Ayudantía Repaso I1

2024-1

Dafne Arriagada, Gustavo Salinas,
Agustín Gutiérrez, Joaquín Viñuela

Invitación

Concierto

Torpedo I1, Inasistencia

(1) El torpedo es una hoja tamaño carta u oficio, escrita con su puño y letra, por ambos lados. Puede ser algo que escribieron en su tablet/ipad y que luego imprimieron, pero tiene que haber sido escrito por ustedes.

No puede incluir: screenshots de diapositivas o similares.

(2) Calendario semanal: lunes repaso con los profes y miércoles ayudantía repaso I1 (no hay clases, solo ayudantía en dicho horario), quedando el horario del viernes libre para su estudio.

(3) Inasistencia a interrogaciones: Si faltan deben justificar a su unidad académica para optar a la instancia de preguntas recuperativas que ocurre justo luego de finalizada la I3.

Repaso Correcitud

¿Qué necesitamos para que un algoritmo sea considerado correcto?

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

I1 2022-1

4) **Demostración de corrección**. Considera dos arreglos A y B de n elementos cada uno; ambos arreglos están ordenados. Demuestra que el siguiente algoritmo encuentra la mediana de la totalidad 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

¿Qué se espera de la respuesta?

Específicamente, demuestra —es decir, da un argumento lógico, bien razonado— que

a) El algoritmo termina (observa que el algoritmo es iterativo, que en cada iteración el tamaño de los subarreglos disminuye, y que la condición de término de la iteración es que el tamaño de ambos subarreglos sea 2).

b) El algoritmo efectivamente encuentra la mediana en tiempo $O(\log n)$ (recuerda que dado que el número total de elementos es par, la mediana es el promedio aritmético de los dos elementos que quedan "al medio" si todos los elementos estuvieran ordenados). Con respecto a los pasos 3 y 4, basta que demuestres uno de los dos.

Consejos

Para demostrar ocupamos inducción. Tenemos otras herramientas?

1. Asegurar cantidad finita de pasos: Mirar los “for”, “while” y las anidaciones entre ellos. Llegar a algún valor finito de pasos por aproximación.

2. Demostrar correctitud de algunos pasos: Ver si se usan algoritmos que ya sabemos funcionan correctamente (ej: uso de QuickSort dentro de algún algoritmo personalizado)

Repaso Complejidad

¿Cómo encontramos la complejidad?

Utilizando el Teorema Maestro (si lo recuerdan)

Si les acomoda lo anterior recomendamos:

1. Ver demostraciones ppt's clases
    a. Nuevamente "podemos poner atención a los "for", "while" y las anidaciones entre ellos, entre otros" e ir operando aritméticamente con las complejidades que van surgiendo para llegar a un resultado (ver ejercicio más adelante)

Tip: Al usar recursión, pensar tiempo que demora el paso recursivo.
Tip 2: Tener a mano las complejidades de algoritmos conocidos

Repaso Sorting

¿Cómo estudiamos sorting?

1. Tomar un arreglo e intentar ordenarlo siguiendo los pasos que indican los algoritmos vistos en clase

2. Entender bien los algoritmos*

   *No se desgasten memorizando, para eso está el torpedo.

I1 2022-2 **Diseño de algoritmos**

Un archivo contiene datos de todos los estudiantes que han rendido cursos del DCC desde 1980 hasta la fecha. El formato cada registro en el archivo es

RUT	primer_apellido	segundo_apellido	nombre

y los registros se encuentran ordenados por RUT.

Solución

(a) [3 ptos.] Proponga el pseudocódigo de un algoritmo para ordenar los registros alfabéticamente, i.e. según (`primer_apellido`, `segundo_apellido`, `nombre`). Especifique qué estructura básica usará (listas o arreglos). Si p es un registro, puede acceder a sus atributos con `p.primer_apellido`, `p.nombre`, etc. Además, puede asumir que todo algoritmo de ordenación visto en clase puede ordenar respecto a un 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 repeticiones de primer apellido entre los índices i y f. De forma similar se define la rutina `SecondLastNameReps` 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 estudiantes en el archivo.

Solución.

El peor caso corresponde a una cantidad $\mathcal{O}(n)$ de repetidos en primer y segundo apellido. Luego, el 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 repeticiones por apellido es muy baja (#repeticiones < 20). ¿Puede proponer alguna mejora a su algoritmo utilizando esta información? Justifique.

