

## PilaCauta

10. Tenemos cajas numeradas de 1 a  $N$ , todas de iguales dimensiones. Queremos encontrar la máxima cantidad de cajas que pueden apilarse en una única pila cumpliendo que:

- sólo puede haber una caja apoyada directamente sobre otra;
- las cajas de la pila deben estar ordenadas crecientemente por número, de abajo para arriba;
- cada caja  $i$  tiene un peso  $w_i$  y un soporte  $s_i$ , y el peso total de las cajas que están arriba de otra no debe exceder el soporte de esa otra.

Si tenemos los pesos  $w = [19, 7, 5, 6, 1]$  y los soportes  $s = [15, 13, 7, 8, 2]$  (la caja 1 tiene peso 19 y soporte 15, la caja 2 tiene peso 7 y soporte 13, etc.), entonces la respuesta es 4. Por ejemplo, pueden apilarse de la forma 1-2-3-5 o 1-2-4-5 (donde la izquierda es más abajo), entre otras opciones.

- a) Pensar la idea de un algoritmo de *backtracking* (no hace falta escribirlo).
- b) Escribir una formulación recursiva que sea la base de un algoritmo de PD. Explicar su semántica e indicar cuáles serían los parámetros para resolver el problema.
- c) Diseñar un algoritmo de PD y dar su complejidad temporal y espacial auxiliar. Comparar cómo resultaría un enfoque *top-down* con uno *bottom-up*.

Tengo que recorrer mi lista  $w$  y  $s$  en orden creciente

Cuáles son sol. candidatas? Todo  $n \in \mathbb{N}$

Cuáles son sol. válidas? Aquella  $n \in \mathbb{N} / 1 \leq n \leq N$

Es un algoritmo de backtracking. Componería qué pasa si digo una caja y qué pasa si no.

Así hasta llegar a  $i=N+1$ , es decir, recorrer toda la lista

$$\begin{aligned}
 & \text{logaritmo } w, s \text{ con } \\
 & \text{PC}(i, b) \quad \left. \begin{array}{l} \text{O si } i = N+1, b \geq 0 \rightarrow \text{c.b., recorrer todas las cajas} \\ -\infty \text{ si } b < 0 \end{array} \right\} \rightarrow \text{un pila. No es válido y las cajas no se soportan} \\
 & \text{más } \left. \begin{array}{l} \text{PC}(i+1, \min(b-w[i], s[i])) + 1 \\ \text{PC}(i+1, b) \end{array} \right\} \text{ C.C.}
 \end{aligned}$$

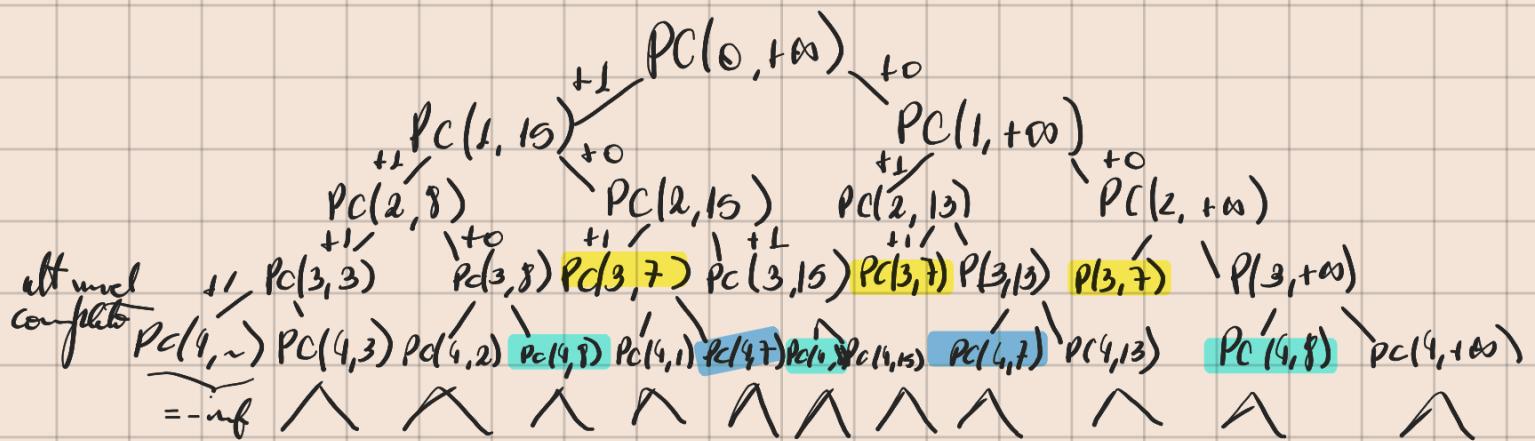
agregó una caja a la pila. lo que la

- la pila pueda soportar es a ser el mínimo entre lo que ya soporta la pila y el peso de la nueva caja y lo que soporta la nueva

