Recuperatorios de las entregas prácticas
Fecha límite: lunes 21 de diciembre

Algoritmos y Estructuras de Datos II, DC, UBA.

Segundo cuatrimestre de 2020

1. Ejercicio X1 — Funciones recursivas

La muchachada del DC quiere preparar el almuerzo de fin de año de forma original. Básicamente cada quien trae lo que quiere, y luego se utiliza un libro de recetas para ver que se puede hacer.
Entonces, dadas ciertas cantidades de cada ingrediente, con el libro de recetas podemos formar distintas combinaciones de comidas posibles. Pero como el que propone la receta tiene coronita, las recetas tienen un orden y si es posible, hay que respetar ese orden.
En definitiva queremos una operación *combinaciones posibles* que dados varios ingredientes y sus cantidades, junto a un recetario, nos de las comidas que se pueden preparar respetando el orden en el que aparecen en el recetario.
Sean los TADs Ingrediente y Receta renombres de String, defina la siguiente función que devuelve un multiconjunto con todas las recetas que se pueden preparar:

- CombinacionesPosibles : dicc(receta, dicc(ingrediente, nat)) $\times$ dicc(ingrediente, nat) $\rightarrow$ multiconj(receta)

Ejemplo ilustrativo:

- Recetas:
  - Pizza: 1 harina, 4 tomates, 2 quesos
  - Ensalada caprese: 2 tomates, 1 queso, 1 albahaca
- Ingredientes con los que contamos: 8 tomates, 4 quesos, 1 harina, 3 albahacas
- La función debería devolver un multiconjunto con 1 pizza y dos ensaladas caprese:
  - { Pizza, Ensalada caprese, Ensalada caprese }

TIP: pueden asumir que la operación claves del diccionario de recetas les devuelve las recetas en el orden de prioridad correspondiente

2. Ejercicio X2 — TADs

En la playa de Mar del Plata venden empanadas y queremos llevar un registro de sus movimientos.
Inicialmente contamos con un grupo de repartidores de empanadas, que se juntan en la confitería productora de las mismas. Cuando tiene ganas, algún repartidor se lleva varias empanadas consigo (la cantidad de quiera). A medida que pasa el tiempo los grupos de personas que están frente al mar llaman a algún repartidor y le piden la cantidad de empanadas que quieran, claramente no le pueden pedir más empanadas de las que tiene el repartidor en ese momento. Cuando esté cansado el repartidor vuelve a la base inmediatamente a descansar (si le sobran empanadas se las devuelve a la confitería). Si se le acaban las empanadas, el repartidor también debe volver y descansa hasta que le toque volver a salir.

Nos interesan saber varias cosas, primero cuál fue la compra más numerosa, es decir, cuál fue la máxima cantidad de empanadas vendidas en una sola compra y por cual vendedor (en caso de empate devuelvan todas las que correspondan). Por otro lado nos interesa saber cuales fueron los vendedores que más veces vendieron todas sus empanadas.

Modelar con el TAD EMP el sistema descripto. La especificación debe estar completa, incluyendo observadores, generadores, otras operaciones, restricciones y la axiomatización correspondiente. Se sugiere escribir la igualdad observacional, pero no es de caracter obligatorio.

3. Ejercicio X3 — Cálculo de complejidad

El siguiente algoritmo llamado "`CametSort`" ordena un arreglo A.

```
función CametSort(in/out A: arreglo(nat))
    inicio ← 0
    fin ← longitud(A) − 2
    mientras inicio ≤ fin hacer
        x ← fin
        y ← inicio
        para i de inicio a fin, sumando 1 hacer
            si A[i] > A[i + 1] entonces
                Intercambiar A[i] y A[i + 1]
                y ← i
            fin
        fin
        fin ← y − 1
        para i de fin a inicio, restando 1 hacer
            si A[i] > A[i + 1] entonces
                Intercambiar A[i] y A[i + 1]
                x ← i
            fin
        fin
        inicio ← x + 1
    fin
    devolver A
fin
```

Se pide:

1. Identificar mejor caso, justificar por qué lo es, y expresar su complejidad temporal en función de longitud(A) usando la notación Θ.
2. Identificar peor caso, justificar por qué lo es, y expresar su complejidad temporal en función de longitud(A) usando la notación Θ.

