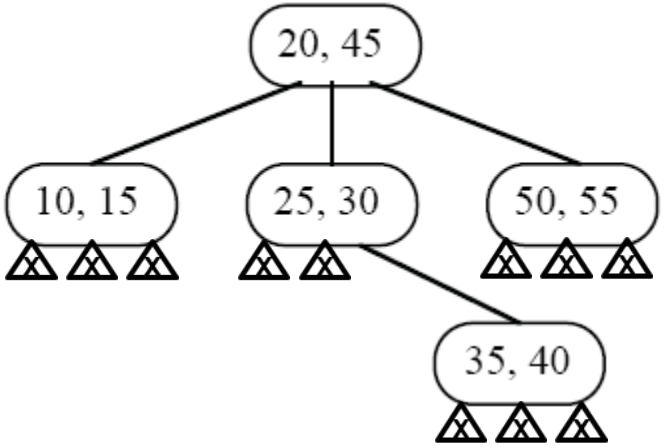
Árboles N-arios de Búsqueda

Lección 16

Definiciones

- Definición informal:
  - Los árboles *n*-arios de búsqueda son árboles multicamino (aquellos en los que cada nodo puede tener más de dos descendientes directos) cuyas ramas están ordenadas "a modo de" un árbol binario de búsqueda[1]…
  - En un árbol multicamino, de grado *n*, cada nodo interno puede tener como máximo *n* nodos descendientes, y puede almacenar como máximo *n-1* claves ordenadas (denominadas: claves de búsqueda)
- Ejemplo de árbol 3-ario de búsqueda:



1) Árboles binarios de búsqueda sin repetidos

Definiciones

- Los árboles *n*-arios de búsqueda (árboles de búsqueda múltiple o multicamino) son árboles de grado *n* definidos de la forma:
  - si el árbol *A* es vacío, entonces es un árbol *n*-ario de búsqueda;
  - si el árbol *A* es no vacío, entonces es un árbol *n*-ario de búsqueda si verifica:
    a) La raíz de *A* tiene *m* subárboles *n*-arios de búsqueda $A_1, \ldots A_m$, con $1 \leq m \leq n$;
    b) La raíz de *A* contiene la siguiente información: $(m, e_1, e_2, \ldots, e_{m-1})$ de forma que $e_{i-1} < e_i$, para $2 \leq i \leq m - 1$;
    c) Todo elemento *e* perteneciente al subárbol A_i es tal que $e < e_i$, para $1 \leq i \leq m-1$;
    d) Todo elemento *e* perteneciente al subárbol A_i es tal que $e > e_{i-1}$, para $1 \leq i \leq m$;

*) Con esta definición en los árboles *n*-arios de búsqueda no puede haber elementos repetidos

Definiciones

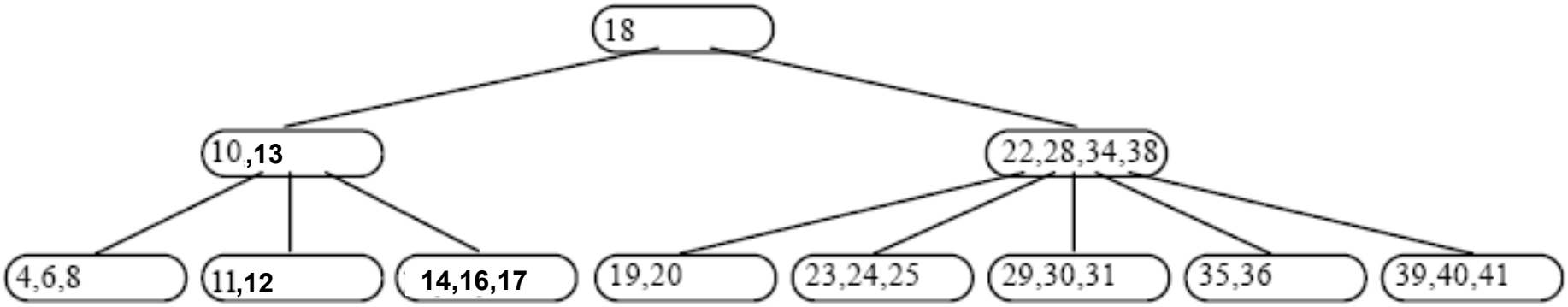
- Los árboles *n*-arios de búsqueda son especialmente útiles cuando se utilizan con datos ubicados en **almacenamiento externo** (los datos almacenados en disco, o similar, no en memoria principal), permitiendo reducir así el número de accesos a disco:
  - → Consultar, para comparar con la información de un nodo, supone un acceso a disco
  - → el grado del árbol se escoge como el valor más grande para que toda la información de un nodo quepa en un solo *bloque* o *página* de disco (unidad de lectura/escritura de disco)
- El coste de las operaciones de acceso y manipulación en los árboles n-arios de búsqueda será del orden de la altura del árbol
  - → Algoritmos de coste lineal en altura del árbol: 1 acceso por nodo consultado o nivel del árbol
  - ➢ Interesará que estos árboles sean lo más equilibrados posible (su altura sea mínima):
    - ➢ Árboles B y sus diferentes optimizaciones
- Los árboles B, y sus optimizaciones, se diseñaron para reducir al mínimo los accesos a disco, y se utilizan para implementar sistemas que requieran gran cantidad de accesos a disco (por ejemplo: sistemas de ficheros, sistemas de bases de datos, etc.)

