

ARBOL

ESTRUCTURAS DE DATOS
y ALGORITMOS
LCC TUPW

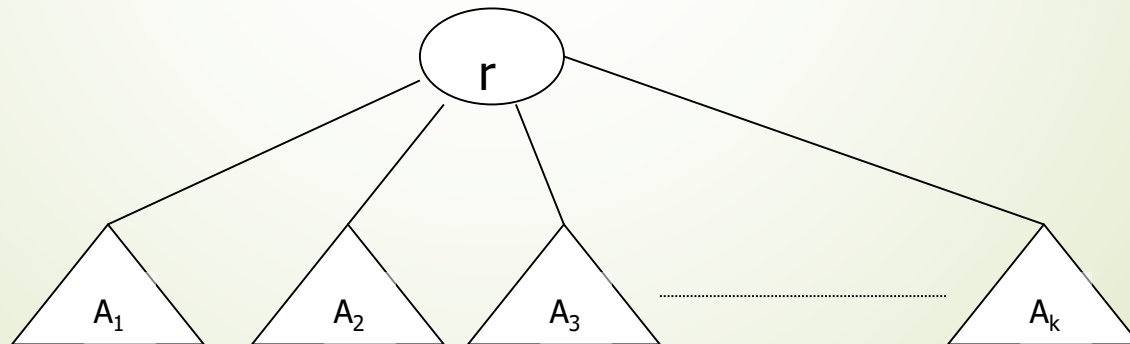
Objetivos

- Conocer distintos tipos de árboles y sus aplicaciones habituales.
- Realizar un análisis comparativo entre los mismos, a partir de la evaluación de costo de ejecución de los algoritmos que los manipulan.
- Construir los TADs: Árbol BB, AVL, B y Montículo Binario.

ARBOL

Un **ÁRBOL** es una colección o conjunto de nodos. La colección puede:

- 1 - Estar vacía
- 2 - Si no está vacía, consiste de un nodo distinguido r , conocido como *raíz*, y cero o más (sub)árboles A_1, A_2, \dots, A_k , cada uno de los cuales tiene su raíz conectada a r por medio de una *arista* (o *rama*) dirigida.



ARBOL

- La raíz de cada subárbol es un **hijo** o **descendiente directo** de r , y r es el **padre** o **antecesor directo** de cada raíz de los subárboles.
- **Camino de un nodo n_i a otro n_k :** secuencia de nodos n_1, n_2, \dots, n_k , tal que n_i es el padre de n_{i+1} para $1 \leq i < k$. En un árbol existe solamente un camino desde la raíz a cada nodo.
- Si hay un camino entre los nodos n_1 y n_2 , entonces n_1 es **antecesor** de n_2 y n_2 es **descendiente** de n_1 .
- **Longitud de camino de un nodo n_i a otro n_k :** Número de aristas que forman el camino, o número de nodos menos 1 que forman la secuencia. Existe un camino de longitud cero desde cada nodo a sí mismo.
- **Nivel de un nodo:** si el nodo x está en el nivel i , entonces sus descendientes directos están en el nivel $i+1$. La raíz de un árbol, se define como localizada en el nivel 1.
- **Profundidad o Altura del árbol:** máximo de los niveles de todos los nodos del árbol.
- **Grado de un nodo:** Número de descendientes directos de un nodo.
- **Grado del árbol:** Grado máximo en todos los nodos.
- **Nodo hoja:** nodo de grado 0.
- **Nodo Interior:** Nodo no hoja.

Árbol Binario

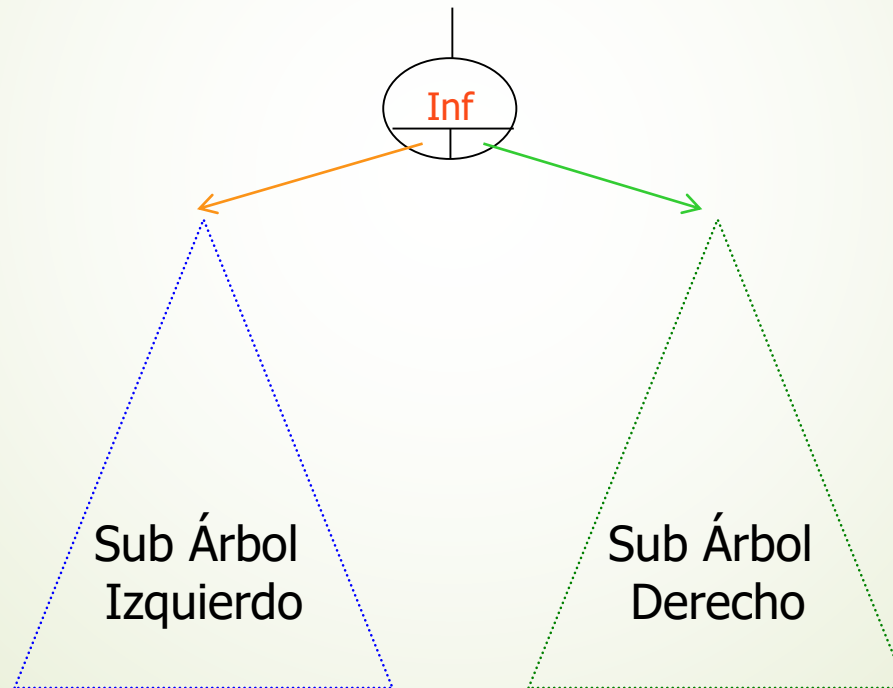
Un árbol ordenado de grado 2 se denomina **Árbol Binario**.

Este árbol se define como un conjunto finito de elementos (nodos) que:

- es vacío o
- consta de una raíz (nodo) con dos árboles binarios disjuntos llamados *subárbol izquierdo* y *derecho* de la raíz.

Árbol Binario Representación

Representación enlazada



Árbol Binario – Aplicación

Generación de Códigos de Huffman

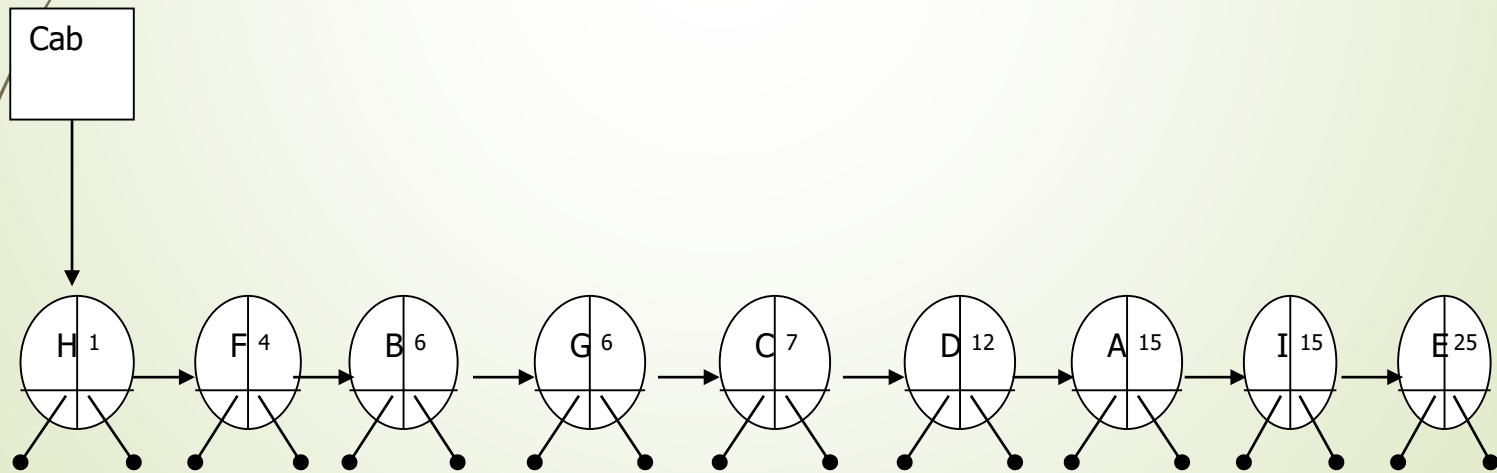
Códigos de longitud variable

<i>Caracter</i>	A	B	C	D	E	F	G	H	I
<i>Frecuencia</i>	15	6	7	12	25	4	6	1	15

Árbol Binario – Aplicación

Generación de Códigos de Huffman

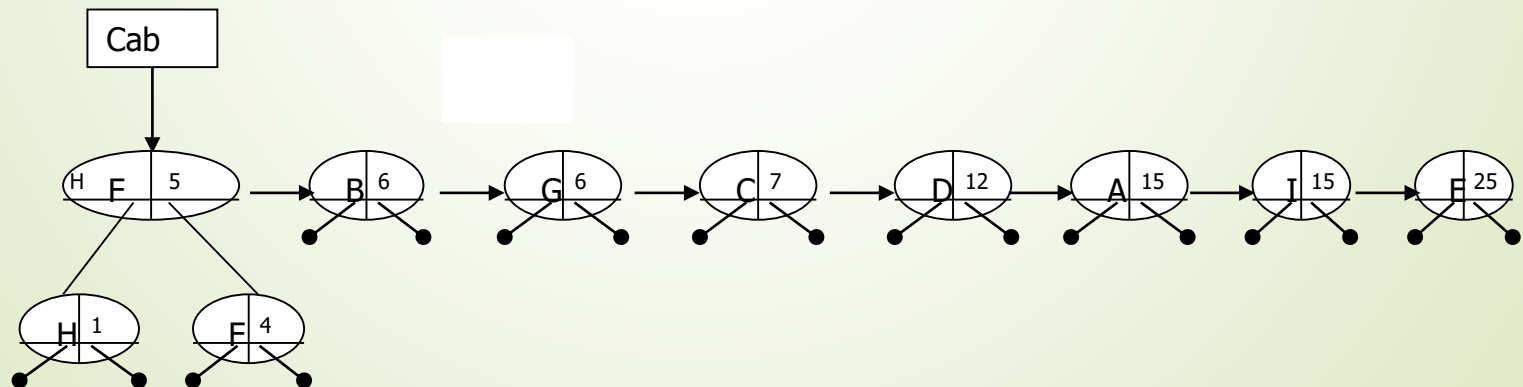
Paso 1: Generar una lista de árboles binarios. Cada raíz, inicialmente, contiene un caracter y su frecuencia de ocurrencia. La lista está ordenada en forma creciente por las frecuencias.



Árbol Binario – Aplicación

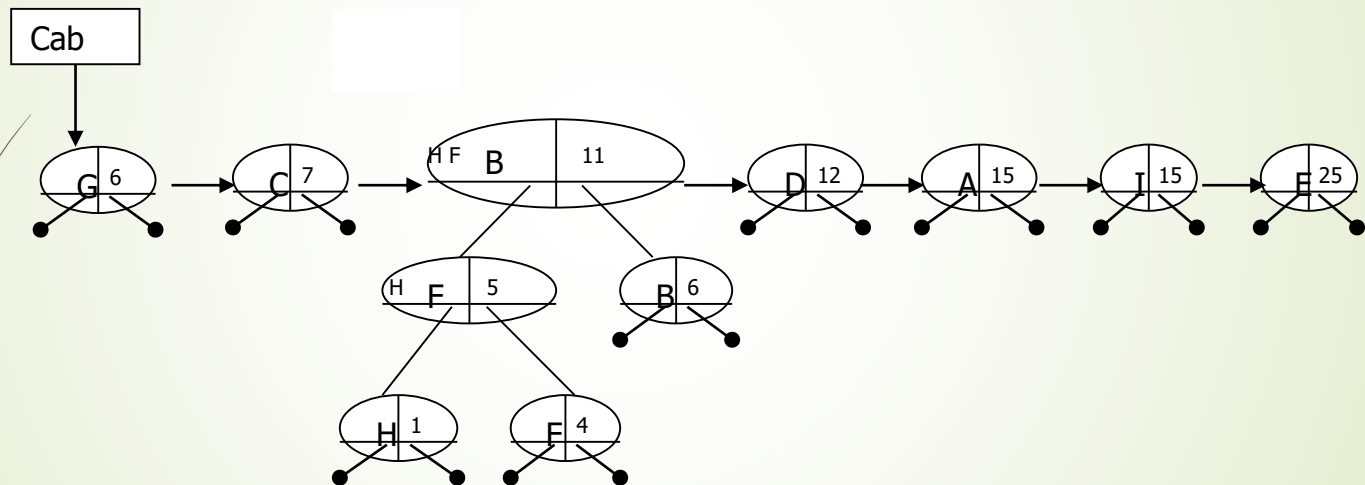
Generación de Códigos de Huffman

- **Paso 2:** Generar repetidamente árboles binarios a partir de aquellos con frecuencias menores, e insertarlos nuevamente en la lista ordenada. La frecuencia asociada a la raíz de cada nuevo árbol es la *suma* de las frecuencias de sus subárboles.



