Ayudantía Repaso I1

¡No se olviden que pueden llevar un resumen escrito a mano por ustedes!

Repaso Correcitud

¿Qué necesitamos para que un algoritmo sea considerado correcto?

1. **Termina en una cantidad finita de pasos**
   a. Decimos porque termina el algoritmo
2. **Cumple con su propósito**, en otras palabras hace lo que tiene q
   hacer
   a. Hacemos uso de inducción
   b. Se debe cumplir para **todo input**

Enunciado

4) Demostración de corrección. Considera dos arreglos A y B de n elementos cada uno arreglos están ordenados. Demuestra que el siguiente algoritmo encuentra la mediana de la t de los elementos de A U B en O(log n).

1) Calcula las medianas *m1* y *m2* de los arreglos ar1[] y ar2[] respectivamente.
2) If *m1* = *m2* → terminamos; return *m1* (o *m2*)
3) If *m1* > *m2*, entonces la mediana está en uno de los siguientes subarreglos:
   a) Desde el primer elemento de ar1 hasta *m1*
   b) Desde *m2* hasta el último elemento de ar2
4) If *m2* > *m1*, entonces la mediana está en uno de los siguientes subarreglos:
   a) Desde *m1* hasta el último elemento de ar1
   b) Desde el primer elemento de ar2 hasta *m2*
5) Repite los pasos anteriores hasta que el tamaño de ambos subarreglos sea 2.
6) If tamaño de ambos subarreglos es 2, entonces la mediana es:
   Mediana = (max(ar1[0], ar2[0]) + min(ar1[1], ar2[1]))/2

Repaso Complejidad

¿Cómo encontramos la complejidad?

1. Si diste Discretas, puedes hacer uso de **Teorema Maestro**
2. Si no, te sugerimos ver las **demostraciones de complejidad vistas en clases**
3. **Tip**: Si se usa recursión pensar cuánto tiempo se puede demorar el paso recursivo

Tips Sorting

¿Cómo estudiamos sorting?

1. Vean si pueden ordenar un arreglo con los algoritmos vistos en clase
2. Entender bien los algoritmos

Enunciado

1) Ordenar por dos criterios.

Como parte del proceso de postulación a las universidades chilenas que realiza el DEMRE con un gran volumen datos en que cada registro representa cada una de las postulaciones estudiante a una carrera en una universidad. Estos registros son de la siguiente forma:

RUT_Postulante	codigo_universidad	codigo_carrera	puntaje_ponderado	…

Nota: Asume que codigo_universidad y codigo_carrera son valores numéricos enteros.

Originalmente las postulaciones se encuentran ordenadas por RUT_Postulante, y se requ ordenarlas por codigo_universidad Y codigo_carrera para distribuir esta información por s cada universidad con los postulantes a cada una de sus carreras.

Preguntas

a) Propón un algoritmo para realizar el ordenamiento requerido. Puedes utiliza
arreglos —especifica. Usa una notación similar a la usada en clases.
b) Calcula su complejidad.
c) Determina su mejor y peor caso.

Solución

(a) [3 ptos.] Proponga el pseudocódigo de un algoritmo para ordenar los registros alfabéticamen según (`primer_apellido`, `segundo_apellido`, `nombre`). Especifique qué estructura básica usará o arreglos). Si p es un registro, puede acceder a sus atributos con `p.primer_apellido`, `p.nombr` Además, puede asumir que todo algoritmo de ordenación visto en clase puede ordenar respect atributo específico.

Solución.

input : Arreglo $A[0, \ldots, n-1]$ e índices i, f
output: Lista de pares de índices L

`FirstLastNameReps` (A, i, f)**:**
    $L \leftarrow$ lista vacía
    $k \leftarrow i$
    $j \leftarrow i$
    for $m = 1 \ldots f$ **:**
        if $A[m].\mathit{primer\_apellido} = A[k].\mathit{primer\_apellido}$ **:**
            $j \leftarrow m$
        else:
            if $k < j$ **:**
                añadir a L el par (k, j)
            $k \leftarrow m$
            $j \leftarrow m$
    return L

