Notas de Consulta Online Parte 3

...

Algoritmos y Estructuras de Datos II

Examen Final de Práctica (24 julio 2019). Ejercicio 4

4. En una localidad cordobesa, vive el José Agustín Goytisolo quien, apenado por los padecimientos de la mayoría de los vecinos, decide comprometerse en solucionarlos postulándose a la intendencia. Gracias a su entusiasmo y creatividad, en pocos minutos enumera una larga lista de N propuestas para realizar. Pronto descubre que a pesar de que cada una de ellas generaría una satisfacción popular $p_1, p_2, \ldots, p_N$ también provocaría desagrado $q_1, q_2, \ldots, q_N$ en el sector más acomodado de la sociedad local. En principio, el desagrado de cada propuesta es insignificante en número de votos ya que la alta sociedad no es muy numerosa. Pero a José le interesa cuidar su relación con este sector, ya que el mismo tiene suficientes recursos como para dificultar su triunfo en caso de proponérselo.

   Pronto descubre que las propuestas elaboradas son demasiadas para ser publicitadas: tantas propuestas (N) generarían confusión en el electorado. Esto lo lleva a convencerse de seleccionar solamente K de esas N propuestas ($K \leq N$). Se dispone, entonces, a seleccionar K de esas N propuestas de forma tal que la suma de satisfacción popular de las K propuestas elegidas sea máxima y que el descontento total de esas K propuestas en la alta sociedad no supere un cierto valor M.

   José Agustín te contrata para que desarrolles un algoritmo capaz de calcular el máximo de satisfacción popular alcanzable con K de esas N propuestas sin que el descontento supere M.

Examen Final de Práctica (24 julio 2019). Ejercicio 4

- N propuestas de gobierno.
- Cada propuesta i, genera satisfacción p_i, y desagrado q_i.
- Debe elegir K propuestas, entre las N.
- Se deben seleccionar K propuestas con satisfacción máxima, sin que el desagrado total supere un monto M.

Ejemplo. N = 4. K = 2. M = 45.
p_1 = 30, q_1 = 24
p_2 = 20, q_2 = 10
p_3 = 50, q_3 = 32
p_4 = 25, q_4 = 18

Es backtracking, así que debo “probar” con todos los subconjuntos de K propuestas cuya suma total de desagrado no supere M.

Una opción serán las propuestas 1 y 2, cuya popularidad total será de 50 y el desagrado de 34.
Las propuestas 1 y 3 no pueden ser elegidas juntas, pues su desagrado supera M.
Otra opción válida son las propuestas 1 y 4, cuya popularidad es 55, y el desagrado 42.

IMPORTANTE: No pide que minimicemos el desagrado. Solo debemos asegurar que no supera M.
Pide que maximicemos la suma total de satisfacción, siempre y cuando el desagrado no supere M.

Examen Final de Práctica (24 julio 2019). Ejercicio 4

Voy a definir una función recursiva

propuestas(i,k,m) = *"La mayor satisfacción popular obtenible al elegir* k *propuestas de entre las propuestas* 1 *hasta la* i, *sin que la suma de sus desagrados supere el monto* m*"*

¿Cuál es la llamada o expresión con esta función que me va a obtener la solución que me piden en el enunciado?
propuestas(N,K,M)

Definamos la función "propuestas":
propuestas(i,k,m) =

- Si i = 0, k = 0 ----> 0
- **i = 0, k > 0** **----> - ∞**
- i > 0, k = 0 ----> 0 {- se junta con la 1ra: **i >= 0, k = 0** -}
- i > 0, k > 0, q_i > m ----> propuestas(i-1,k,m)
- i > 0, k > 0, q_i <= m ----> max (p_i + propuestas(i-1, k-1, m-q_i) , propuestas(i-1,k,m))

EJERCICIO: Pasar a Programación Dinámica

Examen Final de Práctica (24 julio 2019). Ejercicio 4

```
fun propuestas(p : array[1..N], q : array[1..N], K, M : nat) ret r : nat
  var prop : array[0..N,0..K,0..M] of nat
  for i := 0 to N do
    for m := 0 to M do
      prop[i,0,m] := 0
    od
  od
  for k := 1 to K do
    for m := 0 to M do
      prop[0,k,m] := -inf
    od
  od

  { SIGUE ----> }
```

Examen Final de Práctica (24 julio 2019). Ejercicio 4

```
{ ←- CONTINUACIÓN }

  for i := 1 to N do
    for k := 1 to K do
      for m := 0 to M do
        if q[i] > m then
          prop[i,k,m] := prop[i-1,k,m]
        else
          prop[i,k,m] := max(p[i] + prop[i-1,k-1,m-q[i]], prop[i-1,k,m])
        fi
      od
    od
  od
  r := prop[N,K,M]
end fun
```

Final 9/12/2020 - Ejercicio 1 - Tema 1

1. (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.