Árbol Binario – Aplicación

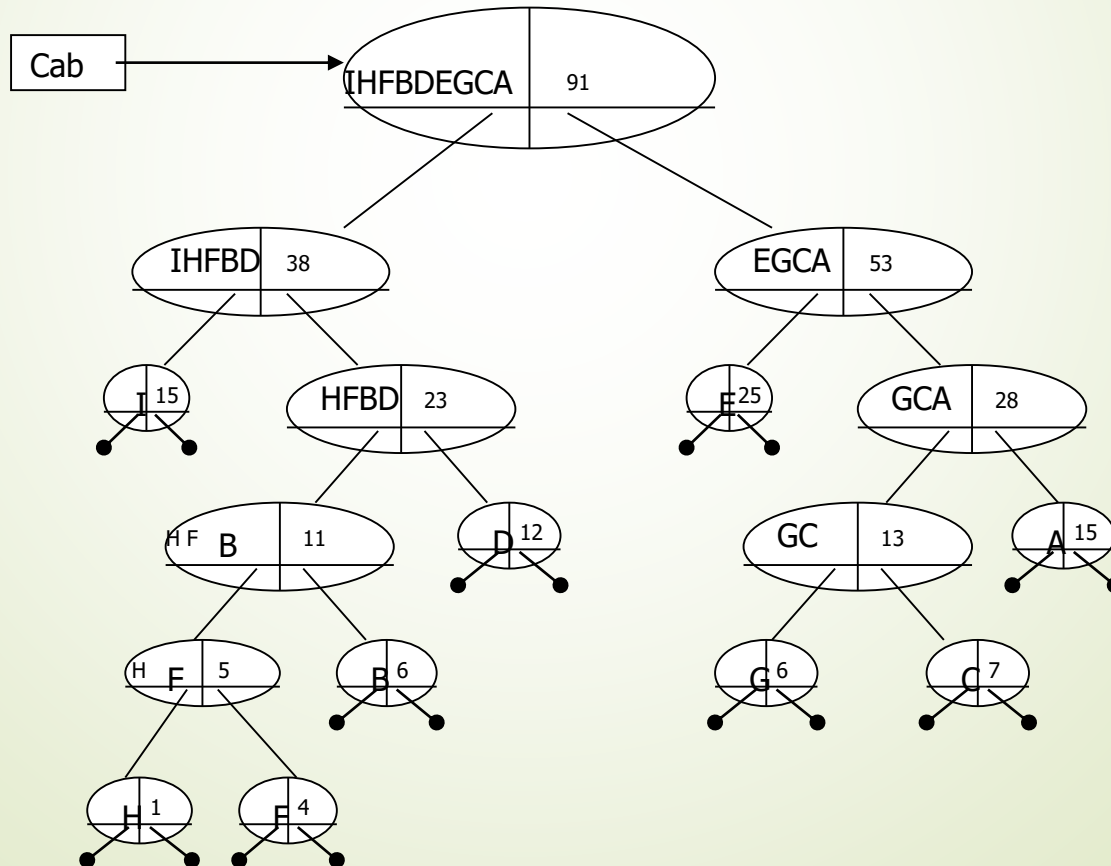
Generación de Códigos de Huffman



Esta secuencia de supresiones, síntesis e inserciones se realiza hasta el momento en que en la lista queda una sola celda, que corresponde a la raíz del árbol binario que tiene a todos los caracteres del diccionario de entrada como nodos hoja.

Árbol Binario – Aplicación

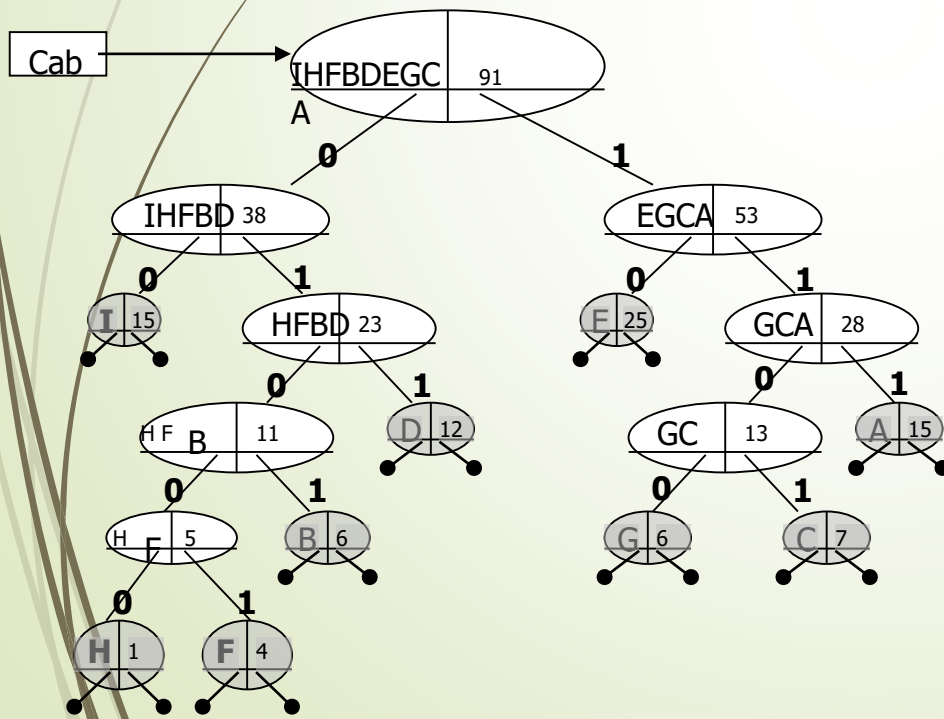
Generación de Códigos de Huffman



Árbol Binario – Aplicación

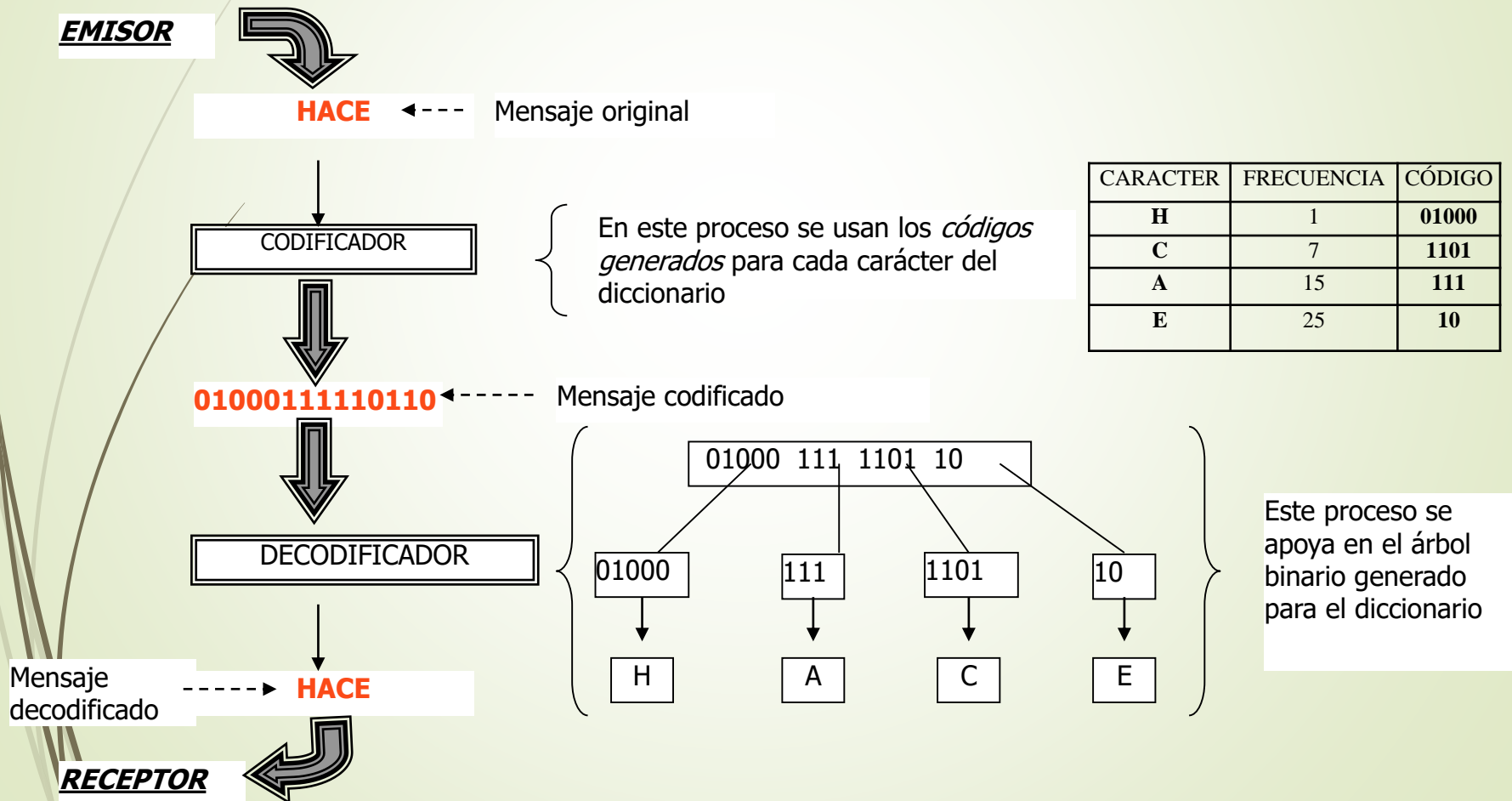
Generación de Códigos de Huffman

Paso 3: Generar el código de longitud variable correspondiente a cada carácter, hoja del árbol, concatenando ceros y unos, según se atravesase en el árbol una rama izquierda o una rama derecha.



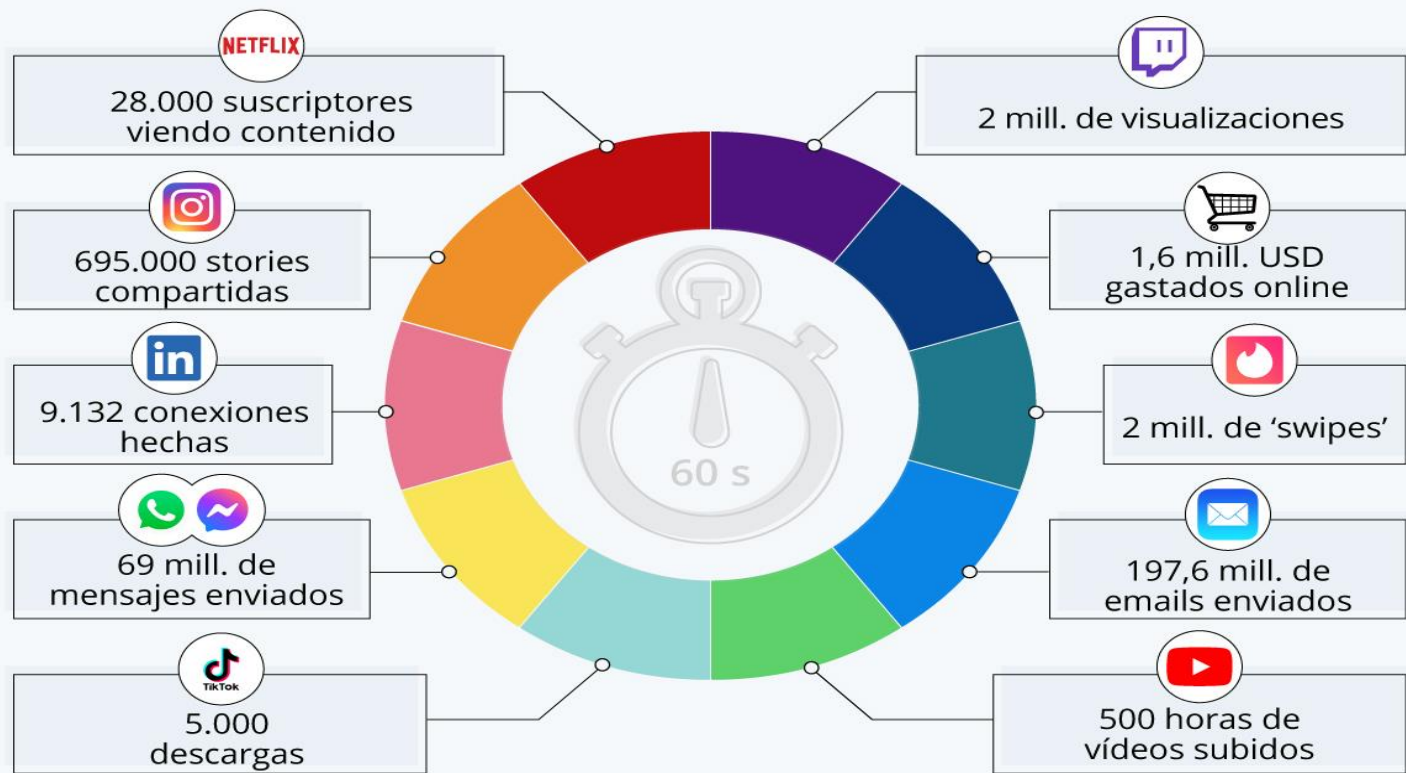
CARACTER	FRECUENCIA	CÓDIGO
H	1	01000
F	4	01001
B	6	0101
G	6	1100
C	7	1101
D	12	011
A	15	111
I	15	00
E	25	10

