Ej1)

Modifique el código del algoritmo que resuelve el problema de la moneda utilizando backtracking, de manera que devuelva qué monedas se utilizan, en vez de sólo la cantidad.

```
type Money : tuple
               cantidad : Nat
               monedas : List of Nat
             end tuple


fun cambio(j : Nat, C: Set of Nat) ret S : Money
      var c, Nat
      var C_aux : Set of Nat
      var aux1, aux2 : Money
      if j = 0 then
            S.cantidad := 0
            S.monedas := empty_list()
      else if is_empty(C) then
            S.cantidad := ∞
            S.monedas := empty_list()
      else
                  C_aux := set_copy(C)
                  c := get(C)
                  elim(C_aux,c)
            if (c ≤ j) then
                  aux1 := cambio(j-c,C)
                  aux2 := cambio(j,C_aux)
                  if(aux1.cantidad + 1 < aux2.cantidad) then
                        addr(S.monedas,c)
                        concatt(S.monedas,aux1.monedas)
                        S.cantidad := aux1.cantidad + 1
                  else
                        S := aux2
                  fi
            else
                  S := cambio(j, C_aux)
            fi
            set_destroy(C_aux)
      fi
end fun
```

Ej2)

En un extraño país las denominaciones de la moneda son 15, 23 y 29, un turista quiere comprar un recuerdo pero también quiere conservar el mayor número de monedas posibles. Los recuerdos cuestan 68, 74, 75, 83, 88 y 89. Asumiendo que tiene suficientes monedas para comprar cualquiera de ellos, ¿cuál de ellos elegirá? ¿Qué monedas utilizará para pagarlo?
Justificar claramente y mencionar el método utilizado.

Seleccionamos la menor cantidad de monedas para pagar el regalo más barato, si quiere guardar monedas que no ande careteando entonces

68 : 29 + 29 + 15

Para cada uno de los siguientes ejercicios:
• Identifique qué parámetros debe tomar la función recursiva que resuelve el problema.
• Describa con palabras qué calcula la misma, en función de sus argumentos.
• Defina la función recursiva en notación matemática y opcionalmente en código.
• Indique cuál es la llamada principal que obtiene el resultado pedido en el ejercicio.

Ej3)

Una panaderıa recibe n pedidos por importes $m_1, \dots, m_n$, pero sólo queda en depósito una cantidad H de harina en buen estado. Sabiendo que los pedidos requieren una cantidad $h_1, \dots, h_n$ de harina (respectivamente), determinar el máximo importe que es posible obtener con la harina disponible.

- El enunciado:
  - n pedidos con importes $m_1, \dots, m_n$ y costos de harina $h_1, \dots, h_n$
  - H es la harina que tenemos disponible para hacer pedidos
  - Determinar el máximo importe que es posible obtener con la harina disponible
- Parámetros:
  - **IMPORTANTE:** El algoritmo deberá obtener el **MÁXIMO** importe, pero no que pedidos hacemos
  - Definimos:
    - *pedidos(i,j) = "el máximo importe posible que puedo generar con la harina disponible haciendo algunos pedidos entre 1 e i, de manera tal que su costo de harina sea menor o igual a h"*
- Función recursiva:

  pedidos(n,H)

$$
pedidos(i,j) = \begin{cases} 0 & i = 0 \lor j = 0 \\ pedidos(i-1, j) & h_i > j \land (j > 0 \land i > 0) \\ \max(m_i + pedidos(i-1, j - h_i), pedidos(i-1, j)) & j \geq h_i \land (j > 0 \land i > 0) \end{cases}
$$

Testeo "rápido" con:

```
m₁ = 5      h₁ = 10
m₂ = 6      h₂ = 5
m₃ = 10     h₃ = 11

H = 15
----
pedidos(3,15)
max( 10 + pedidos(2,4), pedidos(2,15)
=
max( 10 + pedidos(1,4), max( 6 + pedidos(1, 10), pedidos(1, 15)))
=
max( 10 + pedidos(0,4), max( 6 + pedidos(1, 10), pedidos(0, 15)))
=
max( 10 , 6 + pedidos(1, 10))
=
max( 10 , 6 + max( 5 + pedidos(0, 0), pedidos(0, 10)))
=
max( 10 , 6 + 5)
=
11
```

Ej4)

Usted se encuentra en un globo aerostático sobrevolando el océano cuando descubre que empieza a perder altura porque la lona está levemente dañada. Tiene consigo n objetos cuyos pesos p1 , . . . , pn y valores v1 , . . . , vn conoce. Si se desprende de al menos P kilogramos logrará recuperar altura y llegar a tierra firme, y afortunadamente la suma de los pesos de los objetos supera holgadamente P . ¿Cuál es el menor valor total de los objetos que necesita arrojar para llegar sano y salvo a la costa?