- n oficinas
- 3 colores con costo por oficina C1, C2 y C3.
- pintarlas sin usar el mismo color para dos oficinas contiguas
- preferencias de los oficinistas p^i_1, p^i_2, p^i_3.

UNA SOLUCIÓN POSIBLE:

Función: pintar(i, j) = “valor máximo posible de pintar las oficinas 1,2,..., i sin usar el color j para la oficina i (si j = -1 se puede usar cualquier color).”

Llamada principal: pintar(n, -1)

Definición recursiva:

pintar(i, j) = (0 si i = 0
| max(p^i_2/C_2 + pintar(i-1, 2), p^i_3/C_3 + pintar(i-1, 3)) si i > 0, j = 1
| max(p^i_1/C_1 + pintar(i-1, 1), p^i_3/C_3 + pintar(i-1, 3)) si i > 0, j = 2
| max(p^i_1/C_1 + pintar(i-1, 1), p^i_2/C_2 + pintar(i-1, 2)) si i > 0, j = 3
| max(p^i_1/C_1 + pintar(i-1, 1), p^i_2/C_2 + pintar(i-1, 2),
p^i_3/C_3 + pintar(i-1, 3)) si i > 0, j = -1

)

OTRA SOLUCIÓN POSIBLE:

Función: pintar(i, j) = “valor máximo posible de pintar las oficinas 1,2,..., i usando el color j para la oficina i.”

Llamada principal: max(pintar(n, 1), pintar(n, 2), pintar(n, 3))

Definición recursiva:

pintar(i, j) = (0 si i = 0
| p^i_j / C_j + max(pintar(i-1, 2), pintar(i-1, 3)) si i > 0, j = 1
| p^i_j / C_j + max(pintar(i-1, 1), pintar(i-1, 3)) si i > 0, j = 2
| p^i_j / C_j + max(pintar(i-1, 1), pintar(i-1, 2)) si i > 0, j = 3
)

Final 8/2/2021 - Ejercicio de los Dominós

1. (Backtracking) Debemos llenar con dominós una fila de n casilleros con n par. Cada ficha ocupa 2 casilleros. Contamos con infinitas fichas de todos los tipos. Cada ficha tiene dos números i,j (de 0 a 6) y tiene un puntaje $P_{i,j}$ $(= P_{j,i})$. Como siempre en el dominó, al poner dos fichas juntas los números de los casilleros adyacentes deben coincidir. El último número de la última ficha debe ser un 6. Además, cada casillero tiene un número prohibido $c_1, \ldots, c_n$. Al colocar una ficha en dos casilleros, ésta debe respetar los números prohibidos.

   Escribir un algoritmo que utilice la técnica de backtracking para obtener el máximo puntaje posible colocando las fichas de manera que se repeten todas las restricciones. 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.

   Ejemplo: Con n = 6 debemos usar tres fichas. Si los números prohibidos para los casilleros son

3	2	6	5	0	5

   una posible solución es:

4	3	3	1	1	6

Tenemos:

- Fichas de dominó con puntaje P_i,j
- n casilleros con n par
- el número derecho de cada ficha debe corresponderse con el número izquierdo de la que sigue
- en la i-ésima casilla está prohibido el número c_i
- en la última casilla debe ir un 6

¿Cuáles serían los parámetros de la función recursiva?

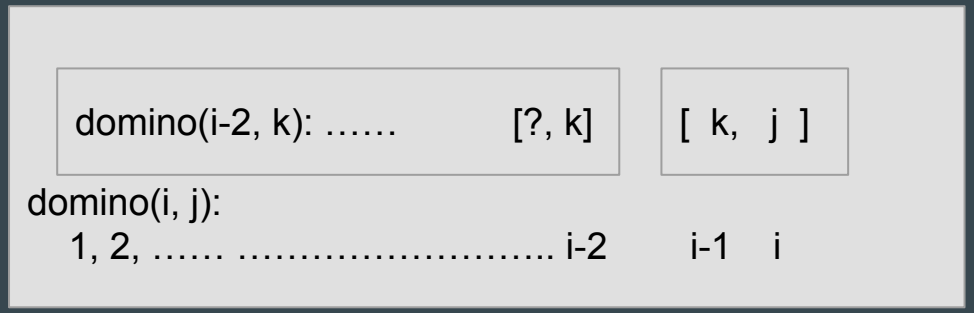
- i: para calcular solución parcial de 1 hasta i (puede ser contando casilleros o dominós). lo haremos contado casilleros.
  - otra opción que no tomamos: solución parcial de i hasta n.
- j: para fijar el valor que debe tener la casilla derecha del último dominó.

