Árboles de búsqueda 1

Fernando Schapachnik[1,2]

[1]En realidad... push('Fernando Schapachnik', push('Esteban Feuerstein', autores))
[2]Departamento de Computación, FCEyN, Universidad de Buenos Aires, Buenos Aires, Argentina

Algoritmos y Estructuras de Datos II, segundo cuatrimestre de 2018

(2) La gran Gugl

- ¿Cómo busco un DNI en el padrón?
- ¿Cómo busco todas las páginas que tengan la palabra 'zapato'?
- ¿Y si además quiero poder agregar, borrar, modificar, y que todo eso sea "barato"?
- Hoy vamos a aprender nuestras primeras estructuras de datos "sin peros".

- En realidad, vamos a hablar de alternativas de diseño para diccionarios y también para conjuntos.
- Notemos que son parecidos, así que vamos a trabajar con el diccionario que es más genérico.

(4) Recordemos qué es una diccionario...

TAD $\textsc{Diccionario}(\alpha,\beta)$

observadores básicos

def? : dicc(α,β) $\times$ α $\longrightarrow$ bool

obtener : dicc(α,β) d $\times$ α c $\longrightarrow$ β {def?(*d*, *c*)}

generadores

vacío : $\longrightarrow$ dicc(α,β)

definir : dicc(α,β) $\times$ α $\times$ β $\longrightarrow$ dicc(α,β)

otras operaciones

borrar : dicc(α,β) d $\times$ α c $\longrightarrow$ dicc(α,β) {def?(*d*, *c*)}

claves : dicc(α,β) d $\longrightarrow$ conj(α)

Fin TAD

(5) Paréntesis terminológico

- A veces se habla de operaciones de ABM o ABMC:
  - **A**ltas: definir una clave en el dicc.
  - **B**ajas: borrado.
  - **M**odificaciones (que pueden traducirse en baja+alta).
  - **C**onsultas: def? y obtener.

(6) Formalicemos lo que veníamos diciendo

- Si trabajamos con listas y arreglos básicamente tenemos una disyuntiva.
- Arreglo ordenado:
  - Búsqueda: $O(\log n)$
  - Inserción/borrado: $O(n)$
- Arreglos no ordenados, listas:
  - Búsqueda: $O(n)$
  - Inserción/borrado: $O(1)$
- Vamos a ver cómo salvar esta disyuntiva.

(7) Yéndonos por las ramas

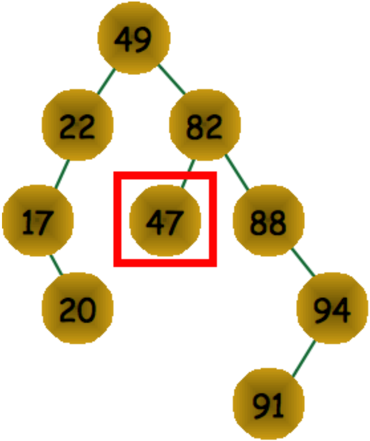
- ¿Y si trabajamos con árboles binarios?
- Recordemos a los AB: nil(), bin(), izq() y der().
- ¿Ganamos algo? No demasiado:
  - Podemos hacer bin() en $O(1)$.
  - ¿Cuánto nos toma buscar? $O(n)$
- ¿Y si somos cuidadosos con la forma que le damos al árbol?
- Es decir, si al buscar pudiésemos saber para qué lado ir...

(8) ABB

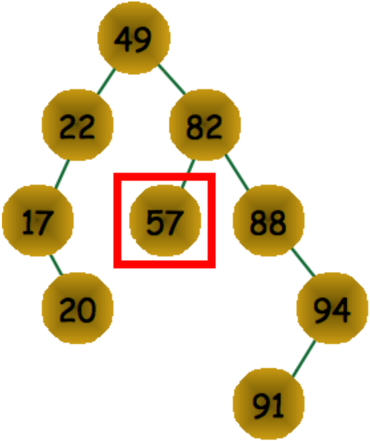
- Un Árbol Binario de Búsqueda (ABB), es un AB con la siguiente propiedad:
  - Para todo nodo, los valores de los elementos en su subárbol izquierdo son menores que el valor del nodo, y
  - los valores de los elementos de su subárbol derecho son mayores que el valor del nodo.
- Dicho de otra forma:
  - El valor de todos los elementos del subárbol izquierdo es menor que el valor de la raíz,
  - el valor de todos los elementos del subárbol derecho es mayor que el valor de la raíz, y
  - tanto el subárbol izquierdo como el subárbol derecho.... son ABB también.

