Parcial 2 - 2021

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 Ci.

   Contás con M pesos en total de presupuesto para gastar en alojamiento y para cada ciudad conocés el costo ki por noche del único hotel que tiene. Cada noche i podés elegir entre dormir en el hotel de la ciudad, lo que te costará ki, 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.

   El criterio de selección voraz es elegir las ciudades cuyo hotel sea mas barato, así podremos maximizar la cantidad de ciudades en la que se duerme en el hotel y minimizar la cantidad de noches que se duerme en carpa.

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

   Voy a utilizar un tipo ciudad que es una tupla de dos elementos el nombre de la ciudad para poder identificarla y el costo de dormir en hotel.

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

   El algoritmo va a recorrer el set de ciudades y mientras el costo del hotel de la ciudad sea menor o igual a mi presupuesto actual, voy a elegir dormir en el hotel de la ciudad.

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

```
type ciudad = tuple
                                nombre : String
                                hotel : Nat
                            end tuple

fun ciudades_en_hotel (ciudades :Array[1..n], M : nat) ret res : List of String
    var presupuesto_act : Nat
    presupuesto_act := M
    {- Funcion que ordena segun el campo de "hotel" -}
    ordenar(ciudades)
    var i : nat
    i:= 1
    while(ciudades[i].hotel < presupuesto_act && i<=n) do
        presupuesto_act := presupuesto_act - ciudades[i].hotel
        addr(res, ciudades[i].nombre)
        i:= i + 1
    od

proc ordenar(in/out ciudades : Array[1..n] of ciudad)
    var min_act : Nat

    for i := 1 to n do
        min_act := i

        for j := i+1 to n do
            if ciudades[j].hotel < ciudades[min_act].hotel then
                min_act := j
            fi
        od
```

```
        if min_act != i then
            swap(ciudades[i], ciudades[min_act])
        fi
    od
    return ciudades
end proc

proc swap(in/out a : ciudad, int/out b : ciudad)
    var temp : ciudad
    temp := a
    a := b
    b := temp
end proc
```

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 C1, C2 y C3 respectivamente. Para cada oficina i, el oficinista ha expresado su preferencia por cada uno de los tres colores dando tres números $p1^i, p2^i, p3^i$ n 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 pji^i/cji. sin utilizar nunca el mismo color para dos oficinas contiguas.

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

   Los argumentos de mi función recursiva son:

   - "i" para representar la i-esima oficina cuyo costo es ci, y ademas tenemos $p1^i, p2^i, p3^i$ que indican la preferencia por color
   - "col" un numero que puede ser 1, 2 o 3 que indica cual es el color que no se puede usar.

   La función recursiva maxValor(i, col) calcula el máximo valor posible al pintar las oficinas desde la oficina "i" hasta la última oficina, sin usar el color "col" para la oficina "i".

   La llamada principal va a ser max(maxValor(1, 1), maxValor(1, 2), maxValor(1, 3))

   Así calcularemos el máximo que podemos obtener sin empezar a pintar con los 3 casos posibles.

$$\text{maxValor}(i, \text{col}) = \begin{cases} 0 & \text{si } i > n \\ \max\left(\frac{p1^i}{C1} + \text{maxValor}(i+1,1), \frac{p2^i}{C2} + \text{maxValor}(i+1,2), \frac{p3^i}{C3} + \text{maxValor}(i+1,3)\right) & \text{si col} = 0 \\ \max\left(\frac{p2^i}{C2} + \text{maxValor}(i+1,2), \frac{p3^i}{C3} + \text{maxValor}(i+1,3)\right) & \text{si col} = 1 \\ \max\left(\frac{p1^i}{C1} + \text{maxValor}(i+1,1), \frac{p3^i}{C3} + \text{maxValor}(i+1,3)\right) & \text{si col} = 2 \\ \max\left(\frac{p1^i}{C1} + \text{maxValor}(i+1,1), \frac{p2^i}{C2} + \text{maxValor}(i+2,2)\right) & \text{si col} = 3 \end{cases}$$

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?

   La tabla dp del algoritmo tendrá dimensiones n * 3, donde:

   - n es el número de oficinas.
   - Cada fila representa una oficina.
   - Cada columna representa uno de los tres colores disponibles (C1, C2, C3).

   La tabla dp[i, j] almacenará el valor máximo de preferencia acumulada al pintar la oficina i con el color j, asegurando que la oficina i y la oficina i-1 no tienen el mismo color.

   (b) En qué orden se llena la misma?

   La tabla dp se llena de manera iterativa, comenzando desde la oficina 1 hasta la oficina n. O sea que se llena de abajo hacia arriba y de izquierda a derecha.

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

Si, se podría llenar de arriba hacia abajo, empezando en n hasta 1, pero habría que cambiar varias cosas en la función.