1. (Algoritmos voraces)

En la juntada del sábado con tus amigos, te mandan a comprar K porciones de helado. Luego de una discusión acalorada, lograron ponerse de acuerdo, asignando a cada sabor i de los N disponibles en la heladería, un puntaje no negativo p_i, y decidieron no repetir ningún sabor. Además, para cada sabor i se sabe si el mismo es al agua o no, mediante un booleano a_i. Se debe encontrar el mayor puntaje obtenible eligiendo K gustos distintos de helado, con la condición de que al menos M gustos sean al agua.

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

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

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

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

Enunciado:

- K porciones de helado
- De los N sabores disponibles, a cada uno se le asignó un puntaje p_i
- No se repiten sabores
- Para cada sabor se sabe si es al agua o no mediante un booleano a_i
- M gustos son al agua

a) Elegir siempre el sabor con mayor p_i

b)
```
type Sabor = tuple
                id : nat
                puntaje : nat
                tipo : bool
             end tuple
```

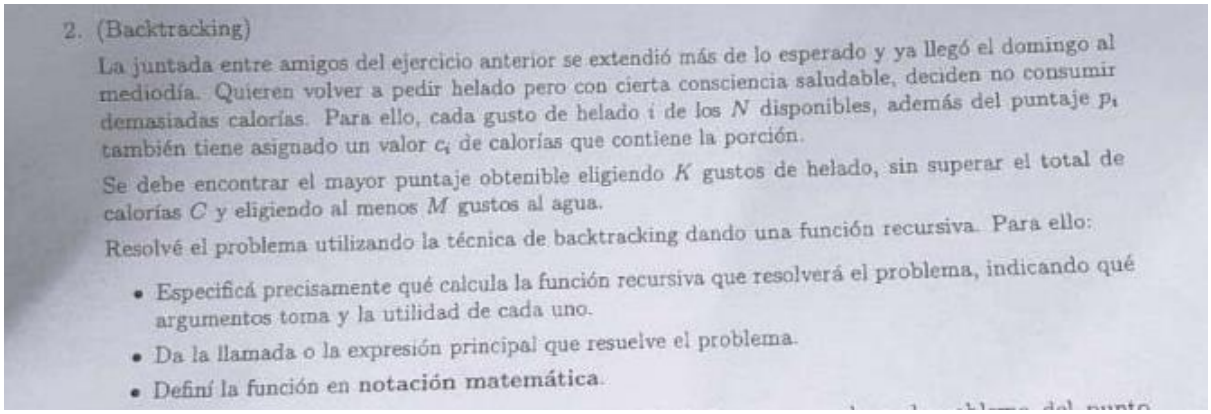
c) El algoritmo toma un conjunto de N sabores. Luego, mediante un ciclo que se ejecuta K veces se selecciona siempre el gusto con mayor puntaje (al agua o a la crema), agregamos el puntaje a la solución y luego eliminamos el sabor elegido del conjunto para buscar el siguiente.

d)

```
fun ice_cream(S: Set of Sabor, K: nat, M: nat) ret result: nat
      var s_aux: Set of Sabor
      var sabor: Sabor
      var i: nat
      var j: nat
      i := 0
      j := 0
      s_aux = copy_set(S)
      while( i < K and not is_empty_set(s_aux)) do
            if ( j < M ) then
                sabor = elegir_sabor(s_aux, true)
            else
                sabor := elegir_sabor(s_aux, false)
            fi
            result := result + sabor.puntaje
            elim(s_aux, sabor)
            i++
            j++
      od
      destroy_set(s_aux)
end fun

fun elegir_sabor(S: Set of Sabor, B: bool) ret res: Sabor
      var s_aux: Set of Sabor
      var sabor: Sabor

      s_aux = copy_set(S)
      while(not is_empty_set(s_aux)) do
            sabor := get(s_aux)
            if (B) then
                  if (sabor.tipo == true and sabor.puntaje > max_aux) then
                        max_aux := sabor.puntaje
                        res := sabor
                  fi
            else
                  if (sabor.puntaje > max_aux) then
                        max_aux := sabor.puntaje
                        res := sabor
                  fi
            fi
            elim(s_aux, sabor)
      od
      destroy_set(s_aux)
end fun
```