(9) Ejemplos de ABB

¿Son ABB?



No



Sí

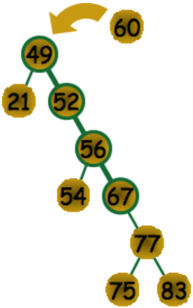
(10) Formalicemos dijo la abuela

- Recordemos la propiedad del ABB:
  - El valor de todos los elementos del subárbol izquierdo es menor que el valor de la raíz,
  - el valor de todos los elementos del subárbol derecho es mayor que el valor de la raíz, y
  - tanto el subárbol izquierdo como el subárbol derecho.... son ABB también.
- Esto mismo, escrito formalmente, es su invariante de representación:
- EsABB?: ab(α) $\rightarrow$ bool
- EsABB?(a) $\equiv$
  - nil?(a) $\vee_{\mathrm{L}}$
    - $\forall\ c : \alpha$, está?(c, izq(a)) $\Rightarrow$ c<clave(raíz(a)) $\wedge$
    - $\forall\ c : \alpha$, está?(c, der(a)) $\Rightarrow$ c>clave(raíz(a)) $\wedge$
    - EsABB?(izq(a)) $\wedge$
    - EsABB?(der(a))
- ¿Podríamos decir que EsABB?(bin(i, x, d)) $\equiv$ raíz(i)$< x \wedge$ raíz(d)$> x \wedge$ *EsABB*?(i) $\wedge$ *EsABB*?(d)

  No, recordar ejemplo anterior.

(11) Algoritmos de ABB

- vacío(): devolver un árbol nil.
- Búsquedas: recorremos el árbol desde la raíz y en cada paso decidimos si vamos a la izquierda o la derecha.
- definir(D, c, s) (definir en el diccionario D la clave c son el significado s).
- Veamos un ejemplo:



- Debemos buscar al padre del nodo a insertar e insertarlo ahí.
- Es decir, vamos bajando por el árbol hasta que llegamos a un padre al que le falta un hijo.

(12) Algoritmos de ABB (cont.)

idefinir(A, c, s) (definir en el ABB A la clave c son el significado s).

if nil?(A) then return bin(nil, $\langle c, s \rangle$, nil)

else

Llamemos I a izq(A)
Llamemos D a der(A)
Llamemos $\langle r_c, r_s \rangle$ a raíz(A)
if $c < r_c$ then return bin(idefinir(I, c, s), $\langle r_c, r_s \rangle$, D)
else return bin(I, $\langle r_c, r_s \rangle$, idefinir(D, c, s))
end if

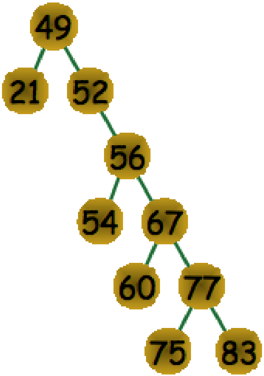
end if

(13) Costos

- Antes de analizar el borrado... ¿logramos mejores resultados que los que teníamos con arreglos y secuencias?
- Inserción:
  - Depende de la distancia del nodo a la raíz.
  - En el peor caso, $O(n)$.
  - En el caso promedio (suponiendo una distribución uniforme de las claves), $O(\log n)$.
- Búsqueda: ídem inserción.

(14) Ahora sí, el borrado

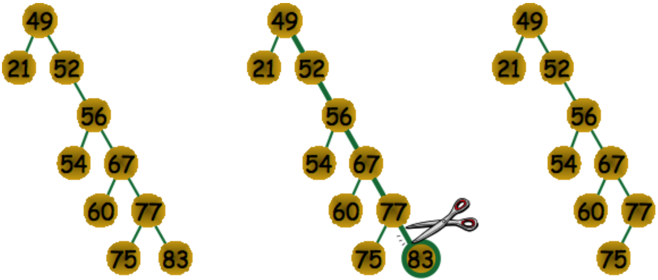
Analicemos cómo haríamos para borrar...



- ¿Cómo borramos el 54?
- ¿Cómo borramos el 52?
- ¿Cómo borramos el 67?
- Es decir borrar(A, e) depende de si
  - e es una hoja,
  - e tiene un sólo hijo, o
  - e tiene dos hijos.