Definiciones

- Un tipo particular de **árboles *n*–arios de búsqueda equilibrados** son los **árboles B**.
- Un **árbol B de orden *n*** tiene las siguientes propiedades:
  - es un árbol *n*–ario de búsqueda;
  - la raíz es una hoja o tiene al menos 2 hijos;
  - cada nodo, excepto la raíz y las hojas, tiene al menos $\lceil n/2 \rceil$ hijos;
  - Un nodo no hoja que tenga *k* hijos, contendrá *k*-1 claves
  - **todas las hojas están en un mismo nivel,**
    - en algunas definiciones se exige también que las hojas estén llenas al menos a la mitad de su capacidad, es decir que al menos tengan ((n-1) div 2) claves
- En la siguiente figura puede verse un árbol B de orden 5:

Nota: las restricciones respecto a los hijos deben cumplirse con hijos no vacíos

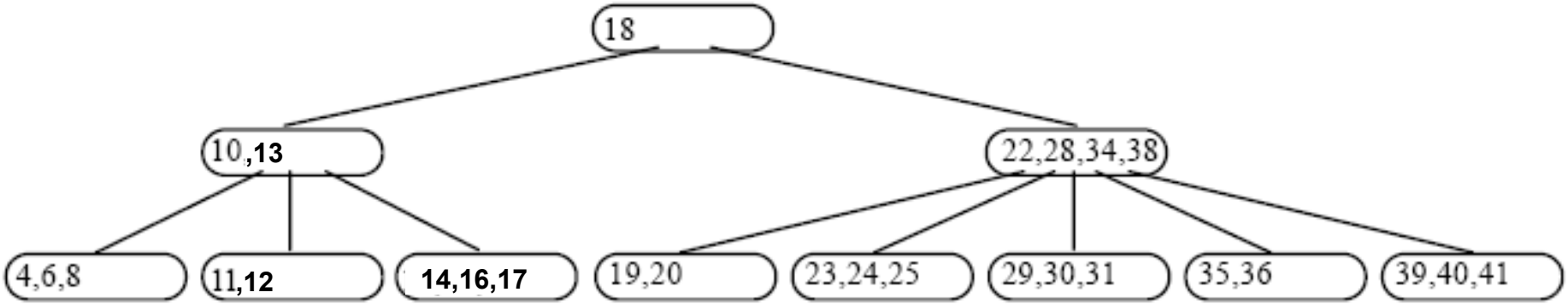
Es decir: ((n+1) div 2)



Definiciones

- Si hay M hojas y las hojas están en el nivel L:
  el nº de nodos en los niveles 1,2,3...
  es por lo menos 2, $2\lceil n/2 \rceil$, $2\lceil n/2 \rceil^2$, ...
  por tanto: $M \geq 2\lceil n/2 \rceil^{L-1}$
  es decir: $L \leq 1 + \log_{\lceil n/2 \rceil}(M/2)$

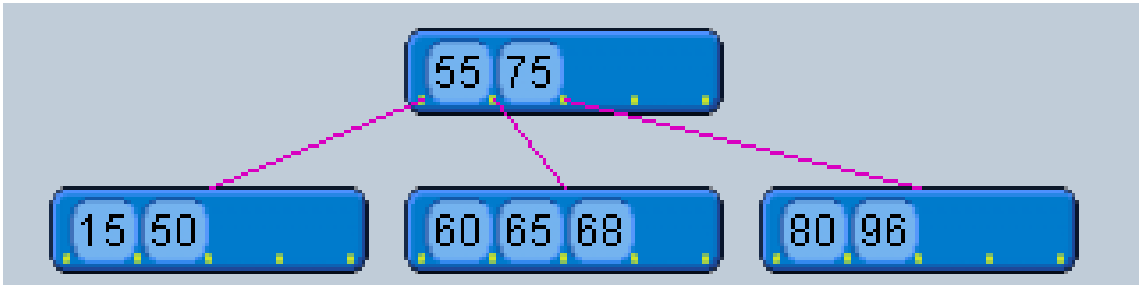
- Es decir, la altura está acotada por el logaritmo del nº de claves



Búsqueda, inserción y borrado en árboles B

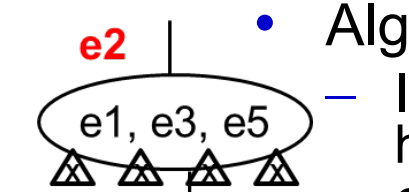
- Algoritmo de **búsqueda**: similar al estudiado con los ABBs…
  - La clave buscada, o se encuentra en el nodo o se baja a buscarla en uno de los hijos (en el único adecuado según las claves de búsqueda en el nodo), así hasta
    - encontrarla entre las claves de un nodo, o
    - llegar a una hoja que no la contiene (tendría que estar en un hijo suyo), y por tanto la clave buscada no está en el árbol
  - Coste lineal en la altura del árbol

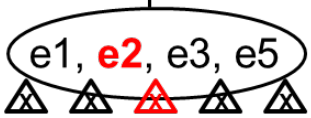
La imagen es un recorte de la ejecución del applet http://slady.net/java/bt/

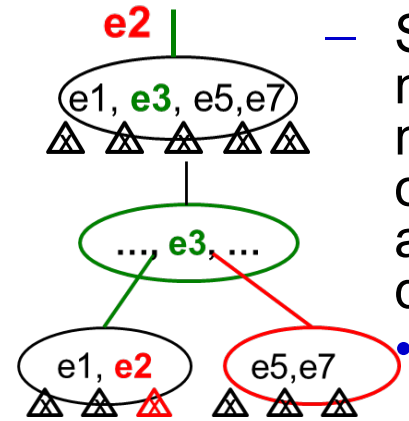


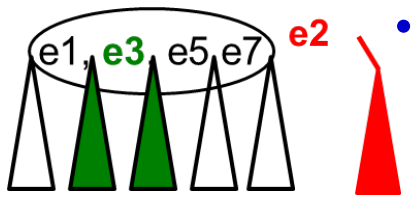
- Los árboles B de orden 3 se denominan también **árboles 2-3**:
  - Cada nodo, excepto las hojas, tiene 2 o 3 hijos

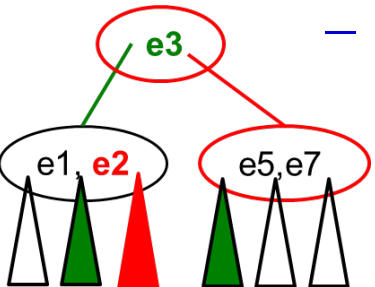
Búsqueda, inserción y borrado en árboles B







