4. Para cada una de las soluciones que propuso a los ejercicios del 3 al 9 del práctico de backtracking, dar una definición alternativa que utilice la técnica de programación dinámica. En los casos de los ejercicios 3, 5 y 7 modificar luego el algoritmo para que no sólo calcule el valor óptimo sino que devuelva la solución que tiene dicho valor (por ejemplo, en el caso del ejercicio 3, cuáles serían los pedidos que debería atenderse para alcanzar el máximo valor).

4. Usted se encuentra en un globo aerostático sobrevolando el océano cuando descubre que empieza a perder altura porque la lona está levemente dañada. Tiene consigo n objetos cuyos pesos $p_1, \ldots, p_n$ y valores $v_1, \ldots, v_n$ conoce. Si se desprende de al menos P kilogramos logrará recuperar altura y llegar a tierra firme, y afortunadamente la suma de los pesos de los objetos supera holgadamente P. ¿Cuál es el menor valor total de los objetos que necesita arrojar para llegar sano y salvo a la costa?

La función recursiva obtenida con backtracking es:

```
globo(c,pm) = ( si pm = 0         ⟶ 0
              | si pm > 0 ∧ c = 0 ⟶ ∞
              | si pm > 0 ∧ c > 0 ⟶ globo(c-1,pm) `min` vc + globo(c-1,pm-pc)
              )
```

donde *c* es el conjunto de objetos a bordo del globo y *pm* el peso mínimo necesario a liberar para mantener el globo a flote.

Siendo su definición en programación dinámica la siguiente:

```
fun globo(v: array[1..n] of nat, p: array[1..n] of nat, P: nat) ret solucion: nat
    var tabla: array[0..n,0..P] of nat   {- tabla[i,j] = globo(i,j) -}

    {- Caso 1 -}
    for i := 0 to n do
        tabla[i,0] := 0
    od

    {- Caso 2 -}
    for j := 1 to P do
        tabla[0,j] := ∞
    od

    {- Caso 3 -}
    for i := 1 to n do
        for j := 1 to P do
            tabla[i,j] := tabla[i-1,j] `min` v[i] + tabla[i-1,j-p[i]]
        od
    od

    solucion := tabla[n,P]
end fun
```

¿Qué forma tiene la tabla? Es un arreglo de ***dos dimensiones***: *[0..n,0..P]*.
¿En qué orden se llena la tabla? Para llenar cada celda debo tener en cuenta:

`tabla[i-1,j-p[i]]`

- **i**: necesito ver la fila anterior (`i-1`) ⇒ ***n* ⟶ *0***
- **j**: necesito ver la columna anterior ⇒ ***P* ⟶ *0***