Diseño de Conjuntos y Diccionarios

Representación de Conjuntos y Diccionarios

- **TAD Diccionario(clave, significado)**
- **Observadores básicos**
- def?: clave c x dicc(clave, significado) d $\rightarrow$ bool
- obtener: clave c $\times$ dicc(clave, significado) d $\rightarrow$ significado (def?(c, d))
- **Generadores**
- vacío: $\rightarrow$ dicc(clave, significado)
- definir: clave $\times$ sign $\times$ dicc(clave, significado) $\rightarrow$ dicc(clave, significado)
- **Otras Operaciones**
- borrar: clave c $\times$ dicc(clave, significado) d $\rightarrow$ dicc(clave, significado) (def?(c, d))
- claves: dicc(clave, significado) $\rightarrow$ conj(clave)
- $\cdot =_{dicc} \cdot$: dicc(α) $\times$ dicc(α) $\rightarrow$ bool

Ejemplos de Conjuntos y Diccionarios

- Un diccionario “clásico” de la lengua: clave son palabras, significado sus significados
- Un diccionario castellano-inglés: clave son palabras en castellano, significado sus traducciones al inglés
- El diccionario del corrector ortográfico: claves son palabras en castellano, significado…
- El “predictivo”: clave son strings, significados las posibles palabras que…
- El padrón electoral, con clave el DNI, significado el nombre y el domicilio
- En el sistema de la facu, el padrón de alumnos, con clave # de libreta, significado la lista de materias aprobadas
- En un sistema operativo, o en un sistema web, clave los usernames, significado (una versión encriptada de) la password.

Representación secuencial de conjuntos y diccionarios

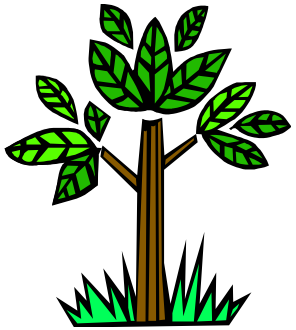
- Conjuntos y diccionarios pueden representarse a través de estructuras secuenciales.

- Intenten hacer Uds. mismos el ejercicio de escribir INV, ABS, y los algoritmos

Representación secuencial de conjuntos y diccionarios

- Complejidad de las operaciones
  - Tiempo: O(n) en el peor caso
  - Espacio: O(n).
  - ¿se podrá hacer mejor?

Representación de conjuntos y diccionarios a través de Arboles Binarios

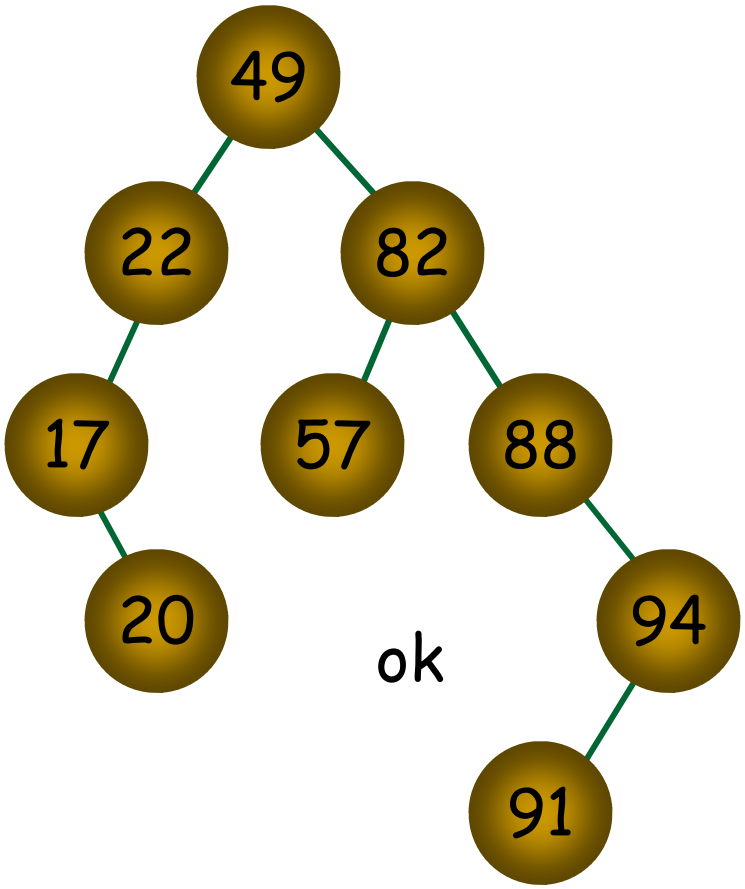


- ¿Podríamos representar conjuntos o diccionarios a través de árboles binarios?
- Claro que podríamos
- ¿Ganaríamos algo? No demasiado en principio ¿no?
- Pero.....