Solución

Es decir, `FirstLastNameReps` (A, i, f) entrega una lista con los rangos entre los cuales hay repet
de primer apellido entre los índices i y f. De forma similar se define la rutina `SecondLastNa`
que entrega rangos de repetidos de segundo apellido. El algoritmo principal es el siguiente

input : Arreglo $A[0, \ldots, n-1]$
output: Arreglo ordenado alfabéticamente

`AlphaSort` (A)**:**

```
1  MergeSort(A, 0, n − 1, primer_apellido)      ▷ ordenamos A según primer apellido
2  F ← FirstLastNameReps(A, 0, n − 1)
3  for (k, j) ∈ F :
4      MergeSort(A, k, j, segundo_apellido)
5      S ← SecondLastNameReps(A, k, j)
6      for (s, t) ∈ S :
7          MergeSort(A, s, t, nombre)
```

Solución

(b) [2 ptos.] Determine la complejidad de peor caso de su algoritmo en función del número de estud
en el archivo.
Solución.
El peor caso corresponde a una cantidad $\mathcal{O}(n)$ de repetidos en primer y segundo apellido. Lue
análisis de complejidad puede resumirse en

- Línea 1 en $\mathcal{O}(n\log(n))$
- Línea 2 en $\mathcal{O}(n)$
- **for** de línea 3 se ejecuta $\mathcal{O}(n)$ veces
  - Línea 4 en $\mathcal{O}(n\log(n))$
  - Línea 5 en $\mathcal{O}(n)$
  - **for** de línea 6 se ejecuta $\mathcal{O}(n)$ veces
    - Línea 7 en $\mathcal{O}(n\log(n))$

Con esto, la complejidad sería

$$\mathcal{O}(n\log(n) + n + n\cdot[n\log(n) + n + n\cdot(n\log(n))]) \quad \Rightarrow \quad \mathcal{O}(n^3\log(n))$$

Solución

(c) [1 pto.] Le informan que, si bien en el archivo hay primeros apellidos repetidos, la cantidad de r
ciones por apellido es muy baja (#repeticiones < 20). ¿Puede proponer alguna mejora a su algo
utilizando esta información? Justifique.

Solución.

Una posible mejora sería utilizar `InsertionSort` para ordenar las secuencias de repetidos, a fin d
tar la recursión y aprovechar el mejor desempeño práctico de `InsertionSort` en instancias peq

Repaso Dividir para Conquistar

¿Cuales son los pasos de dividir para conquistar?

1. **Dividir**. Se divide el problema en subproblemas más pequeños
2. **Conquistar**. Resolvemos los subproblemas al llamarlos de forr recursiva hasta que sea resuelto
3. **Combinar**. Combinamos los subproblemas hasta llegar a la solución del problema.

