IIC2133 – Estructuras de Datos y Algoritmos
Interrogación 2

Hora inicio: 14:00 del 6 de mayo del 2020

Hora máxima de entrega: 23:59 del 7 de mayo del 2020

Rellena lo siguiente e inclúyelo al principio de tu entrega. Nos reservamos el derecho a no corregir tu prueba si no lo haces:

Yo, _____*Nombre Apellido*_____ , doy fe de que todas las respuestas contenidas en esta prueba fueron elaboradas por mí, sin haber consultado sobre la prueba a ninguna persona ajena al cuerpo docente del curso.

_____*Firma*_____

1. La propiedad fundamental de un ABB es que las claves almacenadas en el subárbol izquierdo son todas menores que la clave almacenada en la raíz, la que a su vez es menor que cualquiera de las claves almacenadas en el subárbol derecho. Teniendo presente esta propiedad, responde:

    a. Considera que tienes un ABB vacío T sin autobalance y una lista desordenada L de n números. ¿Cómo puedes utilizar T para ordenar L? ¿Cuál sería la complejidad de este algoritmo en notación Ω? ¿Qué características tiene L cuando se da este caso? Da un ejemplo con $n = 11$.

    b. Definimos B_x como los nodos en la ruta de búsqueda de una hoja x en un árbol T. Definimos A_x como todos los nodos a la izquierda de B_x, y C_x como todos los nodos a la derecha de B_x. ¿Es posible que haya un nodo en C_x de clave menor a la clave de un nodo en A_x?
    Si la respuesta es sí, da un ejemplo. En caso contrario, demuestra que no es posible.

2. La propiedad fundamental de un AVL, además de ser un ABB, es que las alturas del subárbol izquierdo y del subárbol derecho difieren a lo más en 1. Teniendo presente esta propiedad, responde:

    a. Considera que tienes un AVL vacío T y una lista desordenada L de n números.
    ¿Cómo puedes utilizar T para ordenar L? ¿Cuál sería la complejidad de este algoritmo en notación Ω? ¿Qué características tiene L cuando se da este caso? Ejecuta tu algoritmo con $L = [17, 29, 53, 61, 73, 37, 43]$.

    b. Demuestra que basta con hacer una sola rotación (simple o doble) para corregir el desbalance producto de una inserción en un árbol AVL.

3. Los árboles 2-3 son árboles de búsqueda en que los nodos tienen ya sea una clave y dos hijos, o bien dos claves y tres hijos; y todas las hojas del árbol (que se exceptúan de la regla anterior porque no tienen hijos) están a la misma profundidad. Teniendo presente estas propiedades, responde:

    a. ¿Cuál es la altura máxima que puede tener un árbol 2-3 con n elementos? ¿Y la mínima? ¿Cómo es la estructura del árbol cuando ocurre cada uno de estos casos?

    b. Queremos insertar una clave x en un árbol 2-3 T de altura h, que tiene n claves. ¿Qué debe cumplirse para que esta inserción aumente la altura de T? ¿Para qué valores de n está garantizado que **sí** aumentará la altura? ¿Para qué valores de n está garantizado que **no** aumentará la altura?

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Computación
IIC2133 – Estructuras de datos y algoritmos
Primer Semestre 2020

Pauta Interrogación 2

Problema 1

a)

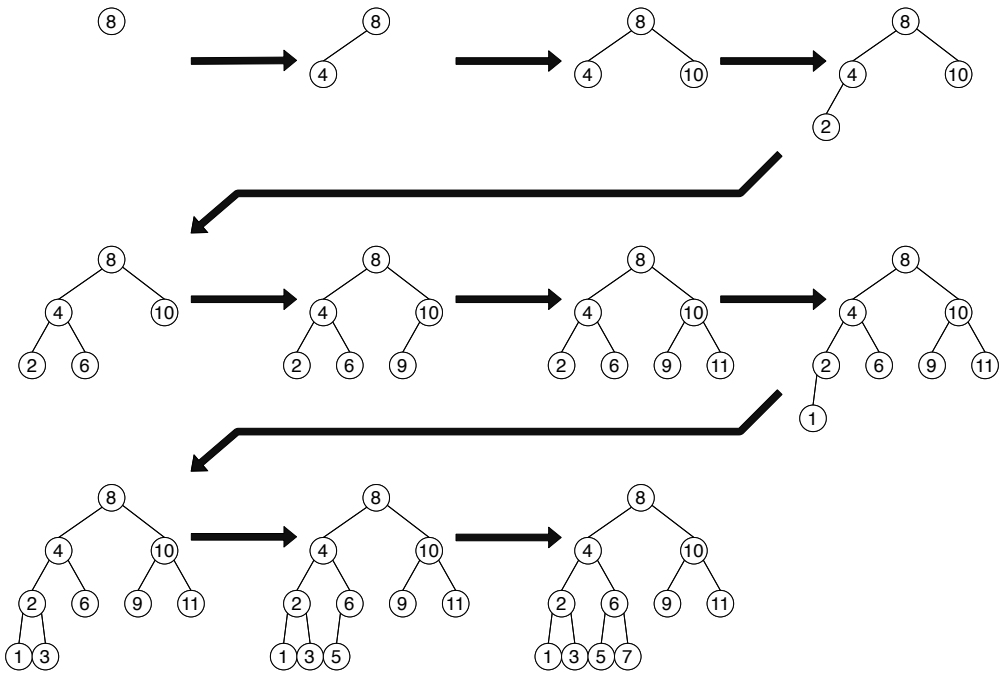
La estrategia es tomar todos los elementos de L e insertarlos en T. Luego podemos obtener los elementos de L en orden haciendo un recorrido *in-order* del árbol. **[0.1 pts]**