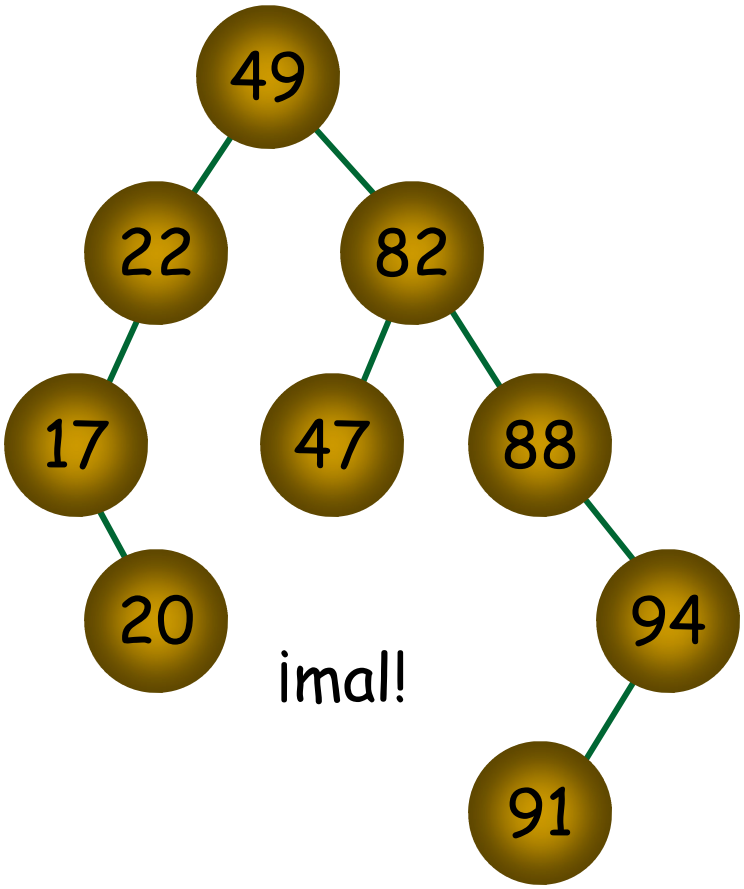
Representación de conjuntos a través de ABB

- ¿Qué es un Árbol Binario de Búsqueda?
- Es un AB que satisface 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.

Ejemplos

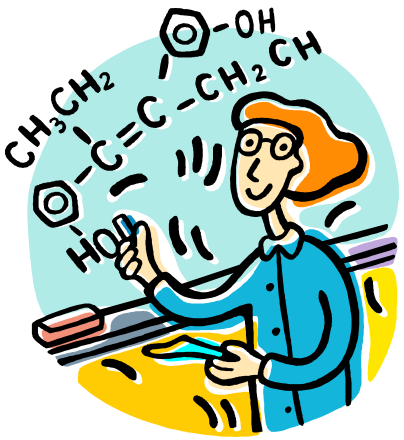


ok



imal!

Un cacho de formalismo, che!



- Invariante de Representación:

ABB: ab(nodo) $\rightarrow$ boolean

$\forall$e: ab(nodo)

ABB(e) $\equiv$ Nil?(e) $\vee_L$

[{($\forall$c: clave: Está(c, Izq(e)) $\Rightarrow$ (c $<_{clave}$ LaClave(Raíz(e)))) $\wedge$

($\forall$ c: clave: Está(c, Der(e)) $\Rightarrow$ (c $>_{clave}$ LaClave(Raíz(e))))**}** $\wedge$

ABB(Izq(e)) $\wedge$

ABB(Der(e))**]**

- ¿Función de abstracción? ¡Ejercicio!