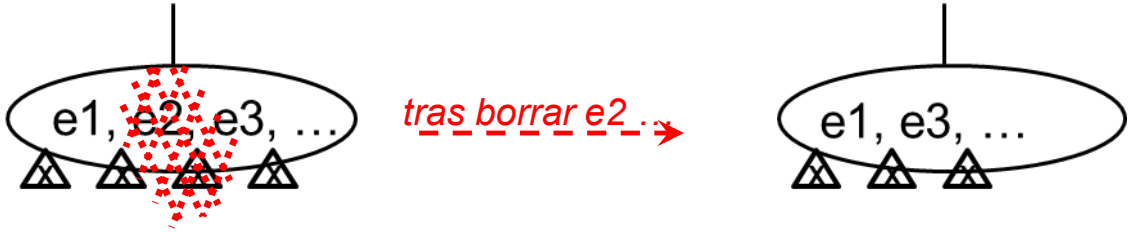




- Algoritmo de **inserción**:
  - Igual que en el algoritmo de búsqueda hasta encontrar el nodo hoja en el que debe insertarse la nueva clave,
  - Si en la hoja no está llena, puede contener la nueva clave. La clave se inserta en ella (y un nuevo hijo vacío a su derecha) dejando ordenadas las claves de la hoja
  - Si la hoja está llena se divide en dos, creándose un nuevo nodo hoja (a su derecha). Se reparten las claves entre los dos nodos, quedándose la mitad (inferior) el nodo que ya existía, y la otra mitad (superior) el nuevo nodo. La clave intermedia se sube al antecesor para ser usada como clave de búsqueda entre los dos árboles hijos. Y se repite el proceso.
    - Si el nodo antecesor no está lleno, aloja la nueva clave y el nuevo hijo (a su derecha), respetando las condiciones del orden
    - Si el nodo antecesor está lleno, no puede alojar la nueva clave y el nuevo árbol hijo, y se repite el proceso anterior, dividiendo el nodo en dos que se reparten las claves y la clave intermedia sube al antecesor del nodo.
  - En el peor de los casos se llega hasta la raíz, que se divide en dos creándose además una nueva raíz, provocando que el árbol crezca.
    - Los árboles B crecen de abajo hacia arriba, desde las hojas hacia la raíz → Coste lineal en la altura del árbol

Búsqueda, inserción y borrado en árboles B

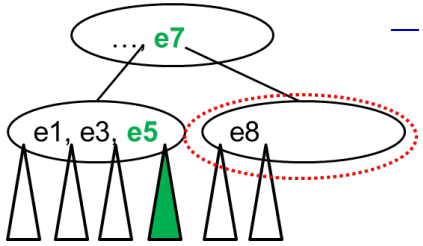
- Algoritmo de **borrado**:
  - Similar al algoritmo en los ABBs, se resuelve recursivamente.
    - Si la clave a borrar no está en una hoja, se sustituye por su predecesora más inmediata (la máxima del subárbol a su izquierda), o por su sucesora inmediata (la mínima del subárbol a su derecha), y dicha clave se borra del árbol en el que estaba.
    - Al final siempre se borra de una hoja.
  - Al borrar una clave de una hoja:
    - Si el número de claves que queda en la hoja es suficiente para cumplir las condiciones de árbol B, se suprime la clave y se acaba.



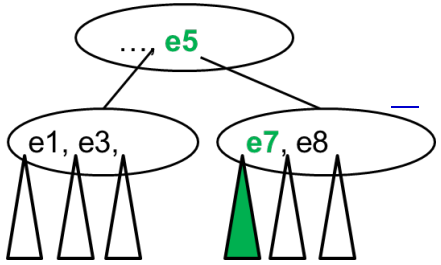
    - Si no quedan claves suficientes, se suprime la clave y se informa al padre (al volver de la llamada recursiva) de que ya no cumple las condiciones de árbol B
    - Al volver de la recursividad, el padre intentará que se vuelvan a cumplir las condiciones
      - ...

Búsqueda, inserción y borrado en árboles B

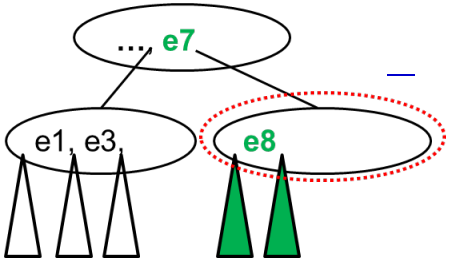
- ...
  - Al volver de la recursividad, el padre intentará que se vuelvan a cumplir las condiciones:
    - Si el hermano izquierdo (si existe) del nodo en el que se ha borrado está más de la mitad de lleno, este envía su última clave y su último hijo al padre. El padre se queda con esa clave como clave divisoria entre los dos nodos, y envía la que él tenía al nodo del que se borró la clave, en el cual quedará como primera clave, y además envía el que era último hijo de su hermano para que sea la rama a su izquierda (primer hijo)



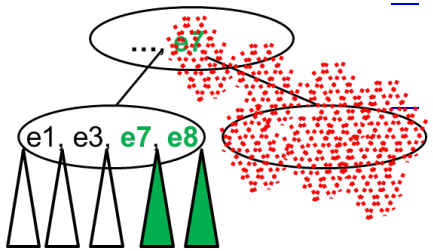
    - Si el hermano izquierdo tiene el mínimo de claves e hijos, se intenta con el hermano a la derecha (si existe) del nodo del que se borró, enviando este al padre su primer hijo y su primera clave. El padre se queda con esa clave como clave divisoria entre los hermanos, y pasa la clave que usaba antes, y el hijo, para que sean la última clave e hijo del nodo del que se borró.



    - Si el hermano no tiene suficientes claves, se baja la clave divisoria entre los dos hermanos, quitándola del padre, y fusionando los dos hermanos en un nodo que contendrá todas las claves que tenían más la que se ha bajado del padre.



    - La eliminación de la clave (y del subárbol hijo) del nodo padre, hace que en el se tenga que repetir el proceso