Este algoritmo es $\Omega(n \cdot log(n))$, ya que son n inserciones y en el **mejor caso** cada una toma $\mathcal{O}(log(n))$, mientras que el recorrido *in-order* es $\Theta(n)$. **[0.2 pts]**

Para ver cuando se da este caso, tenemos que pensar en un ABB perfectamente balanceado T'. Para que una lista L genere el árbol T', se debe cumplir que cada número aparezca en L después que todos sus ancestros en T'. **[0.5 pts]**. Si solo incluye un caso particular como el de las medianas, **[0.3 pts]**

Un ejemplo sería:

$$L = [8,\ 4,\ 10,\ 2,\ 6,\ 9,\ 11,\ 1,\ 3,\ 5,\ 7]$$



[0.2 pts] por mostrar como queda el árbol luego de insertar todos los elementos de L. Basta con mostrar solo el árbol final. Debe cumplir con la propiedad dicha anteriormente.

b)

Primero que todo, la respuesta correcta a la pregunta es, **no**, no puede haber un elemento en A_x mayor a un elemento en C_x. **[0.1 pts]**

Posible demostración [0.9 pts]:

Para demostrar esto, es útil expresar los grupos de nodos en términos de conjuntos:

$$B_x = \{b \mid b \text{ está en la ruta de búsqueda hacia } x\}$$

$$A_x = \{a \mid a \notin B_x \wedge (a \text{ es hijo izquierdo de un nodo en } B_x \text{ o es descendiente de otro elemento en } A_x)\}$$

$$C_x = \{c \mid c \notin B_x \wedge (c \text{ es hijo derecho de un nodo en } B_x \text{ o es descendiente de otro elemento en } C_x)\}$$

Veamos como se relacionan los elementos de A_x y C_x respecto a x.

Para $a \in A_x$, tenemos 2 casos:

- a es hijo izquierdo de un nodo en B_x: Llamemos a dicho nodo "b". Por definición de ABB, a es menor a todo el subárbol derecho de b. Pero por definición de B_x, b forma parte de la ruta hacia x, y como $a \notin B_x$, x tiene que estar en el subárbol derecho de b. Por lo tanto, $a < x$.
- a es descendiente de otro elemento en A_x: Por definición de ABB, si un nodo dado es menor a un número, todos sus descendientes también cumplen con esa desigualdad, sin importar la profundidad. Todos los a que surgen de este caso son descendientes de los a del primer caso, por lo que también se cumple que $a < x$

Es decir,

$$\forall a \in A_x,\ a < x$$

Análogamente,

$$\forall c \in C_x,\ x < c$$

Por lo tanto,

$$\forall a \in A_x,\ \forall c \in C_x,\ a < c$$

Escala en caso de hacer una demostración distinta

En caso de poner una demostración distinta, se hizo la siguiente escala para cada caso.