Explicación: domino(i, j) = “máximo puntaje posible poniendo dominós en las casillas 1..i respetando los números prohibidos y poniendo como último valor del último dominó j.”

Llamada principal: domino(n, 6)

Errores típicos: usar n como parámetro de la función (y en la explicación), y usar variables en la llamada principal (en lugar de las constantes que me da el enunciado).

Definición recursiva



domino(i, j) =

(si i=0 ---> 0

| si i > 0 ---> maximo_{k = 0...6, k != c_i-1 y k != c_i-2} domino(i-2, k) + P_k,j

)

(definimos artificialmente c_0 = -1).

Otra versión:

domino(i,j) =

(si i=0 ---> 0

| si j = c_i ---> -infinito

| si i > 0 ---> maximo_{k = 0...6, k != c_i-1} domino(i-2, k) + P_k,j

)

Final 8/2/2021 - Ejercicio de Anacleta

1. (Voraz) Anacleta reparte pedidos en bicicleta. Todos los clientes viven en la Avenida San Wachín y la dirección del trabajo de Anacleta es Avenida San Wachín 0. Al llegar a trabajar, Anacleta tiene cada pedido con la altura de la casa donde debe entregarlo. En la bici, Anacleta puede transportar cinco pedidos o menos. Escriba un algoritmo que reciba una lista de naturales llamada pedidos y determine la menor distancia que debrá pedalear Anacleta para entregar todos los pedidos.

Tenemos:

- Una lista de naturales de pedidos. Cada pedido es la dirección del lugar (= a la distancia al negocio).

Calcular:

- Menor distancia que se debe pedalear para entregar todos los pedidos.

Criterio para el algoritmo voraz: entregar primero los de más lejos.

```
fun anacleta(pedidos: array[1..N] of nat) ret r : nat
  var pedidos2: array[1..N] of nat
  var i : nat
  copy_array(pedidos, pedidos2)
  reverse_sort(pedidos2)          {- ordenar de manera decreciente -}
  r := 0
  i := 1
  do i <= N →
    {- entregar los pedidos i hasta min(i + 4, N) -}
    r := r + pedidos2[i] * 2
    i := i + 5
  od
end fun
```

Final febrero 2: Problema de las viandas

1. (Algoritmos voraces) Te vas n días de vacaciones al medio de la montaña, lejos de toda civilización. Llevás con vos lo imprescindible: una carpa, ropa, una linterna, un buen libro y comida preparada para m raciones diarias, con $m > n$. Cada ración i tiene una fecha de vencimiento v_i, contada en días desde el momento en que llegás a la montaña. Por ejemplo, una vianda con fecha de vencimiento 4, significa que se puede comer hasta el día número 4 de vacaciones inclusive. Luego ya está fuera de estado y no puede comerse.

   Tenés que encontrar la mejor manera de organizar las viandas diarias, de manera que la cantidad que se vencen sin ser comidas sea mínima. Deberás indicar para cada día j, $1 \leq j \leq n$, qué vianda es la que comerás, asegurando que nunca comas algo vencido.

- n días de vacaciones
- m raciones diarias (m > n)
- la ración i-ésima vence el día v_i.
- organizar las viandas diarias minimizando las viandas que se vencen.
- **a) Criterio: seleccionar primero las viandas que vencen más pronto (siempre que no hayan vencido ya).**

Final febrero 2: Problema de las viandas

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

- input: un arreglo de tamaño m con los vencimientos v_i de las viandas.
- output: un arreglo de tamaño n con el índice de la vianda que toca ese día.

c) Explicar en palabras cómo resolverá el problema el algoritmo:

- armar un arreglo de índices y ordenarlo de menor a mayor de acuerdo al vencimiento.
- **recuerdo en una variable i el índice de la última vianda que consideré**
- para cada día j, me fijo si la vianda actual se puede comer (o sea, si no venció), si venció, voy a la siguiente, hasta encontrar una que se pueda comer.
- cuando encuentro una que se pueda comer, selecciono esa para el día j y actualizo el valor de i.