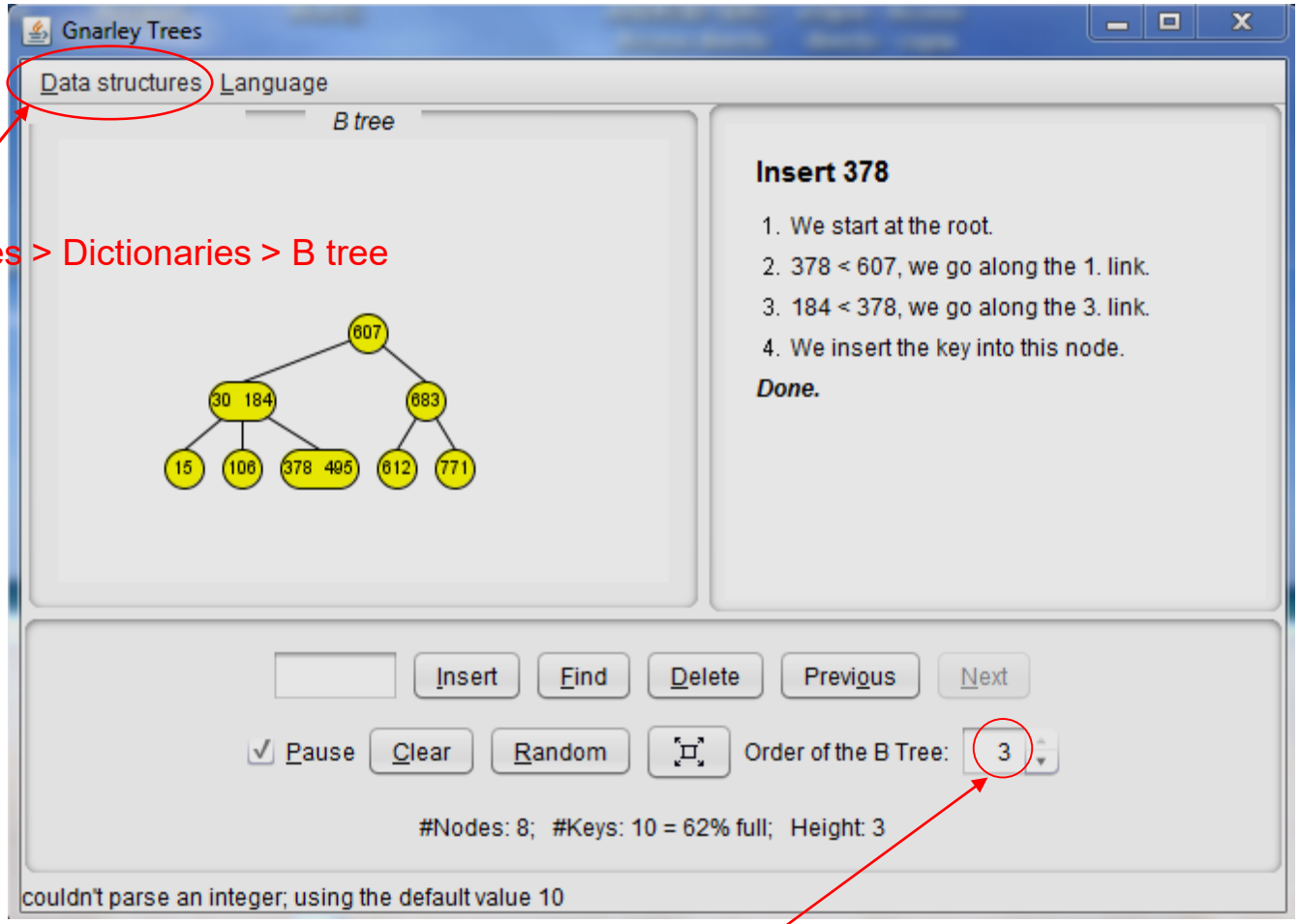
    - En el peor de los casos se llega a eliminar la raíz, fusionándose en un nodo los dos que antes eran sus hijos (y el árbol pierde altura)
      - Coste lineal en la altura del árbol

Definiciones

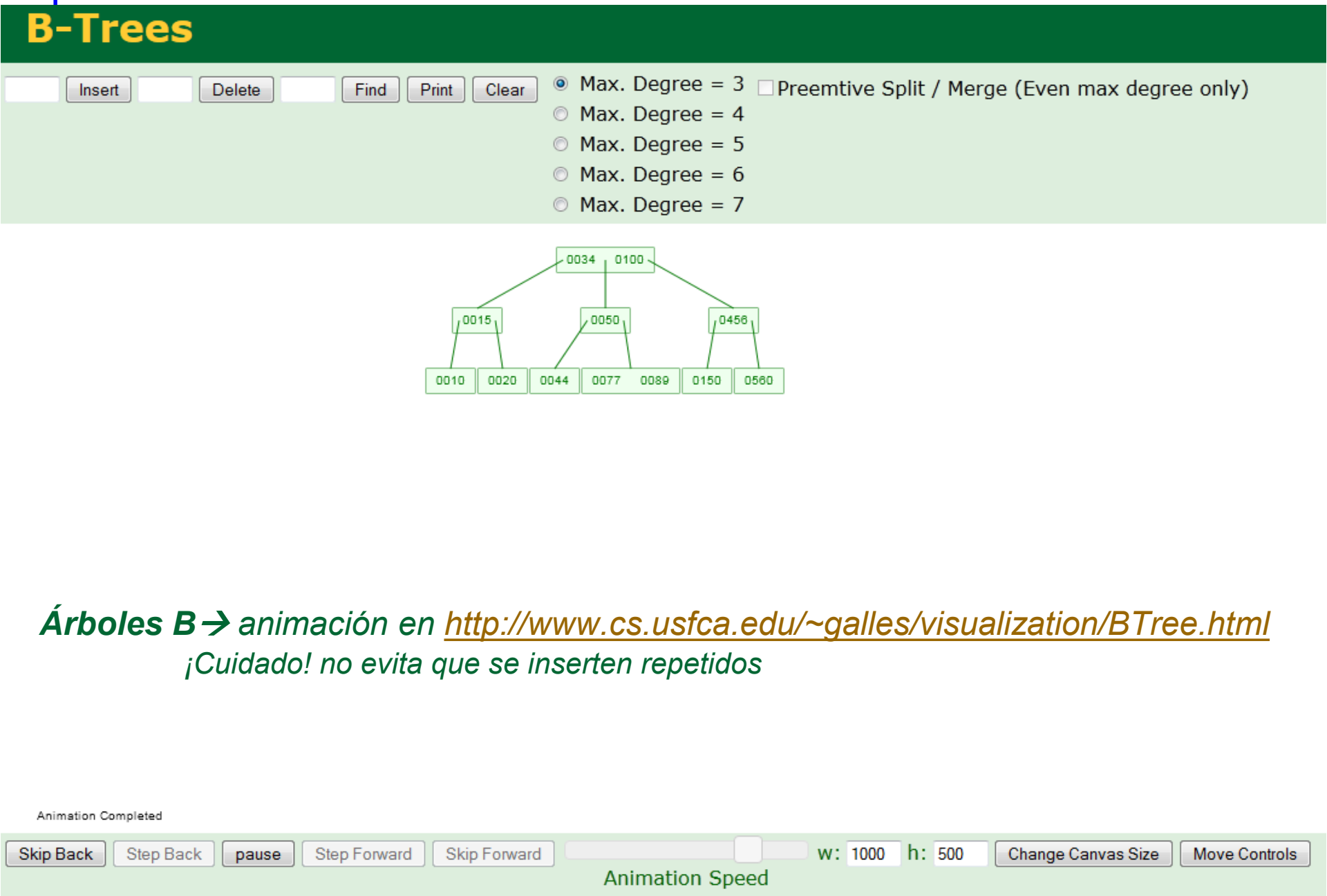
- Otra animación que permite visualizar paso a paso las operaciones sobre árboles B, y otros tipos de árboles:
  - https://kubokovac.eu/gnarley-trees/Btree.html

