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).

5. Sus amigos quedaron encantados con el teléfono satelital, para las próximas vacaciones ofrecen pagarle un alquiler por él. Además del día de partida y de regreso (p_i y r_i) cada amigo ofrece un monto m_i por día. Determinar el máximo valor alcanzable alquilando el teléfono.

La función recursiva obtenida con backtracking es:

```
telefono(a,d) = ( si a = 0           → 0
                | si a > 0 ∧ d > p_a  → telefono(a-1,d)
                | si a > 0 ∧ d ≤ p_a  → telefono(a-1,d) `max`
                                        m_a * (r_a-p_a+1) + telefono(a-1,r_a+1)
                )
```

donde *a* es el conjunto de amigos que viajan y *d* el día actual en el que se encuentra.

Siendo su definición en programación dinámica la siguiente:

```
fun telefono(m: array[1..n] of nat, p: array[1..n] of nat, r: array[1..n] of nat,
             D: nat) ret solucion: nat

    var tabla: array[0..n,0..D] of nat   {- tabla[i,j] = telefono(i,j) -}

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

    {- Caso 2 -}
    for i := 1 to n do
        for j := 0 to D do
            if j > p[i] then
                tabla[i,j] := tabla[i-1,j]
            else    {- Caso 3 -}
                tabla[i,j] := tabla[i-1,j] `max` m[i] * (r[i]-p[i]+1) + tabla[i-1,r[i]+1]
            fi
        od
    od

    solucion := tabla[n,0]
end fun
```

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

`tabla[i-1,r[i]+1]`

- **i**: necesito ver el anterior (`i-1`) ⇒ ***n*** ⟶ ***0***
- **j**: corresponde al día actual, el cual arranca desde 0 ⇒ ***0*** ⟶ ***D***

Otra versión que no solo calcula el valor óptimo sino también cuáles son los amigos en particular a los que conviene alquilar el teléfono es:

```
type Amigos = tuple
                id: nat
                partida: nat
                regreso: nat
                monto: nat
              end tuple

fun telefono(m: array[1..n] of nat, p: array[1..n] of nat, r: array[1..n] of nat,
             D: nat) ret res: List of Amigos

    var tabla: array[0..n,0..D] of nat   {- tabla[i,j] = telefono(i,j) -}
    var solucion: array[0..n,0..D] of (List of Amigos)
    var ganancia: nat
    var maximo: nat

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

    {- Caso 2 -}
    for i := 1 to n do
        for j := 0 to D do
            if j > p[i] then
                tabla[i,j] := tabla[i-1,j]
                solucion[i,j] := copy_list(solucion[i-1,j])
            else    {- Caso 3 -}
                maximo := tabla[i-1,j] `max` m[i] * (r[i]-p[i]+1) + tabla[i-1,r[i]+1]
                tabla[i-1,j] := maximo
                if maximo = tabla[i-1,j] then
                    solucion[i,j] := copy_list(solucion[i-1,j])
                else if maximo = m[i] * (r[i]-p[i]+1) + tabla[i-1,r[i]+1] then
                    solucion[i,j] := copy_list(solucion[i-1,j])
                    addr(solucion[i,j] , (i,p[i],r[i],m[i]))
                fi
            fi
        od
    od

    ganancia := tabla[n,0]
    res := solucion[n,0]
end fun
```