2. (Backtracking)

La juntada entre amigos del ejercicio anterior se extendió más de lo esperado y ya llegó el domingo al mediodía. Quieren volver a pedir helado pero con cierta consciencia saludable, deciden no consumir demasiadas calorías. Para ello, cada gusto de helado i de los N disponibles, además del puntaje p_i también tiene asignado un valor c_i de calorías que contiene la porción.

Se debe encontrar el mayor puntaje obtenible eligiendo K gustos de helado, sin superar el total de calorías C y eligiendo al menos M gustos al agua.

Resolvé el problema utilizando la técnica de backtracking dando una función recursiva. Para ello:

- Especificá precisamente qué calcula la función recursiva que resolverá el problema, indicando qué argumentos toma y la utilidad de cada uno.
- Da la llamada o la expresión principal que resuelve el problema.
- Definí la función en **notación matemática**.

Enunciado:

- K porciones de helado
- De los N sabores disponibles, a cada uno se le asignó un puntaje p_i y además un c_i de calorías que contiene la porción
- No se repiten sabores
- Para cada sabor se sabe si es al agua o no mediante un booleano a_i
- M gustos son al agua
- Se debe obtener el mayor puntaje posible sin superar C calorias

llamada recursiva:

elegir(i, k, c, m) = "máximo puntaje al elegir k gustos de entre {1, ..., i} sin superar c calorías y eligiendo al menos m gustos al agua"

llamada principal:

elegir(N, K, C, M)

función matemática:

elegir(i, k, c, m)
| 0 , si (i = 0 v i > 0) ^ k = 0 ^ m = 0
| -inf , si i = 0 ^ k > 0
| -inf , si k = 0 ^ m > 0
| elegir(i-1, k, c, m) , si i > 0 ^ k > 0 ^ c < c_i
| max (elegir(i-1, k, c, m), p_i + elegir(i-1, k-1 ,c - c_i, max(m-1, 0))) , si i > 0 ^ k > 0 ^ c ≥ c_i ^ a_i
| max (elegir(i-1, k, c, m), p_i + elegir(i-1, k-1, c - c_i , m)) , si i > 0 ^ k > 0 ^ c ≥ c_i ^ not a_i

- Definí la función en notación matemática.

3. Implementá un algoritmo que utilice Programación Dinámica para resolver el problema del punto anterior. Para ello primero respondé:
   - ¿Qué dimensiones tiene la tabla que el algoritmo debe llenar?
   - ¿En qué orden se llena la misma?
   - ¿Se podría llenar de otra forma? En caso afirmativo indique cuál.

fun (K: nat, M: nat, C: nat, c: array[1..n] **of** nat, a: array[1..n] **of** bool, p: array[1..n] **of** nat) **ret** result: nat

```
{- variables -}
var tabla[0..N, 0..K, 0..C, 0..M] of nat
{- casos base -}
{- ya no debo elegir nada k = 0 y m = 0 -}
for i := 0 to N do
   for c := 0 to C do
      tabla[i,0,c,0] := 0
   od
od
{- no tengo disponibles sabores para elegir -}
for k := 1 to K do
   for m := 1 to M do
      for c2 := 0 to C do
         tabla[0,k,c2,m] := -inf
      od
   od
od
{- ya elegí todos los posibles (sobran o no) pero me faltan de agua -}
for i := 0 to N do
   for m := 1 to M do
      for c2 := 0 to C do
         tabla[i,0,c2,m] := -inf
      od
   od
od

{- caso recursivo -}
for i := 1 to N do
   for k := 1 to K do
      for c2 := 0 to C do
          for m := 0 to M do
             if a[i] and c2 ≥ c[i] then
                tabla[i,k,c2,m] := max( tabla[i-1, k, c2, m], p[i] + tabla[i-1,k-1 ,c2 - c[i], max(m-1, 0)])
             else if  c2 ≥ c[i] and not a[i] then
                tabla[i,k,c2,m] := max( tabla[i-1, k, c2, m], p[i] + tabla[i-1,k-1 ,c2 - c[i], m]
             else if c2 < c[i]
                tabla[i,k,c2,m] := tabla[i-1, k, c2, m]
             fi
          od
      od
   od
od
res := tabla[N,K,C,M]
end fun
```

