Parcial 2 - 2022

1. Un colectivero conduce su pequeño colectivo. Muy pequeño. Solamente hay lugar para un pasajero. Mas que un colectivo, parece una moto. Su recorrido o viaje va de la parada 1 hasta la parada n pasando por las paradas intermedias 2,3,..,n-1.

   Hay m pasajeros esperando. Para cada pasajero i sabemos en que parada se quiere subir (Si), y en que parada se va a bajar (b_i con 1<= s_i < b_i < n)
   La intención del colectivero es trasladar en un viaje a la mayor cantidad de pasajeros posible. El colectivero no tiene obligación de levantar a un pasajero por mas que este libre, puede preferir reservarlo para un pasajero que sube después.

   a) indicar de manera simple y concreta, cual es el criterio de selección voraz para construir la solución?

   El criterio de selección es seleccionar al pasajero que se baje antes, esto ya que al bajar a un pasajero lo antes posible, se maximiza la posibilidad de recoger a más pasajeros en las paradas siguientes.

   b) indicar que estructuras de datos utilizaras para resolver el problema.

   Voy a usar un tipo “pasajero” el cual es una tuple con la parada en la que se quiere subir y en la que se quiere bajar.

   c) explicar en palabras como resolverá el problema el algoritmo

   La idea del algoritmo es recorrer la lista de pasajeros (la cual previamente esta ordenada en orden ascendente teniendo en cuenta la parada en la que se baja cada pasajero) y seleccionamos a aquellos que puedan ser transportados, y vamos guardando la ultima parada en la que se bajo el ultimo pasajero.

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

```
type pasajero = tuple
                                    Si : Nat  {- Parada en la que el pasajero se sube -}
                                    Bi : Nat  {- Parada en la que el pasajero se baja -}
                                end tuple

fun max_pasajeros(pasajeros : Array[1..m] of Pasajero) ret res:Nat
    {- Suponer que existe la funcion sort que orden segun el criterio -}
    sort(pasajeros, pasajeros.Bi)  {- Ordenamos segun cuando se baja -}
    var ult_parada : nat {-Guardamos la ultima parada en la que se bajo el ult pasajero -}
    ult_parada := 0
    res := 0
    for i:= 1 to m do
        if (pasajeros[i].Si >= ult_parada) then
            res := res + 1
            ult_parada := pasajeros[i].Bi
        fi
    od
end fun
```

---

2. El presidente de tu país te acaba de elegir como asesor para tomar una serie de medidas de producción que mejoren la situacion económica. En el análisis preliminar se proponen n medidas, donde cada medida i {1...n} producirá una mejora económica de m, puntos con m_i > 0.

   También se analizo para cada una el nivel de daño ecológico d_i que producirá, donde d_i >0, El puntaje que tendrá cada medida i esta dado por la relación m_i/d_i.

   Se debe determinar cual es el máximo puntaje obtenible eligiendo K medidas con K<n, de manera tal que la suma total del daño ecológico no sea mayor a C.

   Se pide lo siguiente.

a) Especifica precisamente que calcula al función recursiva que resolverá el problema, indicando que argumentos toma y la utilidad de cada uno.

La funcion recursiva max_puntaje(i, c, k) calcula el maximo puntaje obtenible seleccionando "k" medidas, de las medidas de la i a la n, tal que la suma del dano ecologico no sea mayor a c.

Los argumentos que toma son:

- "i" → Medida actual que estamos considerando
- "c" → El dano ecologico permitible hasta el momento
- "k" → Cantidad de medidas seleccionadas

b) Da la llamada o la expresión principal que resuelve el problema.

