Algoritmos y Estructuras de Datos I

Facultad de Ciencias Exactas y Tecnología
Universidad Nacional de Tucumán
2023

UNIVERSIDAD NACIONAL DE TUCUMÁN
facet
FACULTAD DE CIENCIAS EXACTAS Y TECNOLOGÍA

Búsqueda(1)

Unidad V - Contenidos

Búsqueda.

El tipo abstracto de datos tabla.

Técnicas básicas de búsqueda: Búsqueda secuencial, Búsqueda binaria. Búsqueda en árboles binarios: en árboles binarios de búsqueda, en árboles binarios de búsqueda balanceados. Análisis de la altura de cada tipo de árbol y costo de las operaciones. Búsqueda en árboles generales: árboles multivías, árboles B. Algoritmos y complejidad.

Dispersión. Funciones de dispersión. Colisión, soluciones. -

Búsqueda

La búsqueda es una operación fundamental, intrínseca a una gran cantidad de tareas de las computadoras, que consiste en recuperar uno o varios elementos particulares de un gran volumen de información previamente almacenada.

Normalmente se considera que la información está almacenada en registros, cada uno de los cuales posee una clave para utilizar en la búsqueda.

Cada elemento x_i será de la forma de registro con los siguientes campos:

clave_i	información asociada a la clave_i

Hay una ***clave de búsqueda*** asociada a cada registro, que se usa para diferenciar unos de otros.

El objetivo de la operación de búsqueda es ***encontrar todos los registros cuyas claves coincidan con una cierta clave de búsqueda***, con el propósito de acceder a la información asociada a esa clave para su procesamiento.

Búsqueda

Las estructuras de datos que se utilizan a menudo para almacenar los registros para búsqueda son: *diccionarios* y las *tablas de símbolos.*

Las operaciones básicas de estas estructura de datos son:

- Inicializar la estructura de datos.
- Buscar un registro con una dada clave.
- Agregar un registro.
- Borrar un registro.

Se supone que hay **n** registros almacenados en una tabla.

Generalmente se requiere que haya **n claves distintas**, de modo que cada clave pueda identificar el registro. Si para una tabla existe un conjunto de claves que sea único, a ese conjunto se le llama **clave primaria** (no existen dos registros con el mismo valor de la clave primaria) .

Búsqueda

En algunas aplicaciones las claves no son siempre únicas.

Los registros con **claves iguales** se pueden tratar de varias formas, según las necesidades de la aplicación.

- Mantener la idea de clave primaria. En ese caso se puede asociar a una clave la lista enlazada de registros con la misma clave. De ese modo todos los registros con la misma clave se pueden encontrar en una sola búsqueda.
- También se puede poner todos los registros con la misma clave en una estructura de datos primaria y una búsqueda retornará cualquiera de ellos.
- Otra posibilidad es suponer que cada registro tiene un identificador único (aparte de la clave) y entonces la búsqueda va a encontrar el registro que tiene el identificador dado, conociendo la clave.
- Otra posibilidad es que el programa de búsqueda llame a una función específica para cada registro que contenga la clave dada.

TABLA(CLAVE,INFORMACION)

Especificación algebraica - OPERACIONES:

A) Sintaxis:

TABVACIA **:** → TABLA

ESTABVACIA **:** TABLA → BOOLEAN

INSERTAR **:** TABLA x CLAVE X INFORMACION → TABLA

BORRAR **:** TABLA x CLAVE → TABLA

PERTENECE **:** TABLA x CLAVE → BOOLEAN

BUSCAR **:** TABLA x CLAVE → INFORMACION U {indefinido}

TABLA(CLAVE,INFORMACION)

B) Semántica: $\forall$ T$\in$ TABLA, $\forall$ a,b $\in$ CLAVE, $\forall$ i $\in$ INFORMACION

ESTABVACIA(TABVACIA) $\equiv$TRUE
ESTABVACIA(INSERTAR (T,a,i)) $\equiv$ FALSE

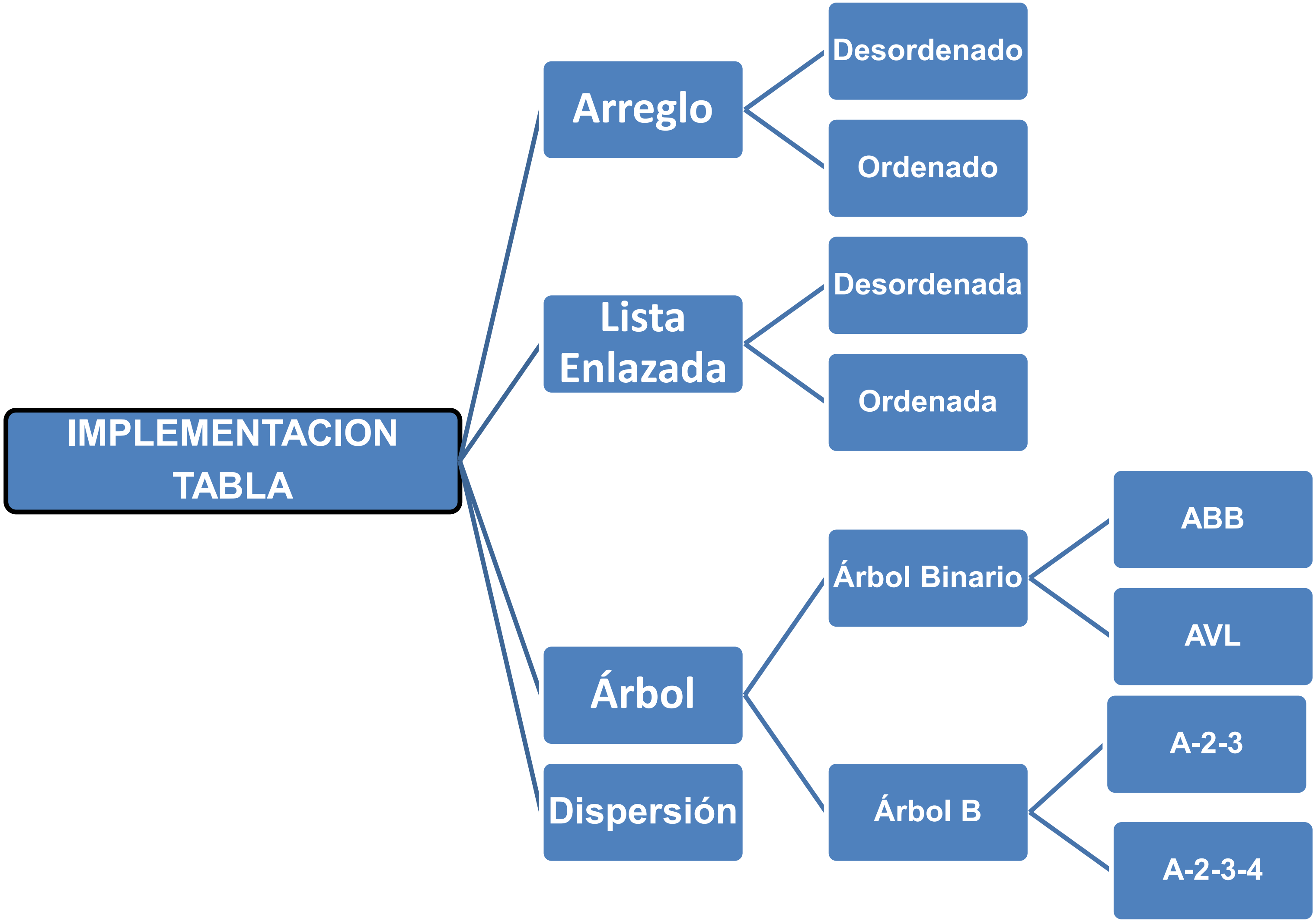
BORRAR(TABVACIA,a) $\equiv$ TABVACIA
BORRAR(INSERTAR (T,a,i),b) $\equiv$ si a **=** b entonces T
sino INSERTAR (BORRAR(T,b),a,i)