Data structures > Dictionaries > B tree

Este applet no permite insertar repetidos



  - Ejemplo visualizando un árbol B de orden 3 o árbol 2-3:



Árboles B → *animación en* http://www.cs.usfca.edu/~galles/visualization/BTree.html

¡Cuidado! no evita que se inserten repetidos

Definiciones

- Otra clase de árboles *n*–arios de búsqueda es la de los **árboles B***:
- Un **árbol B*** de orden *n* (*n*–ario), se define como:
  - es un árbol *n*–ario de búsqueda;
  - Todo nodo excepto la raíz tiene como mucho *n* hijos.
  - Cada nodo de un árbol B* de orden *n*, excepto la raíz y las hojas, tiene como mínimo $\lceil (2n-1)/3 \rceil$ hijos.
    - Es decir, está lleno al menos en 2/3 partes de su capacidad
  - La raíz de un árbol B* de orden *n* tiene como mínimo 2 y como máximo $2\lfloor (2n-2)/3 \rfloor + 1$ hijos.
    - Según definición dada en el libro "*The Art of Computer Programming*" de Knuth (vol 3, pag 488), la raíz puede llegar a tener 4/3 del grado
  - Todas las hojas de un árbol B* de orden *n* están en un mismo nivel.
  - Un nodo interno (no hoja) que tenga *k* hijos, contendrá *k*-1 claves.

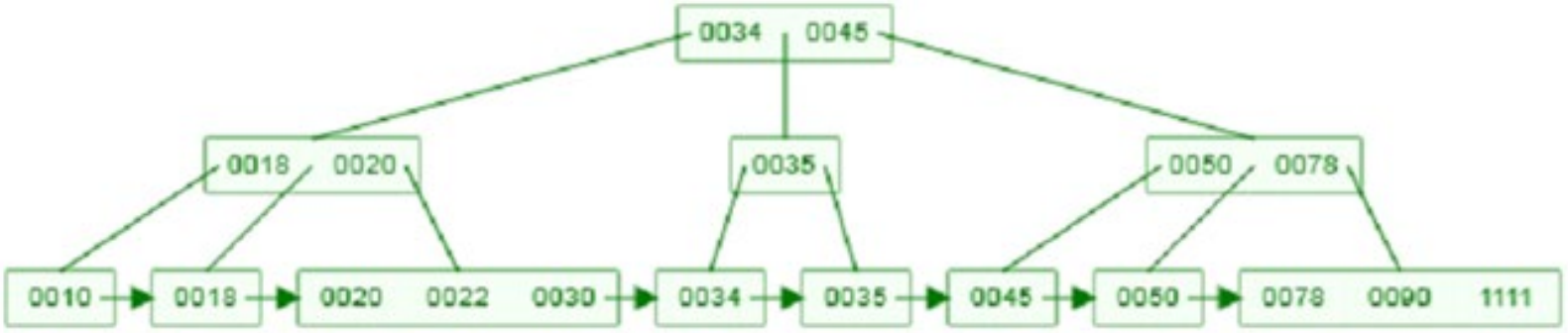
Nota: las restricciones respecto a los hijos deben cumplirse con hijos no vacíos

Definiciones

- Los **árboles B*** constituyen una mejora de los árboles **B** para aumentar el promedio de utilización de los nodos:
  - La inserción de claves en el árbol B* supone que si el nodo que le corresponde está lleno, se mueven las claves (e hijos) a uno de sus hermanos (similar al proceso en el borrado en los árboles B), y así se pospone la división del nodo hasta que ambos hermanos están completamente llenos
    - Cuando finalmente se dividan, esos dos hermanos que estaban llenos, pasarán a ser 3, y cada uno estará lleno en 2/3 partes de su capacidad

Definiciones

- Los **árboles B+** son otra mejora de los árboles **B**:
  - Mantienen la propiedad de acceso rápido en búsquedas para claves cualquiera, y
  - además permiten un recorrido secuencial rápido
- En un árbol B+ **todas las claves se encuentran en las hojas**, duplicándose en la raíz y en los nodos interiores aquellas que definen los caminos de búsqueda
  - Las claves en el nodo raíz e interiores se utilizan únicamente como índices
  - La información que acompañe a las claves solo estará en las hojas (o será accesibles desde ellas)
  - Para facilitar el recorrido secuencial rápido, las hojas (suelen) estar encadenadas

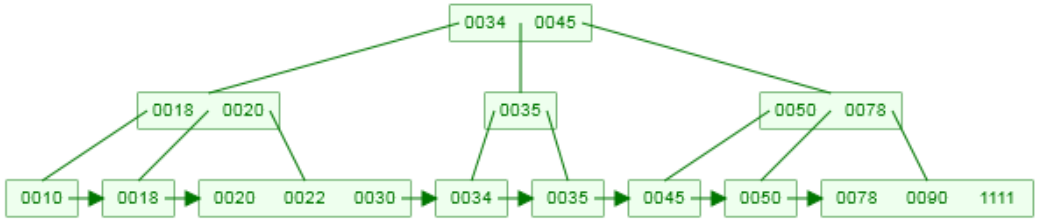


Esta imagen es un recorte de pantalla de una ejecución de la animación disponible en:
http://www.cs.usfca.edu/~galles/visualization/BPlusTree.html

