Practico 3.3 - Backtracking

> **1. Modifique el código del algoritmo que resuelve el problema de la moneda utilizando backtracking, de manera que devuelva que monedas se utilizan, en vez de solo la cantidad.**

```
fun cambio (monto : Nat, monedas : Set of Nat) ret res : set of Nat
    var aux_monedas, cambio_aux, cambio_aux_con_k : Set of Nat
    var k, aux_monto : Nat
    {- Hacemos una copia del conjunto de monedas -}
    aux_monedas = copy_set(monedas)
    aux_monto := monto

    {- Si el monto que queremos es 0 no necesitamos ninguna moneda -}
    if (aux_monto = 0) then
        res := empty_set()
    else
        k := get(aux_monto) {- Seleccionamos alguna moneda "k" -}
        elim(aux_monto, k) {- Eliminamos del set la moneda que elegimos -}

        {- Llamadas recursivas -}

        {- No elegir la moneda "k" -}
        cambio_aux := cambio(monto, aux_monto)
        {- Elegir la moneda "k" -}
        cambio_aux_con_k := cambio(monto - k, monedas)

        {- Elegir la mejor opcion -}
        {- Si no podemos elegir la moenda porque es muy grande o
                si el cambio sin usar la moneda "k" es mejor simplemente
                nuestro resultado va a ser elegir el cambio sin usar la moneda "k" -}
        if (k > monto V set_length(cambio_aux) < set_length(cambio_aux_con_k)) then
                res := cambio_aux
        else
            {- al contrario de lo anterior, no hace falta escribirlo -}
            ( k <= P V set_length(cambio_aux) => set_length(cambio_aux_mas_c))
                res := cambio_aux_con_k
        fi
        destroy_set(cambio_aux)
    fi
fi
end fun
```

> 2. 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 numero de monedas posibles. Los recuerdos cuestan 68, 74, 75, 83, 88 y 89.

> Asumiendo que tiene suficientes monedas para comprar cualquiera
> de ellos, ¿cual de ellos elegirá? ¿que monedas utilizara para pagarlo? Justificar claramente y mencionar el método utilizado.

Veamos que combinaciones necesitamos para elegir cada recuerdo:

- 68 = 29, 29, 15 (3 monedas)
- 74 = 29, 29, 29 (3 monedas)
- 75 =29, 29, 29 (3 monedas)
- 83 = 29, 29, 29, 15 (4 monedas)
- 88 = 29, 29, 29, 15 (4 monedas)
- 89 = 29, 29, 29, 15 (4 monedas)

Vemos que los recuerdos que usan menos monedas son los que cuestas 68, 74 y 75.

Como el problema no especifica si hay que elegir el recuerdo mas caro, o el recuerdo que no tenga tanto vuelto, dependiendo de esto sera el recuerdo que se compre.

Pero de lo que si estoy asegurando que es van a ser de 68, 74 o 75.

El método utilizado es ver cual es la mínima cantidad de monedas para obtener un determinado recuerdo y elegir el que use menos monedas.

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

> **3. Una panadería recibe n pedidos por importes m1, . . . , mn, pero solo queda en deposito una cantidad H de harina en buen estado.**
>
> **Sabiendo que los pedidos requieren una cantidad h1, . . . , hn de harina (respectivamente), determinar el máximo importe que es posible obtener con la harina disponible**

- **Parámetros de la función:** Vamos a tener dos parámetros,
  - el primero "i" el cual nos dice el importe mi y la cantidad de harina necesaria para ese importe hi.
  - Y el otro parámetro es "j" que es la cantidad de harina en buen estado.
- **Descripción de la función recursiva:** La función max_import(i, j) va a calcular el máximo de importe de "i" pedidos con la cantidad de harina "j".
- **Definición recursiva:**

$$maxImport(i,j) = \begin{cases} 0 & \text{si } i = 0 \vee j = 0 \\ maxImport(i-1,j) & \text{si } j < hi \wedge (i > 0 \wedge j > 0) \\ max(maxImport(i-1,j-hi)+mi, maxImport(i-1,j) & \text{si } j \geq hi \wedge (i > 0 \wedge j > 0) \end{cases}$$