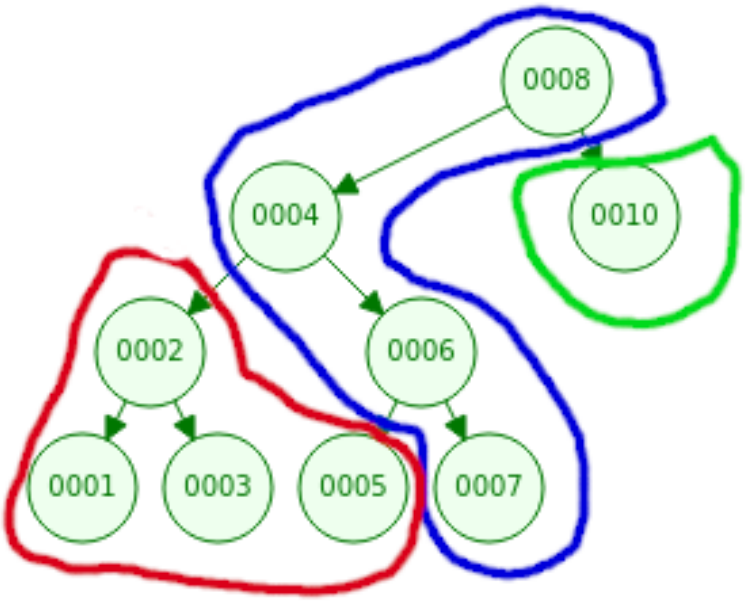
- 0 pts si la demostración no tiene lógica, o no hay formalidad en ella y es incorrecta.
- 0.3 pts si la demostración tiene lógica, pero tiene errores graves de formalidad o planteamientos en la lógica.
- 0.6 pts si la demostración tiene lógica, pero tiene errores pequeños en la formalidad o errores logicos en las ideas planteadas.
- 0.9 pts si la demotración es correcta, tiene la formalidad requerida y demuestra correctamente lo pedido.

Contra Ejemplo

En varios casos se enuncio que

$$\forall a \in A_x, b \in B_x \quad a < b$$

Lo cual es incorrecto, como contraejemplo aca hay un arbol que lo prueba, en el cual x = 7. En azul Bx, rojo Ax y verde Cx



Problema 2

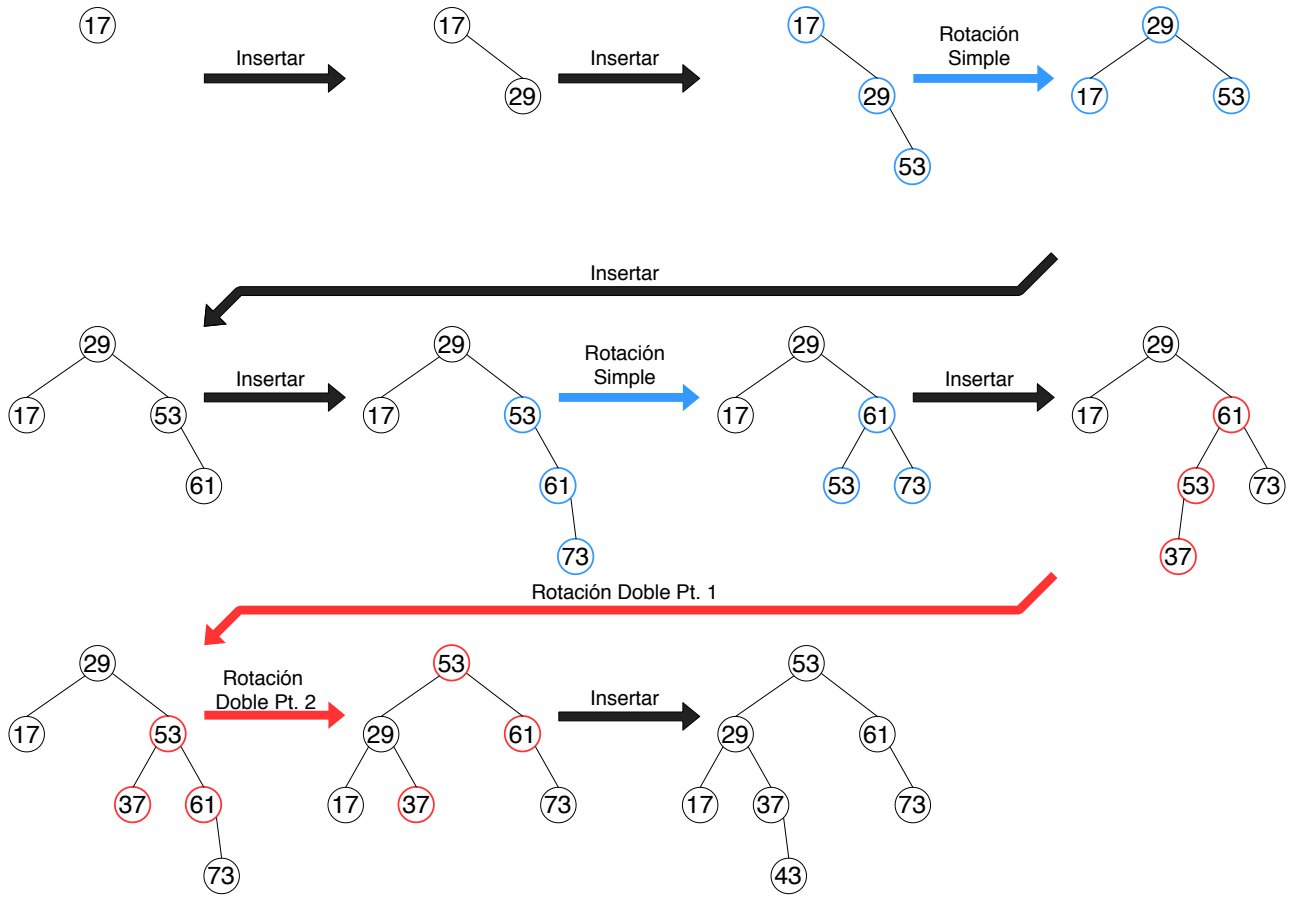
a)

