a) Sea $n = |C|$ la cantidad de elementos de C. Considerar la siguiente función recursiva $\text{ss'}_C\colon \{0,\dots,n\} \times \{0,\dots,k\} \to \{V,F\}$ (donde V indica verdadero y F falso) tal que:

$$\text{ss'}_C(i,j) = \begin{cases} j = 0 & \text{si } i = 0 \\ \text{ss'}_C(i-1,j) & \text{si } i \neq 0 \wedge C[i] > j \\ \text{ss'}_C(i-1,j) \vee \text{ss'}_C(i-1,j-C[i]) & \text{si no} \end{cases}$$

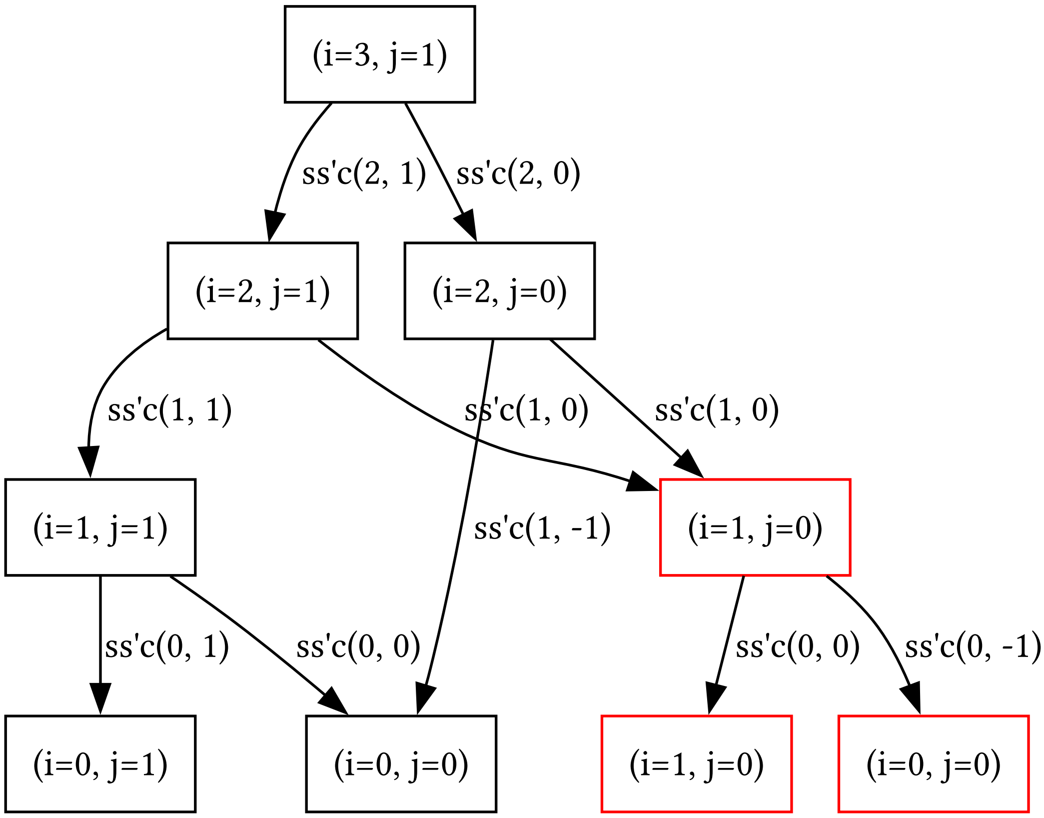
Convencerse de que esta es una definición equivalente de la función ss del inciso *e)* del Ejercicio 1, observando que $\text{ss}(C,k) = \text{ss'}_C(n,k)$. En otras palabras, convencerse de que el algoritmo del inciso *f)* es una implementación por *backtracking* de la función ss'_C. Concluir, pues, que $\mathcal{O}(2^n)$ llamadas recursivas de ss'_C son suficientes para resolver el problema.

✓

---

b) Observar que, como C no cambia entre llamadas recursivas, existen $\mathcal{O}(nk)$ posibles entradas para ss'_C. Concluir que, si $k \ll 2^n/n$, entonces necesariamente algunas instancias de ss'_C son calculadas muchas veces por el algoritmo del inciso *f)*. Mostrar un ejemplo donde se calcule varias veces la misma instancia.

Para $C = \{1,1,1\}, n = |C|, k = 1$



---

c) Considerar la estructura de memoización (i.e., el diccionario) M implementada como una matriz de $(n+1) \times (k+1)$ tal que $M[i,j]$ o bien tiene un valor indefinido $\perp$ o bien tiene el valor $\text{ss'}_C(i,j)$, para todo $0 \leq i \leq n$ y $0 \leq j \leq k$. Convencerse de que el siguiente algoritmo *top-down* mantiene un estado válido para M y computa $M[i,j] = \text{ss'}_C(i,j)$ cuando se invoca $\text{ss'}_C(i,j)$.

