Backtracking (ejercicio del práctico):

8. Una fábrica de automóviles tiene dos líneas de ensamblaje y cada línea tiene n estaciones trabajo, $S_{1,1}, \ldots, S_{1,n}$ para la primera y $S_{2,1}, \ldots, S_{2,n}$ para la segunda. Dos estaciones $S_{1,i}$ y (para $i = 1, \ldots, n$), hacen el mismo trabajo, pero lo hacen con costos $a_{1,i}$ y $a_{2,i}$ respectivame que pueden ser diferentes. Para fabricar un auto debemos pasar por n estaciones de trab $S_{i_1,1}, S_{i_2,2} \ldots, S_{i_n,n}$ no necesariamente todas de la misma línea de montaje ($i_k = 1, 2$). S automóvil está en la estación $S_{i,j}$, transferirlo a la otra línea de montaje (es decir continuar $S_{i',j+1}$ con $i' \neq i$) cuesta $t_{i,j}$. Encontrar el costo mínimo de fabricar un automóvil usando am líneas.

Enunciado:

- Dos líneas de ensamblaje y cada línea tiene n estaciones de trabajo
- Hacen lo mismo pero lo hacen con costos $a_{1,i}$ y $a_{2,i}$ respectivamente
- Para fabricar un auto se debe pasar por n estaciones
- Si se desea cambiar de línea de ensamblaje el costo es de $t_{i,j}$
- Encontrar el costo mínimo de fabricar un automóvil usando ambas líneas

Llamada recursiva:

f(i, j) “costo mínimo de fabricar un automóvil en la línea j y estación i”

Llamada principal:

f(n,0)

Función matemática:

Casos:

- ya no tengo mas estaciones i = 0
- tengo estaciones pero estoy en el inicio j = 0
- tengo estaciones y me cuesta menos quedarme en la línea actual
- tengo estaciones pero cuesta menos cambiarle a la otra línea

f(i,j) | 0 , si i = 0
| min(f(i,1), f(i,2)) , j = 0
| min($a_{1,i}$ + f(i-1,1), $a_{1,i}$ + $t_{1,i}$ + f(i-1,2)) , si i > 0 & j = 1
| min($a_{2,i}$ + f(i-1,2), $a_{2,i}$ + $t_{2,i}$ + f(i-1,1)) , si i > 0 & j = 2

Voraces, backtracking y dinámica (parcial 2 2024):

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
- N disponibles
- Cada sabor i tiene un puntaje p_i
- No se puede repetir sabores
- Cada sabor i tiene un booleano a_i que indica si es al agua o no
- Encontrar el mayor puntaje obtenible eligiendo K gustos distintos de helado tq M gustos sean al agua

a) Seleccionar el máximo puntaje de manera tq los primeros M sean al agua.

b)
```
type Sabor = tuple
                 id: nat
                 p: nat
                 a: bool
             end tuple
```

c) El algoritmo toma un conjunto de helados el cual copia y va seleccionando aquellos helados con el puntaje máximo, primero selecciona los helados al agua y luego puede seleccionar cualquiera de los dos tipos.

