# Ejercitación 2do parcial

Algoritmos y Estructuras de Datos 2

Departamento de Computación,
Facultad de Ciencias Exactas y Naturales,
Universidad de Buenos Aires

10 de noviembre de 2017

# Diseño - Recuperatorio 1c 2017

Estamos desarrollando una agenda que registra tareas a realizar en determinados días del año. Estas tareas pueden agregarse en cualquier momento y se registran para ser realizadas cada año a partir de este momento, en el día agendado (por simplicidad tomamos que un día es un nat entre 1 y 366). La agenda mantiene también el día actual y se le puede preguntar qué tareas hay pendientes para hoy. A lo largo del día, cada una de estas tareas tendrá que darse por cumplida o bien ser pospuesta para otro día. Dado que las tareas son anuales, posponer una tarea altera el día agendado definitivamente, no sólo para esta ocasión sino también para años futuros. Es posible decirle a la agenda que se desea pasar al siguiente día, pero para ello no debe haber tareas pendientes, es decir, todas las tareas del día actual deberán haberse cumplido o pospuesto.

**TAD** AGENDA

**observadores básicos**

| | | | | |
|---|---|---|---|---|
| diaActual | : agenda | $\longrightarrow$ | dia | |
| tareas | : agenda | $\longrightarrow$ | conj(tarea) | |
| diaDe | : agenda $a$ $\times$ tarea $t$ | $\longrightarrow$ | dia | $\{t \in \text{tareas}(a)\}$ |
| pendientesDeHoy | : agenda $a$ | $\longrightarrow$ | conj(tarea) | |

**generadores**

| | | | | |
|---|---|---|---|---|
| iniciar | : dia *hoy* | $\longrightarrow$ | agenda | |
| agendar | : agenda $a$ $\times$ tarea $t$ $\times$ dia $d$ | $\longrightarrow$ | agenda | $\{t \notin \text{tareas}(a)\}$ |
| cumplir | : agenda $a$ $\times$ tarea $t$ | $\longrightarrow$ | agenda | $\{t \in \text{pendientesDeHoy}(a)\}$ |
| posponer | : agenda $a$ $\times$ tarea $t$ $\times$ dia $d$ | $\longrightarrow$ | agenda | $\{t \in \text{pendientesDeHoy}(a)\}$ |
| pasarDeDía | : agenda $a$ | $\longrightarrow$ | agenda | $\{\emptyset?(\text{pendientesDeHoy}(a))\}$ |

**otras operaciones**

| | | | |
|---|---|---|---|
| tareasDe | : agenda $\times$ dia | $\longrightarrow$ | conj(tarea) |

...

**axiomas**

...

**Fin TAD**

# Diseño - Recuperatorio 1c 2017

Asumiendo que comparar tareas es $O(1)$, realizar un diseño que cumpla con las siguientes complejidades de peor caso, siendo $n$ la cantidad total de tareas en la agenda y $t$ la mayor cantidad de tareas agendadas para un mismo día:

- Agendar una nueva tarea en $O(\log n)$.
- Dar por cumplida una tarea del día actual en $O(\log t)$.
- Posponer una tarea del día actual en $O(\log t)$.
- Conocer el día agendado para una cierta tarea en $O(\log n)$.
- Conocer las tareas pendientes del día actual en $O(1)$.
- Conocer las tareas de cualquier otro día (no el actual) en $O(1)$.
- Avanzar de día en $O(1)$.

1. Escriba la estructura de representación del módulo "Agenda" explicando detalladamente qué información se guarda en cada parte de la misma y las relaciones entre las partes. Describa también las estructuras de datos subyacentes.
2. Escriba el algoritmo para posponer una tarea y justifique el cumplimiento de los órdenes solicitados. Para cada una de las demás funciones, descríbalas en castellano, justificando por qué se cumple el orden pedido con la estructura elegida.

# Sorting - Recuperatorio 1c 2017

Se tiene una secuencia $A$ de alturas $h_1, ..., h_n$. Un intervalo $h_t, h_{t+1}, ..., h_{t+k}$ es un *edificio* si para todo $i \in [t, t+k)$, el valor $[h_i - h_{i+1}]$ no supera una tolerancia $\theta$ y tanto $[h_t - h_{t-1}]$ como $[h_{t+k} - h_{t+k+1}]$ (si existen) son mayores que $\theta$. El ancho de un edificio es la cantidad de alturas que lo componen.