Codificación-Decodificación



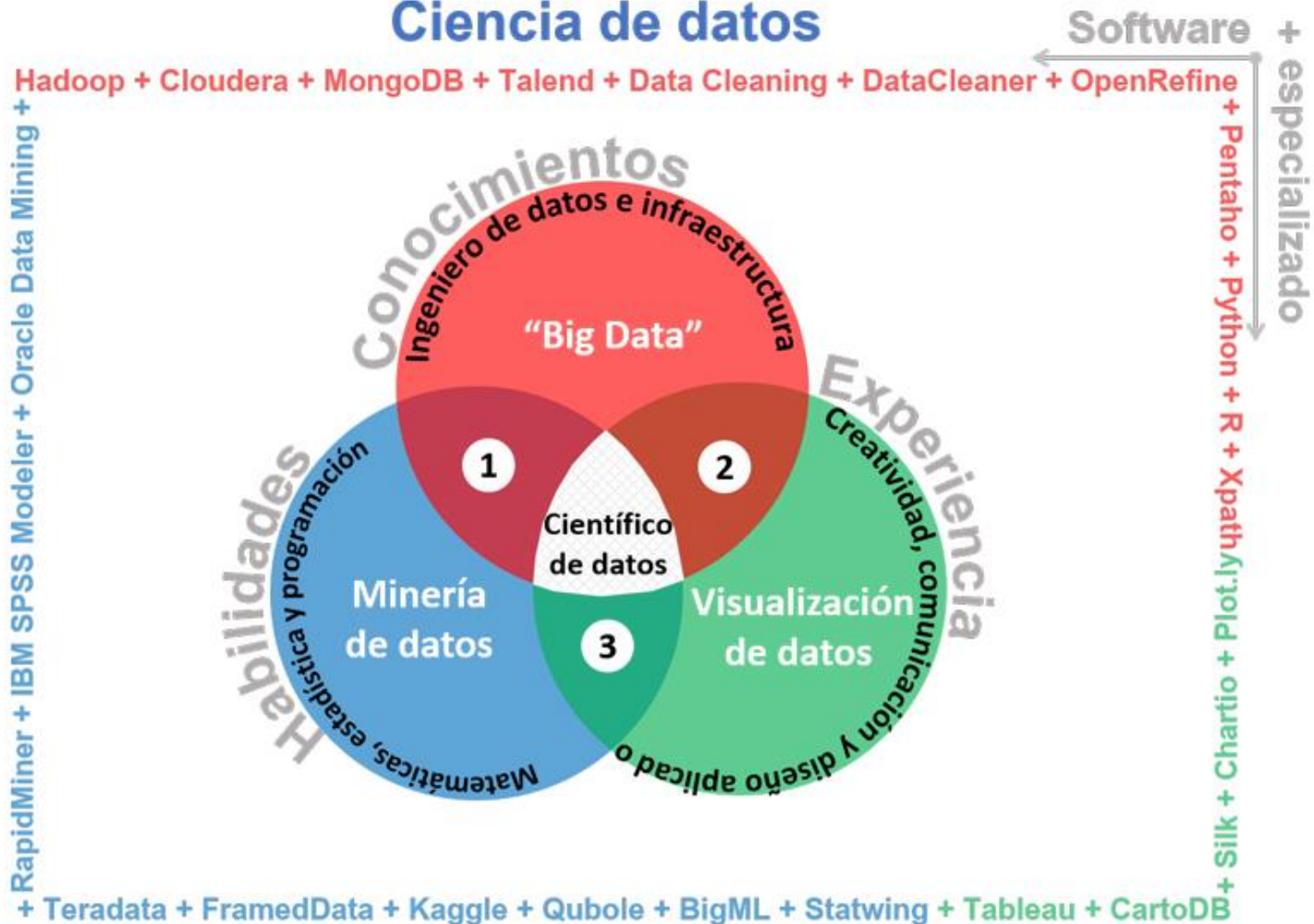
Esto sucede en Internet en un minuto

Estimación de una selección de actividades y datos generados online en un minuto en 2021



Fuente: Lori Lewis vía AllAccess

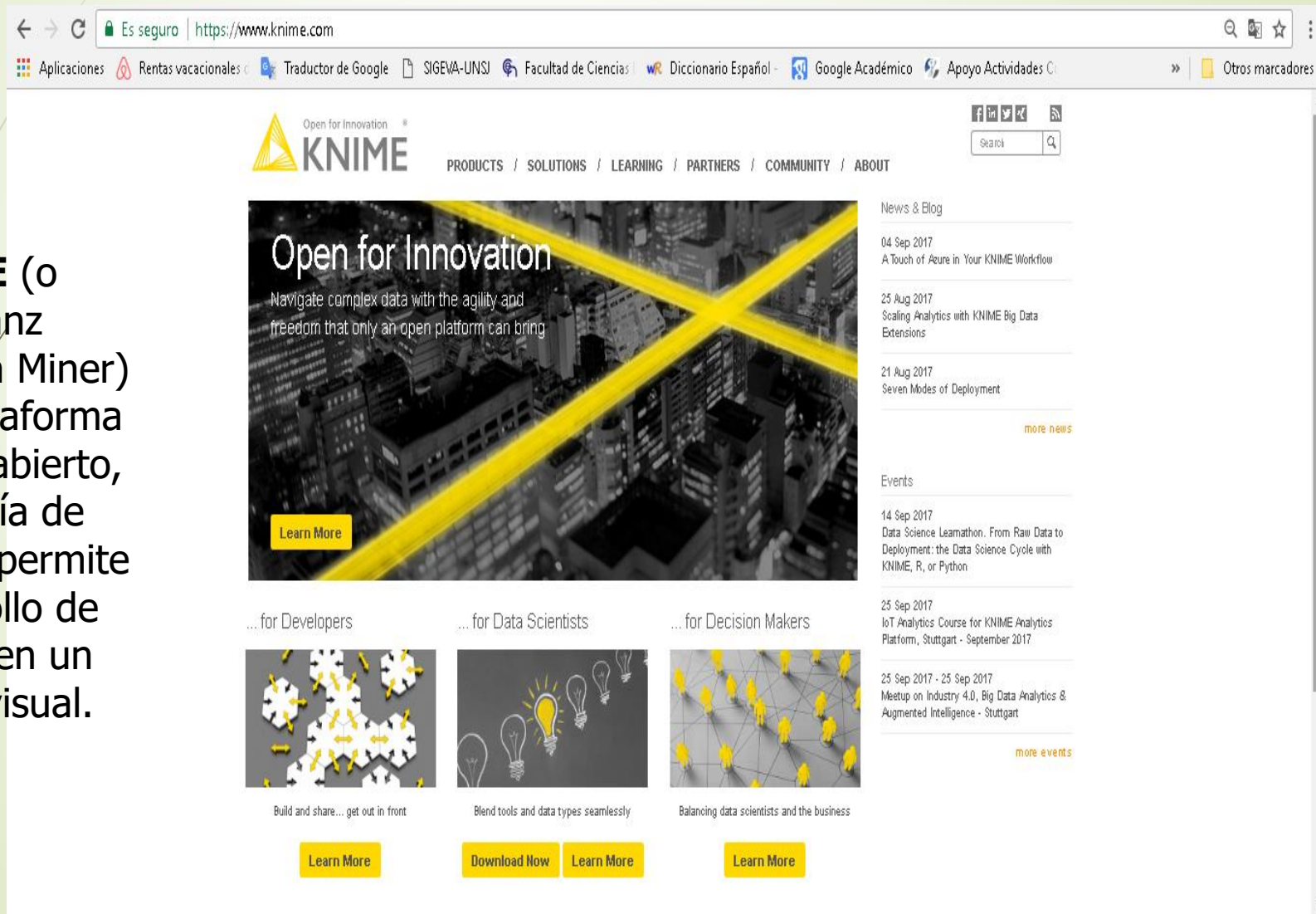
Ciencia de datos



Técnicas: Regresión lineal, Reconocimiento de patrones, Series de tiempo, Árboles de decisión, Estadística Bayesiana, Redes neuronales, Aprendizaje supervisado y no supervisado, K-NN, Sistemas de recomendación, Modelos predictivos, Teoría de juegos, Aprendizaje profundo...

Minería de Datos- KNIME

KNIME (o
Konstanz
Information Miner)
es una plataforma
de código abierto,
de minería de
datos, que permite
el desarrollo de
modelos en un
entorno visual.



The screenshot shows the KNIME website homepage. The browser address bar displays "https://www.knime.com". The website header includes the KNIME logo with the tagline "Open for Innovation" and a navigation menu with links: PRODUCTS / SOLUTIONS / LEARNING / PARTNERS / COMMUNITY / ABOUT. A search bar is located on the right. The main content area features a large banner with the text "Open for Innovation" and "Navigate complex data with the agility and freedom that only an open platform can bring", accompanied by a "Learn More" button. Below the banner are three columns of content: "...for Developers" with a "Learn More" button, "...for Data Scientists" with a "Download Now" button and a "Learn More" button, and "...for Decision Makers" with a "Learn More" button. The right sidebar contains sections for "News & Blog" and "Events", each listing recent updates and dates.



KNIME Explorer



Filter

- EXAMPLES (guest@publicserver.knime.org:47037)
- LOCAL (Local Workspace)

Favorite Nodes

- Personal favorite nodes
- Most frequently used nodes
- Last used nodes

Node Repository

- Data Views
- Statistics
- Mining
 - Bayes
 - Clustering
 - Rule Induction
 - Neural Network
 - Decision Tree
 - Misc Classifiers
 - Ensemble Learning
 - Item Sets / Association Rules
 - Association Rule Learner
 - Association Rule Learner (Borgelt)
 - Create Bit Vector
 - Item Set Finder (Borgelt)**
 - Subset Matcher
- MDS

0: KNIME_project

Welcome to KNIME



Welcome to KNIME!

We are happy to help with your first steps if you let us know your email address and register [here](#). We promise to only send a few getting started emails!

If you do want regular updates on new features, releases and KNIME events (and nothing else!) - do register for our official mailing list [here](#) - if you haven't done so already when downloading KNIME.

If you are new to KNIME, why not open one of the example workflows by double clicking one of the workflows in the explorer in the upper left corner? You can also drag&drop nodes from the node repository (bottom left) to the workflow editor after opening an existing or creating a new workflow. But **first** you may want to install one of the many extensions for processing of additional data types or other tool integrations, such as R and JFreeChart. And don't forget to take a look at our [Quickstart Guide](#) and the [Learning Hub](#), which provides links to great resources for beginners and seasoned KNIMERS.

This page will be displayed upon startup of KNIME but you can customize the content using the checkboxes at the bottom.

Outline

An outline is not available.

Node Description

Item Set Finder (Borgelt)

The node provides different algorithms to searches for frequent items in a list of item sets. The integrated algorithms are:

1. Apriori (Agrawal et al. 1993)
2. FPGrowth (frequent pattern growth, Han et al 2000)
3. RElim (recursive elimination)
4. SaM (Split and Merge)
5. JIM (Jaccard Item Set Mining)

The algorithms have been implemented by Christian Borgelt. The following descriptions have been taken from his [homepage](#).

Apriori: The apriori algorithm (Agrawal et al. 1993) carries out a breadth first search on the subset lattice and determines the support of item sets by subset tests. This is a pretty fast implementation that uses a prefix tree to organize the counters for the item sets. The census data set may be used to test the program.

FPGrowth: The FPGrowth algorithm (frequent pattern growth, Han et al 2000) represents the transaction database as a prefix tree which is enhanced with pointers that organize the nodes into lists referring to the same item. The search is carried out by projecting the prefix tree, working recursively on the result, and pruning the original tree. Since version 1.2 this implementation also contains the alpha-pruning of the FP-Bonsai techniques.

RElim: The RElim algorithm (recursive elimination) is inspired by the FP-growth algorithm, but does its work without prefix trees or any other complicated data structures. The main strength of this algorithm is not its speed (although it is not slow, but even outperforms apriori and eclat on some data sets), but the simplicity of its structure. Basically all the work is done in one recursive function of fairly few lines of code.

RAPIDMINER ... también

Process

MARKET BASKET ANALYSIS

Model associations between products by determining sets of items frequently purchased together and building association rules to derive recommendations.

Step 1:

Load transaction data containing a transaction id, a product id and a quantifier. The data denotes how many times a certain product has been purchased as part of a transactions.

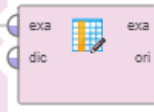
Load Transactions



Aggregate



Rename



Set Role



Step 2:

Edit, transform & load (ETL) - Aggregate transaction data via concatenation so that the products in a transaction are in one entry, separated by the pipe symbol.

Step 3:

Using FP-Growth, determine frequent item sets. A frequent item sets denotes that the items (products) in the set have been purchased together frequently, i.e. in a certain ratio of transactions. This ratio is given by the support of the item set.

FP-Growth



Create Associatio...



Step 4:

Create association rules which can be used for product recommendations depending on the confidences of the rules.