PERTENECE(TABVACIA,a) $\equiv$ FALSE
PERTENECE(INSERTAR(T,a,i),b) $\equiv$ si a **=** b entonces TRUE
sino PERTENECE(T,b)

BUSCAR(TABVACIA,a) $\equiv$ indefinido
BUSCAR(INSERTAR(T,a,i),b) $\equiv$ si a **=** b entonces i
sino BUSCAR(T,b)

*Nota : **=** es el operador que permite comparar si 2 claves son iguales*

Adt TABLA


IMPLEMENTACION TABLA
Arreglo
Desordenado
Ordenado
Lista Enlazada
Desordenada
Ordenada
Árbol
Árbol Binario
ABB
AVL
Árbol B
A-2-3
A-2-3-4
Dispersión

Costo de operación

IMPLEMENTACIÓN del Adt TABLA con:	***BUSCAR***	***INSERTAR***	***BORRAR***
Estructura Lineal:			
Arreglo desordenado	O(n)	O(1)	O(n)
Arreglo ordenado	O($\log_2$n)	O(n)	O(n)
Lista desordenada	O(n)	O(1)	O(n)
Lista ordenada	O(n)	O(n)	O(n)
Arbol:			
ABB			
AVL			
A 2-3			
A 2-3-4			
Árbol B			
Tabla:			
Dispersión			

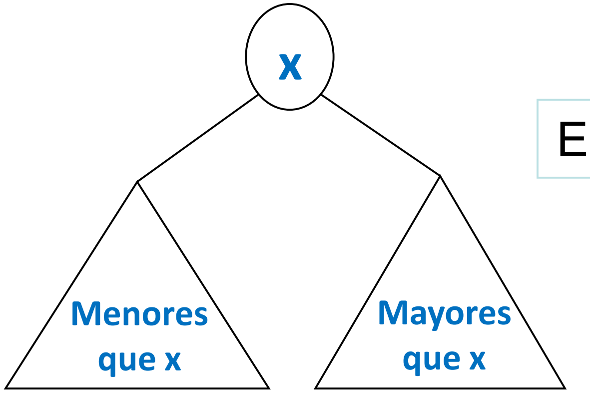
Árbol de Búsqueda Binario(ABB)
(bst: binary search tree)

Un ABB es un árbol binario con una clave en cada uno de sus nodos. El conjunto de claves está formado por elementos en que se puede definir una relación de orden.

Dado un nodo con clave **x**:

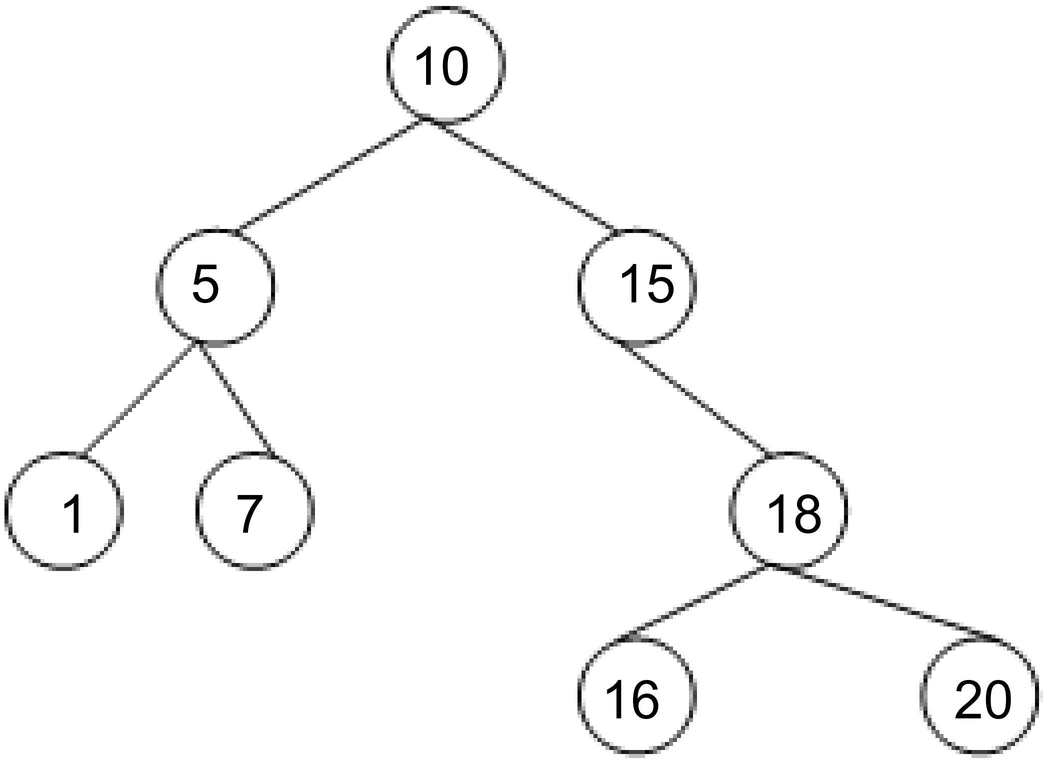
- Las claves almacenadas en los nodos del subárbol izquierdo del mismo, son todas ***menores*** que la clave **x.**
- Las claves almacenadas en los nodos del subárbol derecho son ***mayores*** que la clave **x**.

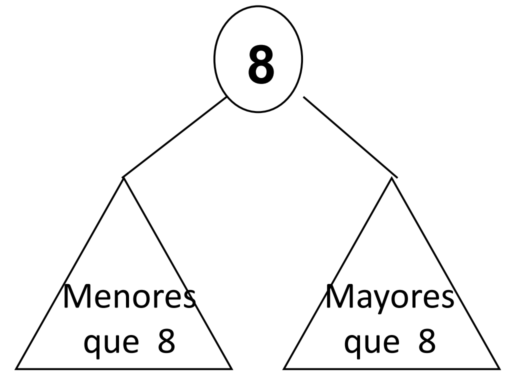
Esta propiedad se mantiene para todos los nodos del árbol.