Por ejemplo:

- Si $A = [100, 101, 100, 103, 80, 77, 74, 200, 32, 30]$ y $\theta = 3$,
- los edificios serían: [100,101,100,103],[80,77,74],[200],[32,30] y sus anchos serían 4, 3, 1 y 2 respectivamente.

Dado un arreglo $A$ de alturas y una tolerancia $\theta$, se desea ordenar $A$ reposicionando los edificios ordenándolos según sus anchos en forma creciente (sin alterar el orden de las alturas dentro de cada edificios). En el ejemplo anterior, el resultado esperado sería: [200,32,30,80,77,74,100,101,100,103]

Escribir un algoritmo que resuelva el problema con complejidad no peor que $O(n)$, donde $n$ es la cantidad de alturas dada. Justifique la correctitud del algoritmo y su complejidad temporal.

# Sorting - Parcial 1c 2017

Se tiene un arreglo $R$ con $n$ strings sin repeticiones que define un *ranking* y un arreglo $A$ de $m$ strings tal que todos ellos aparecen $R$. Se quiere ordenar $A$ en función del ranking dado por $R$. Es decir, dados dos elementos $s$ y $t$ de $A$, $s$ será *"menor"* que $t$, si aparece en $R$ antes que $t$.

Por ejemplo, si tenemos

$R =$ `[Brasil, Argentina, Alemania, Chile, Colombia, Francia]` y

$A =$ `[Chile, Francia, Brasil, Chile, Argentina, Brasil]`,

quedaría $A =$ `[Brasil, Brasil, Argentina, Chile, Chile, Francia]`.

Suponiendo que el largo de todos los strings está acotado por una constante, proponga un algoritmo de ordenamiento que resuelva el problema en una complejidad $O(n + m)$, donde $n$ y $m$ son las cantidades de elementos en el ranking y en el arreglo a ordenar, respectivamente. Justifique la correctitud del algoritmo y su complejidad temporal.

# Divide and conquer

Dados los arreglos *In* y *Pre* correspondientes al resultado de aplicar *inorder* y *preorder* a un árbol binario que no tiene elementos repetidos, se quiere obtener el arreglo correspondiente al *postorder* de dicho árbol.

- Proponer un algoritmo para resolver este problema y calcular su complejidad de peor caso.
- Analizar su complejidad de mejor caso y dar una cota superior para la misma.

# Master theorem (para tener a mano)

Si

$$T(n) = \begin{cases} aT\left(\frac{n}{c}\right) + f(n) & n > 1 \\ \theta(1) & n = 1 \end{cases}$$

Entonces

$$T(n) = \begin{cases} \theta(n^{\log_c a}) & \text{Si } \exists \varepsilon > 0 \text{ tal que } f(n) \in O(n^{\log_c a - \varepsilon}) \\ \theta(n^{\log_c a} \log n) & \text{Si } f(n) \in \theta(n^{\log_c a}) \\ \theta(f(n)) & \text{Si } \exists \varepsilon > 0 \text{ tal que } f(n) \in \Omega(n^{\log_c a + \varepsilon}) \text{ y} \\ & \exists \delta < 1, n_0 \text{ tal que } af\left(\frac{n}{b}\right) \leq \delta f(n), \forall n \geq n_0 \end{cases}$$

# Divide & Conquer - Parcial 2c 2016

Se dice que una matriz A de dimensión $n$ es *pirámide invertida* si, para alguna coordenada (i,j) de la matriz (con $i, j \in [1 \dots n]$) se cumple que:

i) $A_{ij} = 0$;

ii) Toda fila $k$ tiene valores decrecientes hasta la columna $j$ y crecientes desde ésta en adelante;

iii) Toda columna $k$ tiene valores decrecientes hasta la fila $i$ y crecientes desde ésta en adelante.

| i,j | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 20 | 16 | 9 | 5 | 11 |
| 2 | 16 | 7 | 4 | 3 | 6 |
| 3 | 9 | 3 | 1 | **0** | 3 |
| 4 | 11 | 7 | 5 | 4 | 7 |
| 5 | 12 | 10 | 8 | 6 | 9 |

O sea, tiene una celda $(i, j)$ que vale 0 y el resto de las celdas tienen valores estrictamente crecientes a medida que se alejan de la fila $i$ y/o de la columna $j$.

a) Proponga un algoritmo de *Dividir y Conquistar* para encontrar las coordenadas del fondo de una *pirámide invertida*. El algoritmo debe tener complejidad $O(\log n)$.

b) Justifique detalladamente la correctitud del algoritmo y su complejidad temporal, así como qué parte del algoritmo implementa cada una de las diferentes fases de la técnica.