Outputs: association rules, frequent item set

Árbol Aplicación – Algoritmo FP Growth (Frequent Pattern Mining)



Árbol

Aplicación – Algoritmo FP Growth (Frequent Pattern Mining)

- FP Growth (FP-Frequent Pattern) codifica un conjunto de datos usando una estructura de datos compacta llamada FP-tree, y extrae conjuntos de items (itemsets) frecuentes directamente desde esta estructura.
- Transacciones diferentes pueden tener varios items en común, por lo que sus caminos se pueden superponer. Mientras más caminos se superpongan, mayor compresión se puede lograr usando la estructura FP-tree. Si el tamaño del FP-tree es suficientemente pequeño para caber en memoria principal, esto permitirá extraer itemsets frecuentes desde la estructura en memoria en vez de hacer repetidas pasadas sobre los datos almacenados en disco.
- FP-Growth escanea la base de datos dos veces y usa una estructura de árbol (FP-tree) para almacenar toda la información. La raíz representa nulo, cada nodo representa un elemento, mientras que la asociación de los nodos son los conjuntos de elementos con el orden mantenido mientras se forma el árbol. El árbol FP es conciso y se utiliza para generar directamente grandes conjuntos de elementos. Una vez que se ha construido un árbol FP, utiliza un enfoque recursivo de dividir y conquistar para extraer los conjuntos de elementos frecuentes.

T.A.D. Árbol Binario de Búsqueda – Especificación (1)

Un **Árbol Binario de Búsqueda –ABB–** es un Árbol Binario en el que:

- Cada nodo contiene un campo clave (lo identifica en forma única dentro del Árbol).
- Los nodos del árbol están ordenados de tal forma que para cualquier nodo x del árbol,
 - si z es un nodo cualquiera del subárbol izquierdo de x , entonces la clave[z] < clave[x] y
 - si z es un nodo cualquiera del subárbol derecho de x , entonces clave[x] < clave[z].

Esta condición es conocida como *Propiedad del Árbol Binario de Búsqueda*.

T.A.D. ABB – Especificación(2)

Operaciones Abstractas

Sean A: Árbol; X,Z : claves

<i>NOMBRE</i>	<i>ENCABEZADO</i>	<i>FUNCION</i>	<i>ENTRADA</i>	<i>SALIDA</i>
Crear	Crear (A)	Inicializa el árbol A	A	A=()
Insertar	Insertar(A, X)	Ingresa el elemento X en el árbol A, manteniéndolo como árbol binario de búsqueda.	A, X	A con X como hoja, si $X \notin A$; Error en caso contrario
Suprimir	Suprimir(A, X)	Elimina el elemento X del árbol A, manteniéndolo como árbol binario de búsqueda.	A, X	A sin el nodo que contenía a X, si $X \in A$; Error en caso contrario
Buscar	Buscar(A,X)	Localiza el nodo con clave X en el árbol A	A,X	Reporta los datos asociados con X, si $X \in A$; Error en caso contrario
Nivel	Nivel(A, X)	Calcula el nivel del nodo con clave X	A, X	Reporta el nivel del nodo con clave X si $X \in A$; Error en caso contrario
Hoja	Hoja(A, X)	Evalúa si el nodo con clave X es hoja	A, X	Verdadero si el nodo con clave X es hoja, Falso en caso contrario.

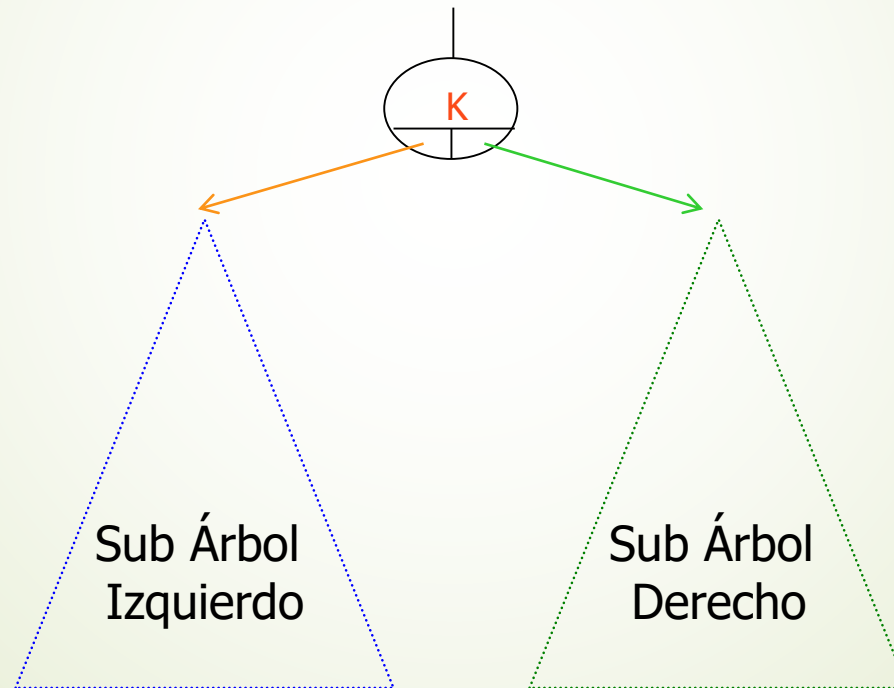
T.A.D. ABB – Especificación(3)

Operaciones Abstractas

Sean A: Árbol; X,Z : claves

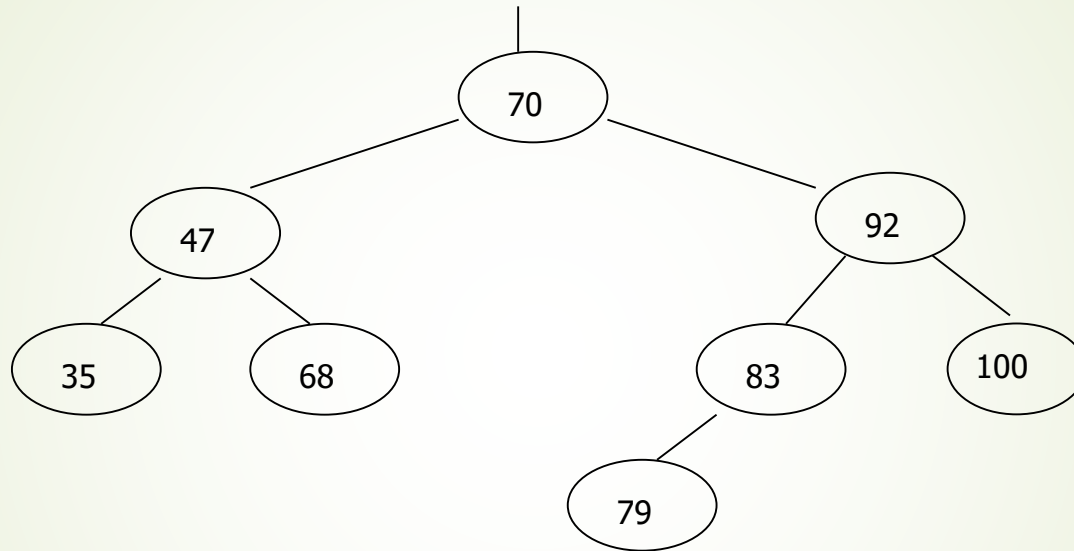
<i>NOMBRE</i>	<i>ENCABEZADO</i>	<i>FUNCION</i>	<i>ENTRADA</i>	<i>SALIDA</i>
Hijo	Hijo(A, X, Z)	Evalúa si X es hijo (descendiente directo) de Z	A, X, Z	Verdadero si el nodo X es hijo del nodo Z, Falso en caso contrario.
Padre	Padre(A, X, Z)	Recíproca de la operación Hijo	A, X, Z	...
Camino	Camino(A, X, Z)	Recupera el camino del nodo con clave X al nodo con clave Z	A, X, Z	Reporta el camino de X a Z, si X es ancestro de Z; Error en caso contrario.
Altura	Altura(A)	Evalúa la altura de A	A	Reporta la altura de A.
InOrden	InOrden(A)	Procesa A en Inorden	A	Está sujeta al proceso que se realice sobre los elementos de A
PreOrden	PreOrden(A)	Procesa A en Preorden	A	Está sujeta al proceso que se realice sobre los elementos de A
PostOrden	PostOrden(A)	Procesa A en Postorden	A	Está sujeta al proceso que se realice sobre los elementos de A

T.A.D. ABB – Representación



T.A.D. ABB – Construcción de Operaciones Abstractas (1)

Buscar(A,X)



Si (sub)árbol vacío

entonces **Error-** Elemento Inexistente

sino

Si Clave [X] =Clave [R] (R:nodo raíz de (sub)árbol)

entonces **Éxito-** Elemento existente

sino

Si Clave [X] < Clave [R] (R:nodo raíz de (sub)árbol)

entonces **Buscar** ((sub)árbol izquierdo(R), X)

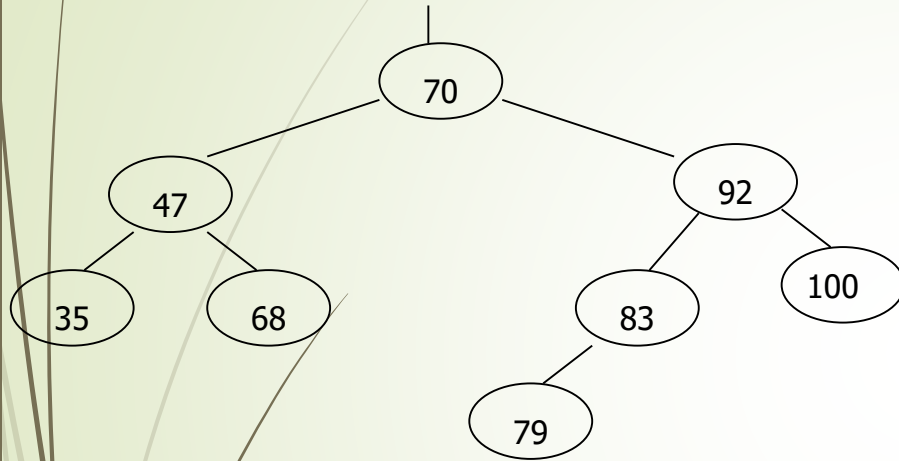
sino

Si Clave[X] >Clave [R] (R: nodo raíz de (sub)árbol)

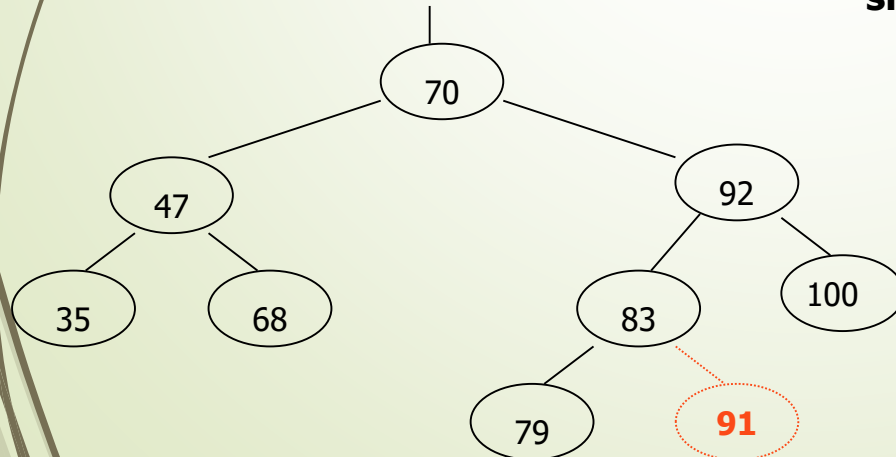
entonces **Buscar** ((sub)árbol derecho(R), X)

T.A.D. ABB –

Construcción de Operaciones Abstractas (2)



Insertar (A,X=91)



Si (sub)árbol vacío

entonces **Éxito- X nueva hoja**

sino

Si Clave [X] = Clave [R]

entonces **Error- Elemento ya existente**

sino

Si Clave [X] < Clave [R]

entonces **Insertar**((sub)árbol izquierdo(R),X)

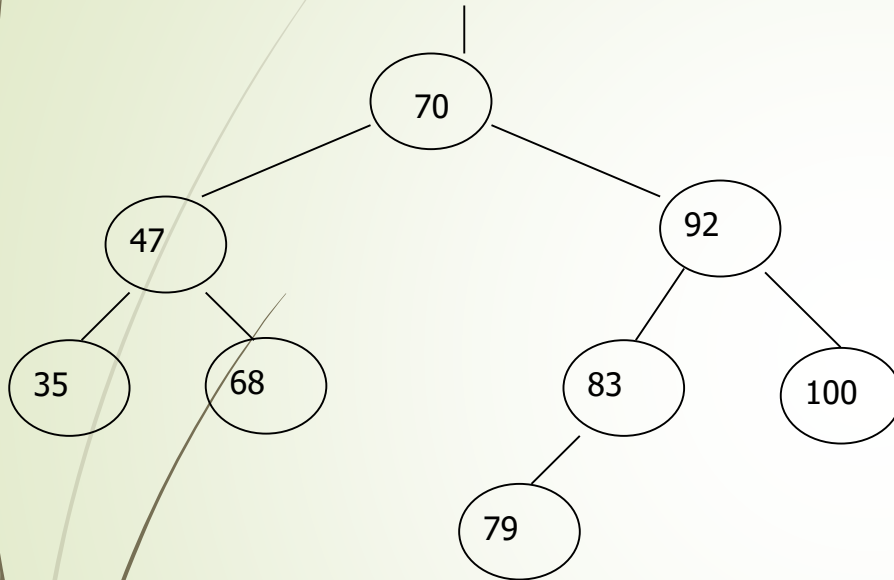
sino

Si Clave[X] > Clave [R]