B+ Trees

Insert | Delete | Find | Print | Clear

- Max. Degree = 3
- Max. Degree = 4
- Max. Degree = 5
- Max. Degree = 6
- Max. Degree = 7



Árboles B+ → *en* http://www.cs.usfca.edu/~galles/visualization/BPlusTree.html

¡Cuidado! no evita que se inserten repetidos

Animation Completed

Skip Back | Step Back | Pause | Step Forward | Skip Forward | w: 1000 h: 500 | Change Canvas Size | Move Controls

Animation Speed

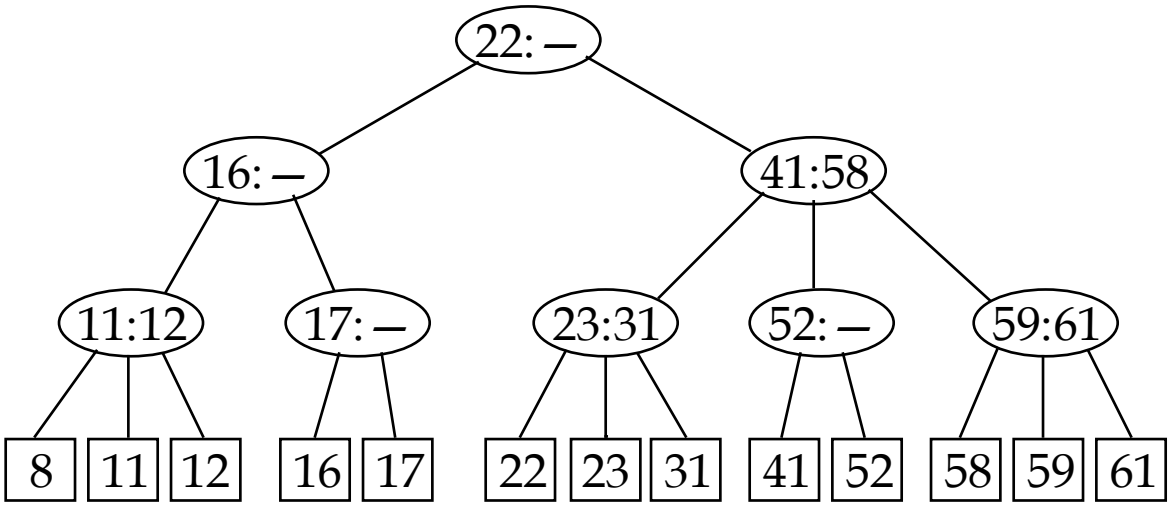
Más detalles sobre árboles **B**, **B***, **B+**, sus algoritmos, y usos:

- En el libro de la bibliografía : *"Clifford A. Shaffer: Data Structures and Algorithm Analysis, Edition 3.2 (C++ Version) March 2013."* (disponible en http://people.cs.vt.edu/shaffer/Book/) :
  - Capítulo 10: sobre arboles B, B*, B+, arboles 2-3, etc.
  - Capítulo 8: para repasar aspectos del acceso y manipulación de datos en almacenamiento secundario
- Estos árboles y sus algoritmos descritos en capítulo 10 del libro de la bibliografía: *"Joyanes, L., Zahonero, I., Fernández, M. y Sánchez, L.: Estructuras de datos. Libro de problemas, McGraw Hill, 1999."*
- Wikipedia: buena entrada sobre B-Trees (https://en.wikipedia.org/wiki/B-tree) que incluye la referencia y enlace a la publicación original de los árboles B
- Capítulo 4 del libro *"Weiss, M.A.: Data Structures and Algorithm Analysis in C++, Fourth Edition, Pearson/Addison Wesley, 2014."*

- *Y muchos otros....*

Ejemplo: árbol B de grado 3 (árboles 2-3)

Árbol de grado 3, a medio camino entre un B y un B+



Cada hoja almacena una clave (o dato), los nodos intermedios contienen claves que definen el camino de búsqueda similar a un B+, los algoritmos son similares a los de un B

Árbol 2–3 (según definición dada en [1]) :

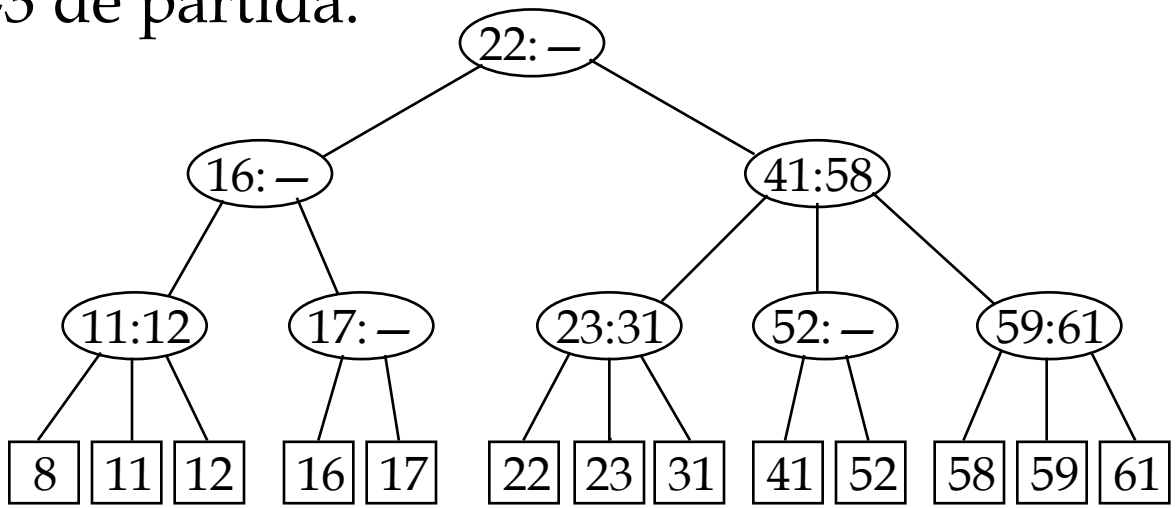
- *Cada nodo no hoja tiene 2 o 3 hijos.*
- *Todas las hojas están en el mismo nivel.*
- *Casos especiales: árbol de 0 o 1 nodo*
- *Si el elemento a está a la izquierda del elemento b, entonces a<b*