Los algoritmos para ABB

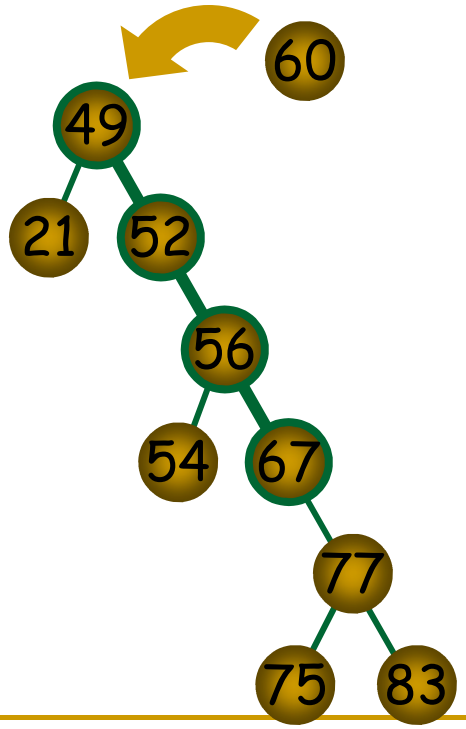
- vacío()
  - nil
- definir(c,s,A)
  - Case
    - A=nil then bin(nil,(c,s),nil) else
    - A=bin(L,(r_c,r_s),D) then

      if c<r_c then bin (definir(c,s,L), (r_c,r_s),D)

      else bin(L, (r_c,r_s),definir(c,s,D))

Los algoritmos para ABB

- O sea:
  - Buscar al padre del nodo a insertar
  - Insertarlo como hijo de ese padre



Los algoritmos para ABB

- **Costo de la inserción:**
  - **Depende de la distancia del nodo a la raiz**
- **En el peor caso**
  - O(*n*)
- **En el caso promedio (suponiendo una distribución uniforme de las claves):**
  - O(lg *n*)