- El enunciado:
  - n objetos con pesos $p_1,p_2,....,p_n$ y valores $v_1,v_2,...,v_n$
  - P es el peso que debemos perder para que el globo no se caiga.
  - Determinar el MENOR valor total de los objetos que debo tirar para que el globo no se caiga.
- Identifiquemos qué parámetros toma la función recursiva, y qué calcula en función de esos parámetros. IMPORTANTE: el algoritmo deberá obtener el MENOR valor tirado, pero no qué objetos tiramos.

```
globo(i,h) = “el menor valor posible del que debo desprenderme tirando
algunos de los objetos entre 1 e i, de manera tal que su peso sea mayor o
igual a h”
```

si entendemos la función que queremos definir, digamos cuál es la “llamada” a esta función que resuelve el problema del ejercicio.

$$globo(i,h) \begin{cases} 0 & h \le 0 \\ +inf & h > 0 \wedge i = 0 \\ min(v_i + globo(i-1,h-p_i), globo(i-1,h)) & h > 0 \wedge i > 0 \end{cases}$$

Testeo "rápido" con:

$p_1 = 2$

$p_2 = 4$

$p_3 = 10$

$P = 8$

----

globo(3,8)

=

min(v_3 + globo(2, -2), globo(2, 8))

=

min(v_3, globo(2, 8))

=

min(v_3, min(v_2 + globo(1, 4), globo(1, 8)))

=

min(v_3, min(v_2 + **min(v_1 + globo(0, 2), globo(0, 4))**, globo(1, 8)))

=

min(v_3, globo(1, 8))

=

min(v_3, +infinito) -- Nos salteamos pasos, pero ya nos damos cuenta que p_i es 2 (no voy a llegar nunca a tirar 8kg)

=

v_3

Ej5)

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 (pi y ri) cada amigo ofrece un monto mi por día. Determinar el máximo valor alcanzable alquilando el teléfono.

- El enunciado:
  - n amigos con p_i , r_i (partida y regreso) y m_i (monto por día)
  - Determinar el MÁXIMO valor alcanzable alquilando el teléfono.
- Identifiquemos qué parámetros toma la función recursiva, y qué calcula en función de esos parámetros. IMPORTANTE: el algoritmo deberá obtener el MÁXIMO valor obtenido alquilando el celular

```
telefono(d) = “el máximo valor obtenible alquilando el celular desde el día d”

teléfono(0) = “el máximo valor posible que se obtiene prestando el celular desde el dia 0”
```

	0	si para todo i, p_i < d
teléfono(d)	telefono(d+1)	si no lo presto el dia d
	max($m_i * (r_i + 1 - p_i) +$ teléfono($r_i + 1$)	si p_i = d

Ej6)

Un artesano utiliza materia prima de dos tipos: A y B. Dispone de una cantidad M A y M B de cada una de ellas. Tiene a su vez pedidos de fabricar n productos p1 , . . . , pn (uno de cada uno). Cada uno de ellos tiene un valor de venta v1 , . . . , vn y requiere para su elaboración cantidades a1 , . . . , an de materia prima de tipo A y b1 , . . . , bn de materia prima de tipo B. ¿Cuál es el mayor valor alcanzable con las cantidades de materia prima disponible?

- El enunciado:
  - n productos a fabricar p1 , . . . , pn (uno de cada tipo de materia)
  - valor de venta de cada producto v1 , . . . , vn
  - requiere a1 , . . . , an de materia prima de tipo A
  - requiere b1 , . . . , bn de materia prima de tipo B
  - Determinar el MÁXIMO valor alcanzable con la materia prima disponible.
- Identifiquemos qué parámetros toma la función recursiva, y qué calcula en función de esos parámetros.

- Llamada función recursiva:

art(i , ma , mb) “ MÁXIMA ganancia obtenible fabricando i productos con ma materia prima tipo A y mb materia prima tipo B”

- Llamada función principal:

    art(n,MA,MB) “MÁXIMA ganancia obtenible fabricando n productos con MA materia prima tipo A y MB materia prima tipo B”

- Función recursiva

art(i, ma, mb) =
| 0 , si i = 0
| max(v_i + art(i - 1, ma - a_i, mb - b_i), art(i - 1, ma, mb)) , si i > 0 ^ ma ≥ a_i ^ mb ≥ b_i
| art(i - 1, ma, mb) ,si i > 0 ^ ma < a_i v mb < b_i

Ej7)

En el problema de la mochila se buscaba el máximo valor alcanzable al seleccionar entre n objetos de valores v_1 , . . . , v_n y pesos w_1 , . . . , 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?