Pueden suponer que intercambiar dos elementos de un arreglo es una operación de tiempo constante.

Pista: pensar en los valores de *inicio* y fin al terminar cada iteración.

4. Ejercicio X4 — Notación "O"

Sean $a \in \mathbb{R}$ y $b \in \mathbb{Z}^+$ constantes, $f, g \colon \mathbb{N} \to \mathbb{R}^+$, y $h(n) = q(n) + 1$, donde $q(n)$ es la cantidad de veces que 2 divide exactamente a n (por ejemplo, $q(1) = 0$, $q(2) = 1$, $q(8) = 3$, $q(10) = 1$).
Decidir si las siguientes afirmaciones son verdaderas o falsas. Justificar detalladamente.

1. $f(n) \in \Omega(f(n)^2) \Rightarrow f(n) \in O(1)$
2. $f(n) \in \Omega(g(n)) \Rightarrow f(n)^2 \in \Omega(g(n))$
3. $n! \in \Theta((n+1)!)$
4. $(n+a)^b \in \Theta(n^b)$.
5. $nf(m) + mg(n) \in O(f(nm) + g(nm))$
6. $h(n) \in O(n)$, y es la cota más justa posible.
7. $\log_2 n \in O(h(n))$
8. Hay una cantidad infinita de funciones $i(n)$, tal que $i(n) \in O(h(n))$

5. Ejercicio X5 — Rep y Abs

En una playa de la costa argentina se decidió particionar el espacio en *ubicaciones* o parcelas donde pueden ir los grupos de personas convivientes.
Cuando una persona llega elige una parcela libre o una en la que haya alguna persona conviviente. No puede suceder que una persona llegue y elija una parcela donde hay una persona no conviviente. Cuando cada visitante quiera puede irse.

Si bien en el párrafo anterior fue comentada la especificación inicial, en la etapa de implementación además se quiso agregar la zona de cada ubicación delimitada por el rectángulo que se genera eligiendo 2 extremos del mismo en el espacio de la playa (por ejemplo las coordenada (1,2) y (2,4) delimitan el rectánculo [(1,2),(1,4),(2,2),(2,4)]). Importante notar entonces que las personas se ubican solo en coordenadas dentro de alguna parcela válida, y las parcelas no se pueden superponer.

TAD PLAYA

observadores básicos

personas : playa $\longrightarrow$ conj(persona)
ubicaciones : playa $\longrightarrow$ conj(identificador)
enUbicación : playa p × identificador i $\longrightarrow$ conj(persona) $\{i \in \text{ubicaciones}(p)\}$
convivientes : playa p $\longrightarrow$ conj(conj(persona))

generadores

iniciar : conj(identificador) × conj(conj(persona)) c $\longrightarrow$ playa $\{\forall d, e \in c \mid d \neq e \Rightarrow d \cap e = \emptyset\}$
llegar : playa p × persona q × identificador i $\longrightarrow$ playa
$\{q \in \text{personas}(p) \wedge i \in \text{ubicaciones}(p) \wedge q \notin \text{enUbicación}(p,i) \wedge (\forall r \in \text{enUbicación}(p,i).\text{conviviente}(p,q,r))\}$
irse : playa p × persona q $\longrightarrow$ playa $\{q \in \text{personas}(p) \wedge (\exists i \in \text{ubicaciones}(p) \mid q \in \text{enUbicación}(p,i))\}$

otras operaciones

conviviente : playa p × persona $q1$ × persona $q2$ $\longrightarrow$ bool $\{q1 \neq q2 \wedge q1, q2 \in \text{personas}(p)\}$

axiomas

...