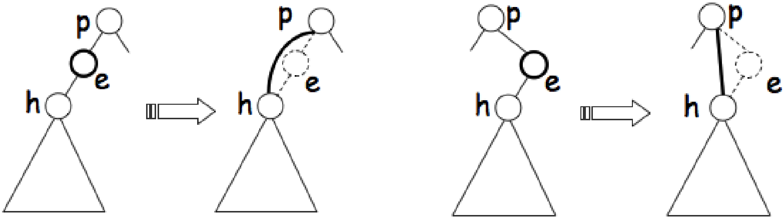
El algoritmo básico es muy sencillo:

- Buscamos al padre.
- Eliminamos la hoja.
- ¡Ojo! No tenemos forma de “retroceder” en la búsqueda.

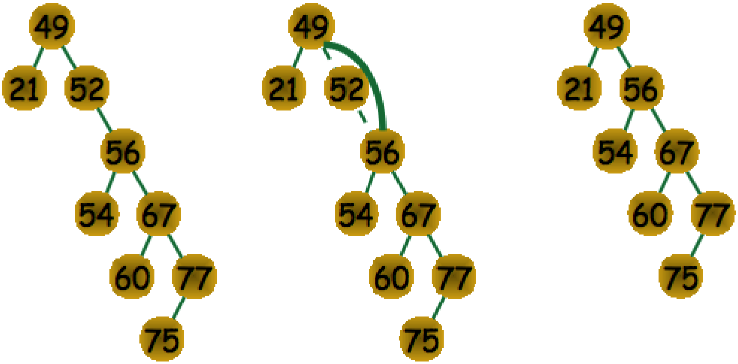


(16) Borrado de nodos con un solo hijo

- Llamemos p al padre del nodo e que estamos buscando.
- Llamemos h al único hijo de e.
- (p podría no existir si e fuese la raíz.)
- Si existe p, reemplazamos la conexión $\langle p, e \rangle$, con la conexión $\langle p, h \rangle$.

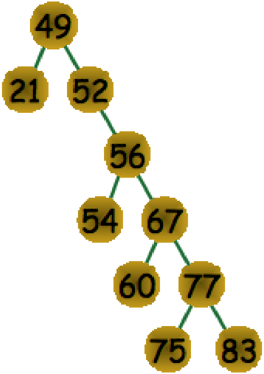


Veamos un ejemplo, borremos el 52:



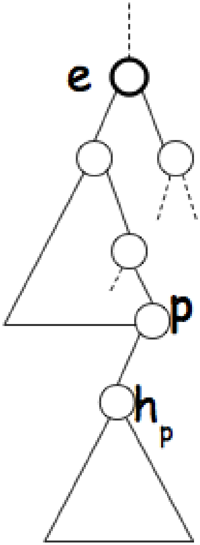
(18) Borrado de nodos con dos hijos

Analicemos el caso borrando al 67:



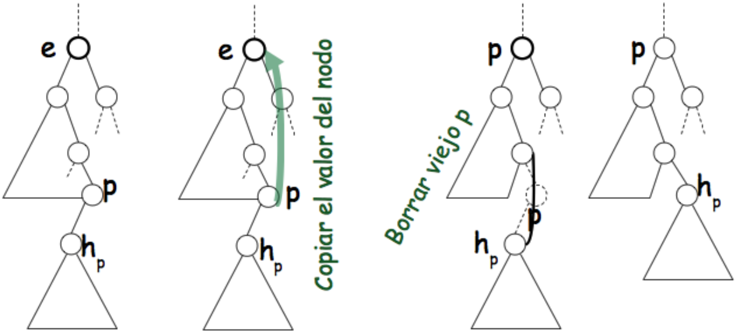
- Tenemos que poner algún nodo en el lugar del borrado.
- Si ponemos el 60, todo funciona.
- Ahora bien, ¿si el 60 tuviese como hijos al 58 y 62?
- ¿Puedo poner al 58? No.
- Pero sí al 62.
- ¿Y si el 62 a su vez tuviese hijos?

(19) Borrado de nodos con dos hijos (cont.)



- Es decir, podemos pensar que e tiene un "predecesor", que es el máximo elemento menor que e, y sería su antecesor si hiciésemos un inorder del árbol.
- Notemos que p no puede tener dos hijos, porque si no no sería el predecesor inmediato.
- Llamemos h_p al único hijo de p.
- Como p es el reemplazo perfecto para e, hay que poner su contenido en el lugar que antes ocupaba e.
- Ahora bien, p es una hoja o tiene un solo hijo. Es decir, volvemos a los casos anteriores.
- (También podemos hacer lo mismo en base al "sucesor", es decir, el mínimo elemento mayor que e.)

(20) Borrado de nodos con dos hijos (cont.)