En un ABB ***no hay nodos con claves repetidas***.

Árbol de Búsqueda Binario
Ejemplos





Árbol de Búsqueda Binario ABB(ITEM)

Especificación algebraica - OPERACIONES:

Sintaxis:

	ABBVACIO	: → ABB
*	ARMARABB	: ABB X ITEM X ABB → ABB
	ESABBVACIO	: ABB → BOOLEAN
	IZQUIERDO	: ABB → ABB
	RAIZ	: ABB → ITEM U {indefinido}
	DERECHO	: ABB → ABB
	PERTENECE	: ABB X ITEM → BOOLEAN
	INSERTAR	: ABB X ITEM → ABB

* *Constructora escondida*

Árbol de Búsqueda Binario ABB(ITEM)

SEMANTICA: $\forall$ i,d $\in$ ABB, $\forall$x1,x2 $\in$ITEM

ESABBVACIO(ABBVACIO) $\equiv$TRUE
ESABBVACIO(ARMARABB(i,x1,d)) $\equiv$ FALSE

IZQUIERDO(ABBVACIO) $\equiv$ ABBVACIO
IZQUIERDO(ARMARABB(i,x1,d)) $\equiv$ i

RAIZ(ABBVACIO) $\equiv$ indefinido
RAIZ(ARMARABB(i,x1,d)) $\equiv$ x1

DERECHO(ABBVACIO) $\equiv$ ABBVACIO
DERECHO(ARMARABB(i,x1,d)) $\equiv$ d

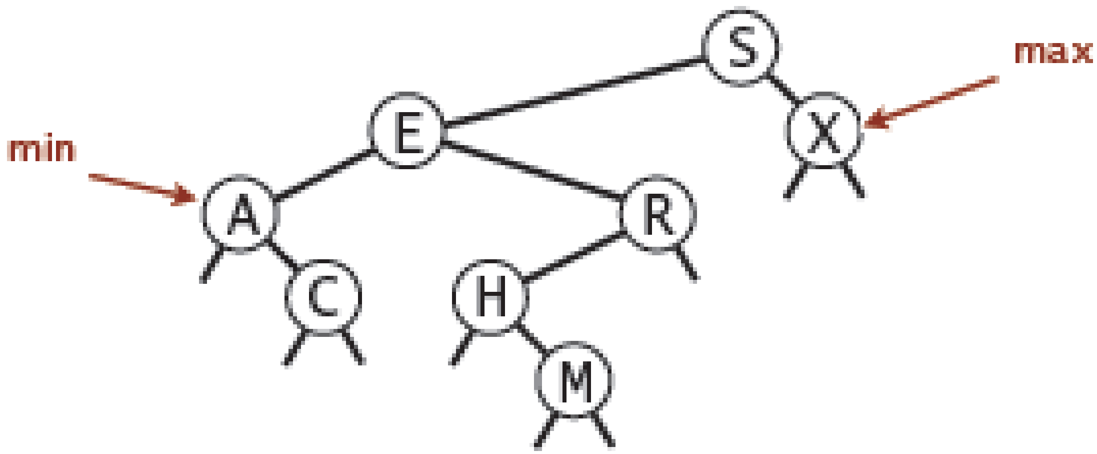
Árbol de Búsqueda Binario ABB(ITEM)

SEMANTICA: $\forall$ i,d $\in$ ABB, $\forall$x1,x2 $\in$ITEM

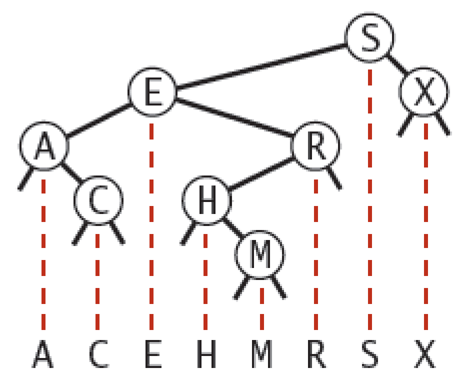
PERTENECE(ABBVACIO,x1) $\equiv$ FALSE
PERTENECE(ARMARABB(i,x1,d),x2) $\equiv$
si x1=x2 entonces TRUE
sino si x1<x2 entonces PERTENECE(d,x2)
sino PERTENECE(i,x2)

INSERTAR(ABBVACIO,x1) $\equiv$ ARMARABB(ABBVACIO,x1,ABBVACIO)
INSERTAR(ARMARABB(i,x1,d),x2) $\equiv$
si x1=x2 entonces ARMARABB(i,x1,d)
sino si x1<x2 entonces ARMARABB(i,x1,INSERTAR(d,x2))
sino ARMARABB(INSERTAR(i,x2),x1,d)

Ejemplo de ABB



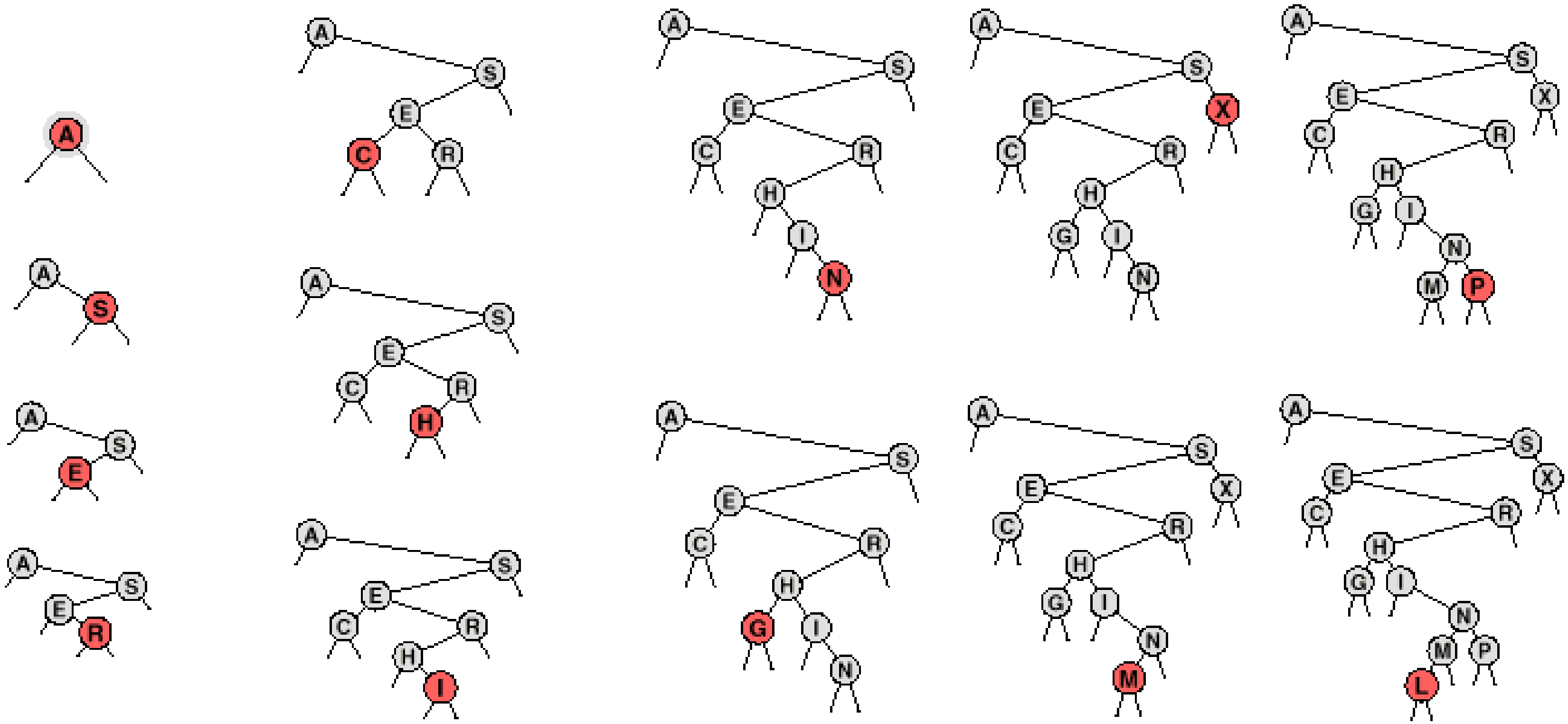
- Recorrido en orden simétrico =<ACEHMRSX>