- **Llamada principal:** maxImport(n, H) Ya que es el máximo importe de "n" pedidos con "H" de harina.
- **Explicacion detallada (no es parte del ejercicio):**
  - Si i = 0 o j = 0 puede signficar dos cosas:
    - no tengo pedidos
    - no tengo harina

    por lo tanto en ambos casos el maximo de importe que puedo tener es 0, ya que no puedo hacer ningun pedido.
  - Si j < hi y bueno i > 0 y j > 0, significa que la harina que tengo no me alcanza para hacer ese pedido, asi que sigo buscando los otros con mi funcion recursiva, es decir, sigo con el otro pedido con la misma cantidad de harina.
  - Por ultimo si j≥ hi y obviamente i>0 y j>0, significa que puedo hacer el pedido i, entonces el maximo importe que voy a obtener va a ser el maximo entre hacer ese pedido o no hacerlo, hacer el pedido lo traducimos a seguir con el otro (i-1) pero con la harina restante despues de hacer el pedido i (j - hi) y ademas le sumamos el importe mi. y no hacer el pedido se traduce en seguir con el otro pedido (i-1) pero sin usar la harina (j)

> 4. Usted se encuentra en un globo aerostático sobrevolando el océano cuando descubre que empieza a perder altura porque la lona esta levemente danada.
>
> Tiene consigo n objetos cuyos pesos p1, . . . , pn y valores v1, . . . , vn conoce.
>
> Si se desprende de al menos P kilogramos lograra recuperar altura y llegar a tierra firme, y afortunadamente la suma de los pesos de los objetos supera holgadamente P.
>
> ¿Cual es el menor valor total de los objetos que necesita arrojar para llegar sano y salvo a la costa?

- **Parámetros de la función:** Los parametros que vamos a usar son dos:
  - "i" para el objeto "i" con peso pi y valor vi
  - "j" para la cantidad de peso que necesitamos para llegar a tierra sano y salvo
- **Descripción de la función recursiva:** La función min_valor(i, j) va a calcular el menor valor teniendo "i" objetos y peso "j"
- **Definición recursiva:**

$$minValor(i,j) = \begin{cases} 0 & \text{si } i > 0 \wedge j = 0 \\ \infty & \text{si } j < pi \vee (i = 0) \\ min(minValor(i-1,j-pi)+vi, minValor(i-1,j) & \text{si } j \geq pi \wedge (i > 0 \wedge j > 0) \end{cases}$$

  - **Llamada principal:** minValor(n, P) ya que es el menor valor posible teniendo "n" objetos para arrojar peso "P"
  - **Explicacion detallada (no es parte del ejercicio):**
    - Si i > 0 y j = 0 significa:
      - tengo objetos para arrojar pero no necesito arrojarlos ya que no necesito desprender ninguno ya que el peso que necesito arrojar para llegar a tierra firme es 0

      por lo tanto en el menor valor de los objetos a arrojar es 0, ya que no arrojo ninguno.

- en el segundo caso tenemos dos opciones:
  - Si j < pi significa que el peso que tengo disponible no me alcanza para llegar a tierra firme.
  - si i = 0, significa que no tengo objetos

  en ambos casos mi respuesta va a ser infinito, ya que nunca voy a queres elegir esta opción. (ver si esta bien lo de j<pi)
- Por ultimo si j≥pi (el peso disponible para llegar a tierra firme es suficiente del peso que necesito) y ademas tengo objetos y obviamente tengo peso disponible.

  en este caso lo que voy a buscar el menor valor de elegir arrojar el objeto i o no elegir, elegirlo significa pasar al siguiente (i-1) y restar el peso del objeto (j - pi) y ademas sumar el valor, y no elegirlo significa seguir con el otro (i-1) pero con el mismo peso disponible (j)

> 5. Sus amigos quedaron encantados con el teléfono satelital, para las próximas vacaciones ofrecen pagarle un alquiler por el. Ademas 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.