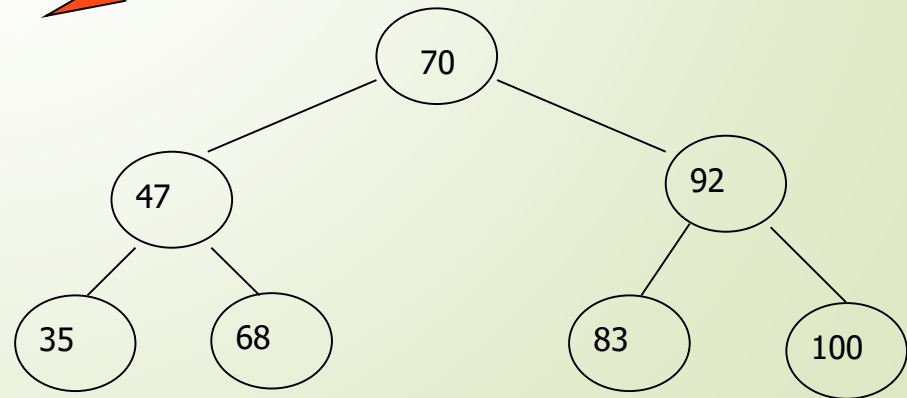
entonces **Insertar** ((sub)árbol derecho(R), X)

T.A.D. ABB –

Construcción de Operaciones Abstractas (3)

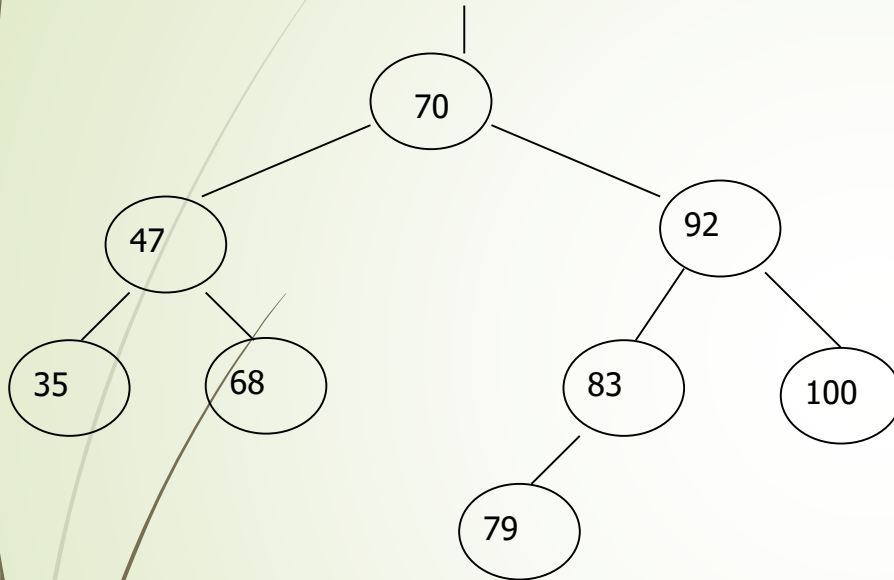


Suprimir (A , X=79)

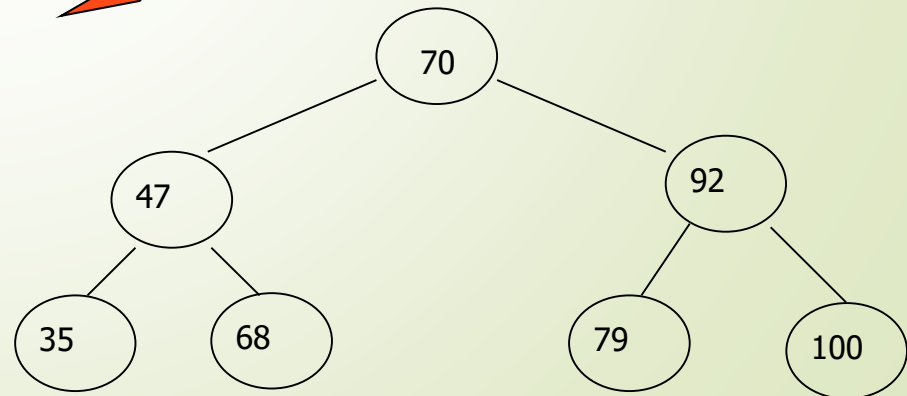


T.A.D. ABB –

Construcción de Operaciones Abstractas (4)

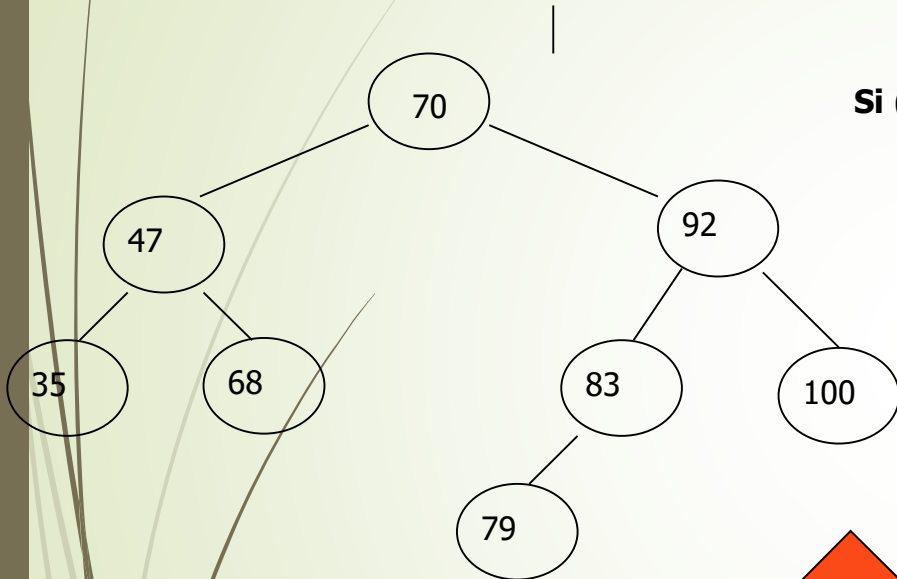


Suprimir (A , X=83)



T.A.D. ABB –

Construcción de Operaciones Abstractas (5)



Suprimir (A , X=70)

Si (sub)árbol vacío

entonces Error -elemento inexistente

sino

Si Clave [X] = Clave [R] (R: nodo raíz de (sub)árbol)

entonces

Si Grado(R) = 0

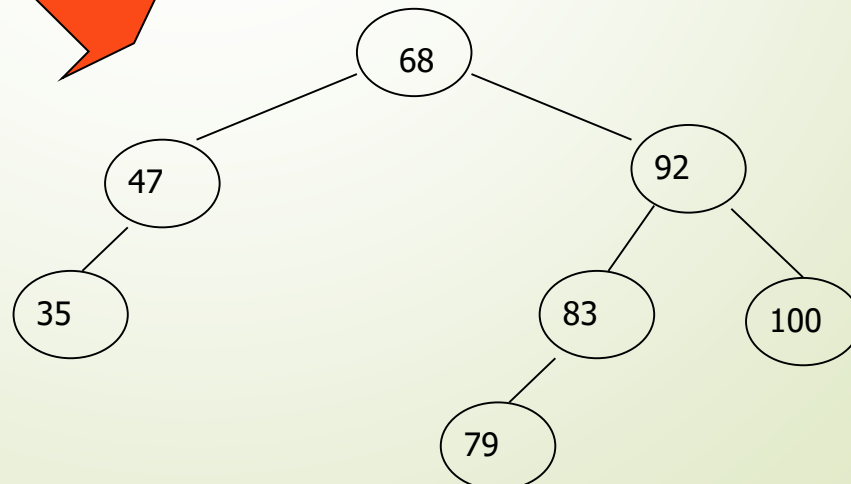
entonces eliminar nodo R

Si Grado(R) = 1

entonces Padre(R) ← Hijo(R)

Si Grado(R) = 2

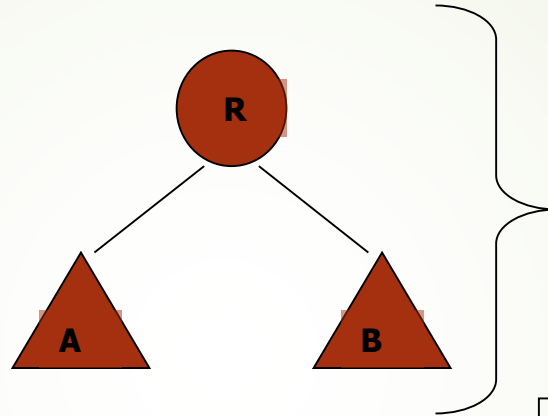
entonces R ← Máximo ((sub)árbol-izquierdo (R))
 Eliminar (Máximo((sub)árbol-izquierdo(R)))



Árbol Binario

Construcción de Operaciones Abstractas-Recorridos (6)

Recorridos



Árbol

R: raíz

A: Sub Árbol izq de R

B: Sub Árbol der de R

Preorden (Árbol)

Si (Árbol no vacío)
entonces

Procesar nodo R

Preorden ((Sub)Árbol izq
de R)

Preorden ((Sub)Árbol der
de R)

Inorden (Árbol)

Si (Árbol no vacío)
entonces

Inorden((Sub) Árbol izq
de R)

Procesar nodo R

Inorden((Sub) Árbol der
de R)

Postorden (Arbol)

Si (Árbol no vacío)
entonces

Postorden((Sub) Árbol izq
de R)

Postorden((Sub) Árbol der
de R)

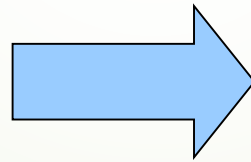
Procesar nodo R

T.A.D. ABB – Aplicación

Indice de referencias cruzadas

ENTRADA

- 1 . ` El editor de lenguaje C
- 2 . presenta un entorno
- 3 . similar al entorno
- 4 . de lenguaje Pascal`



SALIDA

al	3
C	1
de	1, 4
editor	1
entorno	2, 3
el	1
Lenguaje	1, 4
Pascal	4
presenta	2
similar	3
un	2

Árbol Binario – Tipos

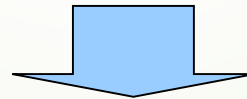
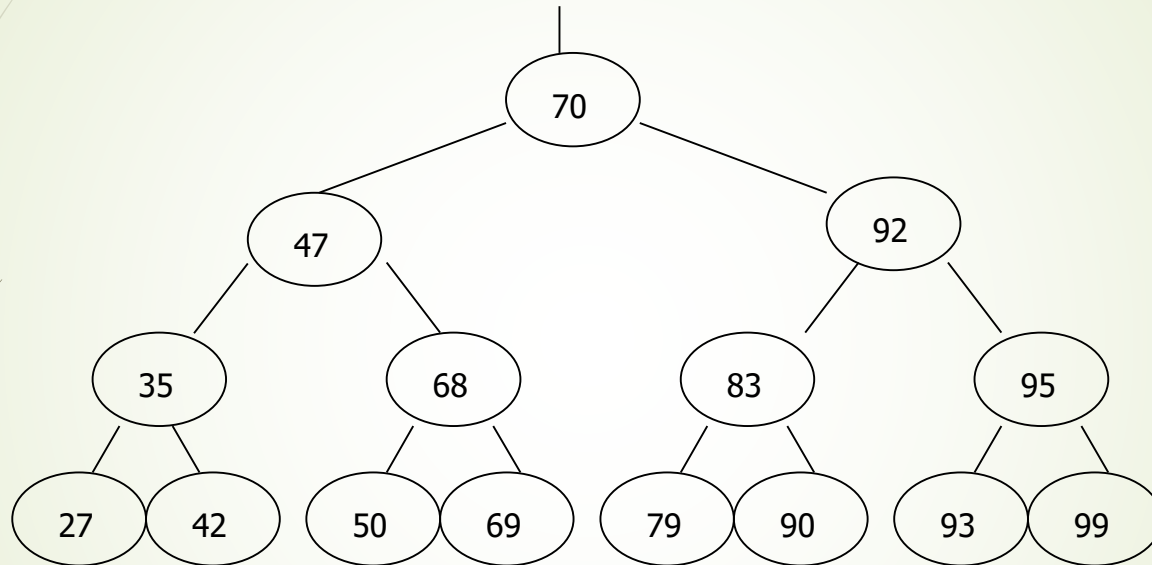
¿Cuántas comparaciones se realizan en una **búsqueda**, en el peor de los casos, en un **ABB** ?

Las operaciones Insertar, Suprimir y Buscar se realizan en h accesos como máximo, siendo h la altura del árbol binario.

- *Árbol Binario Relleno*: Es aquel árbol en el que todo nodo o bien es una hoja, o tiene dos hijos.
- *Árbol Binario Completo*: Es un árbol binario relleno en el que todas las hojas están en el mismo nivel.

¿Cuántas comparaciones se realizan en una **búsqueda**, en el peor de los casos, en un **ABB Completo**?