Fin TAD

La estructura de representación elegida es la siguiente:

PLAYA **se representa con** estr

donde estr es tupla ⟨ *ubicaciones*: dicc(identificador, ⟨A: coordenada, B: coordenada⟩),
personas: dicc(persona, coordenada),
convivientes: secu(conj(persona))⟩

En esta estructura:

- *ubicaciones* contiene los identificadores de ubicaciones, pero en la etapa de implementación se decidió agregar las coordenadas que delimitan el rectángulo de la misma.
- *personas* contiene las personas que están actualmente en la playa y su coordenada.
- *convivientes* indica los grupos de personas convivientes, que pueden estar dentro de la misma ubicación.

Teniendo en cuenta lo descripto arriba se pide:

a) Escribir en castellano el invariante de representación.
b) Escribir formalmente el invariante de representación.
c) Escribir formalmente la función de abstracción.

Se puede asumir que existen funciones útiles para manipular coordenadas, del estilo:

- hayIntersección(coordenada a, coordenada b, coordenada x, coordenada y) que devuelve verdadero si las zonas delimitantes se superponen.
- pertenece(coordenada a, coordenada b, coordenada p) que devuelve verdadero si p se encuentra dentro de la zona delimitada por a y b.

6. Ejercicio X6 — Elección de estructuras

En la playa Varesse hay muchos autos y un estacionamiento que tiene que ser organizado muy prolijamente para que entren y salgan autos sin estrellarse. Podemos pensar el estacionamiento como una gran fila de espacios disponibles, cuando un auto llega va hasta la parte de la playa que le guste y ahí pone el auto-stop y busca la siguiente ubicación libre para ocuparla. Se transcribe un fragmento de la especificación:

TAD ESTACIONAMIENTO

observadores básicos

#lugares : EST $\longrightarrow$ nat	
vehículos : EST e $\longrightarrow$ conj(vehículo)	
ubicación : EST e $\times$ vehículo v $\longrightarrow$ nat	$\{v \in \text{vehículos}(e)\}$

generadores

iniciar : nat u $\longrightarrow$ EST	$\{u > 0\}$
ocupar : EST e $\times$ vehículo v $\times$ nat n $\longrightarrow$ EST	$\{ n < \#\text{lugares}(e)\}$
desocupar : EST e $\times$ vehículo v $\longrightarrow$ EST	$\{ v \in \text{vehículos(e)}\}$

otras operaciones

hayLibre : EST e $\times$ nat n $\longrightarrow$ bool	$\{n < \#\text{lugares}(e)\}$
siguienteLibre : EST e $\times$ nat n $\longrightarrow$ nat	$\{n < \#\text{lugares}(e) \wedge \text{hayLibre(e,n)}\}$

Fin TAD

Diseñar el módulo ESTACIONAMIENTO, de tal modo que provea las siguientes operaciones con las complejidades temporales en peor caso indicadas. Escribir la estructura y detallar los algoritmos de las cuatro operaciones siguientes teniendo en cuenta que u es la cantidad de lugares libres y n es la cantidad de vehículos que se encuentran en el estacionamiento.

1. SIGUIENTELIBRE(**in** e: estacionamiento, **in** p: nat, **out** q: nat) $\longrightarrow$ *res* : bool
   Devuelve si hay lugar libre a partir de la posición p, y de haberlo, escribe dicho lugar en q.
   Complejidad: $O(\log(u))$

2. ESTÁLIBRE(**in** e: estacionamiento, **in** p: nat) $\longrightarrow$ *res* : bool
   Devuelve si en particular la ubicación p está libre.
   Complejidad: $O(1)$

3. OCUPAR(**in/out** e: estacionamiento, **in** v: vehículo, **in** p: nat)
   Ocupa el lugar p con el vehículo v, como precondición tendríamos que p es un lugar válido. Complejidad: $O(\log(n) + \log(u))$

4. UBICACIÓN(**in** e: estacionamiento, **in** v: vehículo) $\longrightarrow$ *res* : nat
   Devuelve la ubicación en el estacionamiento para el vehículo v.
   Complejidad: $O(\log(n))$

Se pide:

1. Dar una estructura de representación del módulo ESTACIONAMIENTO explicando detalladamente qué información se guarda en cada parte, las relaciones entre las partes, y las estructuras de datos subyacentes.