Solución.

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

Repaso Dividir para Conquistar

Pasos de dividir para conquistar:

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

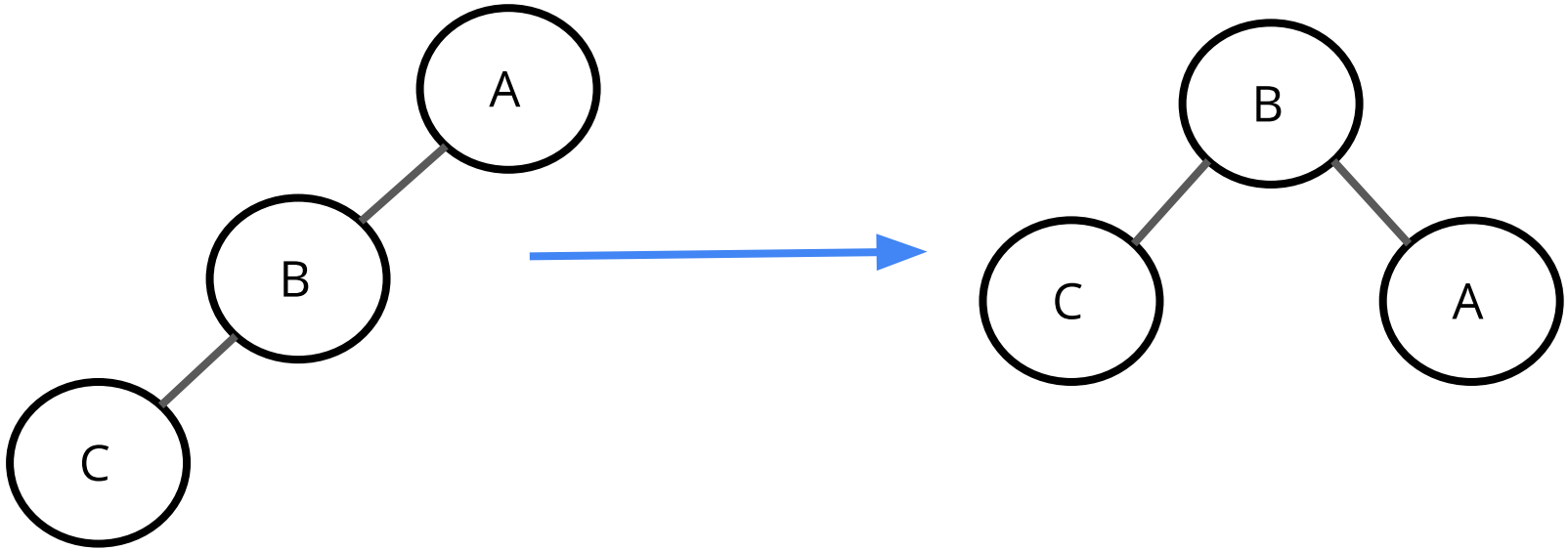
Repaso Árboles

Características de cada tipo de árbol.

1. **AVL.** Las alturas de sus hijos difieren a lo más en 1 entre sí y cada hijo está 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.
4. **B+**.Nodos hojas tienen llave-valor y los nodos internos sólo tienen llaves. Las hojas están al mismo nivel y conectadas por una lista doblemente ligada.