[1] *Extraído del libro "Data Structures and Algorithms", A.V. Aho, J.E. Hopcroft y J.D Ullman (1983)*

Ejemplo (de árbol B): árboles 2-3

Ejemplos de **inserción en árboles 2–3** (según definición dada en [1])

Árbol 2–3 de partida:



Árbol de grado 3, a medio camino entre un B y un B+

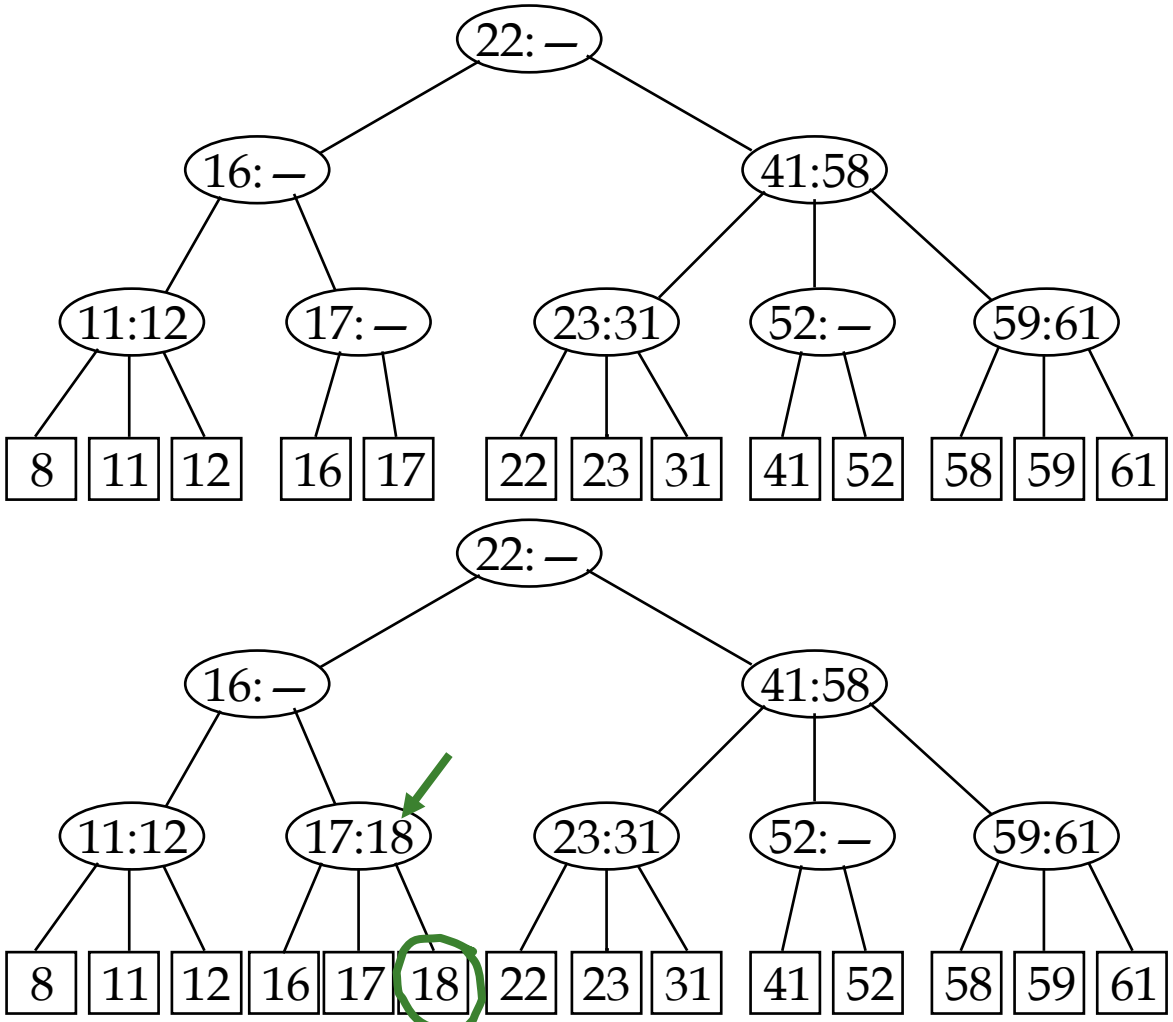
Como en un árbol B+, los datos están siempre en las hojas.

Siempre se inserta y borra una hoja, pero se deberán actualizar también las claves en el camino de búsqueda

[1] *Extraído del "Data Structures and Algorithms", A.V. Aho, J.E. Hopcroft y J.D Ullman (1983)*