- **Parámetros de la función:** EN este caso voy a usar un solo parámetro
  - "i" cantidad de amigo, el amigo "i" tiene día de partida pi, día de regreso ri, y monto mi.
- **Descripción de la función recursiva:** la función maxValor(i, pi) va a calcular el máximo valor que puedo obtener alquilando el teléfono con "i" amigos desde el día "pi"
- **Función recursiva:**

$$maxValor(i,j) = \begin{cases} 0 & \text{si } i = 0 \\ maxValor(i-1, pi) & \text{si } pi > ri \\ max(maxValor(i-1, ri) + (ri - pi) * mi), maxValor(i-1, pi) & \text{si } pi \leq ri \wedge i > 0 \end{cases}$$

- **Llamada principal:** maxValor(n, p0) mayor valor alcanzable alquilando el teléfono teniendo n amigos desde el dia p0
- **Explicación detallada:** Es mas de los mismo de los ejercicios anteriores, lo único que puede sonar medio raro es porque se multiplica, la explicación de eso es que el monto mi, es por cada día, entonces al elegir al amigo para calcular el monto es por el total de días que viaja, entonces para calcular el total de día que viaja es el día que llego - el día que partió, ese resultado nos la cantidad de días que viajo, y obviamente cada día nos pago un monto mi, por eso lo multiplicamos por mi.

> 6. 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. ¿Cual es el mayor valor alcanzable con las cantidades de materia prima disponible?

- **Parámetros de la función:** En este caso voy a usar los siguientes parámetros:
  - "i" para representar la cantidad actual de productos, vi el valor, ai la cantidad de materia prima que necesita de tipo A y bi la cantidad de materia prima de tipo B.
  - "j" representa la cantidad actual de materia prima disponible de tipo "A"
  - "k" representa la cantidad actual de materia prima disponible de tipo "B"
- **Descripción de la función recursiva:** la función recursiva maxValorDisp(i, j, k) representa el máximo valor alcanzable con "i" pedidos, "j" materia prima A y "k" materia prima B.
- **Función recursiva:**

$$maxValorDisp(i,j,k) = \begin{cases} 0 & \text{si } i = 0 \\ maxValorDisp(i-1, j, k) & \text{si } j < ai \vee k < bi \\ max(maxValorDisp(i-1, j-ai, k-bi) + vi, maxValorDisp(i-1, j, k)) & \text{si } j \geq ai \vee k \geq bi \end{cases}$$

- **Llamada principal:** maxValorDisp(n, MA, MB) mayor valor alcanzable con "n" pedidos y MA materia prima de tipo A y MB materia prima de tip "B"

> 7. En el problema de la mochila se buscaba el máximo valor alcanzable al seleccionar entre n objetos de valores v1, . . . , vn y pesos w1, . . . , wn, respectivamente, una combinación de ellos que quepa en una mochila de capacidad W. Si se tienen dos mochilas con capacidades W1 y W2, ¿cual es el valor máximo alcanzable al seleccionar objetos para cargar en ambas mochilas?

MAL: En la función recursiva tengo que ver todos los casos por separado.

- **Parámetros de la función:** En este caso voy a usar 3 parámetros
  - "i" cantidad de objetos actuales, con valor vi y peso wi

- "j" capacidad de la primera mochila
- "k" capacidad de la segunda mochila

- **Descripción de la función recursiva:** la función maxValor(i, j, k) calcula el máximo valor alcanzable con "i" objetos capacidad de la primera mochila 'j" y capacidad de la segunda mochila "k"
- **Función recursiva:**

$$maxValor(i,j,k) = \begin{cases} 0 & \text{si } i = 0 \\ maxValor(i-1,j,k) & \text{si } j < wi \wedge k < wi \\ max(maxValor(i-1,j-wi,k-wi) + vi, maxValor(i-1,j,k) & \text{si } j \geq wi \vee k \geq wi \end{cases}$$

- **Llamada principal:** maxValor(n, W1, W2) mayor valor alcanzable teniendo "n" objetos con la capacidad de la primera mochila "W1" y de la segunda "W2".