Para resolver el problema debes llamar a la función con  $PC(0, +\infty)$

Si no agregas nada, puede reportar cualquier cosa

Hazan un árbol de BT con el ejemplo



Vemos que tenemos una superposición de subproblemas. Veamos cuando ocurre:

# llamadas recursivas = Regresa tres en árbol binario completo hasta que  $k < 0$   
 $= \Omega(2^k)$

# subinstancias = cantidad de valores posibles  $i \times$  los posibles valores de  $k$

Cómo estructura puedes tener una matriz donde cuando en  $M[i][k]$ , el problema es  $k = +\infty$   
 ↪ si recorres de atrás para adelante, quitará solución esto.

Cuando agrega una caja, la función que agregue tiene que ser capaz de aguantar el peso de la que ya estan, de la suma de pesos!

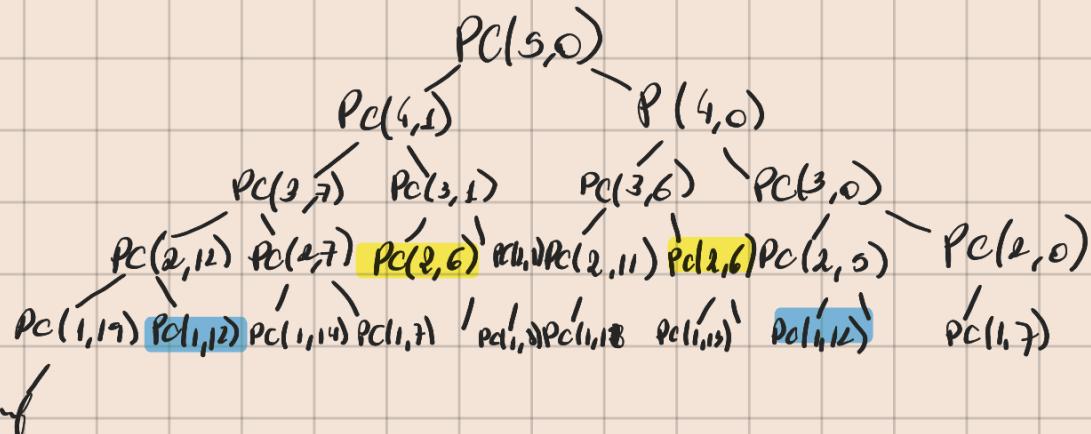
$$PC(i, k) = \begin{cases} 0 & \text{Si } i = -1 \\ \inf & \text{Si } k > S[i] \end{cases}$$

Si ya no puedes soportar peso no es una  
solución

$\max(PC(i-1, k-w[i]) + 1, PC(i-1, k))$

C.C  
Regresando la peso

Resuelve el problema  $PC(|w|-1, 0)$



Vemos que hay subproblemas

# llamada recursiva = es un árbol binario completo hasta que  $k > s[i]$   
=  $\mathcal{O}(2^n)$

# subestimación =  $O(n \cdot k)$   
 $\downarrow$   
     $k$  puede ser cero o la max suma de pesos  
    *i* puede tomar n valores

TD

Matrix  $n \times$  max suma de pesos de las cajas incluyendo en 1

if  $i = -1$

return 0

if  $k > s[i]$

return -inf

else

if  $M[i][k] = 1$

con =  $PC(i-1, k-w[i])$

sin =  $PC(i-1, k)$

$M[i][k] = \max \{ 1 + \text{con}, \text{sin} \}$

endif

return  $M[i][k]$

En el enfoque TD no nos tenemos que preocupar x como llenar la matriz y solo se llena lo necesario. En BV nos tenemos que ocupar de todo esto. Además como debemos llenar toda la matriz, para facilitar la tarea de determinar modificar la recursión para que en vez de devolvernos solamente  $PC(i-1, b)$  sea de modo que podamos llegar a todos los casos!