

SumaSubconjuntosBT

1. En este ejercicio vamos a resolver el problema de suma de subconjuntos con la técnica de *backtracking*. Dado un multiconjunto $C = \{c_1, \dots, c_n\}$ de números naturales y un natural k , queremos determinar si existe un subconjunto de C cuya sumatoria sea k . Vamos a suponer fuertemente que C está ordenado de alguna forma arbitraria pero conocida (i.e., C está implementado como la secuencia c_1, \dots, c_n o, análogamente, tenemos un iterador de C). Las *soluciones (candidatas)* son los vectores $a = (a_1, \dots, a_n)$ de valores binarios; el subconjunto de C representado por a contiene a c_i si y sólo si $a_i = 1$. Luego, a es una solución *válida* cuando $\sum_{i=1}^n a_i c_i = k$. Asimismo, una *solución parcial* es un vector $p = (a_1, \dots, a_i)$ de números binarios con $0 \leq i \leq n$. Si $i < n$, las soluciones *sucesoras* de p son $p \oplus 0$ y $p \oplus 1$, donde \oplus indica la concatenación.

- a) Escribir el conjunto de soluciones candidatas para $C = \{6, 12, 6\}$ y $k = 12$.

Soluciones candidatas = soluciones posibles

Se presume que el orden de C es creciente ya que secuencia

$$SC = \{(0,0,0), (1,0,0), (1,1,0), (1,1,1), (0,1,0), (0,1,1), (0,0,1), (1,0,1)\}$$

- b) Escribir el conjunto de soluciones válidas para $C = \{6, 12, 6\}$ y $k = 12$.

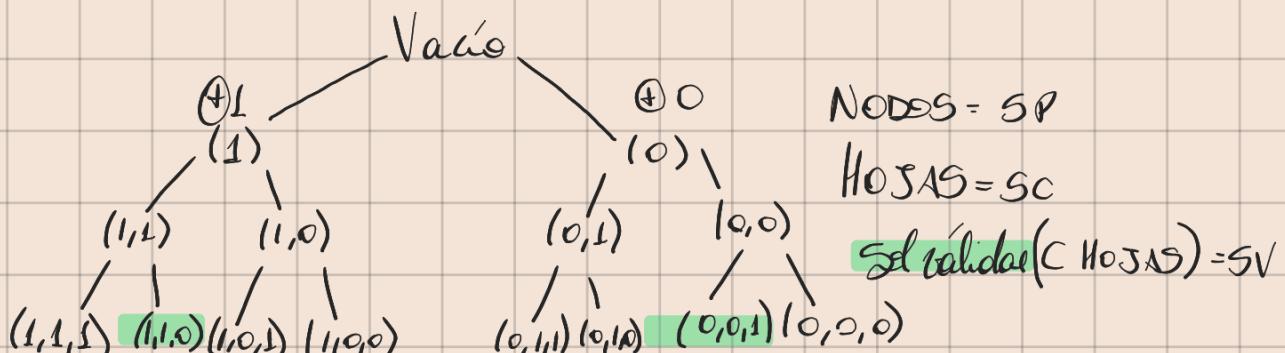
Soluciones válidas = soluciones que resuelven el problema

$$SV = \{(1,1,0), (0,0,1)\}$$

- c) Escribir el conjunto de soluciones parciales para $C = \{6, 12, 6\}$ y $k = 12$.

$$SP = \{(0), (1), (0,0), (0,1), (1,0), (1,1)\} \cup SC$$

- d) Dibujar el árbol de *backtracking* correspondiente al algoritmo descrito arriba para $C = \{6, 12, 6\}$ y $k = 12$, indicando claramente la relación entre las distintas componentes del árbol y los conjuntos de los incisos anteriores.



- e) Sea \mathcal{C} la familia de todos los multiconjuntos de números naturales. Considerar la siguiente función recursiva $ss: \mathcal{C} \times \mathbb{N} \rightarrow \{V, F\}$ (donde $\mathbb{N} = \{0, 1, 2, \dots\}$, V indica verdadero y F falso):

$$ss(\{c_1, \dots, c_n\}, k) = \begin{cases} k = 0 & \text{si } n = 0 \\ ss(\{c_1, \dots, c_{n-1}\}, k) \vee ss(\{c_1, \dots, c_{n-1}\}, k - c_n) & \text{si } n > 0 \end{cases}$$

de atrás para adelante

Convencerse de que $ss(C, k) = V$ si y sólo si el problema de subconjuntos tiene una solución válida para la entrada C, k . Para ello, observar que hay dos posibilidades para una solución válida $a = (a_1, \dots, a_n)$ para el caso $n > 0$: o bien $a_n = 0$ o bien $a_n = 1$. En el primer caso, existe un subconjunto de $\{c_1, \dots, c_{n-1}\}$ que suma k ; en el segundo, existe un subconjunto de $\{c_1, \dots, c_{n-1}\}$ que suma $k - c_n$.