2. Justificar detalladamente de qué manera es posible implementar los algoritmos para cumplir con las complejidades pedidas. Escribir el algoritmo para la operación OCUPAR.

Para la resolución del ejercicio no esta permitido utilizar módulos implementados sobre tablas de *hash* como estructura de representación. Esto responde a dos razones: en primer lugar, el objetivo del ejercicio es que puedan combinar otras estructuras que estudiamos en la materia; en segundo lugar, considerar que los costos de inserción, búsqueda y borrado en un diccionario o conjunto implementado sobre una tabla de *hash* no son constantes en peor caso, y por lo tanto seguramente excedan los costos necesarios para resolver los ejercicios.

7. Ejercicio X7 — Ordenamiento

En la famosa playa La Perla quieren sacar una foto área artística donde todas las sombrillas estén ordenadas formando un cuadro de colores único. Cada sombrilla podríamos considerarla como una tupla de ⟨Color, Diámetro⟩ donde un Color es un código hexadecimal de u dígitos y Diámetro un natural positivo. Como las sombrillas tienen tamaños estándar, nos indican que hay a lo sumo k posibles tamaños de sombrillas (pero no se aclara a priori cuáles k). La muchachada nos solicita un algoritmo que ordene las sombrillas por color creciente y en caso de empate por tamaño decreciente. En concreto, diseñar el siguiente algoritmo:

`SombrillasSort`(**in** A : arreglo(Sombrilla), **in** B : arreglo(Diámetro)) $\longrightarrow$ C : arreglo(Sombrilla)

con **complejidad** $O(n \cdot u + (k + n) \log k)$ **en peor caso**, que genere un arreglo con todas las sombrillas ordenadas como se mencionó previamente. El arreglo A de tamaño n contiene las sombrillas y el B de tamaño k los diámetros posibles. Por ejemplo:

```
SombrillasSort([<"0x6A319",21>, <"0x6A319",27>, <"0xA1719",11>, <"0xA1720",21>], [27,11,21])
==> [<"0x6A319", 27>, <"0x6A319", 21>, <"0xA1719", 11>, <"0xA1720", 21>]
```

En este ejemplo: $u = 5$ y $k = 3$.

8. Ejercicio X8 — Dividir y Conquistar

Se tiene un arreglo de palabras de longitud acotada por una constante y se desea saber cuántas veces es posible leer el nombre de la ciudad `mar del plata` de izquierda a derecha en este arreglo, esto es, alguna manera de encontrar `mar`, en alguna posición posterior `del` y en alguna posición posterior a ésta `plata`, no necesariamente consecutivas. Si hiciera falta puede asumirse que n es una potencia de algún natural mayor que 1.

Ejemplos: en [`del`, `mar`, `del`, `del`, `mar`, `plata`, `tuyú`] se puede leer 2 veces, mientras que en [`mar`, `del`, `playa`, `mar`, `plata`, `mar`, `tuyú`, `mar`, `del`, `arcoiris`, `mar`, `plata`] se puede leer 6 veces.

Usando la técnica de Dividir y Conquistar, escribir un algoritmo que, dado un arreglo de n palabras cualesquiera, devuelva esta cantidad en tiempo estrictamente mejor que $O(n^2)$ (preferentemente $O(n)$).
Se pide el algoritmo en pseudocódigo, explicar con palabras las ideas volcadas en el algoritmo y justificar su complejidad temporal.