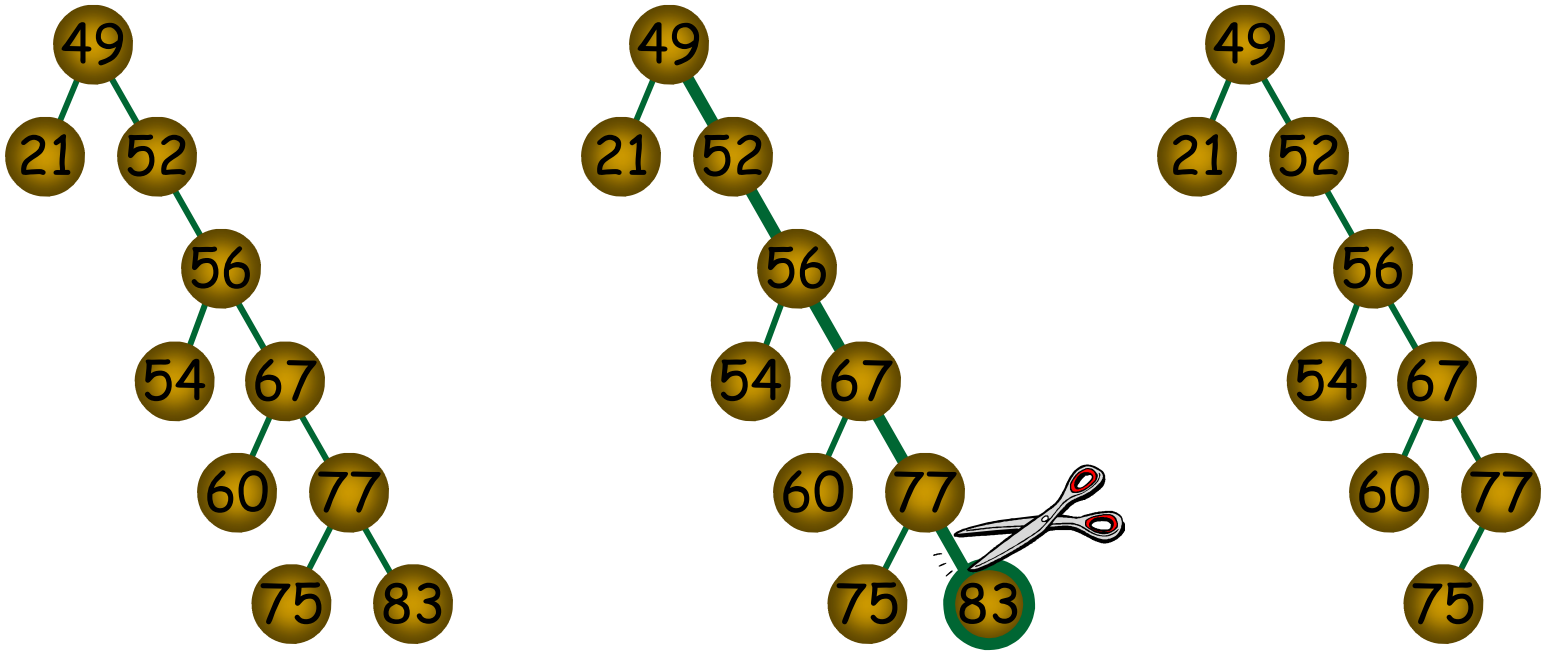
Los algoritmos para ABB: borrado

- Borrar(u,A)
- Tres casos
  1. u es una hoja
  2. u tiene un solo hijo
  3. u tiene dos hijos