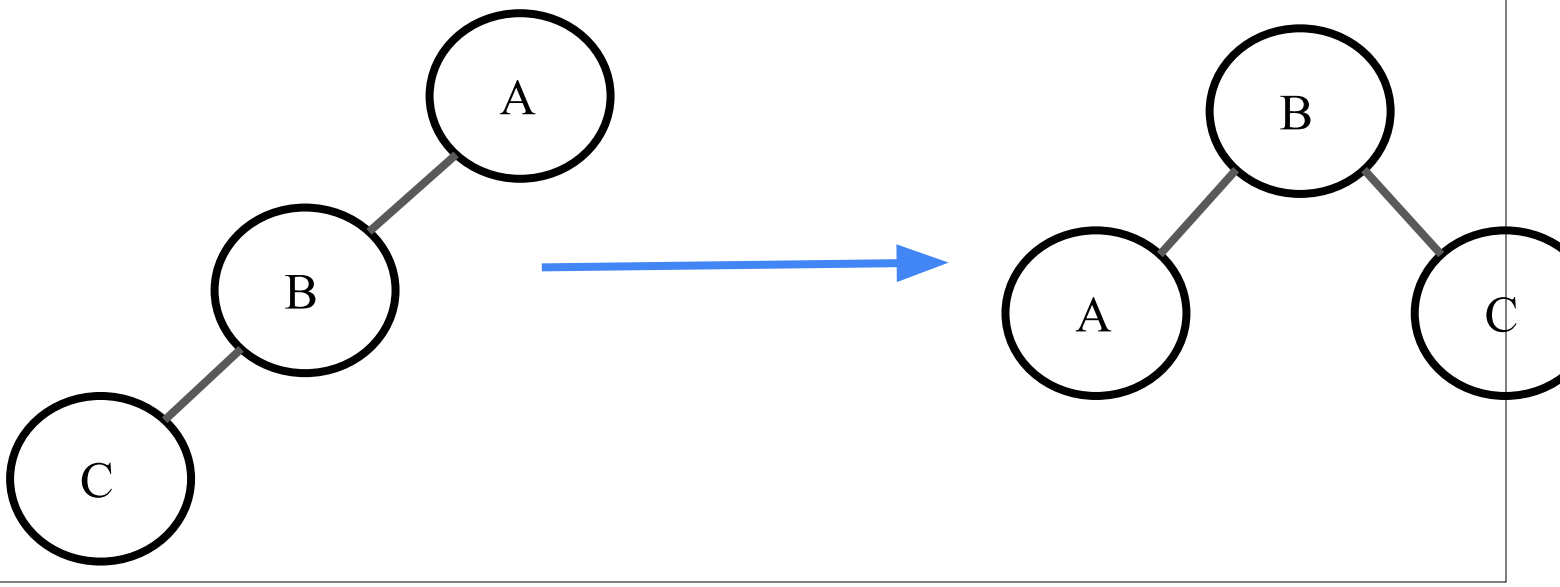
Repaso Árboles

¿Cuales son las características de cada tipo de árbol?

1. **AVL.** Las alturas de sus hijos difieren a lo más en 1 entre sí y cada hijo esta AVL-balanceado.
2. **2-3.** Tiene dos tipos de nodos. Si tienen una sola llave pueden tener hasta 2 hijos y si tienen dos llaves pueden tener hasta 3 hijos.
3. **Rojo-Negro.** Sus nodos pueden ser rojos o negros, la raíz es negra, los hijos de un nodo rojo tienen que ser negros y la cantidad de nodos negros camino a cada hoja desde un nodo cualquiera debe ser la misma.