- El enunciado:
    - n objetos de valores v_1 , . . . , v_n y pesos w_1 , . . . , w_n
    - 2 mochilas con capacidad para W_1 y W_2 respectivamente
    - ¿Cuál es el valor máximo alcanzable al seleccionar objetos para cargar en ambas mochilas? tq no se supere el peso W_i
    - Tengo que ver todas las combinaciones posibles y me quedo con el valor máximo que me de, poner el objeto i en la mochila W_1 o no poderlo y ponerlo en W_2 o no ponerlo.
- Parámetros:
    - Llamada función recursiva:
        - moc(i, w1, w2), i = Cantidad de objetos que tengo para agregar
          w1 = Peso actual disponible de la mochila 1
          w2 = Peso actual disponible de la mochila 2
    - Llamada función principal que se obtiene el máximo valor obtenible pudiendo cargar n objetos en 2 mochilas:
        - moc(n,W_1,W_2)
- Función matemática:

casos: El peso de w1 se acabó, pero todavía entran objetos en w2
El peso de w2 se acabó, pero todavía entran objetos en w2
*Ambas mochilas se quedaron sin capacidad
*No tengo más objetos para agregar
*El objeto es más pesado que el peso disponible de w1 pero no de w2 (entra en w2)
*El objeto es más pesado que el peso disponible de w2 pero no de w1 (entra en w1)
*El objeto no entra en ninguna
El objeto es menos pesado que ambos pesos disponibles(entra en ambos)

$$
moc(i,w1,w2) \begin{cases}
0 & ,\text{si } (w1 = 0 \wedge w2 = 0) \vee i = 0 \\
moc(i-1,w1,w2) & ,\text{si } p_i > w1,\ w2 > 0 \wedge i > 0 \\
\max(v_i + moc(i-1, w1, w2 - p_i), moc(i - 1, w1, w2)) & ,\text{si } w2 \geq p_i > w1 \wedge i > 0 \\
\max(v_i + moc(i-1, w1 - p_i , w2), moc(i-1, w1, w2)) & ,\text{ si } w1 \geq p_i > w2 \wedge i > 0 \\
\max(v_i + moc(i-1, w1 - p_i , w2) , v_i + moc(i-1, w1, w2 - p_i), moc(i-1, w1, w2)) & ,\text{ si } w1 \geq p_i \wedge w2 \geq p_i \wedge i > 0
\end{cases}
$$

Ej8)

Una fábrica de automóviles tiene dos líneas de ensamblaje y cada línea tiene n estaciones de trabajo, $S_{1,1}$, . . . , $S_{1,n}$ para la primera y $S_{2,1}$, . . . , $S_{2,n}$ para la segunda. Dos estaciones $S_{1,i}$ y $S_{2,i}$ (para i = 1, . . . , 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 $S_{i1,1}$, $S_{i2,2}$ . . . , $S_{in,n}$ 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' != i) cuesta $t_{i,j}$. Encontrar el costo mínimo de fabricar un automóvil usando ambas líneas.

- El enunciado:
  - 2 líneas de ensamblaje
  - n estaciones de trabajo por cada línea
  - $S_{1,1}$, . . . , $S_{1,n}$ para la primera y $S_{2,1}$, . . . , $S_{2,n}$ para la segunda
  - hacen el mismo trabajo pero con costos $a_{1,i}$ y $a_{2,i}$
  - Para fabricar un auto pasamos por n estaciones, no deben ser de la misma línea de ensamblaje
  - Pero cambiar de línea de ensamblaje cuesta $t_{i,j}$
  - Encontrar el costo mínimo usando ambas líneas
- Parámetros:
  - La función tomará las n estaciones por las cuales debe pasar el auto para ser ensamblado.
  - Tenemos la opción de hacerlo en una de dos líneas
  - Llamada recursiva:
    - car(i , $S_{J,i}$)
    - Costo mínimo de fabricar un auto en i estaciones en $S_{j,i}$ línea de ensamblaje
  - Llamada principal
    - car(n , $S_{J,i}$)
    - Costo mínimo de fabricar un auto en n estaciones en $S_{j,n}$ línea de ensamblaje
- Función matemática:

  Casos: ya no tengo estaciones de trabajo

  tengo estaciones de trabajo:

  yo tengo el auto en $S_{1,i}$

si $a_{2,i} + t_{i,2} < a_{1,i}$ (debería cambiar a $S_{2,i}$)
yo tengo el auto en $S_{2,i}$
si $a_{1,i} + t_{i,1} < a_{2,i}$ (debería cambiar a $S_{1,i}$)
yo tengo el auto en $S_{1,i}$
si $a_{2,i} + t_{i,2} \geq a_{1,i}$ (debería quedarme en $S_{1,i}$)
yo tengo el auto en $S_{2,i}$
si $a_{1,i} + t_{i,1} \geq a_{2,i}$ (debería quedarme en $S_{2,i}$)

car(i , $S_{J,i}$) =

| 0 , si $i = 0 \vee S_{j,i} = 0$
| min($a_{2,i} + t_{i,2}$ + car(i - 1, $S_{2,i-1}$), $a_{1,i}$ + car(i - 1, $S_{j,i-1}$))
, si $i > 0 \wedge a_{2,i} + t_{i,2} < a_{1,i}$
| min($a_{1,i} + t_{i,1}$ + car(i - 1, $S_{1,i-1}$), $a_{2,i}$ + car(i - 1, $S_{j,i-1}$))
, si $i > 0 \wedge a_{1,i} + t_{i,1} < a_{2,i}$
| min($a_{1,i}$ + car(i - 1, $S_{1,i-1}$), car(i - 1, $S_{1,i-1}$))
, si $i > 0$, $j = 1$, $a_{2,i} + t_{i,2} \geq a_{1,i}$
| min($a_{2,i}$ + car(i - 1, $S_{2,i-1}$), car(i - 1, $S_{2,i-1}$))
, si $i > 0$, $j = 2$, $a_{1,i} + t_{i,1} \geq a_{2,i}$

Ej9)

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 cij (i, j = 1, . . . , 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.
Los dos últimos ejercicios, también pueden resolverse planteando un grafo dirigido y recurriendo al algoritmo de Dijkstra. ¿De qué manera? ¿Serán soluciones más eficientes?

- Enunciado:
  - 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 cij (i, j = 1, . . . , 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
    - Determinar el máximo y el mínimo puntaje que se puede obtener en el juego
- Parámetros:
    - La función deberá tomar la casilla en la cual está parado
    - Llamada función recursiva:
        - up_min($T_{i,j}$) nos da el mínimo valor desde T $_{n,j'}$ hasta $T_{i,j}$
        - up_max($T_{i,j}$) nos da el máximo valor desde T $_{n,j'}$ hasta $T_{i,j}$
    - Llamada función principal:
        - up_min($T_{n,j}$) nos da el mínimo valor desde T $_{n,j'}$ hasta $T_{1,j}$
        - up_max($T_{n,j}$) nos da el máximo valor desde T $_{n,j'}$ hasta $T_{1,j}$
- Función matemática:
  Casos:
  La ficha ya esta arriba ($T_{1,j}$) i = 1
  La ficha está en la columna extrema izquierda   k = 1
  La ficha está en la columna extrema derecha   k = n
  La ficha no está en ninguna columna extrema y no estamos arriba i != 1

up_min($T_{i,k}$) | 0 , si i = 0
| min($c_{i,k}$ + up_min($T_{i-1,k}$), $c_{i,k}$ + up_min($T_{i-1,k+1}$)) , si k = 1, i != 1
| min($c_{i,k}$ + up_min($T_{i-1,k}$), $c_{i,k}$ + up_min($T_{i-1,k-1}$)) , si k = n, i != 1
| min($c_{i,k}$ + up_min($T_{i-1,k}$), $c_{i,k}$ + up_min($T_{i-1,k-1}$), $c_{i,k}$ + up_min($T_{i-1,k+1}$))
, si k != 1,n

up_max($T_{i,k}$) | 0 , si i = 0
| max($c_{i,k}$ + up_max($T_{i-1,k}$), $c_{i,k}$ + up_max($T_{i-1,k+1}$)) , si k = 1, i != 1
| max($c_{i,k}$ + up_max($T_{i-1,k}$), $c_{i,k}$ + up_max($T_{i-1,k-1}$)) , si k = n, i != 1
| max($c_{i,k}$ + up_max($T_{i-1,k}$), $c_{i,k}$ + up_max($T_{i-1,k-1}$), $c_{i,k}$ + up_max($T_{i-1,k+1}$))
, si k != 1,n