(21) Costo del borrado en un ABB

- (Nodos *internos* son aquéllos que no son ni raíz ni hojas.)
- El borrado de un nodo interno requiere encontrar al nodo que hay que borrar y a su predecesor inmediato.
- En el caso peor ambos costos son lineales: $O(n) + O(n) = O(n)$

(22) Repasando

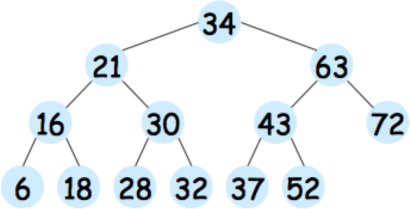
- Es decir, los ABB funcionan razonablemente bien en el caso promedio, pero no dan garantías.
- Nada impide caer en su peor caso, que sigue siendo lineal.
- Debemos notar que en ese sentido, los arreglos también son lineales en el peor caso pero ocupan menos memoria.
- ¿Podemos hacer algo más eficiente?

(23) ¿Y si estuviese balanceado?

- Todos los algoritmos que vimos tienen un peor caso lineal.
- Pero si miramos en detalle, más bien son $O(h)$, donde h es la altura del árbol.
- Si distribuyésemos los nodos del ABB de manera "pareja", de manera tal de que el árbol tuviese la mínima altura y estuviese siempre parejo, ¿qué altura tendría?
- Teorema: un árbol binario perfectamente balanceado de n nodos tiene altura $\lfloor \log_2 n \rfloor + 1$.

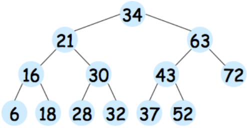
(24) Balanceo perfecto

- Teorema: un árbol binario perfectamente balanceado de n nodos tiene altura $\lfloor \log_2 n \rfloor + 1$.



- Supongamos que cada nodo tiene 0 o 2 hijos.
- Llamemos n_i a la cantidad de nodos internos (más la raíz) y n_h a la cantidad de hojas.
- Prop: Si $n > 1$, $n_h = n_i + 1$.
- Demo: caso base trivial. Supongamos que vale para n_h y n_i y agregamos dos nodos (uno no mantiene la propiedad). Las hojas aumentan en 1 ($n'_h = n_h - 1 + 2$) y los nodos internos también: $n'_i = n_i + 1$.
- Corolario: al menos la mitad de los nodos son hojas.

- Teorema: un árbol binario perfectamente balanceado de n nodos tiene altura $\lfloor \log_2 n \rfloor + 1$.



- Demo (esquema)
  - Sabemos que (1) $n = n_i + n_h$ (1) y (prop) si $n > 1$, $n_h = n_i + 1$.
  - Imaginemos que podamos las hojas: nos queda un árbol con las mismas propiedades, 1 menos de altura (llamémosla h), la mitad de los nodos y ahora todas las ramas de la misma longitud. ¿Cuántas veces más podemos podarlo?
  - Lo podemos pensar al revés: ¿cuánto niveles se pueden agregar desde el comienzo para tener un árbol de altura h?

(26) Balanceo perfecto (cont.)

- Al agregar un nivel la cantidad de nodos se duplica, porque $n'_h = n'_i + 1$, pero $n'_i = n$, entonces $n'_h = n + 1$. Reemplazando en (1) nos queda que $n' = n + (n + 1) + 1$.
- Entonces $n = 1\ .\underbrace{2 \ldots 2}_{h \text{ veces}} = 2^h = 2^{\log_2 n}$.
- Por ende, $h = \log_2 n$ y la altura del árbol era $h + 1$.
- Detalles de la demo, en el libro.
- Nota: este resultado es generalizable a árboles k-arios.

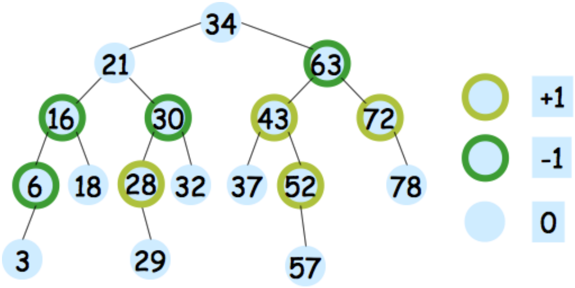
(27) ¿Árboles perfectamente balanceados?