- Observación: en un ABB no hay nodos con rótulos repetidos

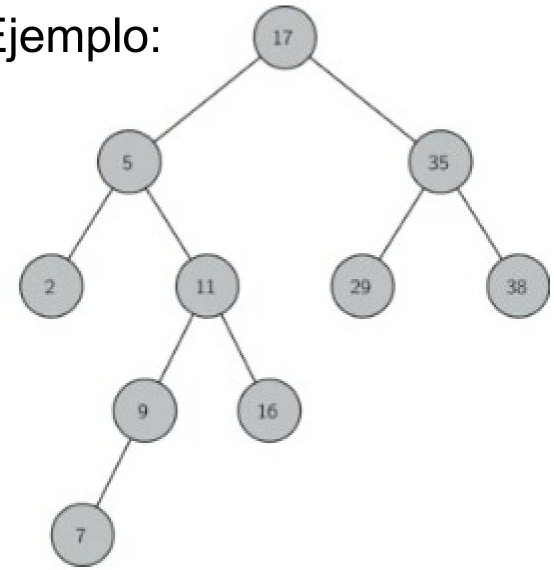
Buscar e Insertar en un ABB

Ejemplo: Insertar las letras: **A S E R C H I N G X M P L**



Borrar en un ABB

Ejemplo:

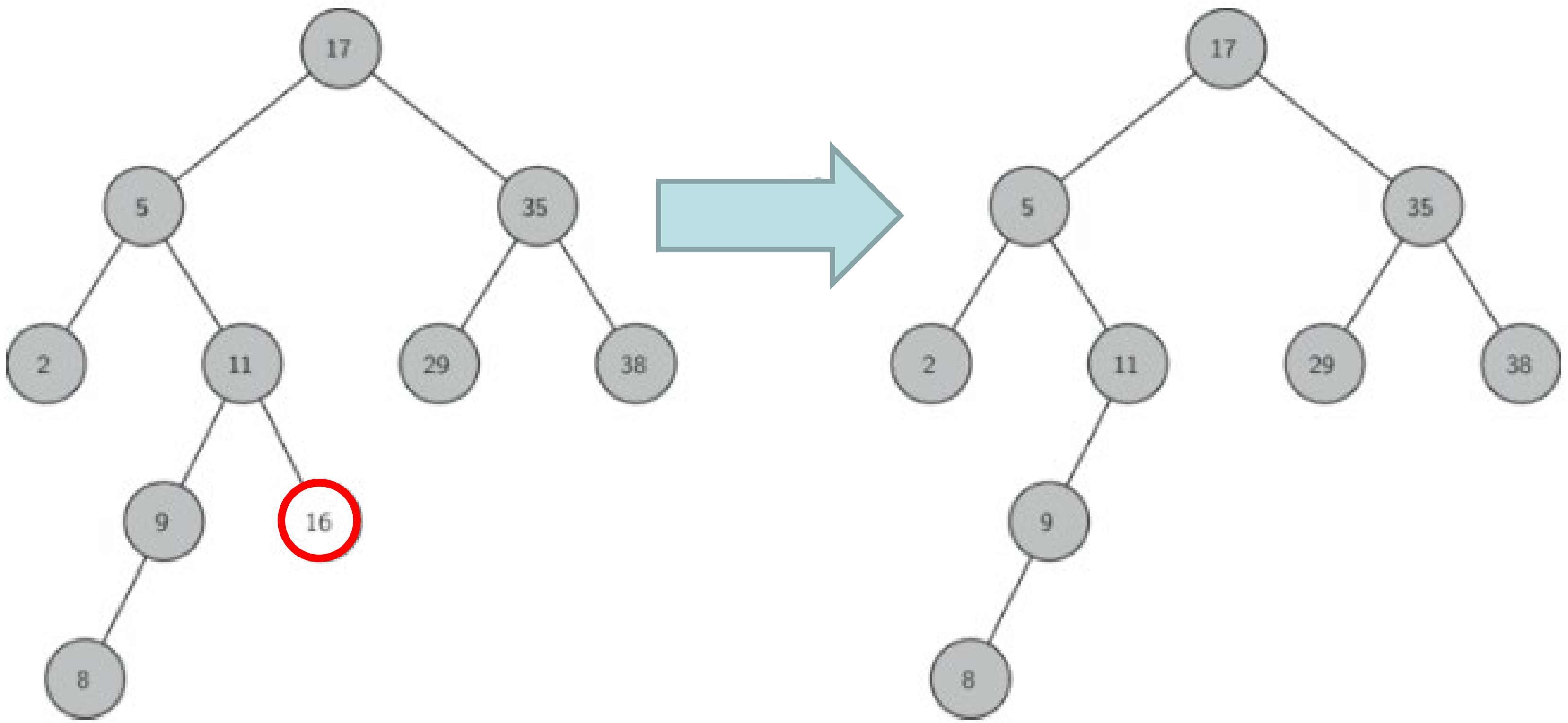


Borrar

- 1) nodo sin hijos (nodo hoja)
- 2) nodo con un solo subárbol
- 3) nodo con dos subárboles
  - Predecesor inmediato
  - Sucesor inmediato

