n == 0$ 
2      return 0
3   $q = -\infty$ 
4  for  $i = 1$  to  $n$ 
5       $q = \max(q, p[i] + \text{CUT-ROD}(p, n - i))$ 
6  return  $q$ 
```

$$T(n) = 1 + \sum_{j=0}^{n-1} T(j) . \quad (15.3)$$

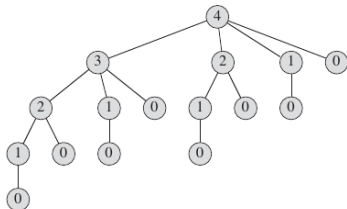
$$T(n) = 2^n , \quad (15.4)$$

# Análisis

CUT-ROD( $p, n$ )

```
1  if  $n == 0$ 
2    return 0
3   $q = -\infty$ 
4  for  $i = 1$  to  $n$ 
5     $q = \max(q, p[i] + \text{CUT-ROD}(p, n - i))$ 
6  return  $q$ 
```

$n=4$



Se resuelven repetidamente algunos problemas.

# Programación dinámica

Al ver que una función recursiva resuelve repetidamente el mismo problema, podemos modificar el algoritmo para solo solucionar estos subproblemas solo una vez y almacenar la solución.

Se hace un trade-off entre memoria y tiempo. Pudiendo convertir soluciones exponenciales a un tiempo polinómico.

# Programación dinámica

CUT-ROD( $p, n$ )

```
1  if  $n == 0$ 
2      return 0
3   $q = -\infty$ 
4  for  $i = 1$  to  $n$ 
5       $q = \max(q, p[i] + \text{CUT-ROD}(p, n - i))$ 
6  return  $q$ 
```

Se escribe un algoritmo recursivo de forma normal, pero lo modificamos para guardar el resultado de cada subproblema. Entonces el procedimiento verifica si ese subproblema ha sido resuelto previamente o no.

MEMOIZED-CUT-ROD( $p, n$ )

```
1  let  $r[0..n]$  be a new array
2  for  $i = 0$  to  $n$ 
3       $r[i] = -\infty$ 
4  return MEMOIZED-CUT-ROD-AUX( $p, n, r$ )
```

MEMOIZED-CUT-ROD-AUX( $p, n, r$ )

```
1  if  $r[n] \geq 0$ 
2      return  $r[n]$ 
3  if  $n == 0$ 
4       $q = 0$ 
5  else  $q = -\infty$ 
6      for  $i = 1$  to  $n$ 
7           $q = \max(q, p[i] + \text{MEMOIZED-CUT-ROD-AUX}(p, n - i, r))$ 
8   $r[n] = q$ 
9  return  $q$ 
```

# Agenda

## ① Rod-Cutting Problem

Formulación

Diseño

Análisis

Programación dinámica

Diseño

## ② Aspectos finales

Ejercicios

## Baldosas

- Dado un table de  $3 \times n$ , encuentre el número de formas en que puede completar con dominós de tamaño  $2 \times 1$ .

**3 x 2**



**3 formas**

**3 x 8**



n	# formas
2	3
8	153
12	2131

# Amigos

Suponga que hay  $n$  amigos, quienes pueden permanecer solos o ser emparejados con otro amigo. Cada amigo puede ser emparejado una sola vez. Encuentre el número de formas en que los amigos pueden quedar solos o emparejados.

3 amigos

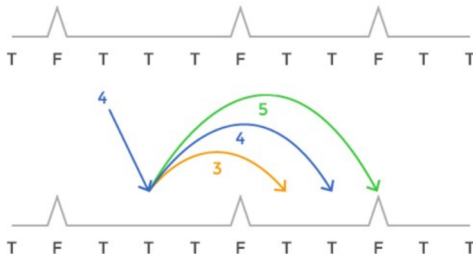
- [1, 2, 3]
- [1], [2, 3]
- [1, 2], [3]
- [1, 3], [2]

**4 formas**

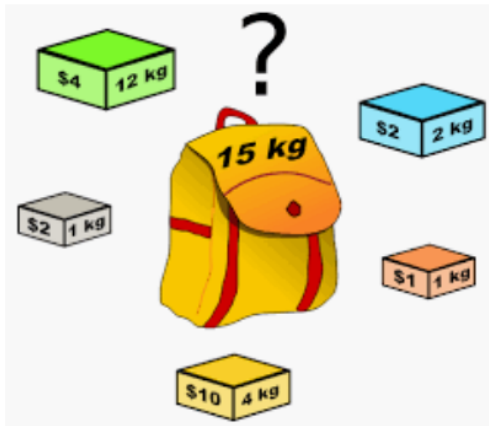


# Clavos

1. Considere un camino plano con clavos en el camino. Usted puede saltar de punto en punto según una velocidad  $S$ ; luego de llegar a un punto, puede ajustar su velocidad en 1 unidad antes de dar el siguiente salto. La idea es no pisar ningún clavo durante el camino, y se detiene una vez su velocidad llega a 0. Dado un camino, una posición y una velocidad inicial, indique si es posible detenerse a lo largo del camino sin pisar ningún clavo.

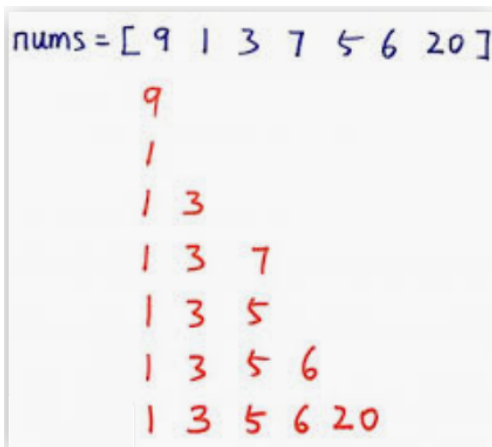


# Knapsack



Dada una mochila con una capacidad de 15 kg que puedo llenar con objetos de distinto peso y valor, ¿qué objetos elijo para maximizar mis ganancias y no exceder los 15 kg de peso permitidos?

# LIS



El problema de la subsecuencia creciente máxima consiste en encontrar una subsecuencia de una secuencia dada donde los elementos de la subsecuencia están ordenados, de menor a mayor, y la subsecuencia sea tan larga como sea posible. Esta subsecuencia no es necesariamente continua, o única.

*LIS: Longest Increasing Subsequence*