- Si tuviésemos árboles perfectamente balanceados todas nuestras operaciones serían $O(\log n)$.
- ¿Pero podemos?
- Es muy costoso mantener el balanceo perfecto.
- Sin embargo, podemos tener un balanceo "casi" perfecto, haciendo que todas las ramas tengan "casi" la misma longitud.
- Ese "casi", lo vamos a interpretar de la siguiente manera: la longitud entre dos ramas cualesquiera de un nodo difiere a lo sumo en 1.
- Notemos que nuestros algoritmos deberían garantizar que sucesiones de inserciones y/o borrados no destruyan ese balance. O mejor dicho, que lo reestablezcan.

- Conozcamos a los AVLs:
  - Árboles Valanceados Lateralmente?
  - Árboles de Validada Longitud?
  - Adel'son-Vel'skii & Landis?
- Un árbol se dice balanceado en altura si las alturas de los subárboles izquierdo y derecho *de cada nodo* difieren en a lo sumo una unidad.
- G. Adel'son-Vel'skii & E. M. Landis (1962). "An algorithm for the organization of information". Proceedings of the USSR Academy of Sciences 146: 263-266.

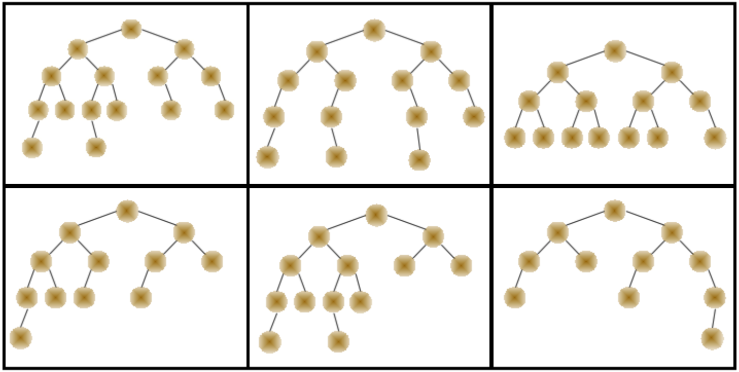
(29) Factor de balanceo

- Para cada nodo se calcula el factor de balanceo (FdB):



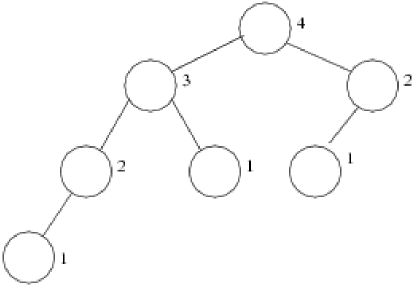
- FdB = altura del subárbol derecho - altura del subárbol izquierdo.
- En un AVL, $\forall n$: nodo, $|FdB(n)| \leq 1$

(30) ¿Cuáles son AVL?



(31) El peor de todos

- ¿Cuál es el peor AVL de todos? O mejor dicho, el más desbalanceado, pero que sigue siendo AVL.
- Definamos a P_h como el peor AVL de altura h.
- P_0 es el árbol vacío, P_1 tiene un solo nodo.
- Ejemplo de P_4 (nodos indican altura):



- Para $h > 1$ tenemos que P_h tiene una raíz y dos subárboles, P_{h-1} y P_{h-2}.

(32) El peor de todos (cont.)

- ¿Cuántos nodos tiene P_h?
- Primero contemos las hojas: P_0 tiene 0, P_1 tiene 1, y luego tenemos ... la sucesión de Fibonacci: $f_h = f_{h-1} + f_{h-2}$.
- Por eso, a los P_h se los llama también árboles de Fibonacci.
- Y que la cantidad de nodos es los internos + las hojas.
- Es decir, que P_h tiene f_h+algo nodos, donde ese algo es $\leq f_h$.
- Como f_h crece exponencialmente con h, eso significa que la altura de P_h, que es h, crece logarítmicamente con la cantidad de nodos de P_h ($\sim f_h$+algo).
- Pero P_h es el "peor" AVL posible y aún así su altura es logarítmica en n.
- De hecho, Adel'son-Vel'skii & Landis demostraron que un árbol de Fibonacci con n nodos tiene altura $<$ $1{,}44 \log_2 (n + 2) - 0{,}328$.
- Por ende, un AVL con n nodos tiene altura $\Theta(\log n)$.

(33) En la próxima...

- Hoy vimos:
  - Diccionarios en base a arreglos y secuencias.
  - Sus limitaciones.
  - Introducción a las estructuras arbóreas.
  - ABBs.
  - Intro a los AVLs.
- En la próxima clase vamos a aprender los algoritmos que permiten mantener a los AVLs balanceados.