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).

9. El juego ↖U↑P↗ consiste en mover una ficha en un tablero de n filas por n columnas desde la fila inferior a la superior. La ficha se ubica al azar en una de las casillas de la fila inferior y en cada movimiento se desplaza a casillas adyacentes que estén en la fila superior a la actual, es decir, la ficha puede moverse a:

   - la casilla que está inmediatamente arriba,
   - la casilla que está arriba y a la izquierda (si la ficha no está en la columna extrema izquierda),
   - la casilla que está arriba y a la derecha (si la ficha no está en la columna extrema derecha).

   Cada casilla tiene asociado un número entero c_{ij} $(i, j = 1, \ldots, n)$ que indica el puntaje a asignar cuando la ficha esté en la casilla. El puntaje final se obtiene sumando el puntaje de todas las casillas recorridas por la ficha, incluyendo las de las filas superior e inferior.

   Determinar el máximo y el mínimo puntaje que se puede obtener en el juego.

Las funciones recursivas obtenidas con backtracking son:

```
maxUP(f,c) = ( si f = 1                  → c_{1,c}
             | si 1 < f ≤ n ∧ c = 1      → c_{f,1} + maxUP(f-1,c) `max` c_{f,1} + maxUP(f-1,c+1)
             | si 1 < f ≤ n ∧ c = n      → c_{f,n} + maxUP(f-1,c-1) `max` c_{f,n} + maxUP(f-1,c)
             | si 1 < f ≤ n ∧ 1 < c < n → c_{f,c} + maxUP(f-1,c-1) `max` c_{f,c} + maxUP(f-1,c)
                                            `max` c_{f,c} + maxUP(f-1,c+1)
             )

minUP(f,c) = ( si f = 1                  → c_{1,c}
             | si 1 < f ≤ n ∧ c = 1      → c_{f,1} + minUP(f-1,c) `min` c_{f,1} + minUP(f-1,c+1)
             | si 1 < f ≤ n ∧ c = n      → c_{f,n} + minUP(f-1,c-1) `min` c_{f,n} + minUP(f-1,c)
             | si 1 < f ≤ n ∧ 1 < c < n → c_{f,c} + minUP(f-1,c-1) `min` c_{f,c} + minUP(f-1,c)
                                            `min` c_{f,c} + minUP(f-1,c+1)
             )
```

donde *f* corresponde a las filas del tablero y *c* a las columnas del mismo.

Siendo su definición en programación dinámica la siguiente:

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

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

    for i := 2 to n do
        for j := 1 to n do
            if j = 1 then   {- Caso 2 -}
                tabla[i,j] := p[i,1] + tabla[i-1,j] `max` p[i,1] + tabla[i-1,j+1]
            else if j = n then   {- Caso 3 -}
                tabla[i,j] := p[i,n] + tabla[i-1,j-1] `max` p[i,n] + tabla[i-1,j]
            else if 1 < j < n then   {- Caso 4 -}
                tabla[i,j] := p[i,j] + tabla[i-1,j-1] `max` p[i,j] + tabla[i-1,j]
                              `max` p[i,j] + tabla[i-1,j+1]
```

```
            fi
        od
    od

    maximo_puntaje := 0

    for j := 1 to n do
        maximo_puntaje := maximo_puntaje `max` maxUP(n,j)
    od

    solucion := maximo_puntaje
end fun
```

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

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

    for i := 2 to n do
        for j := 1 to n do
            if j = 1 then   {- Caso 2 -}
                tabla[i,j] := p[i,1] + tabla[i-1,j] `min` p[i,1] + tabla[i-1,j+1]
            else if j = n then   {- Caso 3 -}
                tabla[i,j] := p[i,n] + tabla[i-1,j-1] `min` p[i,n] + tabla[i-1,j]
            else if 1 < j < n then   {- Caso 4 -}
                tabla[i,j] := p[i,j] + tabla[i-1,j-1] `min` p[i,j] + tabla[i-1,j]
                              `min` p[i,j] + tabla[i-1,j+1]
            fi
        od
    od

    minimo_puntaje := ∞

    for j := 1 to n do
        minimo_puntaje := minimo_puntaje `min` minUP(n,j)
    od

    solucion := minimo_puntaje
end fun
```

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

`for j := 1 to n do`
`  maximo_puntaje := maximo_puntaje max`
`  maxUP(n,j)`
`od`
	`for j := 1 to n do`
	`  minimo_puntaje := minimo_puntaje min`
	`  minUP(n,j)`
	`od`

- **i**: el juego necesariamente empieza de la última fila y va subiendo como su nombre lo indica ⇒ ***n*** ⟶ ***1***
- **j**: el ciclo analiza las posibilidades partiendo de la primera columna hasta la última ⇒ ***1*** ⟶ ***n***