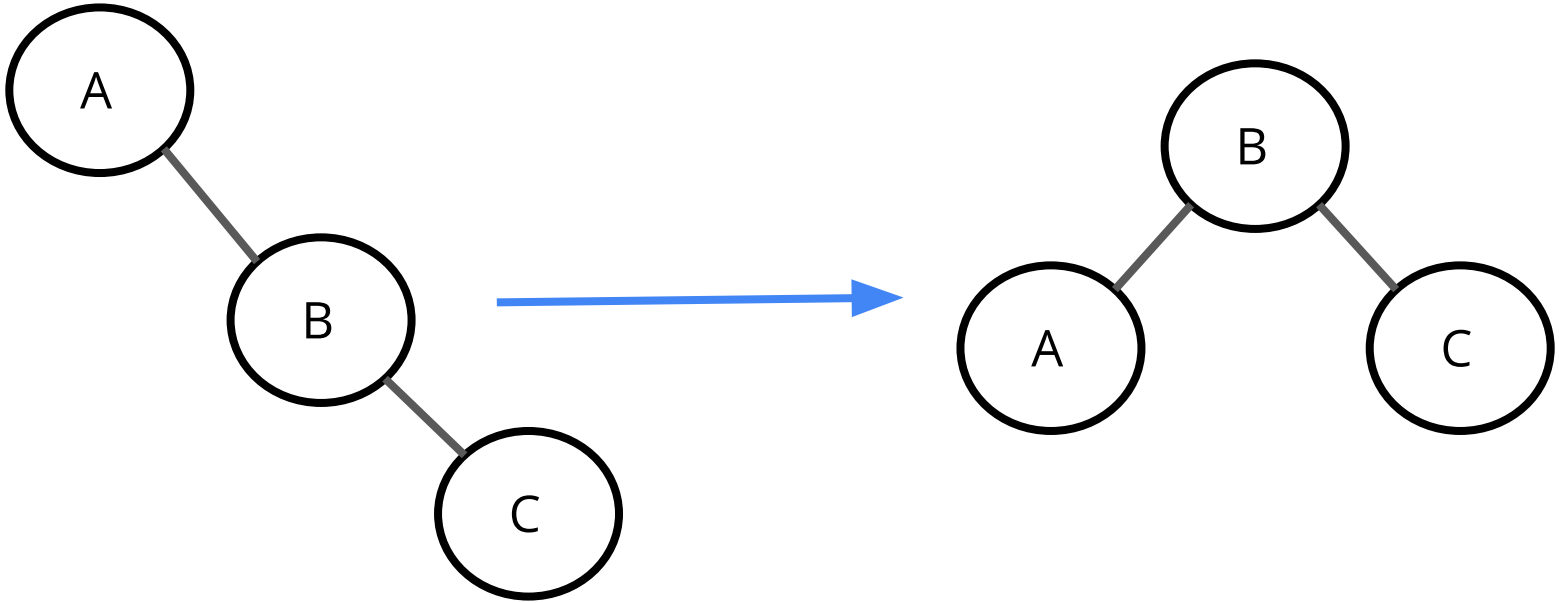
AVL

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



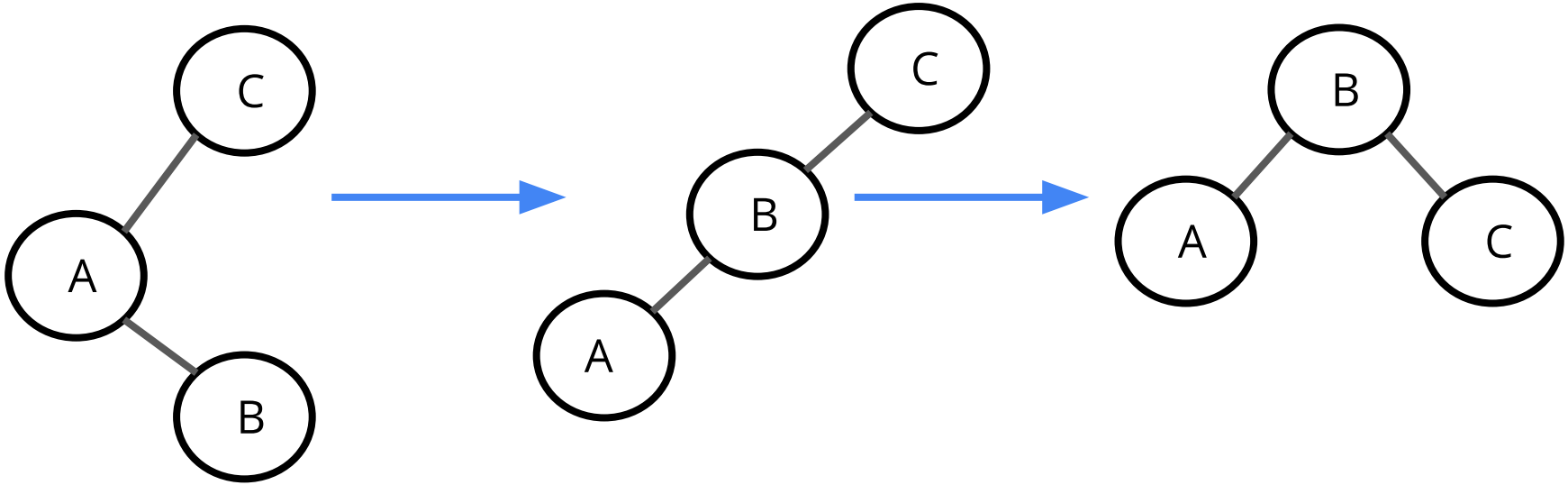
AVL

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



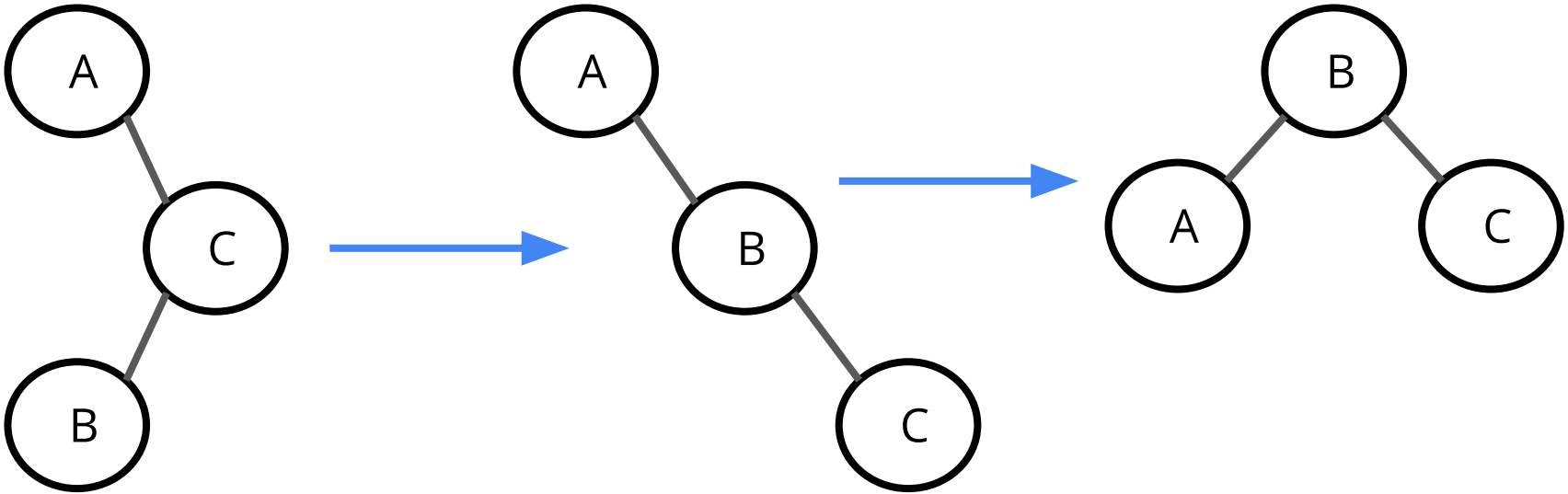
AVL

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



AVL

4. **Right-Left Rotation**. Nodo es insertado en el subárbol izquierdo de un 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 ahora
- La llave central del 4-nodo sube como llave múltiple al padre (**split**)
- Se repite la modificación de forma recursiva hacia la raíz