- Vamos a ver la idea, Uds. la pueden formalizar luego

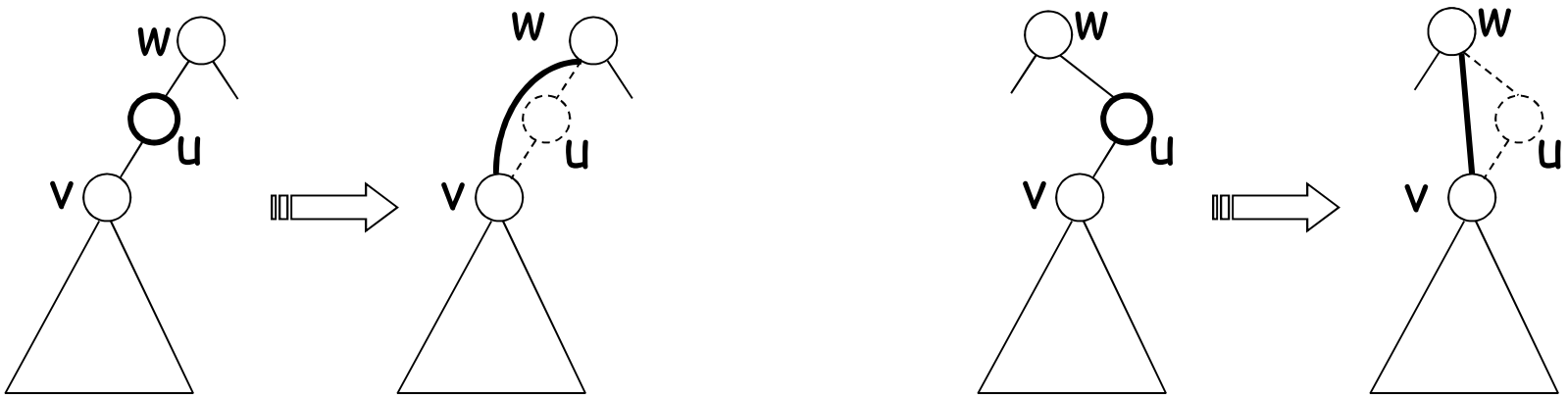
Borrado en ABB

1. **Borrar una hoja**
   - Buscar al padre
   - Eliminar la hoja



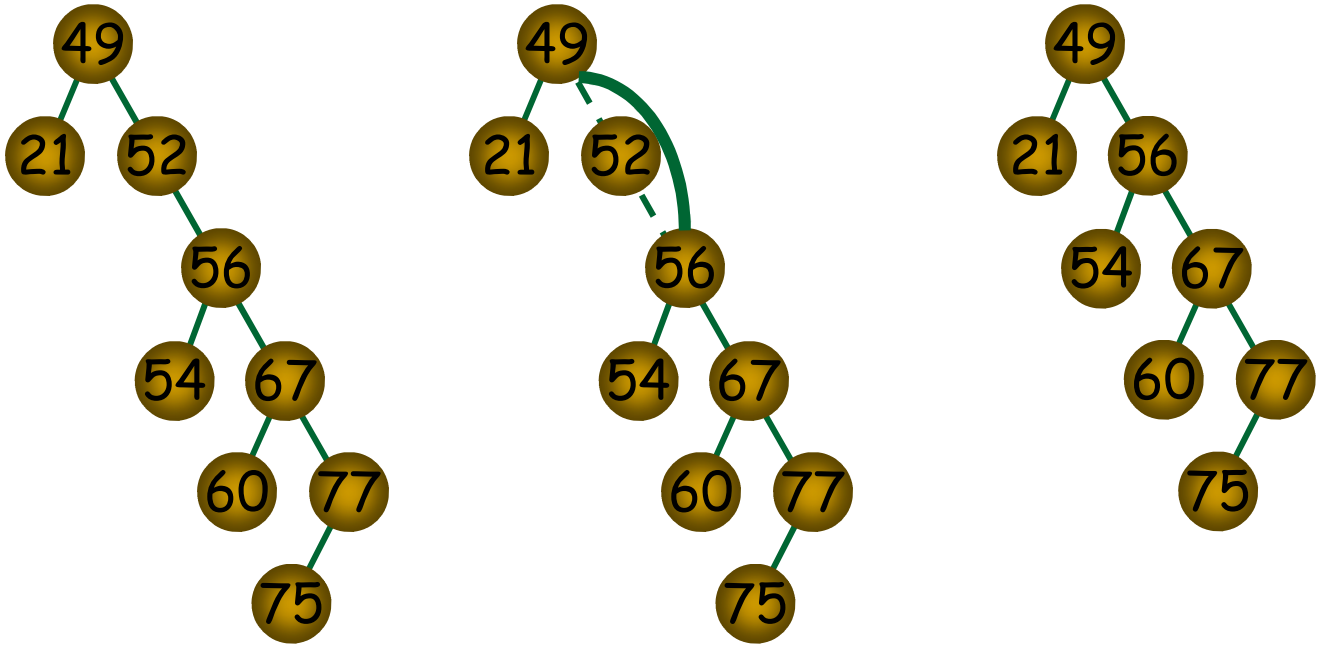
Borrado en ABB

2. **Borrar un nodo u con un solo hijo v**
   - Buscar al padre w de u
   - Si existe w, reemplazar la conexión (w,u) con la conexión (w,v)