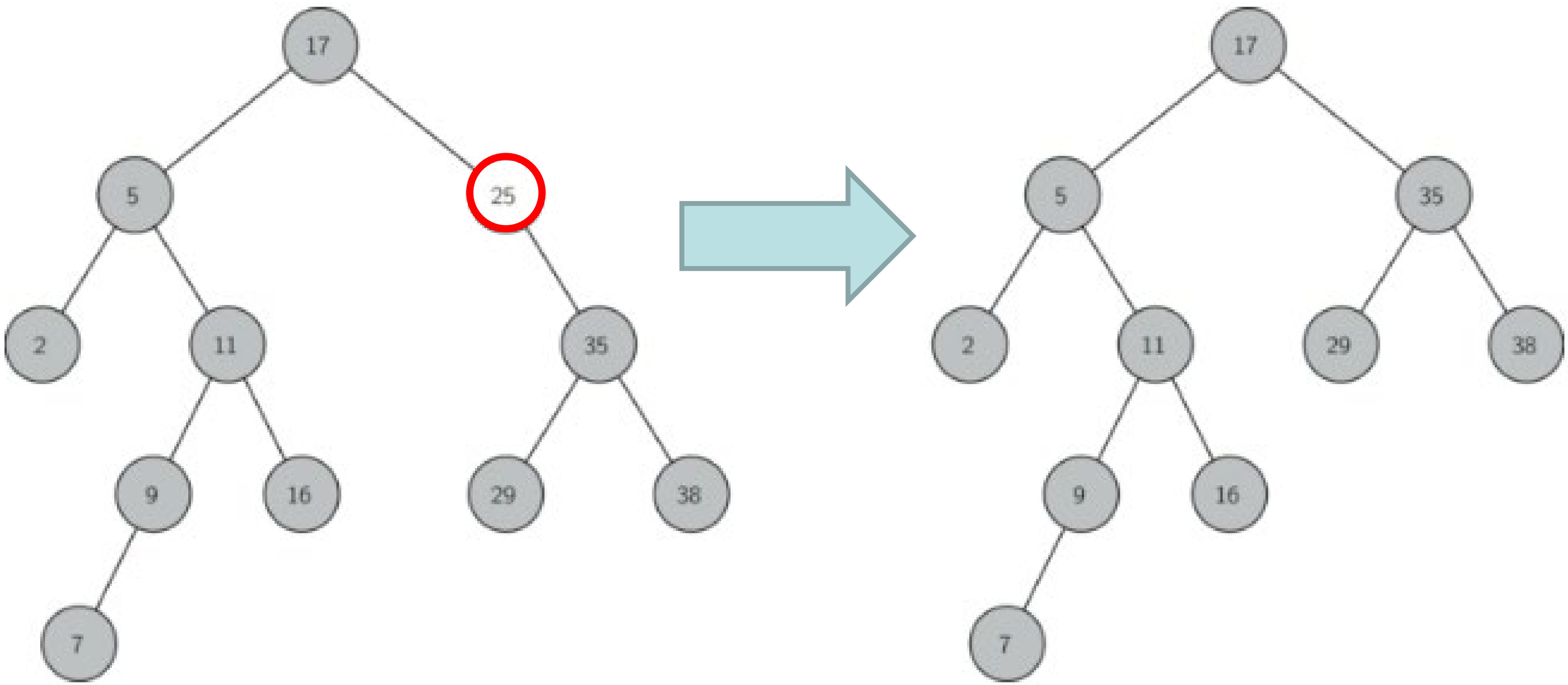
Borrar en un ABB

1) Borrar un nodo sin hijos ó nodo hoja:



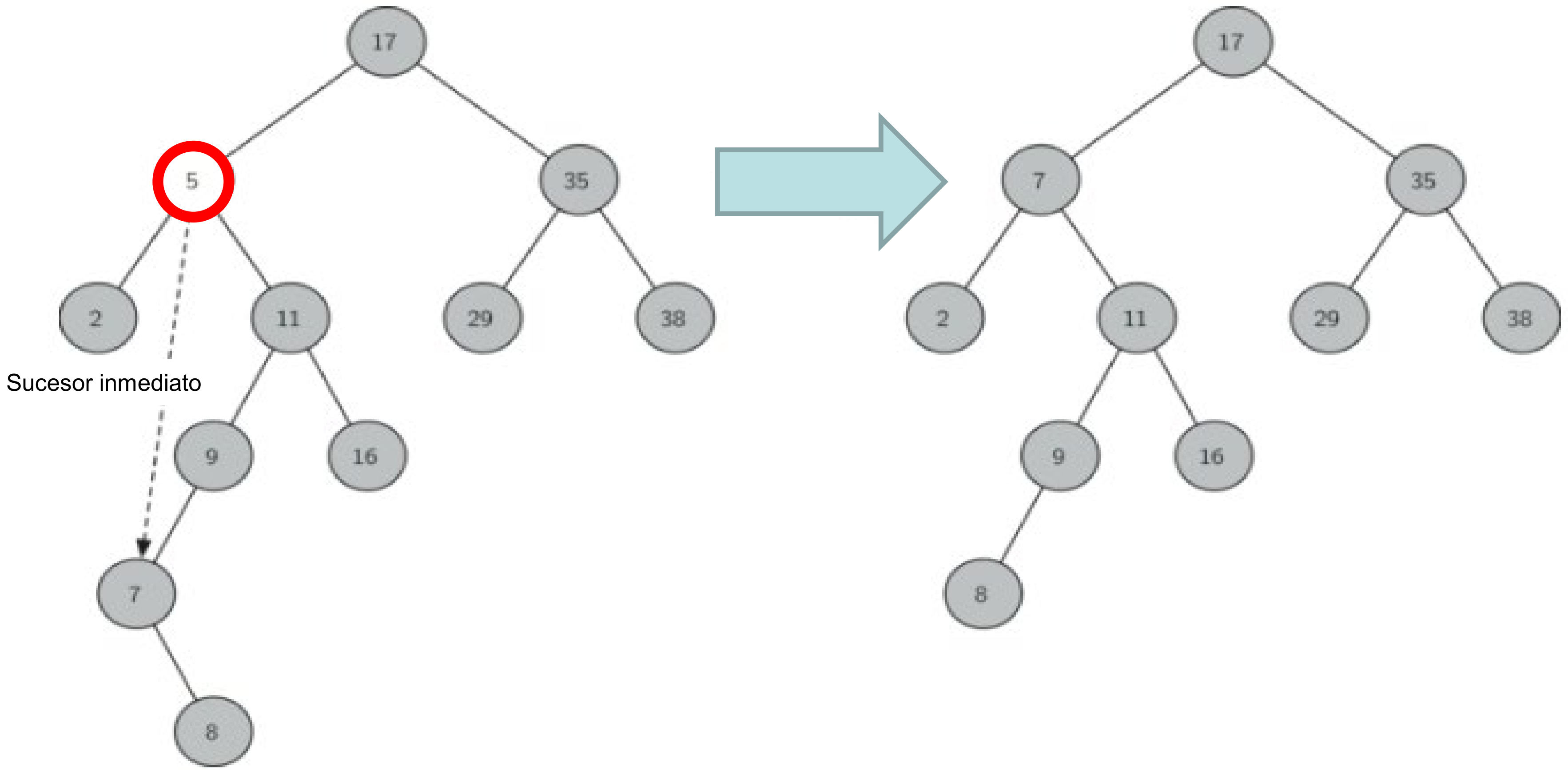
Borrar en un ABB

2) Borrar un nodo con un solo subárbol:



Borrar en un ABB

3) Borrar un nodo con dos subárboles con la estrategia de Sucesor inmediato:



Borrar en un ABB

Función que suprime el mínimo de un abb y devuelve su valor:

FUNCION SUPMIN (a): Tipoabb → Tipoabb x Tipoclave

```
    SI ESABBVACIO(IZQUIERDO(a)) ENTONCES
            valor ← RAIZ(a)
            a ← DERECHO(a)
            RETORNA valor
    SINO
            RETORNA SUPMIN(IZQUIERDO(a))
FIN
```

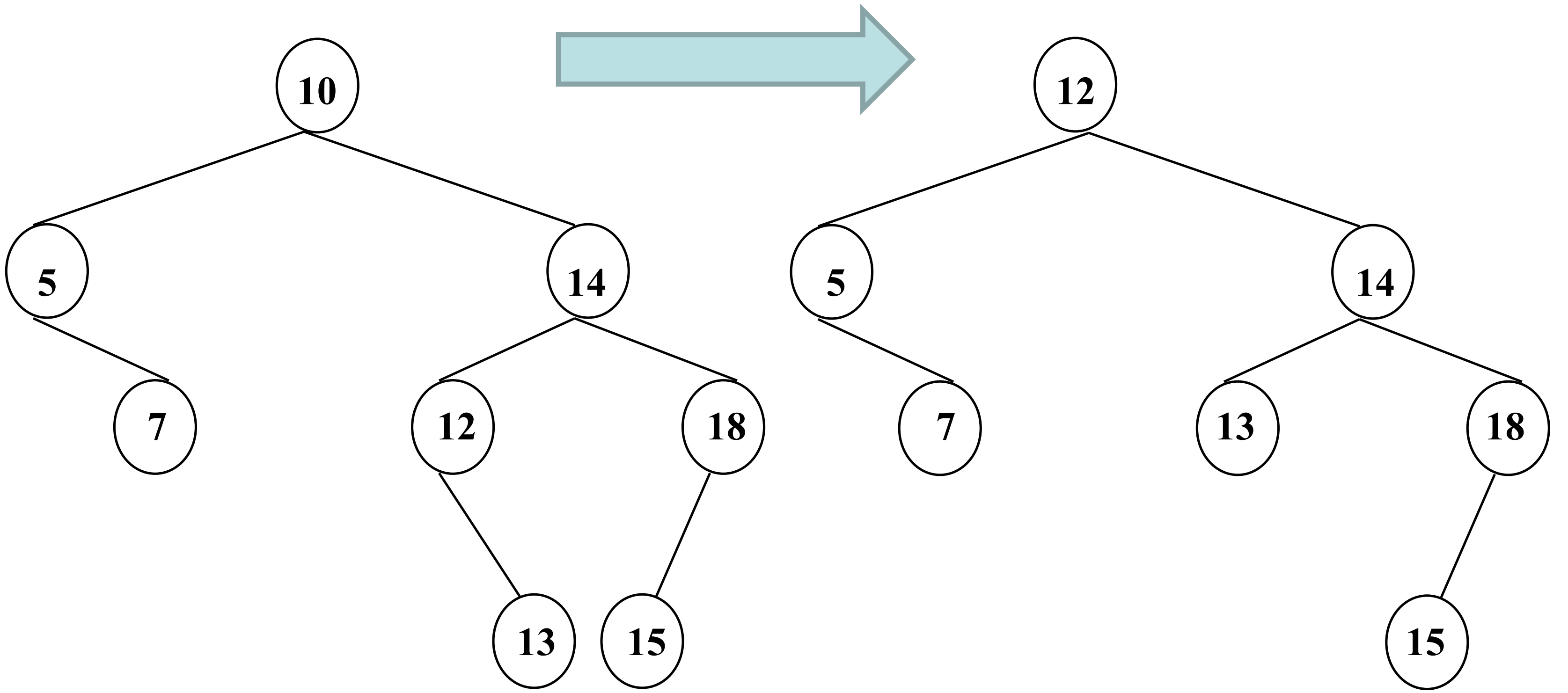
Borrar en un ABB

FUNCION BORRAR (a,x): Tipoabb x Tipoclave → Tipoabb

```
SI NOT ESABBVACIO(a) ENTONCES
   SI PERTENECE(a,x) ENTONCES     // x esta en el arbol
           SI x < RAIZ(a) ENTONCES     // x esta en el subarbol izquierdo
                       BORRAR(IZQUIERDO(a),x)
           SINO SI x > RAIZ(a) ENTONCES)  // x esta en el subarbol derecho
                       BORRAR(DERECHO(a),x)
            SINO // x es la raiz del arbol
               SI ESABBVACIO(IZQUIERDO(a)) AND ESABBVACIO(DERECHO(a))  ENTONC
                       a ← ABBVACIO   // es hoja
               SINO SI ESABBVACIO(IZQUIERDO(a)) ENTONCES
                       a ← DERECHO(a)    // tiene un solo hijo el derecho
               SINO SI ESABBVACIO(DERECHO(a)) ENTONCES
                       a ← IZQUIERDO(a)    // tiene un solo hijo el izquierdo
               SINO  // tiene 2 hijos
                       valor ← SUPMIN(DERECHO(a))
                       ARMARABB(IZQUIERDO(a), valor, DERECHO(a))
FIN
```

Borrar en un ABB

Ejemplo: BORRAR(a,10)



Borrado perezoso en ABB (lazy deletion)

Si se espera pocas operaciones de borrado, se puede usar esta popular estrategia.

Cuando un elemento se tiene que borrar, se lo deja en el árbol y se lo marca como borrado.

En la búsqueda se agrega un control adicional para detectar los nodos borrados.

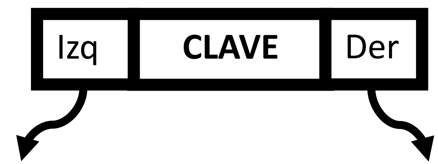
Si se reinsertara la clave se evita el trabajo de armar un nuevo nodo para ella.

Esta estrategia funciona bien cuando el número real de nodos del ABB es igual al número de nodos borrado, en este caso se puede esperar que la altura del mismo sólo aumente en 1.

Si se necesita eliminar los espacios ocupados por los nodos borrados, se puede reconstruir periódicamente la tabla excluyendo dichos nodos.