La estrategia es tomar todos los elementos de L e insertarlos en T. Luego podemos obtener los elementos de L en orden haciendo un recorrido *in-order* del árbol. **[0.1 pts]**

Este algoritmo es $\Omega(n \cdot log(n))$, ya que son n inserciones y cada una toma $\Theta(log(n))$, mientras que el recorrido *in-order* es $\Theta(n)$. **[0.2 pts]**

Este caso se da independiente de las características de L. **[0.2 pts]**

$$L = [17,\ 29,\ 53,\ 61,\ 73,\ 37,\ 43]$$



[**0.1 pts**] por las inserciones, [**0.2 pts**] por las rotaciones simples y [**0.2 pts**] por la rotación dobles.
Si el alumno copia y pega una imagen del procedimiento realizado por un sitio web, sin explicación alguna, es 0/0.5 puntos en esta parte.

b)

Por propiedad sabemos que si un árbol es AVL todos sus subárboles son AVL. Esto implica que la diferencia de altura entre el subárbol derecho y izquierdo de todo subárbol del árbol AVL es 0 o 1.

Para que una inserción produzca un desbalance, tiene que cumplirse que el se inserta en el subárbol de la altura mayor, donde ya había una diferencia de altura de 1.
Esto, por que al hacerse está inserción donde ambos subárboles tienen la misma altura, la nueva diferencia sería de 1, y por lo tanto sigue siendo AVL balanceado.

Solo hay 4 posibles casos para los que ocurre esto.
Estos son desbalance externo (clase 7, slide 9) y desbalance interno (clase 7, slide 13), que puede ocurrir para el subárbol derecho u izquierdo. Estos casos son simétricos y las rotaciones se hacen como se indican ahí.

(0.4) Explica cada caso (interno y externo) correctamente.
(0.2) Explica solo un caso (interno y externo) u ambos casos con errores menores.
(0) No explica ningún caso u los explica incorrectamente.

(0.2) Explica cada caso simétrico al ya explicado (implícita o explícitamente).
(0.1) Explica solo un caso simétrico al ya explicado (implícita o explícitamente).

(0) No explica ningún caso simétrico al ya explicado (implícita o explícitamente).

(0.2) Muestra o menciona como para cada uno de estos casos basta con una rotación para volver a balancear.
(0) No lo hace.

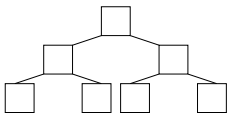
(0.2) Por explicar por que esos cuatro son los únicos casos posibles de desbalance.
(0.1) Por decir o indicar que esos cuatro son los únicos casos posibles de desbalance.
(0) Por decir o indicar que hay más de casos donde puede haber desbalance (sin demostrarlo correctamente).

Problema 3

a)

La altura será mayor mientras menos elementos tenga el árbol por nivel, y viceversa. Siguiendo esta lógica la altura será mayor cuando todos los nodos sean nodos 2, será menor cuando todos sus nodos sean nodos 3.