Borrado en ABB

Ejemplo del caso 2

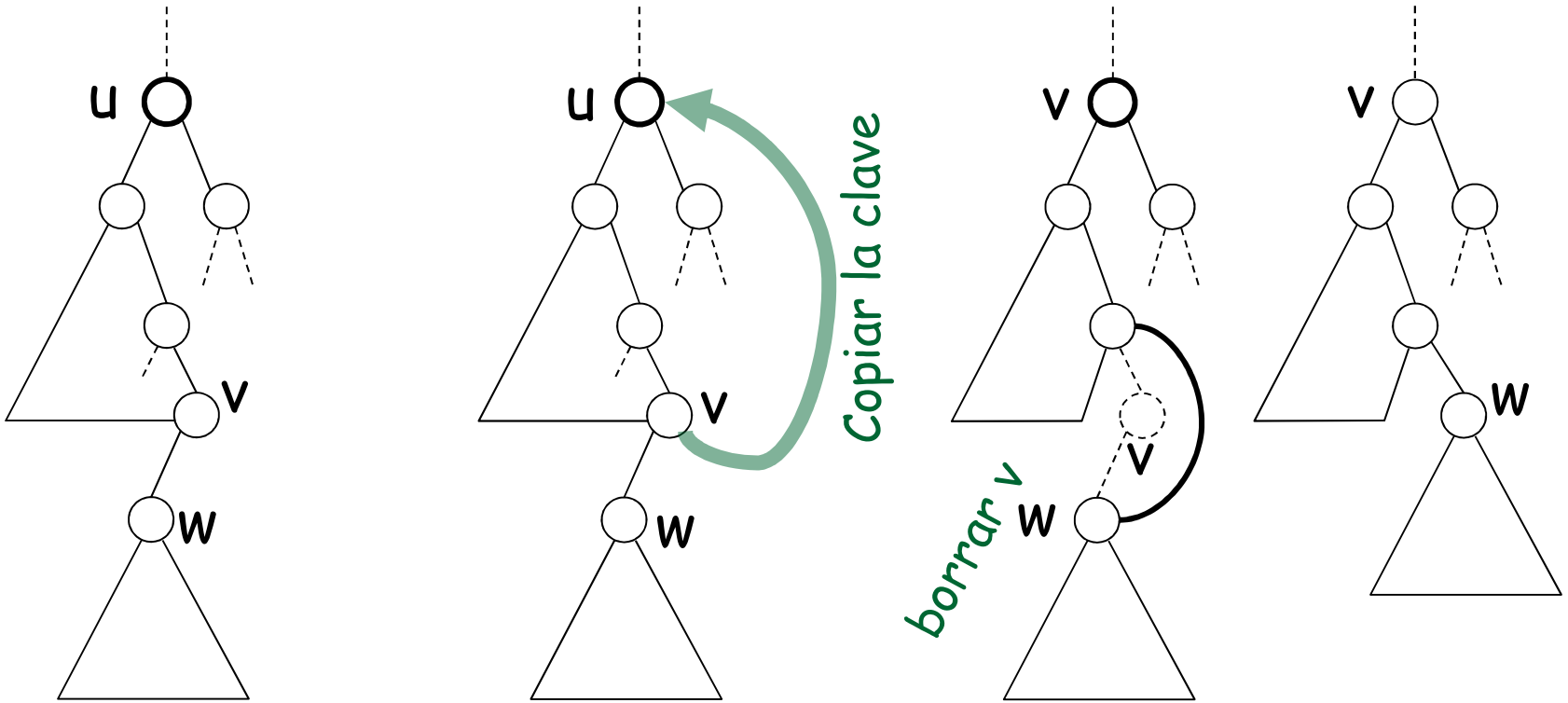


Borrado en ABB

3. **Borrado de un nodo u con dos hijos**

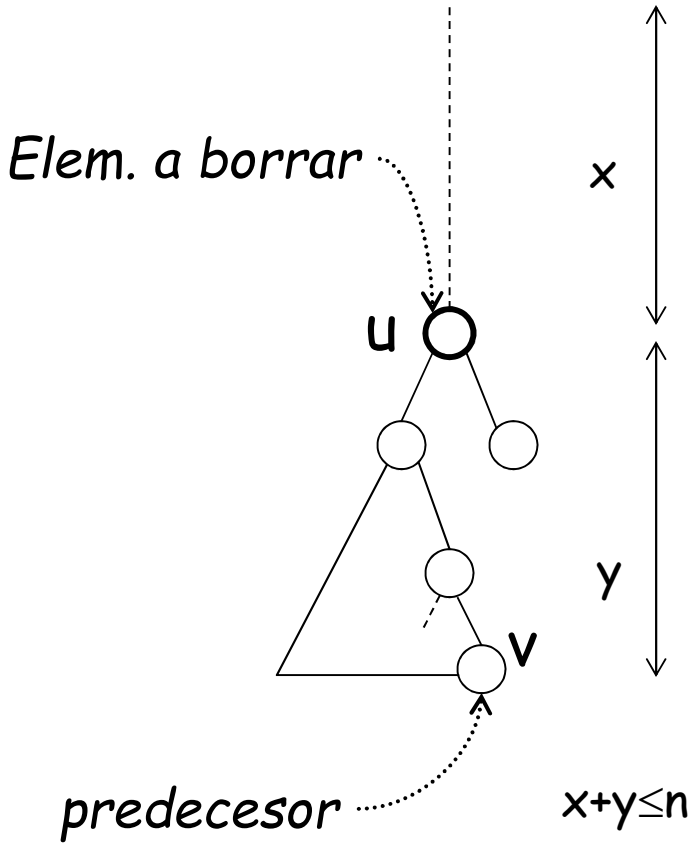
- Encontrar el “predecesor inmediato” v (o sucesor inmediato) de u
  - v no puede tener dos hijos, en caso contrario no sería el predecesor inmediato (sucesor)
- copiar la clave de v en lugar de la de u
- Borrar el nodo v
  - v es hoja, o bien tiene un solo hijo, lo que nos lleva los casos anteriores

Borrado en ABB



Costo del borrado en un ABB

- 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)