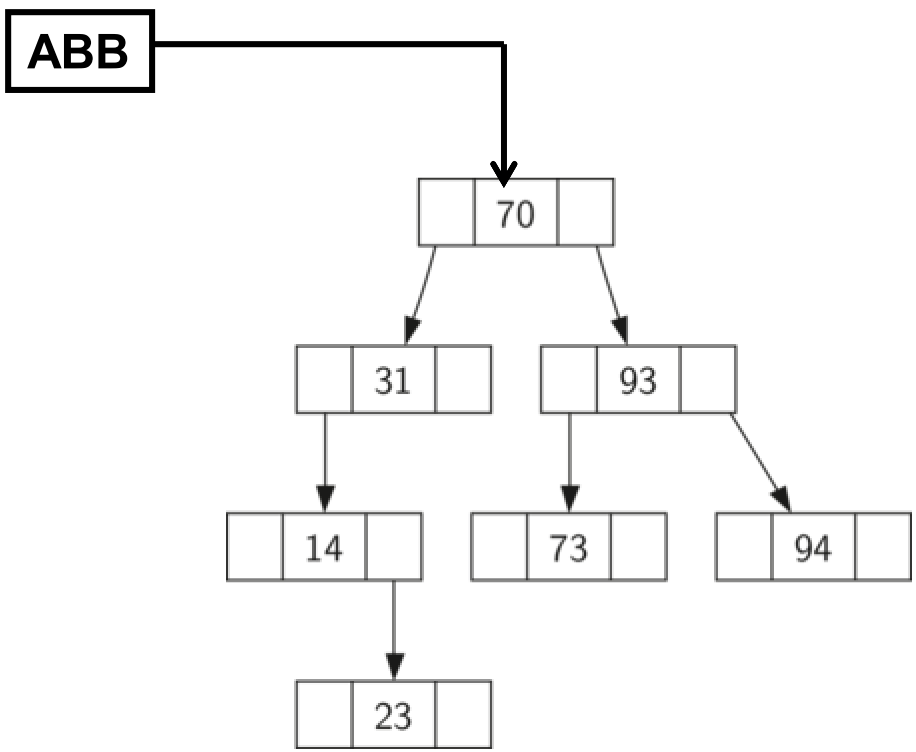
Implementación de ABB

Los ABB se implementan con nodos con clave y 2 punteros de la forma:



Un ABB es un puntero a uno de estos nodos.

Ejemplo:

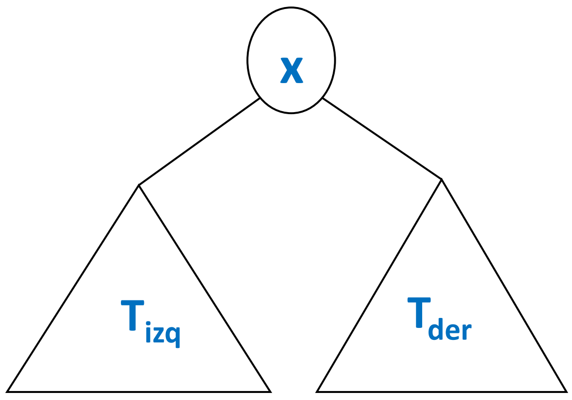


Altura de un ABB

Definición de altura:

Sea T un abb y sean T_{izq} y T_{der} subárboles de T

- $h(T) = 0$ si el árbol es vacío
- $h(T) = 0$ si es árbol hoja
- $h(T) = 1+maximo(h(T_{izq}),h(T_{der}))$ si contiene más nodos

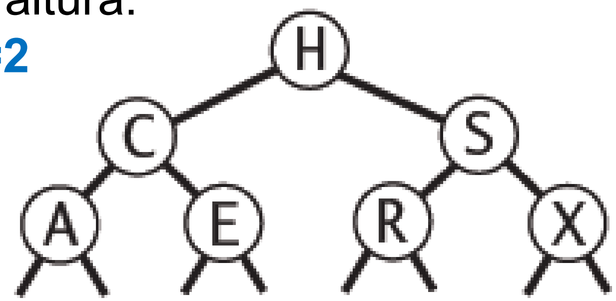


Distintas formas de ABB

Ejemplo: ABB , Número de claves n = 7

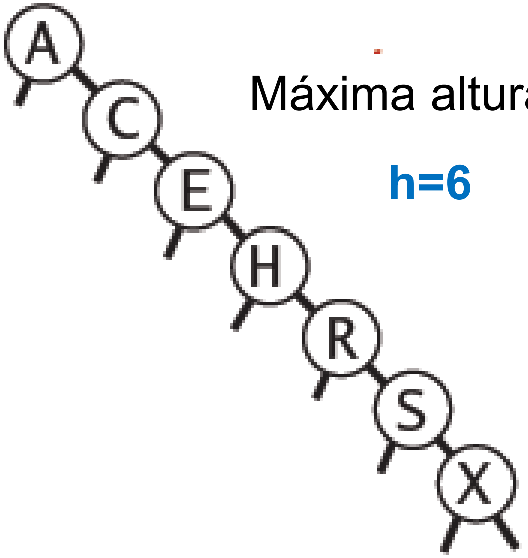
Minima altura:

h=2



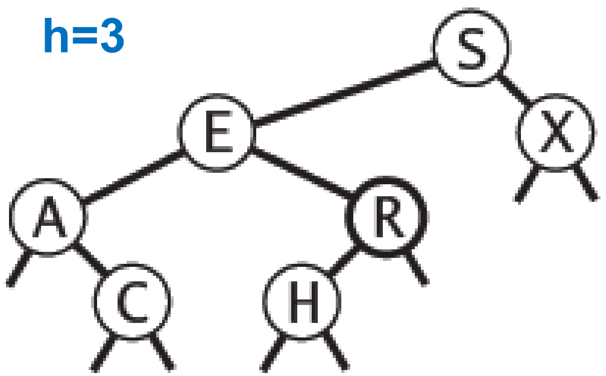
Máxima altura:

h=6



Otro caso:

h=3



Altura de un ABB

Dadas **n claves** ubicadas en **n** nodos de un árbol binario de búsqueda:

$$h_{minimo} = \lfloor \mathbf{\log_2 (n+1)} \rfloor \mathbf{- 1}$$

$$h_{maximo} = \mathbf{n-1}$$

De modo que la altura h del ABB se encuentra en el siguiente rango:

$$\lfloor \mathbf{\log_2 (n+1)} \rfloor \mathbf{- 1 \leq h \leq n - 1}$$

Entonces:

$$h_{minimo} \ \mathbf{\epsilon\ O(\log_2 n)}$$

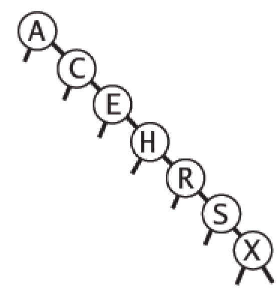
$$h_{maximo} \ \mathbf{\epsilon\ O(n)}$$

Costo de las operaciones en ABB

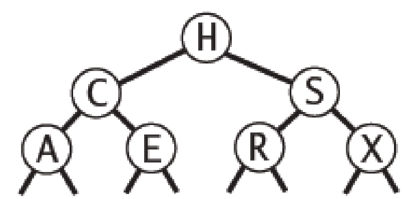
La altura del ABB depende del orden de inserción de las claves:.