ABB Completo



N	1	3	7	15
C	1	2	3	4



$$N = 2^C - 1 \Rightarrow N + 1 = 2^C$$

$$\Rightarrow \log_2 (N+1) = C (\log_2 2)$$

$$\Rightarrow \log_2 (N+1) = C$$

$t(\text{Buscar (ABB Completo)}) \in O(\log n)$



Operaciones Abstractas: BUSCAR

¿A que Orden de Complejidad pertenece la operación Buscar?

LISTA ?

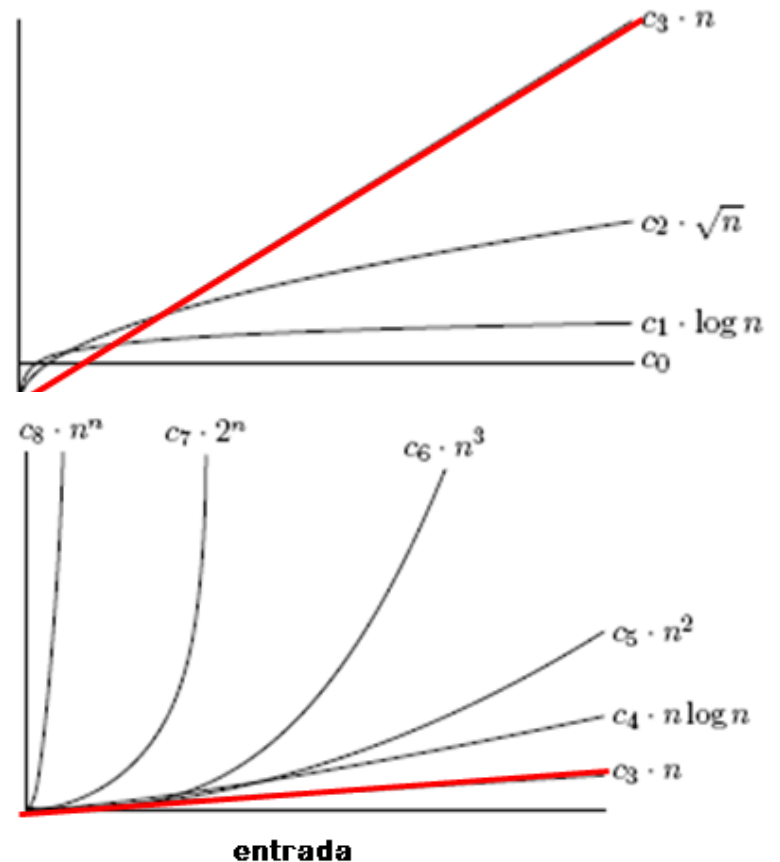
LISTA
ordenada
por
contenido ?

ABB
Completo?

ABB ?

tiempo

tiempo



Arbol Balanceado- AVL

Un árbol es ***perfectamente equilibrado***, si para cada nodo los números de nodos en sus subárboles izquierdo y derecho difieren cuanto más en uno.

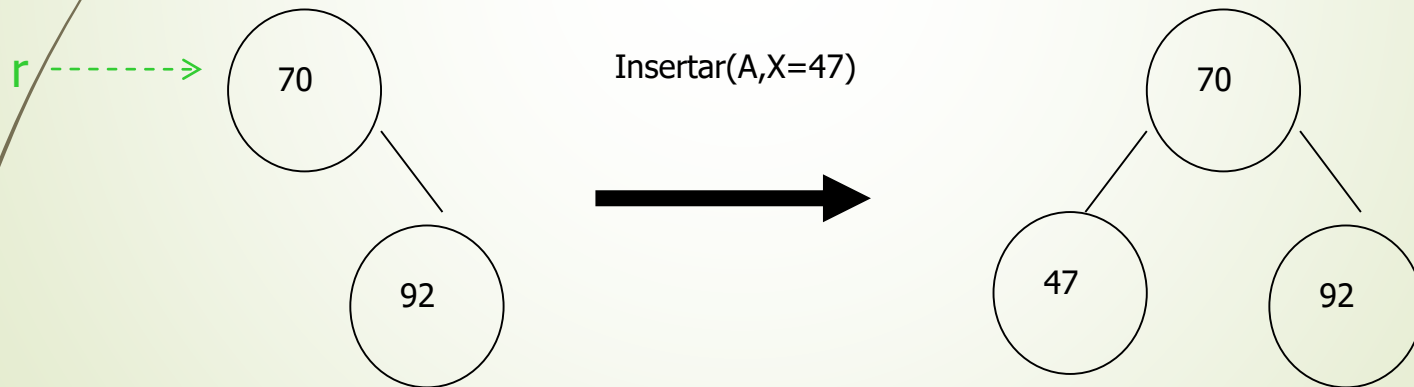
Un árbol está ***balanceado*** si y solo si en cada nodo las alturas de sus dos subárboles difieren a lo máximo en uno.

Los árboles balanceados reciben el nombre de **Árboles AVL** por ser Adelson-Velski y Landis quienes propusieron esta definición de equilibrio.

T.A.D. Arbol Balanceado – Construcción de operaciones abstractas (1)

Insertión en un Árbol Balanceado

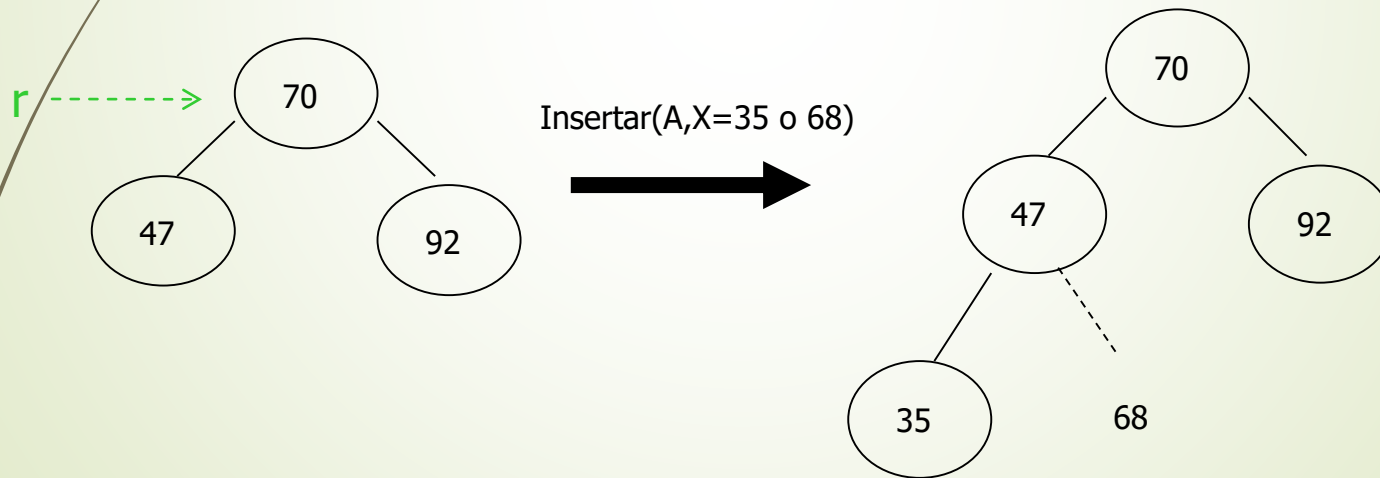
a) Si $\text{altura}(I(r)) < \text{altura}(D(r))$ y X se inserta en $I(r)$



T.A.D. Arbol Balanceado – Construcción de operaciones abstractas (2)

Inserción en un Árbol Balanceado

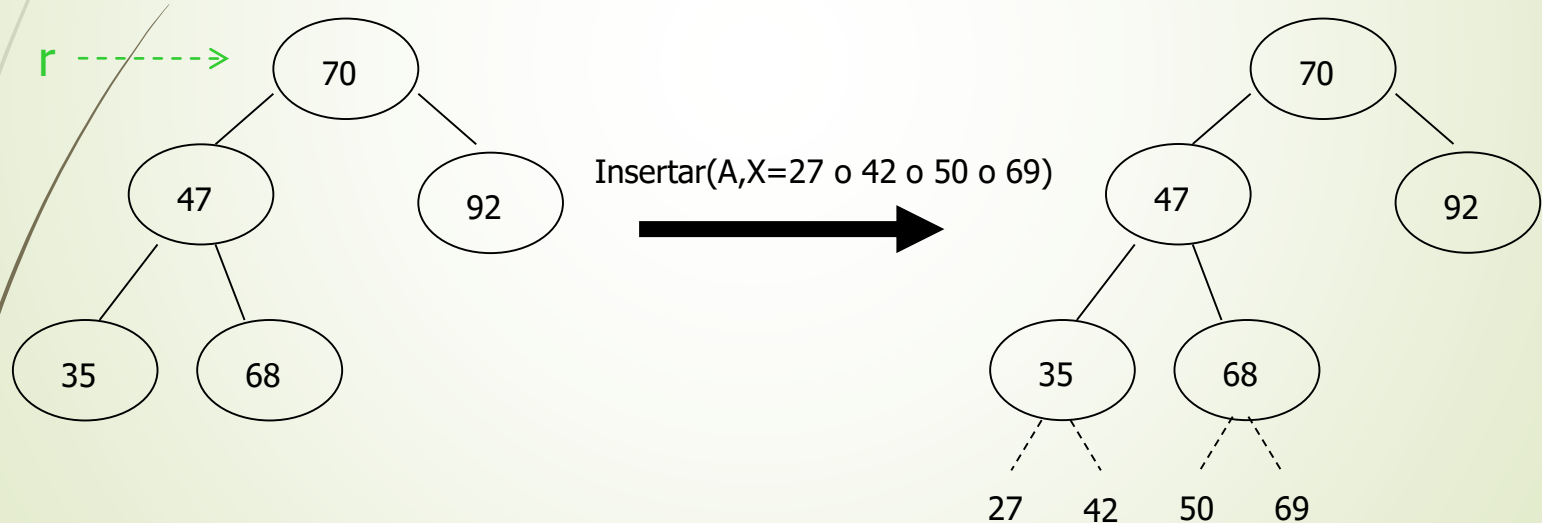
b) Si $\text{altura}(\text{I}(r)) = \text{altura}(\text{D}(r))$ y X se inserta en $\text{I}(r)$,



T.A.D. Arbol Balanceado – Construcción de operaciones abstractas (3)

Inserción en un Árbol Balanceado

c) Si $\text{altura}(I(r)) > \text{altura}(D(r))$ y X se inserta en $I(r)$

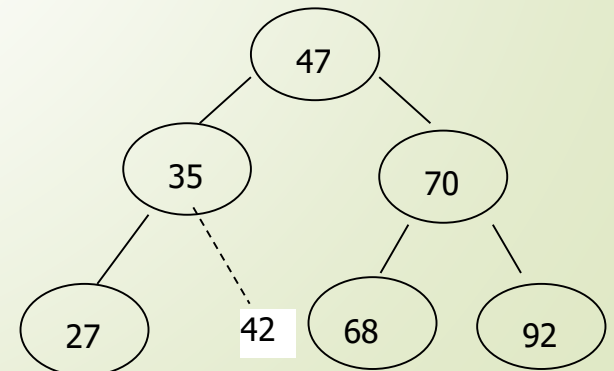
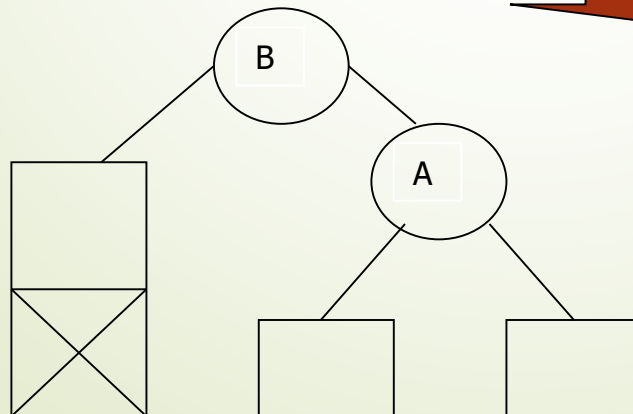
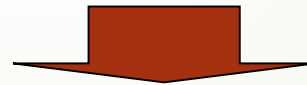
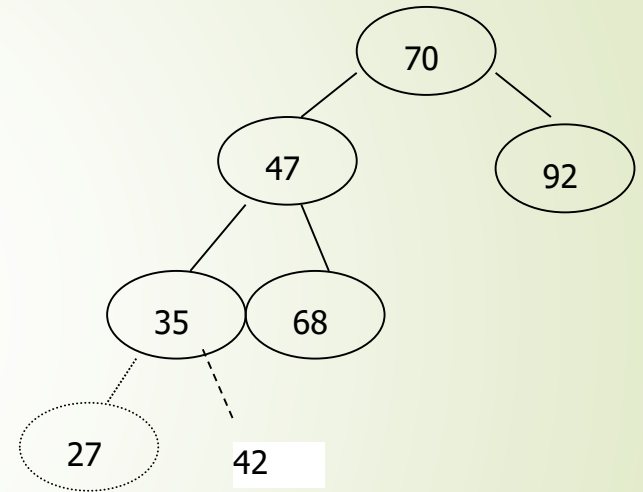
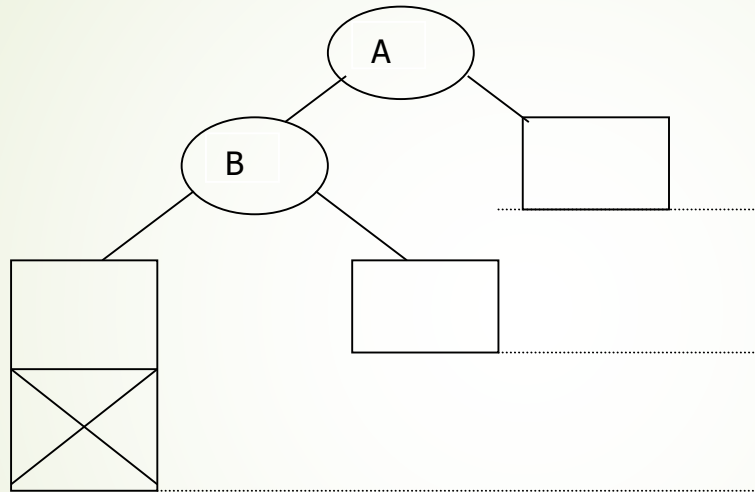


REBALANCEAR !!