La llamada principal es: **max_puntaje(1, C, K)**

c) Definí la función en notación matemática.

$$maxPuntaje(i,c,k) = \begin{cases} 0 & \text{si } i = 0 \\ -\infty & \text{si } i = 0 \wedge k \\ maxPuntaje(i+1,c,k) & \text{si } dk > c \\ max(maxPuntaje(i+1,c-di,k+1)+mi/di, maxPuntaje(i+1,c,k)) & \text{si } dk \leq c \wedge \end{cases}$$

---

3. Se tiene la siguiente definicion recursiva para 0<= i,j <= n

$$guntHonacci(i,j) = \begin{cases} 1 & \text{si } i = 0 \wedge j = 0 \\ 1 & \text{si } i = 0 \wedge j = 1 \\ 1 & \text{si } i = 1 \wedge j = 0 \\ guntHonacci(i,j-2) + guntHonacci(i,j-1) & \text{si } i = 0 \wedge j > 1 \\ guntHonacci(i-2,j) + guntHonacci(i-1,j) & \text{si } i > 1 \wedge j = 0 \\ guntHonacci(i,j-1) + guntHonacci(i-1,j) & \text{si } i > 0 \wedge j > 0 \end{cases}$$

Llamada principal: *guntHonacci(n, n)*

a) dar una definicion de la misma funcion usando programacion dinamica

```
fun guntHonacci(n:Nat) ret res : Nat
    var dp : Array[0..n, 0..n] of Nat

    {- Lleno la tabla de dp con los casos bases -}
    dp[0, 0] := 1
    dp[0, 1] := 1
    dp[1, 0] := 1

    for i:=0 to n do
        for j:= 0 to n do
            if (i = 0 && j>1) then
                dp[i,j] := dp[i, j-2] + dp[i, j-1]
            else
                if (i>1 && j = 0) then
                    dp[i, j] := dp[i-2, j] + dp[i-1, j]
                else
                    dp[i, j] := dp[i, j-1] + dp[i-1, j]
                fi
          fi
     od
 od
 res := dp[n, n]
end fun
```

b) explicar la eleccion de las dimensiones de la tabla de valores, del orden en que la misma se completa, y el valor del retorno.

Las dimensiones de la tabla es de n * n, esto se debe a la necesidad de almacenar los resultados de subproblemas para todos los posibles indices i, j y es de n*n ya que tanto i como j van de 0 a n.

El orden de llenado de la tabla es de abajo arriba hacia abajo y de izquierda a derecha

El valor de retorno esta dado en dp[n, n]

---

4. Sea T un árbol (no necesariamente binario) y supongamos que deseamos encontrar la hoja que se encentra mas cerca de la raiz.¿cuales son las distintas maneras de recorrer T?¿cual de ellas elegirías para encontrar esa hoja y porque?

   De las siguientes tres maneras de recorrer un arbol binario ¿cuales son ejemplos de recorridas en DFS y cuales son ejemplos de recorridas en BFS? justificar sus respuestas explicando con clardiad.

- recorrida en pre-orden
- recorrida en in-orden
- recorrida en pos-orden

Las tres maneras son ejemplos de DFS.

Recorrido en Preorden:

- **Descripción**: En el recorrido en preorden, se visita el nodo actual antes de visitar sus hijos.
- **Justificación**: Se explora un nodo y luego se visitan sus hijos en profundidad antes de volver atrás. Esto es característico de DFS, ya que va tan profundo como sea posible antes de retroceder.

Recorrido en Inorden:

- **Descripción**: En el recorrido en inorden, se visita el nodo actual entre la visita de su hijo izquierdo y su hijo derecho.
- **Justificación**: Se visita el hijo izquierdo, luego el nodo actual, y finalmente el hijo derecho. Esto implica una exploración en profundidad, típica de DFS.

Recorrido en Posorden:

- **Descripción**: En el recorrido en posorden, se visita el nodo actual después de visitar a sus hijos.
- **Justificación**: Se exploran todos los hijos de un nodo antes de visitar el nodo mismo. Se trata de un enfoque DFS, ya que se sigue profundizando en el árbol antes de volver atrás para visitar el nodo actual.