4. (a) Indicá breve y concisamente qué hace el algoritmo de Prim. Para ello describí precisamente qué recibe como argumento y qué devuelve.

(b) Indicá breve y concisamente qué hace el algoritmo de Dijkstra. Para ello describí precisamente qué recibe como argumento y qué devuelve.

(c) ¿Los dos algoritmos mencionados previamente son voraces? Justifique la respuesta.

(d) De las siguientes tres maneras de recorrer un árbol binario ¿cuáles son ejemplos de recorridas en DFS y cuáles son ejemplos de recorridas en BFS? Justificar sus respuestas explicando con claridad.

- recorrida en pre-orden,
- recorrida en in-orden,
- recorrida en pos-orden.

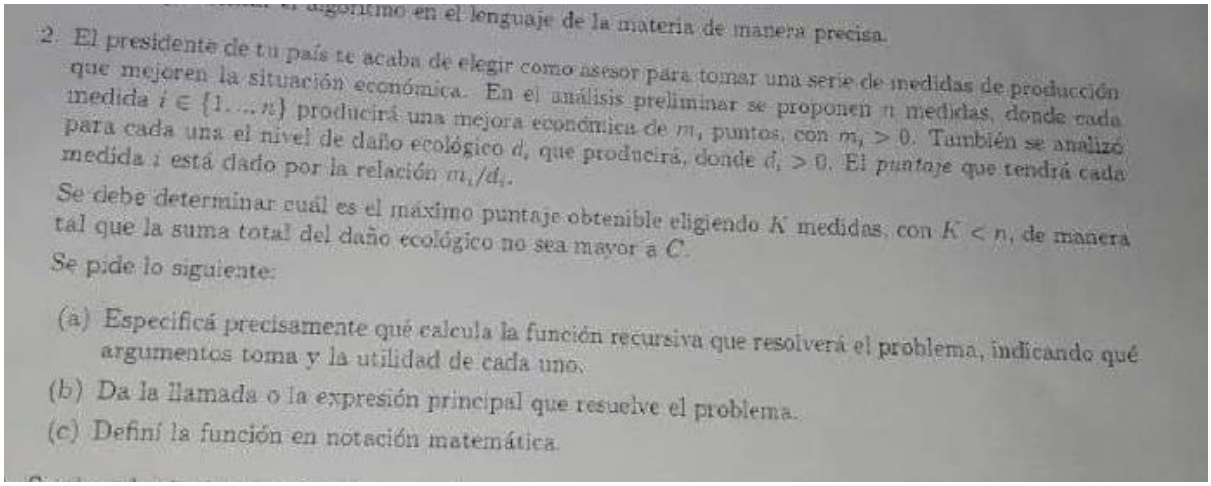
a) Encuentra un árbol generador de costo mínimo. Se parte desde un vértice origen y se va extendiendo el tendido a partir de ahí: en cada paso se une el tendido ya existente con alguno de los vértices aún no alcanzados, seleccionando la arista de menor costo capaz de incorporar un nuevo vértice

   Toma como argumentos un grafo G y un vértice inicial k
   Devuelve un conjunto de aristas

b) Busca obtener los caminos de menor costo desde v hacia cada uno de los demás vértices. En una secuencia de n pasos, donde n es el número de vértices. En cada paso, “aprende” el camino de menor costo desde v a un nuevo vértice. A ese nuevo vértice lo pinta de azul. Tras esos n pasos, conoce los costos de los caminos de menor costo a cada uno de los vértices.