T.A.D. Arbol Balanceado – Construcción de operaciones abstractas (4)

REBALANCEO

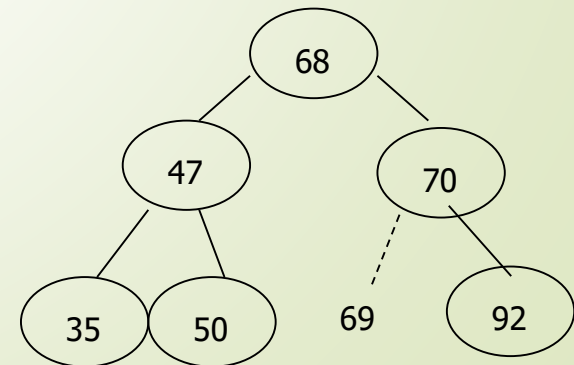
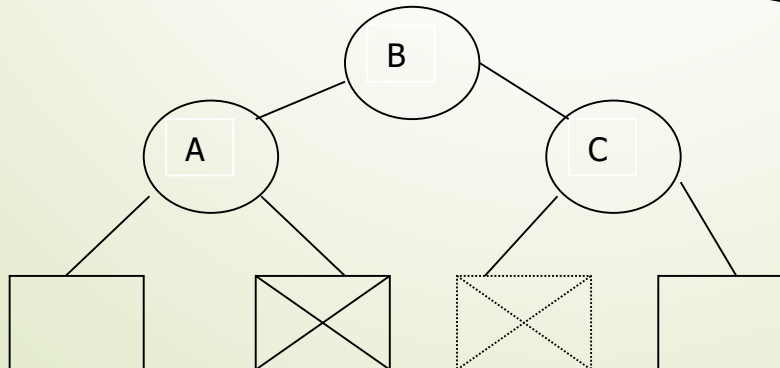
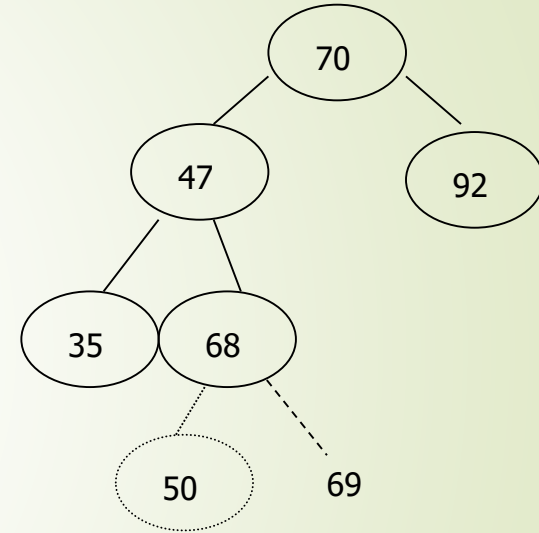
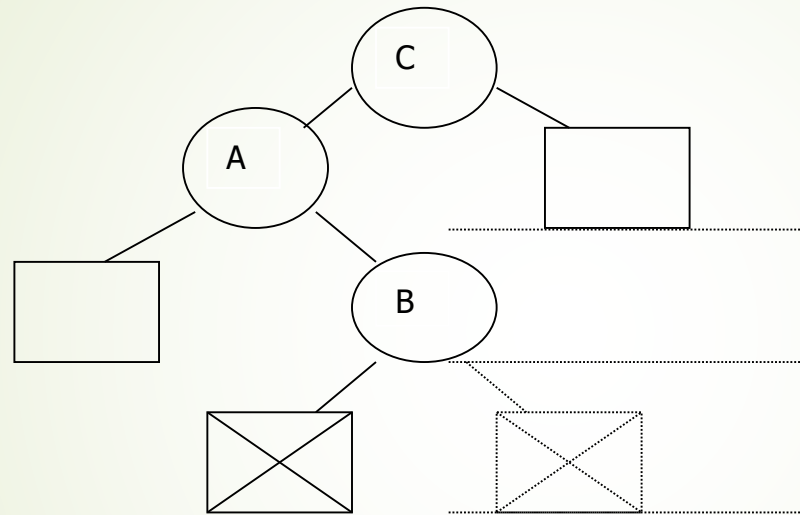
CASO 1



T.A.D. Arbol Balanceado – Construcción de operaciones abstractas (5)

REBALANCEO

CASO 2



Arbol Multicamino

Construcción y mantención de árboles de búsqueda a gran escala, que se almacenan en memoria secundaria.

Almacenar datos de 1 millón de elementos árbol balanceado →

$$\log_2 10^6 = 20 \text{ comp.}$$

acceso a disco por cada comp → 20 accesos a disco

Se incorpora un tipo particular de árbol multicamino

cantidad de accesos en el peor de los casos sería: $\log_{100} 10^6 = 3$ accesos

Cada página (salvo una) contiene entre **n y $2n$** nodos para determinada constante **n** .

De ahí que, en un árbol con N elementos y un tamaño máximo de página de $2n$ nodos por página, en el peor caso requiere $\log_n N$ accesos de página.

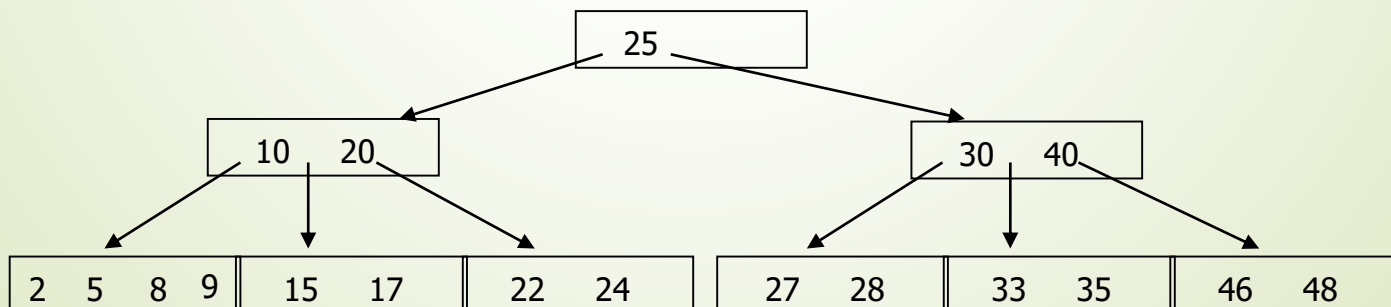
Árbol B, árbol multcamino de *orden n*:

- 1) Cada página contiene a lo sumo $2n$ elementos (claves).
- 2) Cada página, excepto la pagina raíz, contiene n elementos por lo menos.
- 3) Cada página es una página de hoja, o sea que no tiene descendientes, o tiene $m+1$ descendientes, donde m es su número de claves en esa página ($n \leq m \leq 2n$).
- 4) Todas las páginas hoja aparecen al mismo nivel.

T.A.D. Árbol B de orden 2

Especificación

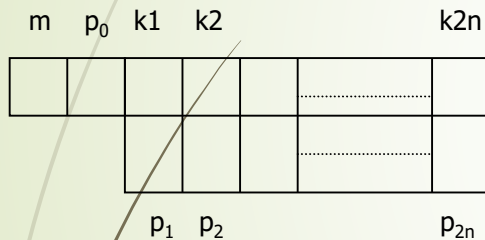
- 1) Cada página contiene a lo sumo 4 ($2*2$) elementos (claves).
- 2) Cada página, excepto la pagina raíz, contiene 2 elementos por lo menos.
- 3) Cada página es una página de hoja, o sea que no tiene descendientes, o tiene $m+1$ descendientes, donde m es su número de claves en esa página ($2 \leq m \leq 4$).
- 4) Todas las páginas hoja aparecen al mismo nivel.



T.A.D. Árbol B

Representación

Estructura de la Página



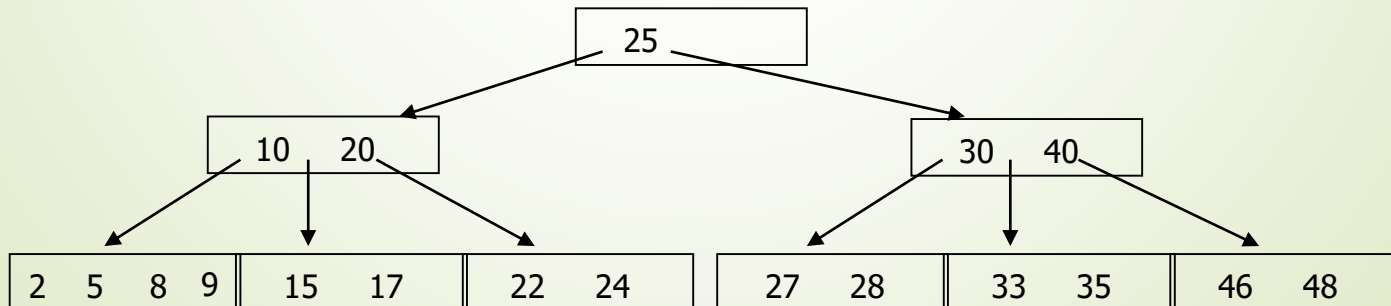
m : cantidad de claves en la página

k_i : clave; $1 \leq i \leq m$

p_0 : dirección de la página que contiene claves menores que k_1

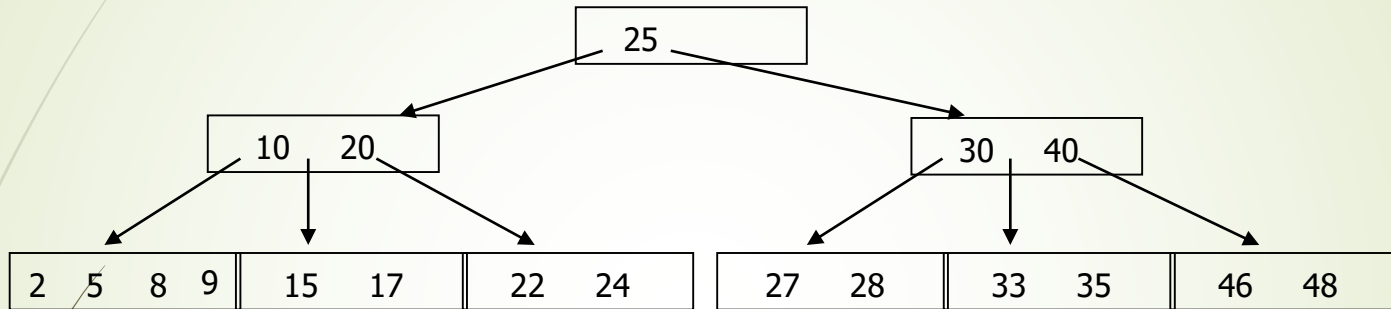
p_i : dirección de la página que contiene claves mayores que k_i y menores que k_{i+1}

p_m : dirección de la página que contiene claves mayores que k_m



T.A.D. Árbol B

Operaciones Abstractas: Buscar



¿Cómo se realiza la búsqueda de una clave **X**, en una página dada?