> 8. Una fabrica de automóviles tiene dos líneas de ensamblaje y cada línea tiene n estaciones de trabajo, S1,1, . . . , S1,n para la primera y S2,1, . . . , S2,n para la segunda.
>
> Dos estaciones S1,i y S2,i (para i = 1, . . . , n), hacen el mismo trabajo, pero lo hacen con costos a1,i y a2,i respectivamente, que pueden ser diferentes.
>
> Para fabricar un auto debemos pasar por n estaciones de trabajo Si1,1, Si2,2 . . . , Si,n, no necesariamente todas de la misma línea de montaje (ik = 1, 2).
>
> Si el automóvil esta en la estación Si,j , transferirlo a la otra linea de montaje (es decir continuar en Si',j+1 con i' != i) cuesta ti,j.
>
> Encontrar el costo mínimo de fabricar un automóvil usando ambas líneas

- **Parámetros de la función:** En este caso voy a usar dos parámetros (i, j):
  - "i" para representar la estación actual en la que estamos
  - "j" para representar la linea de ensamblaje (j = 1,2)
  - Entonces sj,i representa la estación de trabajo "i" de la linea de ensamblaje "j".
  - También tenemos aj, i el costo de hacer el trabajo en la estación "i" y linea "j"
  - Cambiar de estación va a costar ti,j
- **Descripción de la función recursiva:** la función recursiva minCost(i, j) calcula el minimo costo teniendo i estaciones de trabajo y usando la linea "j"
- **Función recursiva:**

$$minCost(i,j) = \begin{cases} 0 & \text{si } i = 0 \vee si, j = 0 \\ min(minCost(i-1,j) + aj,i, minCost(i-1, jmod2+1) + a(jmod2+1), i + ti,j) & \text{si } i > 0 \end{cases}$$

- **Llamada principal:** min(MinCost(n, 1), MinCost(n, 2))
- **Explicación detallada:** En este caso creo conveniente hacer una explicación detallada, ya que use varias cosas raras, primero el j mod 2 + 1, es para elegir la estación no actual, si estoy en la estación j = 1, 1 mod 2 = 1 + 1 = 2, que es la estación diferente a la 1, y si j = 2, 2 mod 2 = 0 + 1 = 1 que es la estación diferente a la 2. Entonces usando ese truquito puedo elegir cambiar de estación sin tantos casos.

  Luego en mi llamada principal, busco el min(MinCost(n, 1), MinCost(n, 2)) ya que voy a buscar el mínimo empezando en la estación 1 o empezando en la estación 2, y ahí estarían todos los casos cubiertos.

> 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 esta inmediatamente arriba
> - la casilla que esta arriba y a la izquierda (si la ficha no esta en la columna extrema izquierda),
> - la casilla que esta arriba y a la derecha (si la ficha no esta en la columna extrema derecha).
>
> Cada casilla tiene asociado un numero entero cij (i, j = 1, . . . , n) que indica el puntaje a asignar cuando la ficha este 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.

- **Parámetros de la función:** En este caso voy a usar dos parámetros (i, j):

- Tenemos “i” que corresponde a la fila actual del tablero.
- Tenemos “j” que correspnde a la columna actual del tablero,
- Entonces Ti, j es la posición actual en la que estoy en el tablero.
- Ademas tenemos Ci,j que es el puntaje a asignar que tiene la posición Ti, j

- **Descripción de la función recursiva:** la función recursiva maxPj(i, j) calcular el puntaje máximo que se puede obtener en el juego desde la fila “i” y la columna “j”.

  a(j mod 2 + 1), i + ti, jY la función minPj(i, j) calcula el mínimo puntaje que se puede obtener en el juego desde la fila “i” y la columna “j”.

- **Función recursiva:**

$$minPj(i,j) = \begin{cases} \infty & \text{si } i < 0 \vee si, j < 0 \vee i > n \vee j > n \\ min(minPj(i+1,j) + ci,j, minPj(i+1,j+1) + ci,j, minPj(i+1,j-1) + ci,j) & \text{si } i > 0 \wedge j > 0 \wedge i \leq n \wedge j \leq n \end{cases}$$

$$maxPj(i,j) = \begin{cases} -\infty & \text{si } i < 0 \vee si, j < 0 \vee i > n \vee j > n \\ max(maxPj(i+1,j) + ci,j, maxPj(i+1,j+1) + ci,j, maxPj(i+1,j-1) + ci,j) & \text{si } i > 0 \wedge j > 0 \wedge i \leq n \wedge j \leq n \end{cases}$$

- **Llamada principal:** minPj(0, 0), maxPj(0, 0)