1) Inicializar $M[i,j] = \perp$ para todo $0 \leq i \leq n$ y $0 \leq j \leq k$.
2) subset_sum(C, i, j): // implementa $\text{ss}(\{c_1, \ldots, c_i\}, j) = \text{ss'}_C(i,j)$ usando memoización
3) Si $j < 0$, retornar **falso**
4) Si $i = 0$, retornar $(j = 0)$
5) Si $M[i,j] = \perp$:
6) Poner $M[i,j] = \text{subset\_sum}(C, i-1, j) \vee \text{subset\_sum}(C, i-1, j - C[i])$
7) Retornar $M[i,j]$

✓

---

d) Concluir que subset_sum(C, n, k) resuelve el problema. Calcular la complejidad y compararla con el algoritmo subset_sum del inciso *f*) del Ejercicio 1. ¿Cuál algoritmo es mejor cuando $k \ll 2^n$? ¿Y cuándo $k \gg 2^n$?

1.f. es $O(2^{|C|}) = O(2^n)$

Luego, tenemos una matriz de tamaño $n \cdot k$, entonces es de a lo sumo $O(n \cdot k)$

Si $k \ll 2^n \Rightarrow O(n \cdot k)$ es mejor ss′ b

1) subset_sum(C, k): // computa $M[i,j]$ para todo $0 \leq i \leq n$ y $0 \leq j \leq k$.
2) Inicializar $M[0,j] := (j = 0)$ para todo $0 \leq j \leq k$.
3) Para $i = 1, \ldots, n$ y para $j = 0, \ldots, k$:
4) Poner $M[i,j] := M[i-1,j] \vee (j - C[i] \geq 0 \wedge M[i-1, j - C[i]])$

✓

---

f) (Opcional) Modificar el algoritmo *bottom-up* anterior para mejorar su complejidad espacial a $O(k)$.

```
memo = [0...k] en False
memo[0] = True

para c in C
  para j in k...c
    si memo[j-c] = True: memo[j] = True

ret memo[k]
```

---

g) (Opcional) Demostrar que la función recursiva del inciso *a)* es correcta. **Ayuda:** demostrar por inducción en i que existe algún subconjunto de $\{c_1, \dots, c_i\}$ que suma j si y solo si ss'$_C(i, j) = V$.

Inducción en i

Sea $i \in \{0...n\}, j \in \{0...k\}$

$P(i) : \exists s \subseteq \{c_1, ..., c_i\} \mid \sum s = j \iff \text{ss'}_C(i, j) = V$

Caso base:

$P(0) : \exists s \subseteq \{c_1, ..., c_0\} \mid \sum s = j \iff \text{ss'}_C(0, j) = V$

$(\Longrightarrow)$

Hipótesis: $\exists s \subseteq \{c_1, ..., c_0\} \mid \sum s = j \Longrightarrow s = \emptyset \land j = 0$

Luego, $\text{ss'}_C(0, j) = (j = 0) \underset{\text{Hipotesis}}{=} V$ ✓

$(\Longleftarrow)$

Hipótesis: $\text{ss'}_C(0, j) = V \Longrightarrow j = 0$

Luego, $\exists s \subseteq \{c_1, ..., c_0\} \mid \sum s = j \Longrightarrow s = \emptyset \mid \sum s = \sum \emptyset = j = 0$ ✓

Paso inductivo:

HI: $P(i-1) : \exists s \subseteq \{c_1, ..., c_{i-1}\} \mid \sum s = j \iff \text{ss'}_C(i-1, j) = V$

Queremos probar que $\forall j :: P(i-1) \Rightarrow P(i)$

Caso $i \neq 0 \land C[i] > j$

$\text{ss'}_C(i, j) = \text{ss'}_C(i-1, j) \underset{\text{HI}}{=} V$ ✓

$\exists s \subseteq \{c_1, ..., c_{i-1}\} \mid \sum s = j \land C[i] > j \Longrightarrow$

$\exists s \subseteq \{c_1, ..., c_i\} \mid \sum s = j, C[i] \notin s \Longrightarrow$

$\exists s \subseteq \{c_1, ..., c_i\} \mid \sum s = j$ ✓

Caso $C[i] \leq j$

Definición: $\text{ss'}_C(i, j) = \text{ss'}_C(i-1, j) \lor \text{ss'}_C(i-1, j - C[i])$

Si $\text{ss'}_C(i, j) = \text{ss'}_C(i-1, j) = V$:

Ya lo probamos antes ✓

Si $\text{ss'}_C(i, j) = \text{ss'}_C(i-1, j - C[i]) = V$:

$C[i] \leq j \land_{\text{HI}} \exists s \subseteq \{c_1, ..., c_{i-1}\} \mid \sum s = j \Longrightarrow 0 \leq j - C[i] \leq k$

Alcanza con probar que

$\exists s' \subseteq \{c_1, ..., c_{i-1}\} \mid \sum s' = j - C[i] \Longrightarrow \exists s = s' \cup \{c_i\} \subseteq \{c_1, ..., c_i\} \mid \sum s = j$

Vemos que trivialmente $s' \subseteq \{c_1, ..., c_{i-1}\} \Rightarrow s = s' \cup \{c_i\} \subseteq \{c_1, ..., c_i\}$

Y $\sum s' = j - C[i] \Rightarrow \sum s' \cup \{c_i\} = j - C[i] + C[i] = j$ ✓

□