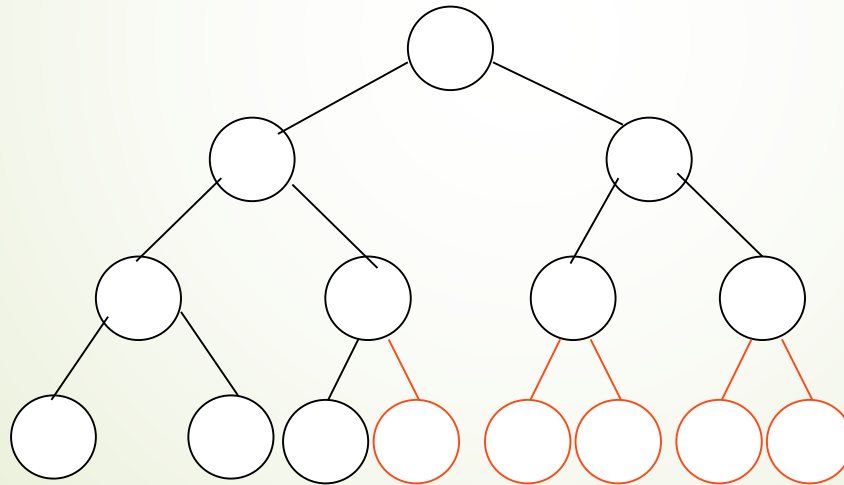
- 1) $k_m < \mathbf{X}$;
- 2) $\mathbf{X} < k_1$;
- 3) $k_i < \mathbf{X} < k_{i+1}$ ($1 \leq i < m$)

¿Si la clave **X** no se encuentra en una página, cómo continúa la búsqueda?

entonces la búsqueda continúa por la página apuntada por p_m
 entonces la búsqueda continúa por la página apuntada por p_0
 entonces la búsqueda continúa por la página apuntada por p_i

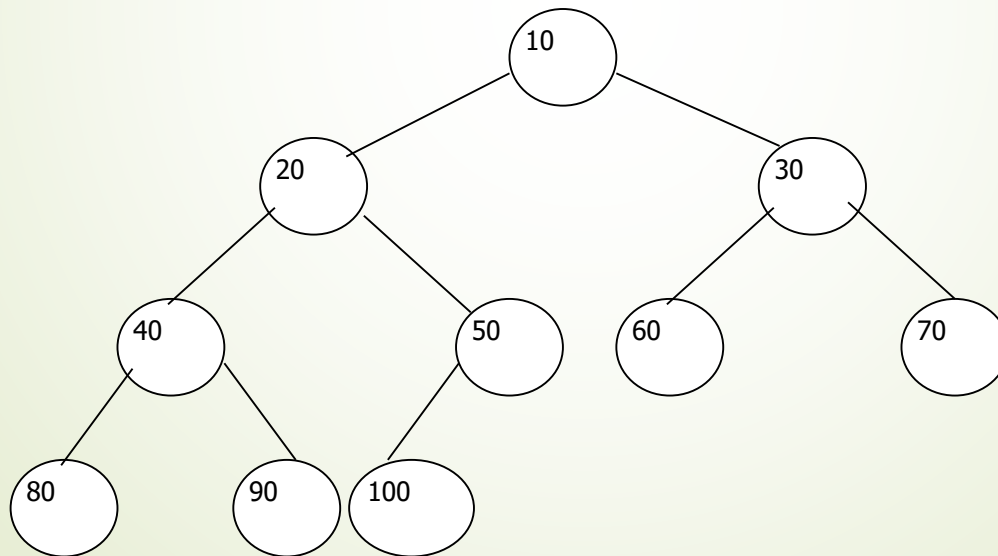
Árbol Binario Semicompleto

Árbol Binario Semicompleto: Un Árbol Binario Semicompleto de n nodos, se forma a partir de un árbol binario completo de $n+q$ nodos, quitando las q hojas extremo derechas del árbol binario completo.



T.A.D. Montículo Binario - Especificación

Montículo Binario: Un *Montículo Binario* es un árbol binario semicompleto en el que el valor de clave almacenado en cualquier nodo es menor o igual que el valor de clave de sus hijos.



Montículos Binarios

Colas de Prioridad



Los valores –claves- que representan prioridades deben interpretarse de la siguiente manera: **a menor valor-mayor prioridad**, por lo que **la máxima prioridad se encuentra en la raíz del árbol**.

T.A.D. Montículo Binario - Especificación

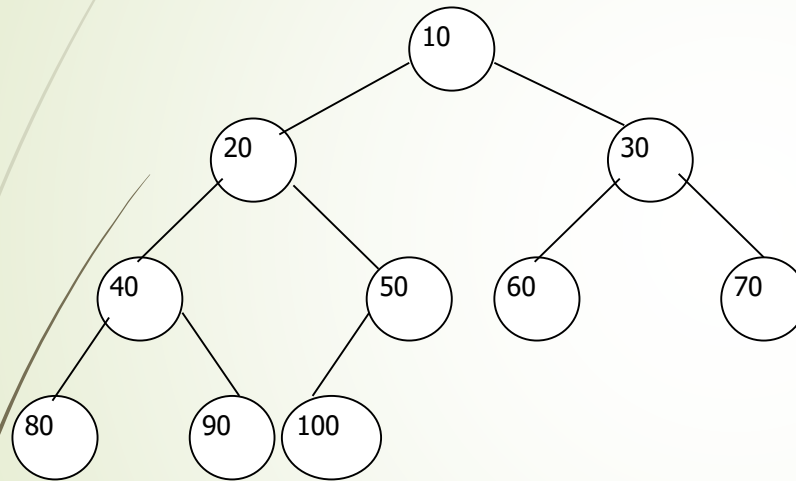
Operaciones Abstractas

M: Montículo Binario y X: Clave

<i>NOMBRE</i>	<i>ENCABEZADO</i>	<i>FUNCIÓN</i>	<i>ENTRADA</i>	<i>SALIDA</i>
Insertar	Insertar(M, X)	Ingresa el elemento X al montículo M	M, X	M con el nuevo elemento
Eliminar_Mínimo	Eliminar_Mínimo(M, X)	Suprime del montículo M el elemento de máxima prioridad- mínimo valor de clave	M	M y X: elemento de máxima prioridad

T.A.D. Montículo Binario - Representación

Montículo Binario desde una perspectiva conceptual



Elementos

	10	20	30	40	50	60	70	80	90	100
0	1	2	3	4	5	6	7	8	9	10	

Elementos [i]

Padre : Elementos [$\lfloor i / 2 \rfloor$]

Hijo Izquierdo : Elementos [$i * 2$]

Hijo Derecho : Elementos [$i * 2 + 1$]

Montículo Binario en su almacenamiento

T.A.D. Montículo Binario – Construcción de operaciones abstractas (1)

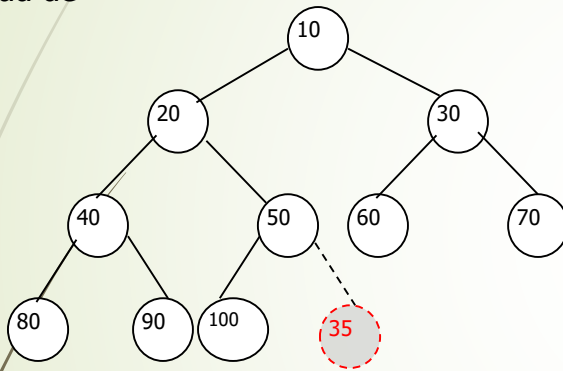
Tanto la operación ***Insertar*** como ***Eliminar_Mínimo***, deben garantizar que en el Montículo Binario se mantengan las siguientes dos propiedades:

- ***Propiedad de Estructura*** : el objeto de datos debe ser un árbol binario semicompleto.
- ***Propiedad de Orden*** : el valor de clave almacenado en cualquier nodo debe ser menor o igual que el valor de clave de sus hijos.

T.A.D. Montículo Binario – Construcción de operaciones abstractas (2)

a) Propiedad de Estructura

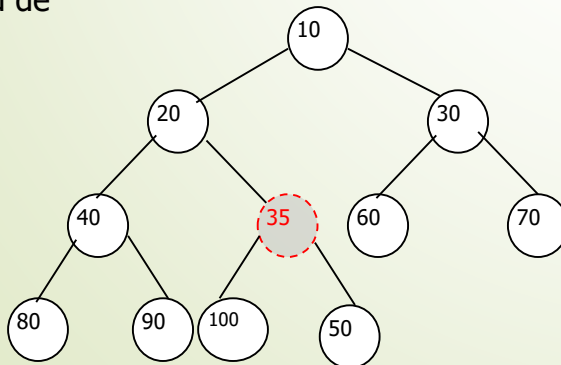
Insertar (M,X=35)



Elementos

	10	20	30	40	50	60	70	80	90	100	35
0	1	2	3	4	5	6	7	8	9	10	11	

b) Propiedad de Orden

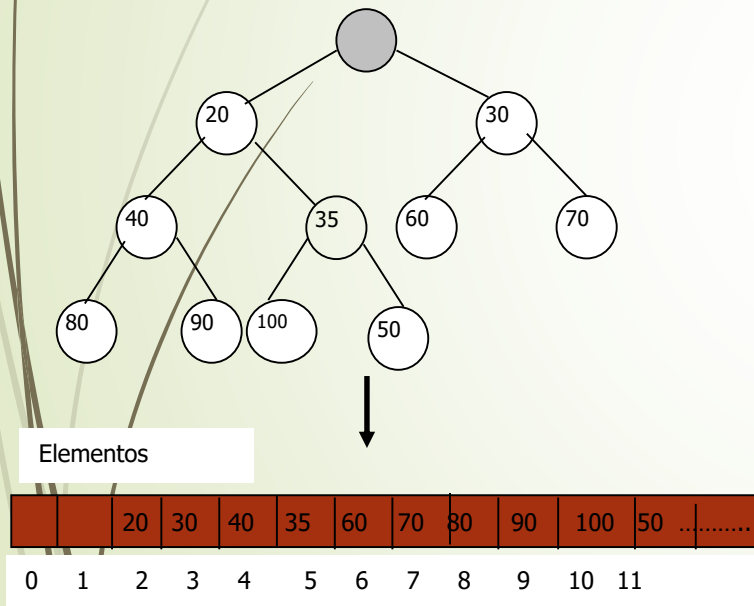


Elementos

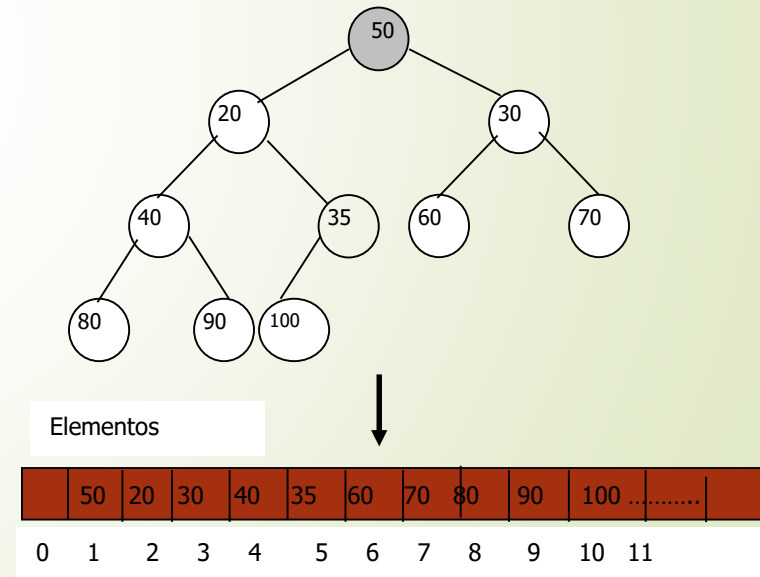
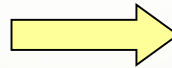
	10	20	30	40	35	60	70	80	90	100	50
0	1	2	3	4	5	6	7	8	9	10	11	

T.A.D. Montículo Binario – Construcción de operaciones abstractas (3)

Eliminar_Mínimo (M, X)



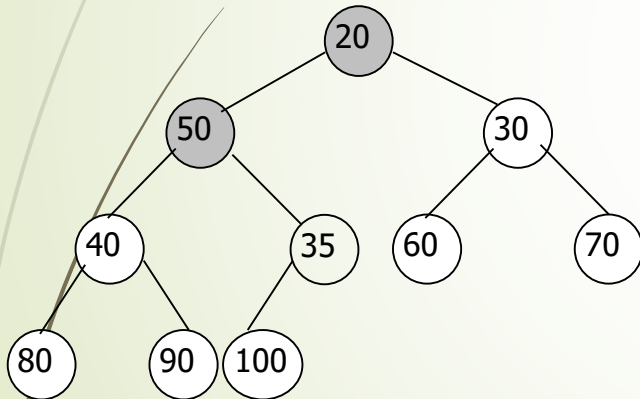
a) Propiedad de Estructura



T.A.D. Montículo Binario – Construcción de operaciones abstractas (4)

Eliminar_Mínimo (M, X)

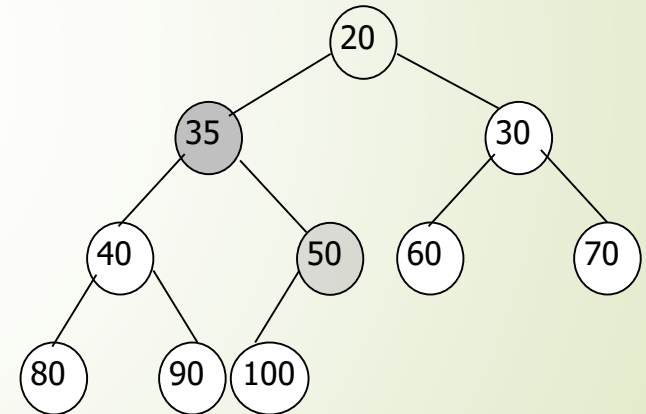
b) Propiedad de
Orden



Elementos

	20	50	30	40	35	60	70	80	90	100	
--	----	----	----	----	----	----	----	----	----	-----	-------	--

0 1 2 3 4 5 6 7 8 9 10 11



Elementos

	20	35	30	40	50	60	70	80	90	100	
--	----	----	----	----	----	----	----	----	----	-----	-------	--

0 1 2 3 4 5 6 7 8 9 10 11