Si repasaste la tabla de verdad de \vee tienes que
en $V \wedge$ una o ambas proposiciones son verdaderas

de forma de que al menos 1 sea V es que cuando $n=0, k=0$

Eso solo pasa si \exists un subconjunto $S \subseteq C / k - \sum_{i=0}^{|S|} S[i] = 0$, es decir,

\exists el menor una sol. válida. Si $\not\exists \rightarrow k/0$ se pone entonces

$\text{NO sera } V \text{ nunca.}$

q	v	p
v	v	f
v	v	v
f	v	v
f	f	f

- f) Convencerse de que la siguiente es una implementación recursiva de ss en un lenguaje imperativo y de que retorna la solución para C, k cuando se llama con $C, |C|, k$. ¿Cuál es su complejidad?

- 1) `subset_sum(C, i, j): // implementa $ss(\{c_1, \dots, c_i\}, j)$`
- 2) Si $i = 0$, retornar ($j = 0$) \rightarrow ~~caso base~~
- 3) Si no, retornar `subset_sum(C, i - 1, j) \vee subset_sum(C, i - 1, j - C[i])`

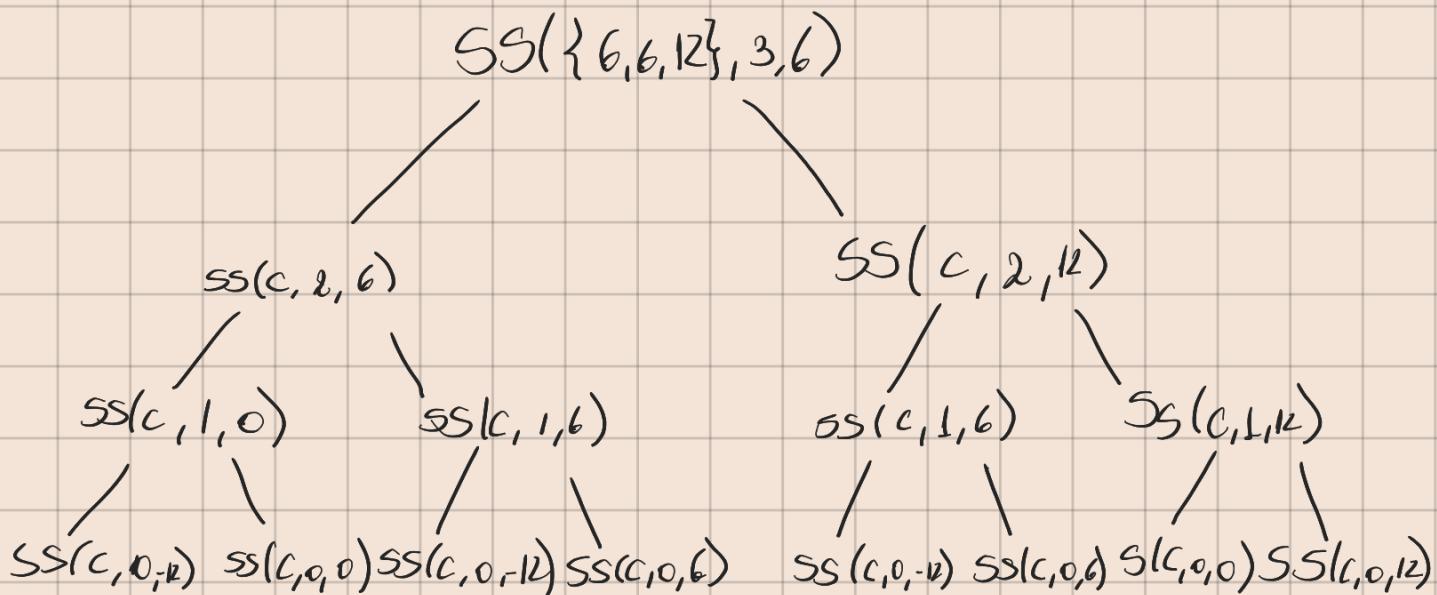
¿Cuál sería la altura del árbol?