Si las n claves se insertar en el árbol binario de búsqueda usando la operación INSERTAR, la estructura del árbol depende del orden en que se inserten las claves.

- Si las claves se insertan en orden (o en orden inverso) el árbol resultante tiene todos sus subárboles derechos (o izquierdos) vacíos. El ABB tiene la mayor altura posible para n claves.



- Si las claves se insertan de manera que la mitad son menores que la clave que se inserta y ya se insertaron y la otra mitad a insertar son mayores que la clave, se alcanza un árbol balanceado.



Costo de las operaciones en ABB

Puesto que la altura del árbol determina el número máximo de comparaciones en una ABB, la forma del árbol es muy importante.

- **Peor caso**: Si las n claves se ingresan en forma ordenada (ascendente o descendente) el ab será una lista y las operaciones seran de $O(n)$.

- **Caso medio**: Si las n claves se presentan en forma aleatoria, es más frecuente que resulten árboles balanceados, de menor altura. En promedio el tiempo de las operaciones será $O(\log_2 n)$.

- **Mejor caso**: Si el árbol binario de búsqueda de n claves es completo o casi completo, cada una de las operaciones toman $O(\log_2 n)$ ya que ningún camino tiene longitud mayor que esta.

Árbol de búsqueda óptimos

Un ABB que ***minimice el número esperado de comparaciones*** para un conjunto dado de claves y probabilidades se llama **árbol de búsqueda óptimo**.

No existe un algoritmo eficiente para construir un árbol óptimo en el caso general, algunas de las técnicas son: método de balanceo, método exhaustivo, árbol de división, árbol de división por medianas.

Como es costoso armar y mantener un ABB óptimo se trata de buscar una estructura menos estricta que tenga también un buen tiempo de búsqueda.

En lugar de trabajar con ABB óptimos se usan **ABB balanceados en altura**, que se comportan tan bien como el árbol de búsqueda perfectamente equilibrado, y se pueden construir y mantener con algoritmos mas eficientes.

Arboles balanceados en altura – AVL

Los AVL son ABB balanceados con respecto a la altura de subárboles.

Definición de árbol AVL:

- Un árbol vacío T es balanceado
- Si T es un árbol no vacío, con sus subárboles T_{izq} y T_{der} con alturas $\mathbf{h_{izq}}$ y $\mathbf{h_{der}}$ respectivamente, entonces T es balanceado si:

  a) $\left| \mathbf{h_{izq} - h_{der}} \right| \leq \mathbf{1}$,

  donde $h_{izq} - h_{der}$ son las alturas de T_{izq} y T_{der} respectivamente.

  b) $\mathbf{T_{izq}}$ **y** $\mathbf{T_{der}}$ **son árboles balanceados**.

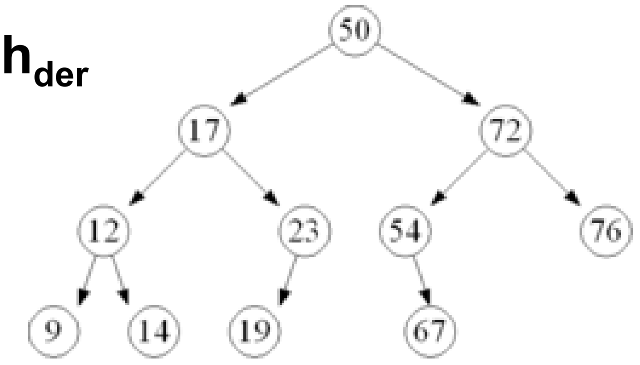
Georgy Maximovich Adelson-Velsky and Yevgeniy Mikhailovich Landis, "*An algorithm for the organization of information*" , DANSSSR, 146, 263-233, 1962

Arboles balanceados en altura – AVL

Factor de balance:

El factor de balance de un nodo de un árbol binario se define como la diferencia de la altura de su subárbol izquierdo menos la altura de su subárbol derecho:

$$\mathbf{FB(nodo) = h_{izq} - h_{der}}$$



Para cada nodo T en un árbol AVL se cumple que FB(T) = -1, 0, 1

Altura:

Sus autores demostraron que un árbol AVL de **n** nodos su **altura** es:

$$\log_2(n+1)-1 \leq h \leq 1.44 \log_2(n+2)-0.328$$

Arboles balanceados en altura – AVL

Operación de inserción:

El peor caso de inserción es **$O(\log_2 n)$**.

La **inserción** de un nodo en un AVL puede desbalancearlo. Para rebalancearlo se haces rotaciones que preservan la condición de AVL. Las rotaciones que sólo involucran algunos cambios de punteros pueden ser a izquierda, a derecha, simples o dobles. En promedio se requiere una rotación en el 46.5% de las inserciones, lo que significa que en término medio se necesita reequilibrar el árbol cada dos inserciones aproximadamente.

Si la altura del árbol antes de la inserción es h, entonces el tiempo para insertar un nuevo nodo es O(h). Esto es lo mismo que para los ABB no balanceados pero ahora hay más trabajo (las rotaciones). En el caso de los AVL la altura puede ser a los sumo $O(\log_2 n)$.

Arboles balanceados en altura – AVL

Operación de borrado:

Para el **borrado** de un nodo se puede demostrar que es posible encontrar y borrar nodos de un árbol binario de búsqueda balanceado en tiempo **$O(\log_2 n)$**.

En general será una operación más complicada que la inserción, a pesar de que la operación de reequilibrado es esencialmente la misma que en el caso de inserción. Hay una diferencia esencial que existe entre los procedimientos de inserción y de borrado. Mientras al realizar una inserción de una sola clave, se puede producir como máximo una rotación (de dos o tres nodos), el borrado puede requerir una rotación en todos los nodos del camino de búsqueda.

Los análisis empíricos dan como resultado que mientras se presenta una rotación por cada dos inserciones, aproximadamente, sólo se necesita una cada cinco borrados.

Arboles balanceados en altura – AVL

Costo de las Operaciones:

En un árbol AVL se pueden realizar las operaciones de:

- Buscar un nodo con una clave dada
- Insertar un nodo con una clave dada
- Borrar un nodo con una clave dada

en tiempo T ϵ $O(\log_2 n)$

Numero de nodos según su altura:

h	0	1	2	3	4	5	6	7	8	9	**h**
n_{min}	1	2	4	7	12	20	33	54	88	143	$n_{h-1}+n_{h-2}+1$
n_{max}	1	3	7	15	31	63	127	255	511	1023	$2^{h+1}-1$