1. (Voraz) Finalmente tenés la posibilidad de irte N días (con sus respectivas noches) de viaje y en el recorrido que armaste, cada día/noche i estarás en una ciudad C_i. Contás con M pesos en total de presupuesto para gastar en alojamiento y para cada ciudad conocés el costo k_i por noche del único hotel que tiene. Cada noche i podés elegir entre dormir en el hotel de la ciudad, lo que te costará k_i, o dormir en una carpa que llevaste, que te cuesta 0. Se pide indicar en qué ciudades dormirás en hotel, de manera tal que el monto total gastado en alojamiento en los N días no supere el presupuesto M, minimizando la cantidad de noches que dormís en carpa. Para ello:

   (a) Indicá de manera simple y concreta, cuál es el criterio de selección voraz para construir la solución.

   (b) Indicá qué estructuras de datos utilizarás para resolver el problema.

   (c) Explicá en palabras cómo el algoritmo resolverá el problema.

   (d) Implementá el algoritmo en el lenguaje de la materia de manera precisa.

Enunciado:

- N días de viaje
- Cada dia i estaras en C_i
- Contamos con M pesos
- Costo de hotel por cada ciudad i es k_i
- Se puede dormir en carpa o en hotel, se pide indicar en qué ciudades dormirás en hotel tq el total gastado no supere el monto M, minimizando la cantidad de noches que se duerme en carpa

a) Se elige el hotel más barato

b)
```
type City = tuple
                id: nat
                hotel_cost: nat
            end tuple
```

c) El algoritmo toma un conjunto de City de N elementos y una cantidad M de dinero, este copia el conjunto para seleccionar paso por paso la ciudad con el hotel más barato disponible, luego, si alcanza la cantidad de dinero para costearlo se agrega a la solución.

d)
```
fun trip(C: Set of City, M: nat) ret S: List of City
      var c_aux: Set of City
      var m_aux: nat
      var current_city: City

      S := empty_list()
      c_aux := copy_Set(C)
      m_aux := M
      while(M != 0 and not is_empty_set(c_aux) do
            current_city := select(c_aux)
            if(current_city.hotel_cost ≤ m_aux) then
                  m_aux := m_aux - current_city.hotel_cost
                  addr(S,current_city)
```

```
            fi
            elim(c_aux, current_city)
      od
      destroy_set(c_aux)
end fun

fun select(C: Set of City) ret res: City
      var c_aux: Set of City
      var current_city: City
      var min_aux: nat

      min_aux := inf
      c_aux := copy_Set(C)
      while(not is_empty_set(c_aux) do
            current_city := get(c_aux)
            if(current_city.hotel_cost < min_aux) then
                  min_aux := current_city.hotel_cost
                  res := current_city
            fi
            elim(c_aux, current_city)
      od
      destroy_set(c_aux)
end fun
```

2. (Backtracking) En el piso 17 de un edificio que cuenta con n oficinas iguales dispuestas de manera alineada una al lado de la otra, se quieren pintar las mismas de modo tal que no haya dos oficinas contiguas que resulten pintadas con el mismo color. Se dispone de 3 colores diferentes cuyo costo por oficina es C_1, C_2 y C_3 respectivamente. Para cada oficina i, el oficinista ha expresado su preferencia por cada uno de los tres colores dando tres números p_1^i, p_2^i y p_3^i, un número más alto indica mayor preferencia por ese color. Escribir un algoritmo que utilice la técnica de backtracking para obtener el máximo valor posible de (sumatoria para i desde 1 a n, de $p_{j_i}^i/C_{j_i}$, es decir, que maximice $\sum_{i=1}^{n} p_{j_i}^i/C_{j_i}$), sin utilizar nunca el mismo color para dos oficinas contiguas.

   Antes de dar la solución, especificá con tus palabras qué calcula la función recursiva que resolverá el problema, detallando el rol de los argumentos y la llamada principal.

Enunciado:

- n oficinas.
- Se pide pintar todas con 3 colores tq dos oficinas contiguas no tengan el mismo color.
- Cada color tiene un coste por oficina C_1 C_2 y C_3.
- Para cada oficina i tenemos 3 preferencias por cada color p_i _1, p_i _2 y p_i _3.
- Se pide calcular el máximo valor posible.

Función recursiva:

pintar(i,c) “máximo valor obtenible al pintar i oficinas tq la siguiente es del color c”

Llamada principal:

pintar(n,0)

Función matemática:

pintar(i,c)

| 0	,si i = 0	
| max(pintar(i,1), pintar(i,2), pintar(i,3))	,si i > 0 & c = 0	
| max($p^{i_{1\_i}} / C_{1\_i}$ + pintar(i-1, 2), $p^{i_{1\_i}} / C_{1\_i}$ + pintar(i-1, 3))	,si i > 0 & c = 1	
| max($p^{i_{2\_i}} / C_{2\_i}$ + pintar(i-1, 1), $p^{i_{2\_i}} / C_{2\_i}$ + pintar(i-1, 3))	,si i > 0 & c = 2	
| max($p^{i_{3\_i}} / C_{3\_i}$ + pintar(i-1, 1), $p^{i_{3\_i}} / C_{3\_i}$ + pintar(i-1, 2))	,si i > 0 & c = 3	

3. (Programación dinámica) Escribí un algoritmo que utilice Programación Dinámica para resolver el ejercicio del punto anterior.

   (a) ¿Qué dimensiones tiene la tabla que el algoritmo debe llenar?

   (b) ¿En qué orden se llena la misma?

   (c) ¿Se podría llenar de otra forma? En caso afirmativo indique cuál.

```
fun pintar(p: array[1..n, 1..3] of nat, c: array[1..3] of nat) ret
res: nat
      {- variables -}
      var dp: array[0..n,0..3] of nat
      {- caso base -}
      for k := 0 to 3 do
            dp[0,k] := 0
      od
      {- caso recursivo -}
      for i := 1 to n do
            for k := 3 downto 0 do
                  if k == 0 then
                        dp[i,k] := max(dp[i,1], dp[i,2], dp[i,3])
                  else if k == 1 then
                        dp[i,k] := max(p[i,k] / c[k] + dp[i-1,2],
                                       p[i,k] / c[k] + dp[i-1,3])
                  else if k == 2 then
                        dp[i,k] := max(p[i,k] / c[k] + dp[i-1,1],
                                       p[i,k] / c[k] + dp[i-1,3])
                  else if k == 3 then
                        dp[i,k] := max(p[i,k] / c[k] + dp[i-1,1],
```

```
                                        p[i,k] / c[k] + dp[i-1,2])
                fi
            od
        od
        res := dp[n,0]
end fun
```