¿Cuántos nodos tiene? N^n

¿Cuánto tiempo opera en un nodo? $O(1)$

Recorro todos los nodos y hago operaciones $O(1)$ \rightarrow complejidad = $O(N^n)$

- g) Dibujar el árbol de llamadas recursivas para la entrada $C = \{6, 12, 6\}$ y $k = 12$, y compararlo con el árbol de *backtracking*.



En el mismo

- h) Considerar la siguiente *regla de factibilidad*: $p = (a_1, \dots, a_i)$ se puede extender a una solución válida sólo si $\sum_{q=1}^i a_q c_q \leq k$. Convencerse de que la siguiente implementación incluye la regla de factibilidad.

- 1) `subset_sum(C, i, j): // implementa $ss(\{c_1, \dots, c_i\}, j)$`
- 2) Si $j < 0$, retornar **falso** // regla de factibilidad
- 3) Si $i = 0$, retornar ($j = 0$)
- 4) Si no, retornar `subset_sum(C, i - 1, j) ∨ subset_sum(C, i - 1, j - C[i])`

Chequear que la suma no se pase y visto que *teorema esencial*

- i) Definir otra regla de factibilidad, mostrando que la misma es correcta; no es necesario implementarla.

$p = (a_1, \dots, a_i)$ se puede extender a una solvible si $k - \sum_{j=i+1}^n c[j] \leq 0$

↳ si los elementos que quedan no alcanzan para que $\sum_{i=1}^n a_i c[i]$ sume k no tiene sentido ni por los caminos ya que llevan a situaciones no factibles

j) Modificar la implementación para imprimir el subconjunto de C que suma k , si existe.

Ayuda: mantenga un vector con la solución parcial p al que se le agregan y sacan los elementos en cada llamada recursiva; tenga en cuenta de no suponer que este vector se copia en cada llamada recursiva, porque cambia la complejidad.

$sol = \text{vector vacío}$

$\{6, 6, 12\} \quad 3, 12, \langle 0, 0, 0 \rangle$

$ss(C, i, j, sol) \leftarrow \text{por referencia}$

if $j < 0$ then

$\text{BorrarUltPos}(sol)$; return false

if $i = 0$ then

 if $j = 0$ then

 print $sol \wedge$ return true

else

 borrarUltPos(sol)

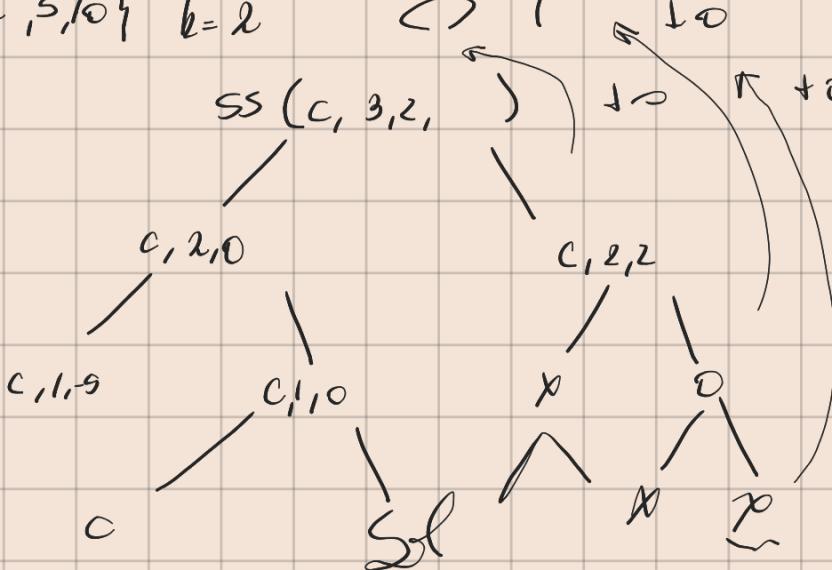
else

 return $ss(C, i-1, j, sol \oplus 0) \vee ss(C, i-1, j - C[i], sol \oplus 1)$

$$C = \{2, 5, 10\} \quad k = 2$$

\leftarrow

NO es de O(n!) O se le
agregó algo
NO



siempre recorre esta rama
durante la recursión \rightarrow se llega
más al final y $k > 0$ Borra esa ultima

Vamos a realizar una implementación que dado un vector lo vaya modificando de acuerdo a la siguiente función

$v = \langle \rangle$

$ss(c, i, j, bv)$

if $j < 0$ then
return false

endif

if $i = 0$ then
return $j = 0$

else

return $ss(c, i, j, bv) \vee ss(c, i-1, j - c[i], bv \oplus c[i])$