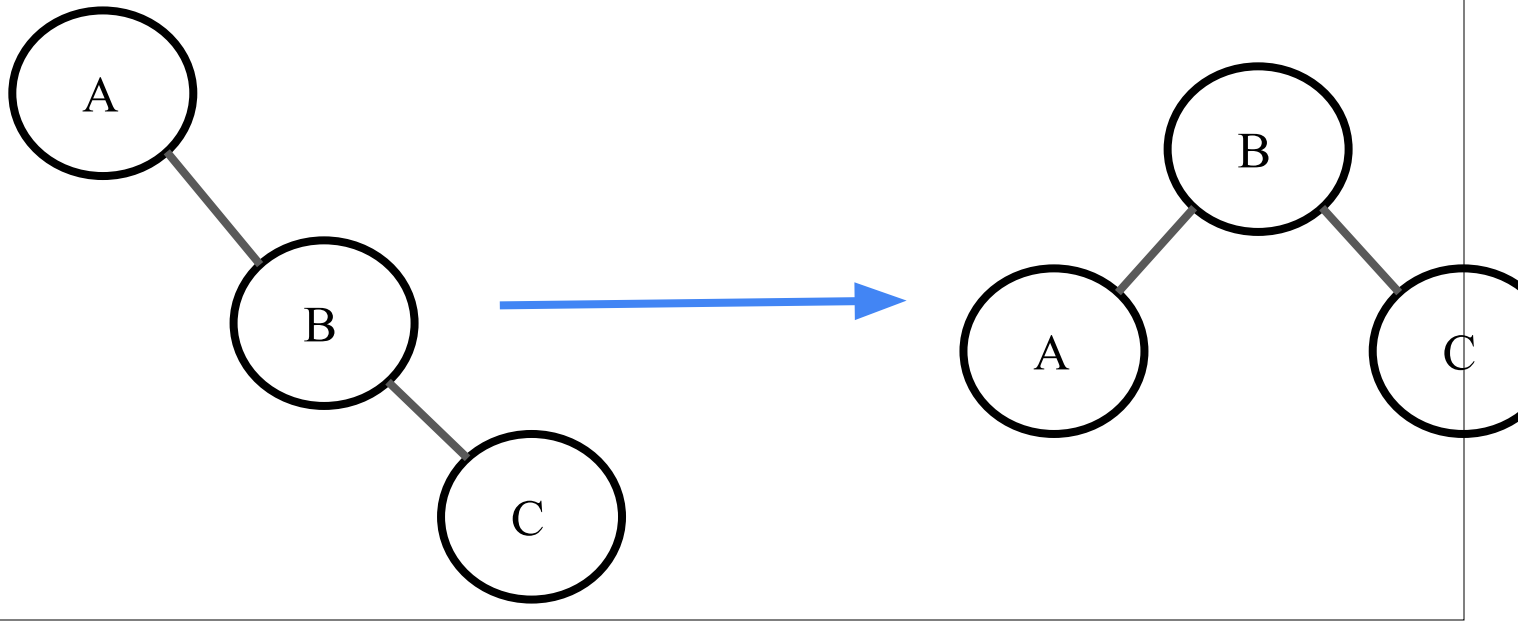
AVL

1. **Right Rotation**. Nodo es insertado en el subárbol izquierdo d
   subárbol izquierdo



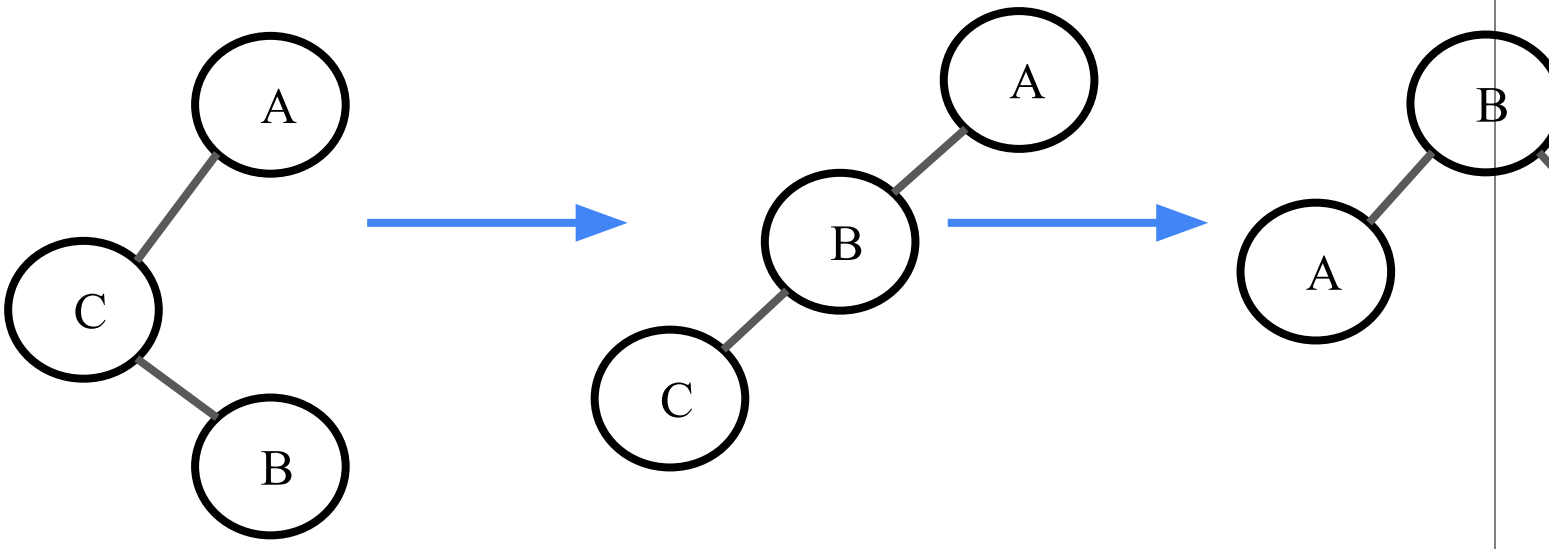
AVL

2. **Left Rotation**. Nodo es insertado en el subárbol derecho de un
subárbol derecho