Ejemplo: venc = [10, 6, 7, 5]
indices: 1 2 3 4

venc_aux = [4,2,3,1]

¿cuál es en venc el vencimiento más próximo? venc[venc_aux[1]] = 5.

¿el segundo? venc[venc_aux[2]] = 6

Final febrero 2: Problema de las viandas

Versión que asume existencia de la solución:

```
fun viandas(venc : array[1..m] of nat, n : nat) ret r : array[1..n] of nat
  var venc_aux : array[1..m] of nat
  var i : nat;
  index_sort(venc, venc_aux)
  {- devuelve en venc_aux los índices del arreglo venc ordenados de acuerdo a los
     valores de venc de menor a mayor -}
  i := 1
  for j := 1 to n do
    {- elegir ración para el día j -}
    while venc[venc_aux[i]] < j do
      i := i + 1
    od
    {- el while terminó: venc[venc_aux[i]] >= j -}
    r[j] := venc_aux[i]
    i := i + 1
end fun
```

Versión que NO asume existencia de la solución:

```
fun viandas(venc : array[1..m] of nat, n : nat) ret r : array[1..n] of nat
  var venc_aux : array[1..m] of nat
  var i : nat;
  index_sort(venc, venc_aux)
  {- devuelve en venc_aux los índices del arreglo venc ordenados de acuerdo a los
     valores de venc de menor a mayor -}
  i := 1
  for j := 1 to n do
    {- elegir ración para el día j -}
    while i <= m && venc[venc_aux[i]] < j do
      i := i + 1
    od
    {- el while terminó: i > m || venc[venc_aux[i]] >= j -}
    if i > m then
      r[j] := -1
    else
      r[j] := venc_aux[i]
      i := i + 1

    fi
end fun
```

Final 23 de febrero: CDs

(Backtracking) Te vas de viaje a la montaña viajando en auto k horas hasta la base de un cerro, donde luego caminarás hasta el destino de tus vacaciones. Tu auto no es muy nuevo, y tiene un stereo que solo reproduce cds (compact-disks). Buscás en tu vasta colección que compraste en los años 90 y tenés p cds, con $p > k$, que duran exactamente una hora cada uno. Encontrás también un cuaderno donde le diste una puntuación entre 1 y 10 a cada cd de tu colección. Cuanto mayor la puntuación, más es el placer que te da escucharlo. Dado que no sos tan exigente, querés que el puntaje promedio entre dos discos consecutivos que escuches, no sea menor a 6. Así por ejemplo si en la hora 2 escuchás un cd que tiene puntaje 8, en la hora 3 podrías escuchar uno que tenga puntaje al menos 4.

Encontrar una combinación de cds para escuchar en las k horas de viaje, cumpliendo la restricción de que en dos horas consecutivas el puntaje promedio de los dos discos sea mayor o igual a 6, maximizando el puntaje total de los k discos que escucharás.

- viaje de k horas
- p CDs (p > k) con puntuaciones s_1, ..., s_p (del 1 al 10).
- dos discos seguidos deben puntuar en promedio más de 6
- seleccionar los discos a escuchar maximizando el puntaje total.

Final 23 de febrero: CDs

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

   cds(i, J, a) = “máxima puntuación al escuchar en las horas 1...i i CDs seleccionados del conjunto J, con puntaje promedio >= 6 entre dos CDs consecutivos de 1..i-1, salvo el último (el i-ésimo) que en promedio con a debe ser >= 6.”

   Importante: fijarse que no se usan los parámetros del problema (k, p, s_i).

b) **Llamada principal:** max_{1 ≤ a ≤ 10} cds(k, {1, 2, ..., p}, a)

   Otra opción: cds(k, {1, 2, ..., p}, 1000) (o cualquier valor mayor que 11)

Final 23 de febrero: CDs

c)

$$
\begin{aligned}
cds(i, J, a) = (\ & \text{si } i = 0 \rightarrow 0 \\
& \mid \text{si } i > 0 \rightarrow \\
& \quad \{\text{- acá debo elegir el i-ésimo CD sabiendo q la nota del i+1-ésimo es a -}\} \\
& \quad \max_{j \in J \mid (s_j + a)/2 \geq 6} cds(i - 1, J / \{ j \}, s_j) + s_j \\
&)
\end{aligned}
$$