(a) Sólo nodos 2



(b) Sólo nodos 3

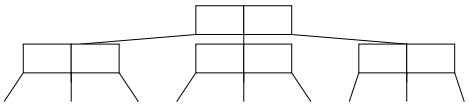


Figura 1: Comparación de estructura de árboles 2-3

Viendo la figura a), podemos notar que la cantidad de elementos para el k-ésimo nivel es de 2^{k-1}. Podemos notar también, que la cantidad de elementos acumulados hasta el k-ésimo nivel es $2^k - 1$. Análogamente para b), la cantidad de elementos para el k-ésimo nivel es $2 \cdot 3^{k-1}$, y la cantidad de elementos acumulados hasta el k-ésimo nivel es $3^k - 1$. Si despejamos la altura en cada una, obtenemos para la altura mínima:

$$h = \log_3(n+1)$$

y para la altura máxima

$$h = \log_2(n+1)$$

Cabe mencionar que estos casos son cuando el árbol esta completamente lleno. Para alcanzar un nivel k de altura, debe exceder necesariamente los $k-1$ niveles anteriores , y no superar los k niveles. Por lo tanto, la altura mínima en función de la cantidad de elementos quedaría expresada como:

$$h = \lceil \log_3(n+1) \rceil$$

En el caso de la altura máxima, como las hojas deben estar a la misma altura, al añadir un valor extra, el árbol se rebalancea, manteniendo su altura, por lo que quedaría expresada como:

$$h = \lfloor \log_2(n+1) \rfloor$$

en los casos que $n > 0$. Si $n = 0$ en realidad no existiría estructura de datos, por lo que las fórmulas pierden sentido.

Son 0.5 pt por cada fórmula:

- ◇ **0.2 por indicar que el máximo era solo con nodos 2 y el mínimo sólo con nodos 3,**
- ◇ **0.2 por llegar a la fórmula sin función techo/piso (despejando h o n),**
- ◇ **0.1 por la formula con función techo/piso, despejando h**

Si llega a una fórmula incorrecta, se realiza descuento

b)

En primer lugar, llamaremos al "camino" de un árbol T a los nodos que hay que recorrer desde la raíz para llegar a un nodo hoja. De esta forma, para que una inserción provoque el aumento de altura de un árbol T se debe cumplir que, en el camino a la hoja donde se insertará la nueva clave, todos los nodos sean nodos 3 (es decir, que tengan 2 claves). Así, se hará un *split* desde la hoja donde insertamos hasta la raíz, aumentando finalmente la altura **(0.3 puntos)**.

Ahora evaluaremos dos casos en dónde siempre aumenta la altura, y donde nunca aumenta la altura para una altura h con n claves.

Garantía de que aumenta la altura: El árbol T debe tener solo nodos 3. Notemos que si pasa lo contrario (al menos uno no es un nodo 3), entonces podemos buscar el camino en donde podamos insertar de tal forma que este nodo pase a ser nodo 3, lo que haría que no aumente la altura. Teniendo esto, la cantidad de claves en función de la altura h es

$$claves = \sum_{i=1}^{h} 2 \cdot \left(3^{i-1}\right)$$

dado que en un nivel i hay 3^{i-1} nodos, con cada uno 2 claves. Resolviendo obtenemos **(0.3 puntos)**

$$claves = 3^h - 1$$

Garantía de que no aumenta la altura: El árbol T no debe tener ningún camino que tenga únicamente nodos 3. Notemos por la definición del comienzo que la condición para que aumente la altura es que agreguemos una clave en la hoja que tenga un camino con sólo nodos 3, por lo que debemos asegurarnos que no exista tal camino. Luego, para garantizar esto debemos contar la cantidad de claves para que no exista un árbol con dicho camino.
La menor cantidad de claves con la que podría aumentar la altura es tal que todos sean nodos 2 excepto un camino en donde sean nodos 3. Teniendo esto, si tomamos esa cantidad de claves - 1 no podremos tener un camino con nodos 3, y garantizaremos desde esa cantidad hacia abajo que no aumente la altura.

Un árbol con las características antes mencionadas tendría la siguiente estructura:

- ◇ nivel $1 \rightarrow 2$ claves (nodo 3)
- ◇ nivel $2 \rightarrow 4$ claves (nodo 3 + 2 nodos 2)
- ◇ nivel $3 \rightarrow 8$ claves (nodo 3 + 6 nodos 2)

  $\vdots$

- ◇ nivel $h \rightarrow 2^h$ claves (nodo $3 + 2^h - 2$ nodos 2)

Si sumamos obtenemos

$$claves = \sum_{i=1}^{h} 2^i - 1$$

siendo la suma de las claves de cada nivel - 1 por lo comentado anteriormente. Resolviendo obtenemos

$$claves = 2^{h+1} - 2$$

Luego, para garantizar que la inserción en el árbol T no tendrá efectos en la altura h, la cantidad de claves debe ser menor a $2^{h+1} - 2$. Además, para la factibilidad de un árbol 2-3, se debe cumplir que las claves sean mayores a $2^h - 1$, que es la cantidad mínima de claves que un árbol 2-3 de altura h puede tener **(0.4 puntos)**.