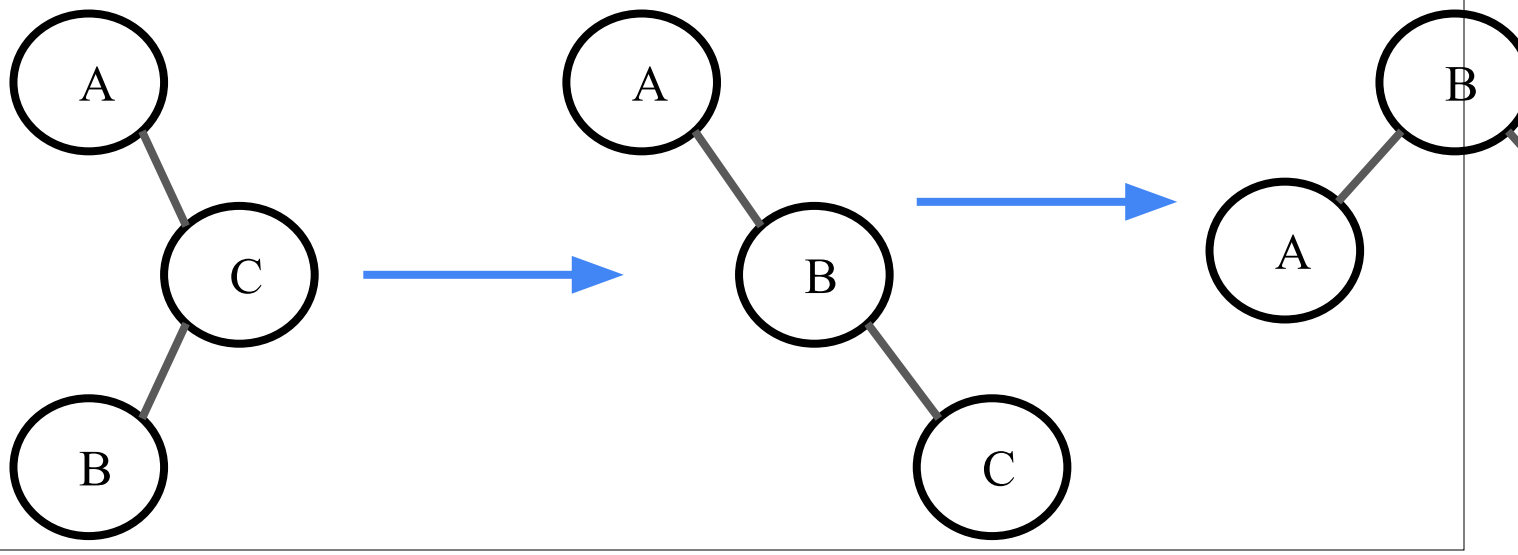
AVL

3. **Left-Right Rotation**. Nodo es insertado en el subárbol derecho
subárbol izquierdo



AVL

4. **Right-Left Rotation**. Nodo es insertado en el subárbol izquier
subárbol derecho



Árbol 2-3

Al insertar llaves:

- Se inserta como llave múltiple en una hoja existente
- Si la hoja era 2-nodo, queda como 3-nodo y terminamos
- Si la hoja era 3-nodo, queda como 4-nodo (con 3 llaves) por
- La llave central del 4-nodo sube como llave múltiple al padr
- Se repite la modificación de forma recursiva hacia la raíz

