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).

7. En el problema de la mochila se buscaba el máximo valor alcanzable al seleccionar entre n objetos de valores $v_1, \ldots, v_n$ y pesos $w_1, \ldots, w_n$, respectivamente, una combinación de ellos que quepa en una mochila de capacidad W. Si se tienen dos mochilas con capacidades W_1 y W_2, ¿cuál es el valor máximo alcanzable al seleccionar objetos para cargar en ambas mochilas?

La función recursiva obtenida con backtracking es:

```
2mochilas(c,m1,m2) = ( si c = 0                        → 0
                     | si m1 = 0 ∧ m2 = 0              → 0
                     | si c > 0 ∧ w_c > m1 ∧ w_c > m2  → 2mochilas(c-1,m1,m2)
                     | si c > 0 ∧ w_c > m1 ∧ w_c ≤ m2  → v_c + 2mochilas(c-1,m1,m2-w_c)
                                                        `max` 2mochilas(c-1,m1,m2)
                     | si c > 0 ∧ w_c ≤ m1 ∧ w_c > m2  → v_c + 2mochilas(c-1,m1-w_c,m2)
                                                        `max` 2mochilas(c-1,m1,m2)
                     | si c > 0 ∧ w_c ≤ m1 ∧ w_c ≤ m2  → v_c + 2mochilas(c-1,m1-w_c,m2)
                                                        `max` v_c + 2mochilas(c-1,m1,m2-w_c)
                                                        `max` 2mochilas(c-1,m1,m2)
                     )
```

donde *c* es el conjunto de objetos a colocar en la mochila, *m1* es la capacidad disponible de la mochila 1 y *m2* es la capacidad disponible de la mochila 2.

Siendo su definición en programación dinámica la siguiente:

```
fun 2mochilas(v: array[0..n] of nat, w: array[0..n] of nat, W1: nat,
              W2: nat) ret solucion: nat

    var tabla: array[0..n,0..W1,0..W2] of nat    {- tabla[i,j,k] = 2mochilas(i,j,k) -}

    {- Caso 1 -}
    for j := 0 to W1 do
        for k := 0 to W2 do
            tabla[0,j,k] := 0
        od
    od

    {- Caso 2 -}
    for i := 0 to n do
        tabla[i,0,0] := 0
    od

    for i := 1 to n do
        for j := 1 to W1 do
            for k := 1 to W2 do
                if w[i] > j ∧ w[i] > k then     {- Caso 3 -}
                    tabla[i,j,k] := tabla[i-1,j,k]
                else if w[i] > j ∧ w[i] ≤ k then     {- Caso 4 -}
                    tabla[i,j,k] := tabla[i-1,j,k] `max` v[i] + tabla[i-1,j,k-w[i]]
                else if w[i] ≤ j ∧ w[i] > k then     {- Caso 5 -}
                    tabla[i,j,k] := tabla[i-1,j,k] `max` v[i] + tabla[i-1,j-w[i],k]
                else if w[i] ≤ j ∧ w[i] ≤ k then     {- Caso 6 -}
                    tabla[i,j,k] := tabla[i-1,j,k] `max` v[i] + tabla[i-1,j-w[i],k]
                                   `max` v[i] + tabla[i-1,j,k-w[i]]
```

```
                fi
            od
        od
    od

    solucion := tabla[n,W1,W2]
end fun
```

¿Qué forma tiene la tabla? Es un arreglo de ***tres dimensiones***: *[0..n,0..W1,0..W2]*.
¿En qué orden se llena la tabla? Para llenar cada celda debo tener en cuenta:
`tabla[i-1,j-w[i],k]` y `tabla[i-1,j,k-w[i]]`

- **i**: necesito ver el anterior (`i-1`) ⇒ ***n*** ⟶ ***0***
- **j**: necesito ver el anterior (`j-w[i]`) ⇒ ***W1*** ⟶ ***0***
- **k**: necesito ver el anterior (`k-w[i]`) ⇒ ***W2*** ⟶ ***0***

Otra versión que no solo calcula el valor máximo de los objetos guardados en la mochilas sino también cuáles son las cosas a guardar para obtener dicho valor es:

```
type Objetos = tuple
                 id: nat
                 valor: nat
                 peso: nat
               end tuple

fun 2mochilas(v: array[0..n] of nat, w: array[0..n] of nat, W1: nat,
              W2: nat) ret res: List of Objetos

    var tabla: array[0..n,0..W1,0..W2] of nat   {- tabla[i,j,k] = 2mochilas(i,j,k) -}
    var solucion: array[0..n,0..W1,0..W2] of (List of Objetos)
    var ganancia: nat
    var maximo4: nat
    var maximo5: nat
    var maximo6: nat

    {- Caso 1 -}
    for j := 0 to W1 do
        for k := 0 to W2 do
            tabla[0,j,k] := 0
            solucion[0,j,k] := empty_list()
        od
    od

    {- Caso 2 -}
    for i := 0 to n do
        tabla[i,0,0] := 0
        solucion[i,0,0] := empty_list()
    od

    for i := 1 to n do
        for j := 1 to W1 do
            for k := 1 to W2 do
                if w[i] > j ∧ w[i] > k then    {- Caso 3 -}
                    tabla[i,j,k] := tabla[i-1,j,k]
                    solucion[i,j,k] := copy_list(solucion[i-1,j,k])
                else if w[i] > j ∧ w[i] ≤ k then    {- Caso 4 -}
                    maximo4 := tabla[i-1,j,k] `max` v[i] + tabla[i-1,j,k-w[i]]
                    tabla[i,j,k] := maximo4
```

```
                    if maximo4 = tabla[i-1,j,k] then
                        solucion[i,j,k] := copy_list(solucion[i-1,j,k])
                    else if maximo4 = v[i] + tabla[i-1,j,k-w[i]] then
                        solucion[i,j,k] := copy_list(solucion[i-1,j,k])
                        addr(solucion[i,j,k] , (i,v[i],w[i]))
                    fi
                else if w[i] ≤ j ∧ w[i] > k then    {- Caso 5 -}
                    maximo5 := tabla[i-1,j,k] `max` v[i] + tabla[i-1,j-w[i],k]
                    tabla[i,j,k] := maximo5

                    if maximo5 = tabla[i-1,j,k] then
                        solucion[i,j,k] := copy_list(solucion[i-1,j,k])
                    else if maximo5 = v[i] + tabla[i-1,j,k-w[i]] then
                        solucion[i,j,k] := copy_list(solucion[i-1,j,k])
                        addr(solucion[i,j,k] , (i,v[i],w[i]))
                    fi
                else if w[i] ≤ j ∧ w[i] ≤ k then    {- Caso 6 -}
                    maximo6 := tabla[i-1,j,k] `max` v[i] + tabla[i-1,j-w[i],k]
                               `max` v[i] + tabla[i-1,j,k-w[i]]
                    tabla[i,j,k] := maximo6

                    if maximo6 = tabla[i-1,j,k] then
                        solucion[i,j,k] := copy_list(solucion[i-1,j,k])
                    else if (maximo6 = v[i] + tabla[i-1,j-w[i],k]) v
                              (maximo6 = v[i] + tabla[i-1,j,k-w[i]]) then
                        solucion[i,j,k] := copy_list(solucion[i-1,j,k])
                        addr(solucion[i,j,k] , (i,v[i],w[i]))
                    fi
                fi
            od
        od
    od

    ganancia := tabla[n,W1,W2]
    res := solucion[n,W1,W2]
end fun
```