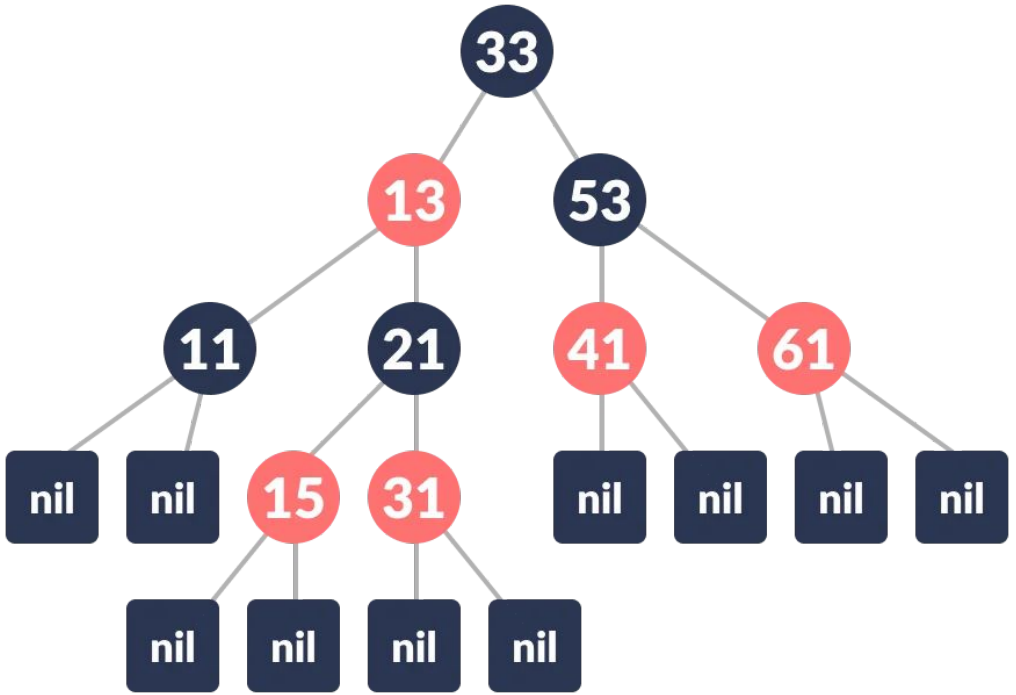
Árbol Rojo-Negro

Propiedades

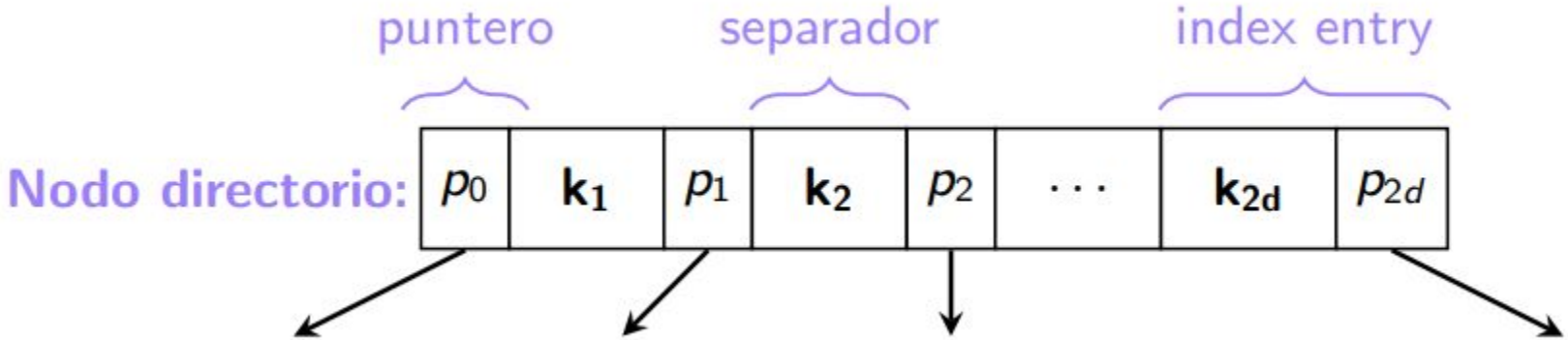
- Un nodo es **rojo** o **negro**
- Los nodos raíz o hojas son **negros**
- Si un nodo es **rojo**, entonces sus hijos son **negros**
- Todas las rutas de un nodo a sus descendientes hoja contienen el mismo número de nodos **negro**