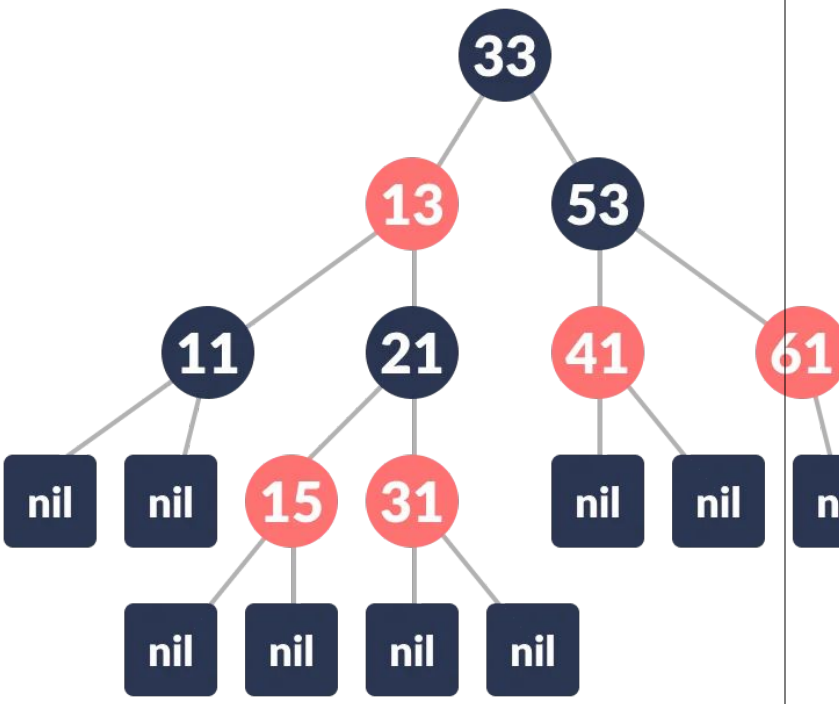
Árbol Rojo-Negro

Propiedades

- Un nodo es **rojo** o **negro**
- Los nodos raíz o hojas son **negros**
- Si un nodo es **rojo**, entonces sus hijos son **negros**
- Todas las rutas de un nodo a sus descendientes hoja contienen mismo número de nodos **negro**

Árbol Rojo-Negro

```
FixBalance (x):
    while x.p ≠ ∅ ∧ x.p.color = rojo :
        t ← x.uncle   ▷ tío de x
        if t.color = rojo :
            x.p.color ← negro
            t.color ← negro
            x.p.p.color ← rojo
            x ← x.p.p
        else:
            if x es hijo interior de x.p :
                Rotation(x.p, x)
                x ← x.p
            x.p.color ← negro
            x.p.p.color ← rojo
            Rotation(x.p.p, x)
    A.root.color ← negro
```



Pregunta 3

a) Sea $\boldsymbol{T}$ un ABB de altura $\boldsymbol{h}$. Escribe un algoritmo que posicione un elemento arbitrario de T de $\boldsymbol{T}$ en $\boldsymbol{\mathcal{O}(h)}$ pasos.

b) Sean $\boldsymbol{T_1}$ y $\boldsymbol{T_2}$ dos ABB de $\boldsymbol{n}$ y $\boldsymbol{m}$ nodos respectivamente. Explica, de manera clara y pr realizar un merge entre ambos árboles en $\boldsymbol{\mathcal{O}(n+m)}$ pasos para dejarlos como un solo ABB $\boldsymbol{T}$ con los nodos de ambos árboles.

Solución Pregunta 3a)

La solución puede o no considerar el proceso de búsqueda del nodo solicitado (de ahora en adelante, y
paso tiene complejidad $\mathcal{O}(h)$, por lo que sigue dentro de la cota especificada.

Tras este proceso, es necesario hacer rotaciones de tal forma que y quede en la raíz del árbol. Esto se
realizar utilizando las rotaciones AVL vistas en clases. A continuación se propone un pseudocódigo.

```
1: procedure VolverRaiz(nodo y)
2:     while y tiene padre do
3:         x ← y.padre
4:         if y es hijo izquierdo de su padre then
5:             rotar hacia la derecha en torno a x-y          ▷ esto modifica el padr
6:         else
7:             rotar hacia la izquierda en torno a x-y        ▷ esto modifica el padr
8:         end if
9:     end while
10: end procedure
```

Vimos en clases que cada rotación es $\mathcal{O}(1)$ y en cada una de estas y sube un nivel. En el peor caso y
hoja por lo tanto es necesario hacerlo subir h niveles $\implies$ se hacen $\mathcal{O}(h)$ rotaciones.

