6. Tenemos un multiconjunto B de valores de billetes y queremos comprar un producto de costo c de una máquina que no da vuelto. Para poder adquirir el producto debemos cubrir su costo usando un subconjunto de nuestros billetes. El objetivo es pagar con el mínimo exceso posible a fin de minimizar nuestra pérdida. Más aún, queremos gastar el menor tiempo posible poniendo billetes en la máquina. Por lo tanto, entre las opciones de mínimo exceso posible, queremos una con la menor cantidad de billetes. Por ejemplo, si $c = 14$ y $B = \{2, 3, 5, 10, 20, 20\}$, la solución es pagar 15, con exceso 1, insertando sólo dos billetes: uno de 10 y otro de 5.

a) Considerar la siguiente estrategia por *backtracking* para el problema, donde $B = \{b_1, \ldots, b_n\}$. Tenemos dos posibilidades: o agregamos el billete b_n, gastando un billete y quedando por pagar $c - b_n$, o no agregamos el billete b_n, gastando 0 billetes y quedando por pagar c. Escribir una función recursiva $\text{cc}(B, c)$ para resolver el problema, donde $\text{cc}(B, c) = (c', q)$ cuando el mínimo costo mayor o igual a c que es posible pagar con los billetes de B es c' y la cantidad de billetes mínima es q.

Defino $\prec$ como el orden c', q' tal que $\min_{\prec}$ implica ver primero el mínimo valor c' y luego, el mínimo q' para "desempatar"

$$\text{cc}(B, c) = \begin{cases} (0, 0) \text{ si } c \leq 0 \\ (\infty, \infty) \text{ si } B = \emptyset \wedge c > 0 \\ \min_{\prec}\{(\max(0, c' + b_n - c), q' + 1), \text{cc}(B - \{b_n\}, c)\} \text{ si } c > 0 \wedge B \neq \emptyset \end{cases}$$

$$(c', q') = \text{cc}(B - \{b_n\}, c - b_n)$$

---

b) Implementar la función de *a*) en un lenguaje de programación imperativo utilizando una función recursiva con parámetros B, i, j que compute $\text{cc}(\{b_1, \ldots, b_i\}, j)$. ¿Cuál es la complejidad del algoritmo?

TO DO

---

c) Reescribir cc como una función recursiva $\text{cc'}_B(i, j) = \text{cc}(\{b_1, \ldots, b_i\}, j)$ que implemente la idea anterior **dejando fijo el parámetro** B. A partir de esta función, determinar cuándo cc'_B tiene la propiedad de *superposición de subproblemas*.

$B = \{b_1 ... b_n\}$

$$\text{cc}'_B(i, j) = \begin{cases} (0, 0) \text{ si } j \leq 0 \\ (\infty, \infty) \text{ si } i = 0 \wedge j > 0 \\ \min_{\prec}\{(\max(0, c'' + b_i - j), q'' + 1), \text{cc}'_B(i - 1, j)\} \text{ si } i > 0 \wedge j > 0 \end{cases}$$

$$(c'', q'') = \text{cc}'_B(i - 1, j - b_i)$$

---

d) Definir una estructura de memoización para cc'_B que permita acceder a $\text{cc'}_B(i, j)$ en $\mathcal{O}(1)$ tiempo para todo $0 \leq i \leq n$ y $0 \leq j \leq k$.

Definimos una matriz de tamaño $(n + 1) \cdot (c + 1)$ tal que cada posible combinación de cantidad de billetes $(0...n)$ y costos $(0...c)$ sea accesible en $O(1)$

La complejidad de esto es $O(n \cdot c)$

---

e) Adaptar el algoritmo de *b*) para incluir la estructura de memoización.

TO DO

---

f) Indicar cuál es la llamada recursiva que resuelve nuestro problema y cuál es la complejidad del nuevo algoritmo.

$\text{cc}'_B(n, c)$ resuelve el problema.