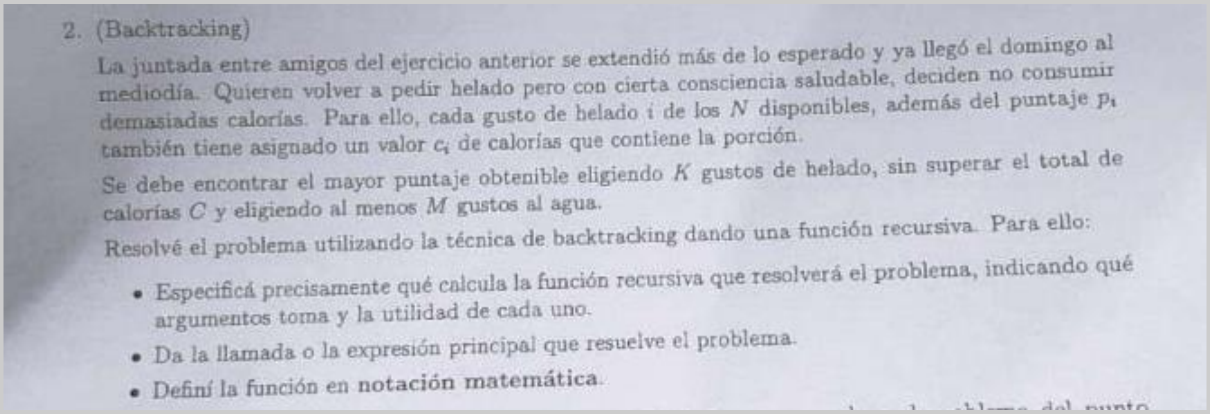
```
fun helado(H: Set of Sabor, K: nat, M: nat) ret puntaje: nat
     var h_aux: Set of Sabor
     var sabor: Sabor
     var k_aux: nat
     var m_aux: nat

     k_aux := K
     m_aux := M
     h_aux := copy_set(H)
     puntaje := 0
     while( not is_empty_set(h_aux) and k_aux != 0) then
```

```
        if ( m_aux != 0 ) then
            sabor := seleccionar(h_aux, true)
            m_aux := m_aux - 1
        else
            sabor := seleccionar(h_aux, false)
        fi
        elim(h_aux, sabor)
        puntaje := puntaje + sabor.p
        k_aux := k_aux - 1
    od
    destroy_set(h_aux)
end fun

fun seleccionar(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.a == true and sabor.p > max_aux) then
                max_aux := sabor.p
                res := sabor
            fi
        else
            if (sabor.p > max_aux) then
                max_aux := sabor.p
                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:

- Además de p_i cada sabor i de los N helados tiene un c_i cantidad de calorías
- Obtener mayor puntaje obtenible eligiendo K gustos sin superar el total de C calorias y al menos M son al agua

Llamada recursiva:

helado(i, k, m, j) “máximo puntaje obtenible seleccionando k helados entre {1,2,..,i} tq almenos m sean al agua sin superar c calorias”

Llamada principal:

helado(N,K,M,C)

Función matemática:

casos:

No tengo mas helados para elegir i = 0 and k > 0 (imposible)
Ya elegí todos los helados pero me faltaron helados al agua (imposible)
Ya elegí todos los helados i > 0 v i = 0 & k = 0 & m = 0 (caso base)
Tengo para elegir helados al agua
Ya no tengo para elegir helados al agua (sigo eligiendo a la crema)

helado(i,k,m,j)

| -inf , si i = 0 & k > 0
| -inf , si k = 0 & m > 0
| 0 , si k = 0 & m = 0
| helado(i-1,k,m,j) , si i > 0 & k > 0 & $c_i > j$
| max(helado(i-1,k,m,j), p_i + helado(i-1,k-1,max(m-1,0),j - c_i))
, si i > 0 & k > 0 & j ≥ c_i & a_i
| max(helado(i-1,k,m,j), p_i + helado(i-1,k-1,m, j - c_i))
, si i > 0 & k > 0 & j ≥ 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 helado(p: array[1..n] of nat, a: array[1..n] of bool, c:
array[1..n] of nat, C: nat, K: nat, M: nat) ret puntaje: nat
{- variables -}
var dp: array[0..n, 0..K, 0..M, 0..C] of nat

{- casos base -}
for k := 1 to K do
  for m := 0 to M do
    for c’ := 0 to C do
      dp[0,k,m,c’] := -inf
    od
  od
od

for i := 0 to n do
  for m := 1 to M do
    for c’ := 0 to C do
      dp[i,0,m,c’] := -inf
    od
  od
od

for i := 0 to n do
  for c’ := 0 to C do
    dp[i,0,0,c’] := 0
  od
od

{- caso recursivo -}
for i := 1 to n do
  for k := 1 to K do
    for m := 1 to M do
      for c’ := 0 to C do
        if c[i] > c’ then
          dp[i,k,m,c’] := dp[i-1,k,m,c’]
        else if c[i] ≤ c’ and a[i] then
          dp[i,k,m,c’] := max(dp[i-1,k,m,c’], p[i] + dp[i-1,k-1,max(m-
1,0),c’ - c_i])
```

```
      else if c[i] ≤ c' and not a[i] then
        dp[i,k,m,c'] := max(dp[i-1,k,m,c'], p[i] + dp[i-1,k-1,m,c' - cᵢ])
      fi
    od
   od
  od
od
  puntaje := dp[N,K,M,C]
end fun
```