Solución Pregunta 3b)

La pregunta no solicita un algoritmo, por lo que basta con describir el proceso:

1. Se itera por sobre los nodos de ambos árboles de manera ordenada, copiando los nodos a un a
   Este paso consiste en un proceso recursivo que visita siempre el nodo izquierdo antes que el der
   (recorrido *in-order* o algoritmo visto en la ayudantía, ver solución pregunta 2a). Haciendo esto tene
   dos arreglos ordenados, con los elementos de $\boldsymbol{T_1}$ y $\boldsymbol{T_2}$ respectivamente.

2. Combinamos estos dos arreglos usando la subrutina ***merge*** de MergeSort en $\boldsymbol{\mathcal{O}(n+m)}$. Con
   obtenemos un arreglo ordenado $\boldsymbol{A}$ con los $\boldsymbol{n+m}$ elementos de ambos árboles.

3. Finalmente, se convierte el array ordenado $\boldsymbol{A}$ en un ABB balanceado poniendo la mediana como
   e insertando los valores menores y mayores a esta en las ramas izquierdas y derechas respectivam
   Notar que encontrar la mediana en un array ordenado es $\boldsymbol{O(1)}$ por lo que proceso se realiza en $\boldsymbol{O(m}$
   pasos.

Ejercicio Árboles (I2 - 2022-1)

a) Considera una secuencia de inserciones de claves distintas que se ejecuta tanto en un árbol AVL como en un rojo-neg
inicialmente vacíos. La secuencia es tal que mantiene los árboles balanceados tanto como sea posible. ?¿Cuál de los
desbalancea primero?

a) Dibuja un árbol rojo-negro, tal que si nos olvidamos de los colores **No** es un AVL.

a) Demuestra que cualquier AVL, sus nodos pueden ser pintados tal que sea un arbol rojo-negro.

Solución

a. 1 pto

El árbol Rojo-Negro se desbalancea primero. Un ejemplo a considerar: Se han colocado 3 nodo
en ambos árboles AVL y rojo-negro, ambos están balanceados. El árbol rojo-negro tiene negr
rojo ambos hijos (por sus propiedades). Al colocar un nodo con la misma clave en ambos árbol
parte, el árbol AVL seguirá siendo balanceado puesto que la diferencia entre las hojas difiere
(propiedad AVL). Por otra parte, el árbol rojo-negro, dicha inserción (independiente del nodo ho
haya colocado) es un nodo inicialmente rojo, y como tanto su padre como su tío son rojos, ya es
la propiedad 3 del árbol rojo-negro (si un nodo es rojo, sus hijos son negros), por lo que despué
colocado el cuarto nodo al árbol rojo-negro, éste debe balancearse.

- 0.3 pts por señalar cual se desbalancea primero
- 0.3 pts por señalar propiedades involucradas
- 0.4 pts por dar justificación

Solución

b. 0.5 pto

En general, deben dibujar un árbol rojo-negro, y señalar cual propiedad AVL se violaría al olvida
colores rojo-negro.

- 0.2 ptos por dibujar un árbol rojo-negro que cumple sus propiedades (es un ABB más las 4 pr
del rojo-negro)
- 0.3 ptos por señalar cual propiedad del AVL estaría incumpliendo al olvidarse de los colore
este caso, sería la propiedad del balance (que al olvidar los colores de nodos, es simplemente
desbalanceado).

Cualquier otra forma de resolver el problema queda a criterio del ayudante.

Solución

c. 0.5 ptos

Se puede resolver de distintas formas, pero una explicación puede ser la siguiente: La raíz siempre
de negra, se puede pintar los nodos por nivel (alternándose). Si el árbol tiene todos sus nodos en
0, basta con pintar de manera alternada para tener las propiedades de rojo-negro. Luego, es im
que mencionen que el caso crítico sería cuando hay una diferencia de 1 en algún subárbol, donde al
mencionar como podrán pintar los nodos de forma que se pueda mantener las propiedades.
El puntaje se distribuye a criterio del ayudante con respecto a la justificación.

¡Muchas gracias y suerte en la prue