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).

6. Un artesano utiliza materia prima de dos tipos: $A$ y $B$. Dispone de una cantidad $MA$ y $MB$ de cada una de ellas. Tiene a su vez pedidos de fabricar $n$ productos $p_1, \ldots, p_n$ (uno de cada uno). Cada uno de ellos tiene un valor de venta $v_1, \ldots, v_n$ y requiere para su elaboración cantidades $a_1, \ldots, a_n$ de materia prima de tipo $A$ y $b_1, \ldots, b_n$ de materia prima de tipo $B$. ¿Cuál es el mayor valor alcanzable con las cantidades de materia prima disponible?

La función recursiva obtenida con backtracking es:

```
artesania(p,ma,mb) = ( si p = 0                          → 0
                     | si p > 0 ∧ (a_p > ma ∨ b_p > mb)  → artesania(p-1,ma,mb)
                     | si p > 0 ∧ a_p ≤ mb ∧ b_p ≤ mb    → artesania(p-1,ma,mb) `max`
                                                           v_p + artesania(p-1,ma-a_p,mb-b_p)
                     )
```

donde *p* es el conjunto de pedidos que se hicieron al artesano, *ma* es la cantidad de materia prima de tipo A disponible y *mb* la cantidad de materia prima de tipo B disponible.

Siendo su definición en programación dinámica la siguiente:

```
fun artesania(v: array[0..n] of nat, a: array[0..n] of nat, b: array[0..n] of nat,
              MA: nat, MB: nat) ret solucion: nat

    var tabla: array[0..n,0..MA,0..MB] of nat   {- tabla[i,j,k] = artesania(i,j,k) -}

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

    {- Caso 2 -}
    for i := 1 to n do
        for j := 0 to MA do
            for k := 0 to MB do
                if a[i] > j v b[i] > k then
                    tabla[i,j,k] := tabla[i-1,j,k]
                else   {- Caso 3, pues ¬(a[i]>j v b[i]>k) ≡ a[i]≤j ∧ b[i]≤k -}
                    tabla[i,j,k] := tabla[i-1,j,k] `max` v[i] + tabla[i-1,j-a[i],k-b[i]]
                fi
            od
        od
    od

    solucion := tabla[n,MA,MB]
end fun
```

---

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

`tabla[i-1,j-a[i],k-b[i]]`

- **i**: necesito ver el anterior (`i-1`) ⇒ ***n*** ⟶ ***0***
- **j**: necesito ver el anterior (`j-a[i]`) ⇒ ***MA*** ⟶ ***0***
- **k**: necesito ver el anterior (`k-b[i]`) ⇒ ***MB*** ⟶ ***0***