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
```



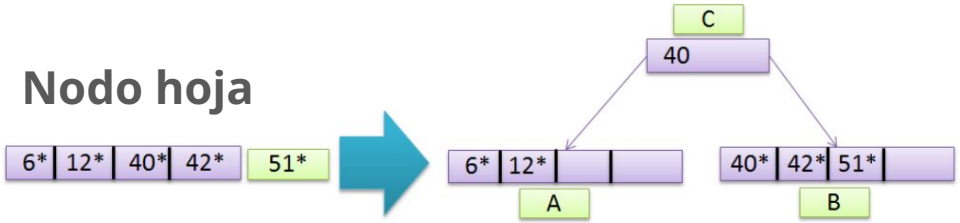
Árbol B+



	# Punteros (Hijos)	# Separadores (Llaves)
Raíz	$2 \leq n \leq 2d+1$	$1 \leq n \leq 2d$
Nodos internos	$d+1 \leq n \leq 2d+1$	$d \leq n \leq 2d$

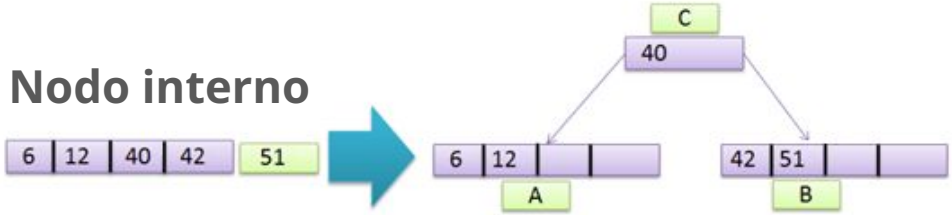
Árbol B+

Nodo hoja



d = 2

Nodo interno



Inserción

- Buscar hoja e insertar. Si hoja tiene más de 2d llaves → Split
- **Split en nodo hoja: llave central** y llaves a la derecha en el nuevo **nodo derecho**, el resto en el izquierdo. Llave central se copia en nodo padre. Obs: siempre que el nodo está **sobrecargado** tiene número **impar** de llaves.
- **Split en nodo interno:** Se divide en dos nodos, llaves a la izquierda y a la derecha de la llave central. Nodo central pasa al nodo padre (no queda en uno de los dos nodos nuevos como en el caso anterior).
- Si nodo padre estaba lleno, split (nodo padre)
- Alternativa a split si hay nodos hojas con espacios disponibles: **redistribución de las llaves**, permite **evitar crecimiento** del árbol

Ejercicio Árboles (P4 - I2 2022-1)

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

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

c) 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 nodos en orden en ambos árboles AVL y rojo-negro, ambos están balanceados. El árbol rojo-negro tiene negra su raíz y rojo ambos hijos (por sus propiedades). Al colocar un nodo con la misma clave en ambos árboles, por una parte, el árbol AVL seguirá siendo balanceado puesto que la diferencia entre las hojas difiere a lo más 1 (propiedad AVL). Por otra parte, el árbol rojo-negro, dicha inserción (independiente del nodo hoja al que se haya colocado) es un nodo inicialmente rojo, y como tanto su padre como su tío son rojos, ya está violando la propiedad 3 del árbol rojo-negro (si un nodo es rojo, sus hijos son negros), por lo que después de haber 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 olvidarse de los colores rojo-negro.

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

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

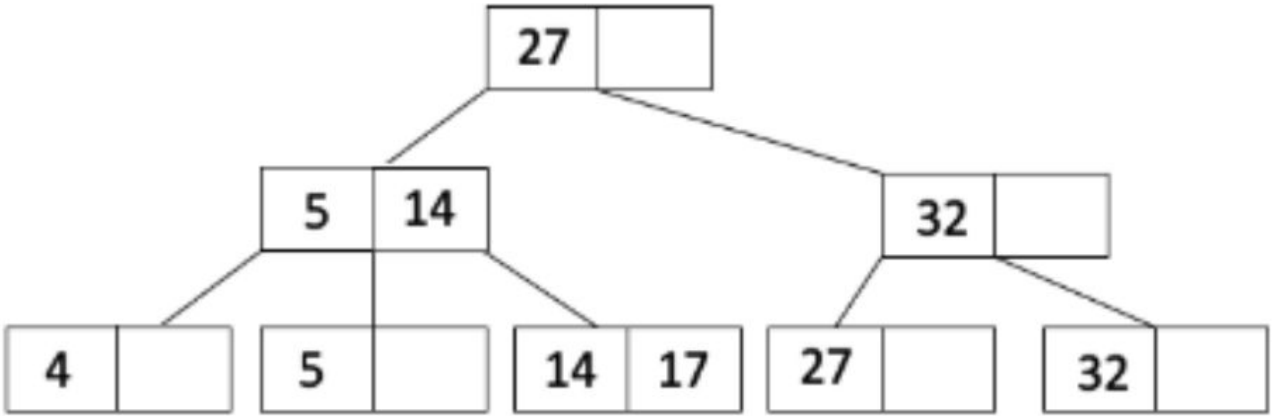
Solución

c. 0.5 ptos

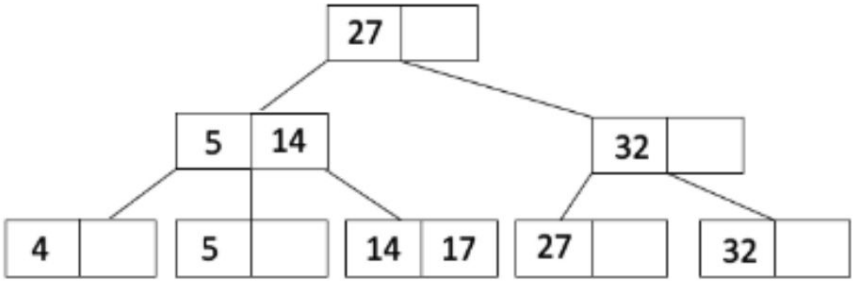