   Toma un arreglo de dos dimensiones que representan un grafo y un vértice inicial v.
   Devuelve un arreglo

c) Si, los dos son voraces. Porque si qcyo odio esta materia
d) pre-orden, post-orden, e in-orden son DFS.



el algoritmo en el lenguaje de la materia de manera precisa.

2. El presidente de tu país te acaba de elegir como asesor para tomar una serie de medidas de producción que mejoren la situación económica. En el análisis preliminar se proponen n medidas, donde cada medida $i \in \{1...n\}$ producirá una mejora económica de m_i puntos, con $m_i > 0$. También se analizó para cada una el nivel de daño ecológico d_i que producirá, donde $d_i > 0$. El *puntaje* que tendrá cada medida i está dado por la relación m_i/d_i.

Se debe determinar cuál 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) Especificá precisamente qué calcula la función recursiva que resolverá el problema, indicando qué argumentos toma y la utilidad de cada uno.

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

(c) Definí la función en notación matemática.

Enunciado(Parcial 2 2022):

- Se proponen n medidas
- Cada medida i ∈ {1,...,n} produce una mejor m_i en la economía $m_i > 0$
- También cada i tiene $d_i > 0$ daño ecológico
- El puntaje de cada pedida está dado por m_i / d_i
- Se debe determinar el máximo puntaje obtenible eligiendo K medidas, con K < n, de manera tq la suma del daño ecológico no sea mayor a C

Llamada recursiva:

- elegir(i, k, c) “máximo puntaje obtenible eligiendo k medidas entre i ∈ {1,...,i} tq la suma del daño ecológico es menor a c”

Llamada principal:

- elegir(n,K,C)

Función matemática:

- Casos:
  - i = 0 & k > 0 (no tenemos medidas analizadas)
  - i = 0 & k = 0 (terminamos)
  - i > 0 & k = 0 (terminamos antes)
  - i > 0 & k > 0 & c ≥ d_i (podemos tomar la medida)
  - i > 0 & k > 0 & d_i > c (se pasa de daño ecológico)

elegir(i, k, c) | 0 , si i = 0 & k = 0
| 0 , si i > 0 & k = 0
| -inf , si i = 0 & k > 0
| max(elegir(i-1, k, c), m_i / d_i + elegir(i-1, k-1, c - d_i)) , si i > 0 & k > 0 & c ≥ d_i
| elegir(i-1, k, c) , si i > 0 & k > 0 & d_i > c

En programación dinámica:

fun elegir(n: nat, K: nat, C: nat, m: array[1..n] **of** nat, d: array[1..n] **of** nat) **ret** res: nat

```
    {- variables -}
    var tabla: array[0..n, 0..K, 0..C] of nat
    {- casos base -}
    for i := 0 to n do
        for c := 0 to C do
            tabla[i, 0, c] := 0
        od
    od
    for k := 1 to K do
        for c := 0 to C do
            tabla[0,k,c] := -inf
        od
    od
    {- caso recursivo -}
    for i := 1 to N do
        for k := 1 to K do
            for c := 0 to C do
                if c ≥ d[i] then
                    tabla[i,k,c] := max(tabla[i-1,k,c], m[i] / d[i] + tabla[i-1, k-1, c - d[i]])
                else
                    tabla[i,k,c] := tabla[i-1, k, c]
                fi
            od
        od
    od
    res := tabla[N,K,C]
end fun
```

3. Dados $c_1, c_2, \ldots, c_n$ y $d_1, d_2, \ldots, d_k$, la siguiente definición recursiva de la función m, para $1 \leq i \leq n$ y $1 \leq j \leq k$, determinar un programa que utilice la técnica de programación dinámica para calcular el valor de $m(1,1)$.

$$m(i,j) - \begin{cases} c_i & \text{si } j = k \\ d_j & \text{si } i - n \wedge j < k \\ m(i, j+1) + m(i+1, j) & \text{si } j < k \wedge i < n \end{cases}$$

Enunciado (parcial 2 2018)