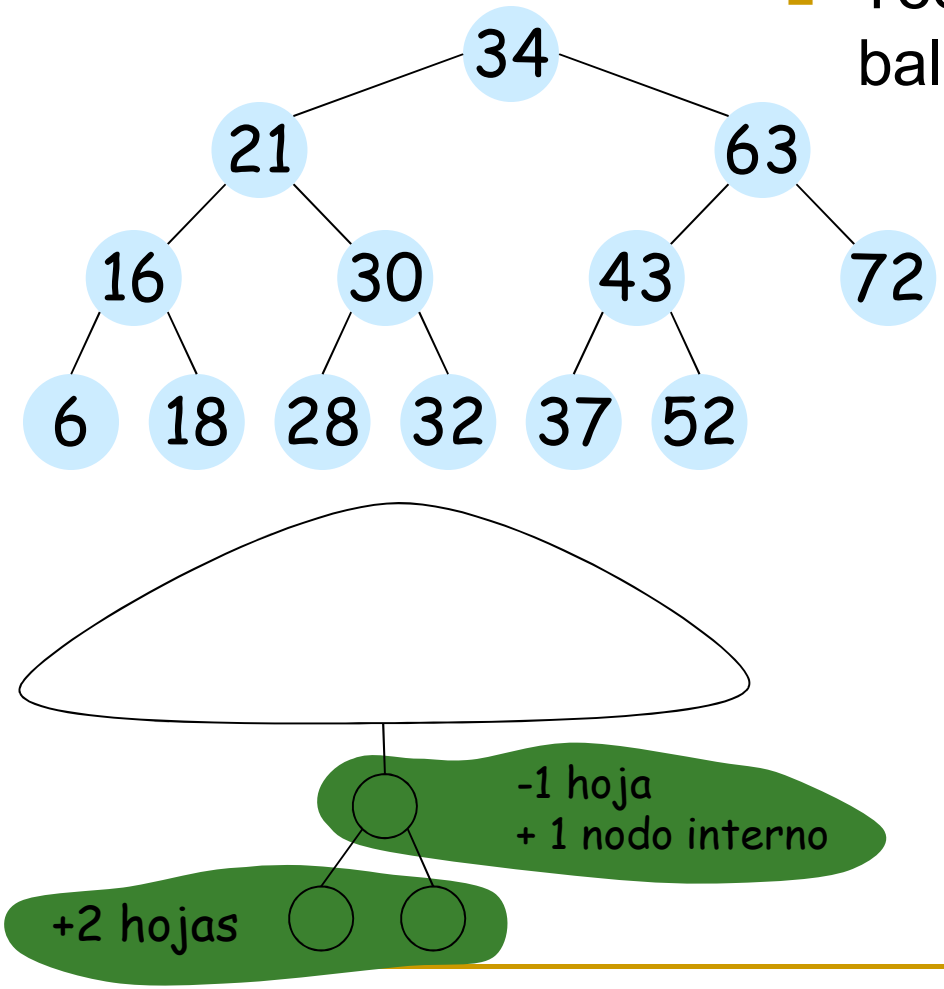
Representación de conjuntos y diccionarios a través de AVL

- Todas las representaciones vistas hasta ahora tienen al menos una operación de costo linear en función de la cantidad de elementos
- En muchos casos, eso puede ser inaceptable
- ¿Habrá estructuras más eficientes?

Introducción al balanceo

- ¿Qué altura tiene un árbol completo?
- Pero...no podemos pretender tener siempre árboles completos (¡mantenerlos completos sería demasiado caro!)
- Quizás con alguna propiedad más débil...
- Noción intuitiva de balanceo
  - Todas las ramas del árbol tienen “casi” la misma longitud
  - Todos los nodos internos tienen “muchos” hijos
- Caso ideal para un arbol *k*-ario
  - Cada nodo tiene 0 o *k* hijos
  - La longitud de dos ramas cualesquiera difiere a lo sumo en una unidad

balanceo perfecto



- Teo: Un árbol binario perfectamente balanceado de *n* nodos tiene altura

$$\lfloor \lg_2 n \rfloor + 1$$

Dem: Si cada nodo tiene 0 o 2 hijos

$n_h = n_i + 1$

n_h = # hojas

n_i = # nodos internos

$n = n_h + n_i$

$n_h' = n_h + 2 - 1 = n_h + 1 = n_i + 1 + 1 = n_i' + 1$

¡Las hojas son más del 50% de los nodos¡

balanceo perfecto/2

- "Podamos" el árbol eliminando primero las hojas de las ramas más largas
- Luego podamos todas las hojas, nos queda otro árbol con las mismas características (cantidad de hijos por nodo y balanceo)
- ¿Cuántas veces podemos "podar" el árbol?
- Fácilmente generalizable a árboles k-arios

$$n_h = (k-1)n_i + 1 \quad \Rightarrow \quad n_h = \frac{(k-1)n+1}{k}$$

- costo de búsqueda/inserción/borrado O(log *n*)
- Pero....sucesiones de inserciones y borrados pueden destruir el balanceo!
  - Degradación de la performance