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).

8. Una fábrica de automóviles tiene dos líneas de ensamblaje y cada línea tiene n estaciones de trabajo, $S_{1,1}, \ldots, S_{1,n}$ para la primera y $S_{2,1}, \ldots, S_{2,n}$ para la segunda. Dos estaciones $S_{1,i}$ y $S_{2,i}$ (para $i = 1, \ldots, n$), hacen el mismo trabajo, pero lo hacen con costos $a_{1,i}$ y $a_{2,i}$ respectivamente, que pueden ser diferentes. Para fabricar un auto debemos pasar por n estaciones de trabajo $S_{i_1,1}, S_{i_2,2} \ldots, S_{i_n,n}$ no necesariamente todas de la misma línea de montaje ($i_k = 1, 2$). Si el automóvil está en la estación $S_{i,j}$, transferirlo a la otra línea de montaje (es decir continuar en $S_{i',j+1}$ con $i' \neq i$) cuesta $t_{i,j}$. Encontrar el costo mínimo de fabricar un automóvil usando ambas líneas.

La función recursiva obtenida con backtracking es:

```
ensamblado(m,e) = ( si e = n                  → a_{m,e}
                  | si m = 1 ∧ 1 ≤ e < n → a_{1,e} + ensamblado(1,e+1)
                                            `min` a_{1,e} + t_{1,e} + ensamblado(2,e+1)
                  | si m = 2 ∧ 1 ≤ e < n → a_{2,e} + ensamblado(2,e+1)
                                            `min` a_{2,e} + t_{2,e} + ensamblado(1,e+1)
                  )
```

donde *m* es la línea de montaje que está trabajando y *e* es la estación en la que se encuentra el vehículo actualmente.

Siendo su definición en programación dinámica la siguiente:

```
fun ensamblado(a: array[1..2,1..n] of nat, t: array[1..2,0..n] of nat) ret solucion: nat
    var tabla: array[1..2,1..n] of nat   {- tabla[i,j] = ensamblado(i,j) -}

    {- Caso 1 -}
    for i := 1 to 2 do
        tabla[i,n] := a[i,n]
    od

    {- Caso 2 -}
    for j := 1 to n-1 do
        tabla[1,j] := a[1,j] + tabla[1,j+1] `min` a[1,j] + t[1,j] + tabla[2,j+1]
    od

    {- Caso 3 -}
    for j := 1 to n-1 do
        tabla[2,j] := a[2,j] + tabla[2,j+1] `min` a[2,j] + t[2,j] + tabla[1,j+1]
    od

    solucion := tabla[1,1] `min` tabla[2,1]
end fun
```

¿Qué forma tiene la tabla? Es un arreglo de ***dos dimensiones***: *[1..2,1..n]*.
¿En qué orden se llena la tabla? Para llenar cada celda debo tener en cuenta:

`tabla[1,j+1]` y `tabla[2,j+1]`

- **i**: da igual el orden en que se llene, pues indica la línea de montaje que varía entre 1 ó 2.
- **j**: necesito ver la siguiente columna (`j+1`) ⇒ ***1*** ⟶ ***n***