Se puede resolver de distintas formas, pero una explicación puede ser la siguiente: La raíz siempre se pinta de negra, se puede pintar los nodos por nivel (alternándose). Si el árbol tiene todos sus nodos en balance 0, basta con pintar de manera alternada para tener las propiedades de rojo-negro. Luego, es importante que mencionen que el caso crítico sería cuando hay una diferencia de 1 en algún subárbol, donde ahí deben 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.

Ejercicio B+

a. ¿Cuál es el número máximo de datos que podemos añadir al siguiente árbol B+ sin aumentar su altura?
b. ¿Cuál es el número mínimo de datos que podemos añadir al árbol B+ para que aumente su altura?

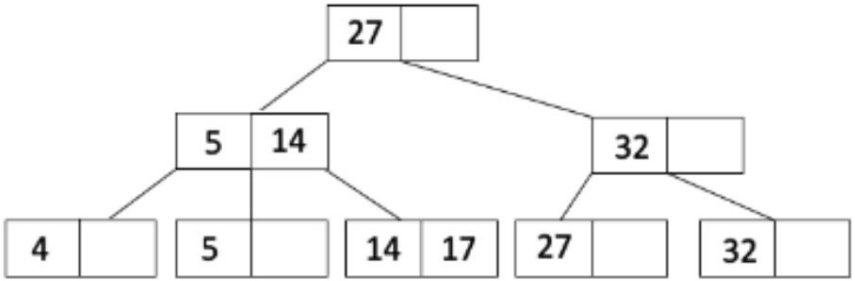


Solución



a. El número máximo de entradas que podemos añadir sin aumentar la altura es la capacidad total de un árbol de altura 2, orden d = 1 menos el número actual de entradas.
Capacidad total = #max número de datos en la lista doblemente ligada (hojas)
Capacidad total = (#max llaves por nodo) x (#max punteros)^altura
Capacidad total = (2d)(2d + 1)^h = (2)(3^2) = 18
El número actual de entradas es 6 porque las entradas están en los nodos hoja. Por lo tanto, podemos añadir un máximo de 18 - 6 = 12. Se puede evidenciar ya que la raíz puede tener un hijo más a la derecha, el cual a su vez puede tener 3 hijos más.

Solución



b. El número mínimo de entradas que podemos añadir para que aumente la altura será 3. Añadiremos 20 al nodo hoja más lleno, que es 14, 17, lo que hará que se divida en 14 y 17, 20. Luego copiamos 17 a su padre 5, 14, lo que hace que se divida en 5 y 17 y 14 será empujado hacia arriba por lo que la raíz tendrá 14, 27. A continuación, vamos a insertar 21 en la hoja más completa que ahora será 17, 20, lo que hará que se divida en 17 y 20, 21. Entonces copiamos 20 a su padre que se convertirá en 17, 20. Luego podemos insertar 22 en la hoja más completa 20, 21 que se dividirá en 20 y 21, 22. Luego copiamos 21 a su padre 17, 20, que se dividirá en 17 y 21 porque 20 será empujado a su padre (la raíz). Esta raíz también se dividirá porque ya tiene 14, 27. Cuando la raíz se divide, sabemos que la altura ha aumentado. Ten en cuenta que se podrían sumar tres números cualesquiera entre 17 y 27 para que la altura del árbol aumentará hasta 3.

Ejercicios recomendados

Ejercicios sugeridos de temas I1:

- P3 - i2 2022-2 (orden lineal)
- P1 - i1 2017-1 (quicksort, mergesort, heaps)
- P3 - i1 2022-2 (dividir para conquistar)
- P2 - i1 2023-2 (AVL)
- P1 - I2 2022-2 (AVL, RN)

De parte del equipo de cátedra les deseamos mucho

Éxito en la prueba!<3