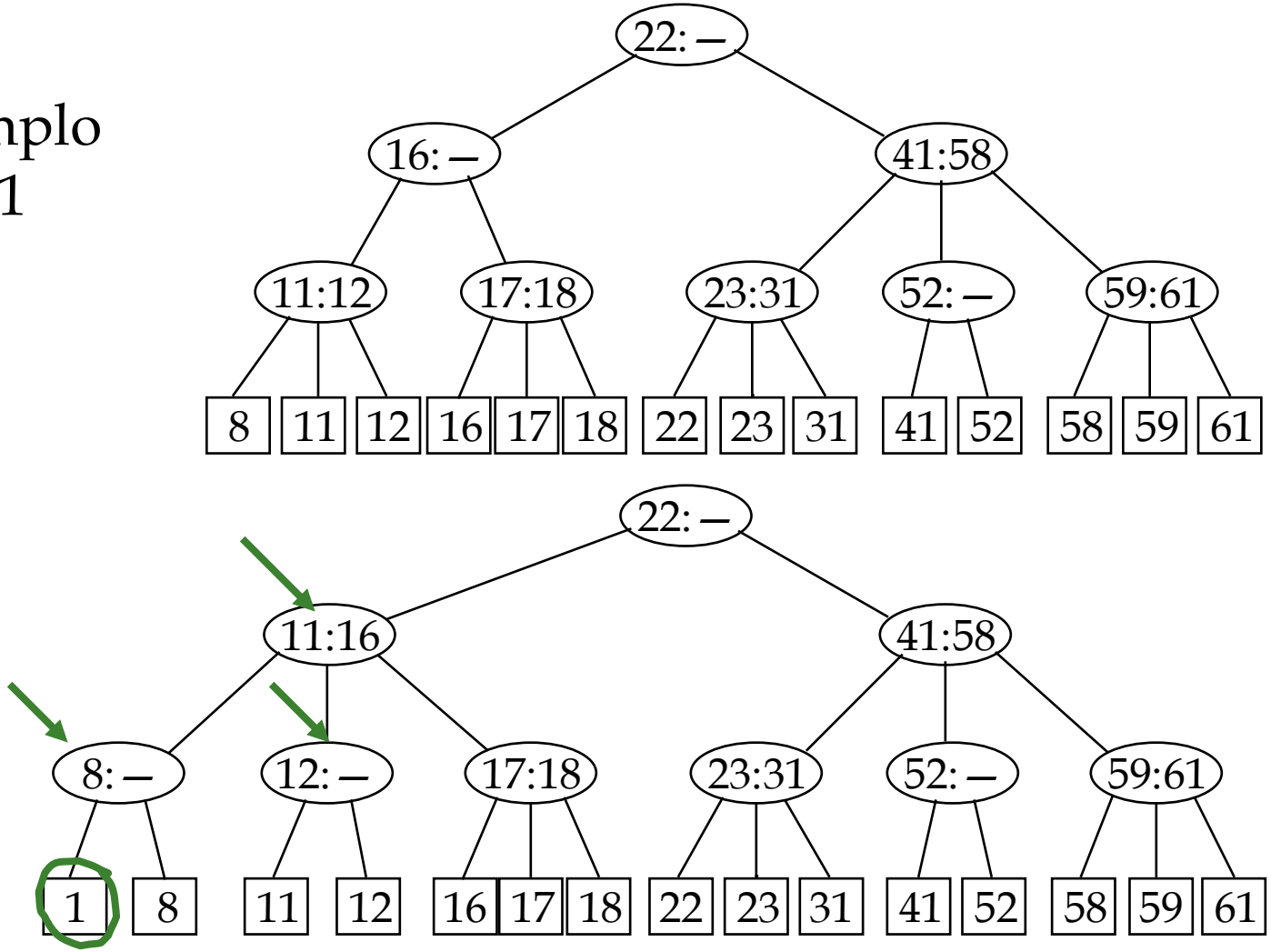
Ejemplo (de árbol B): árboles 2-3

Primer ejemplo de inserción: 18



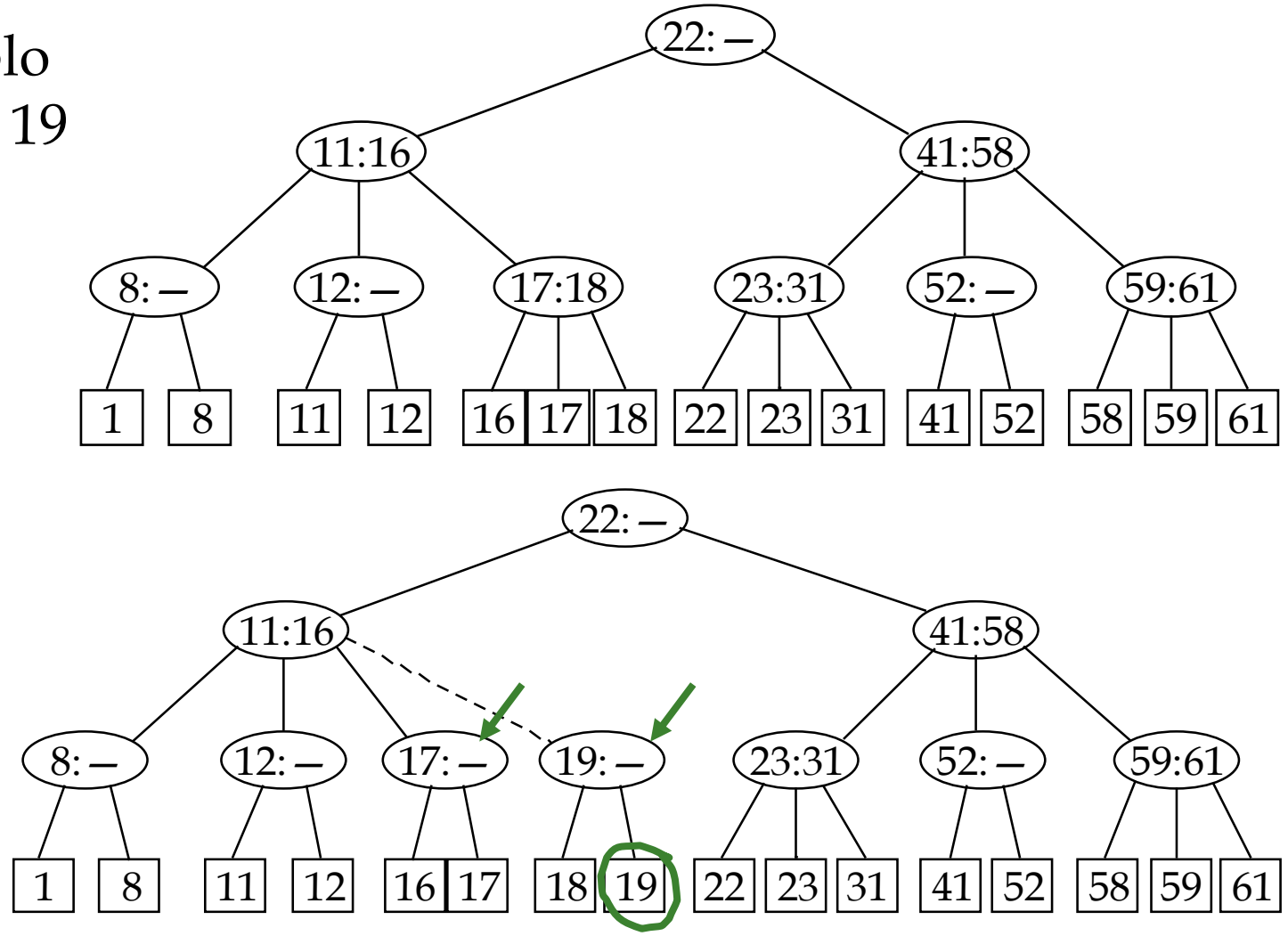
Ejemplo (de árbol B): árboles 2-3

Segundo ejemplo de inserción: 1



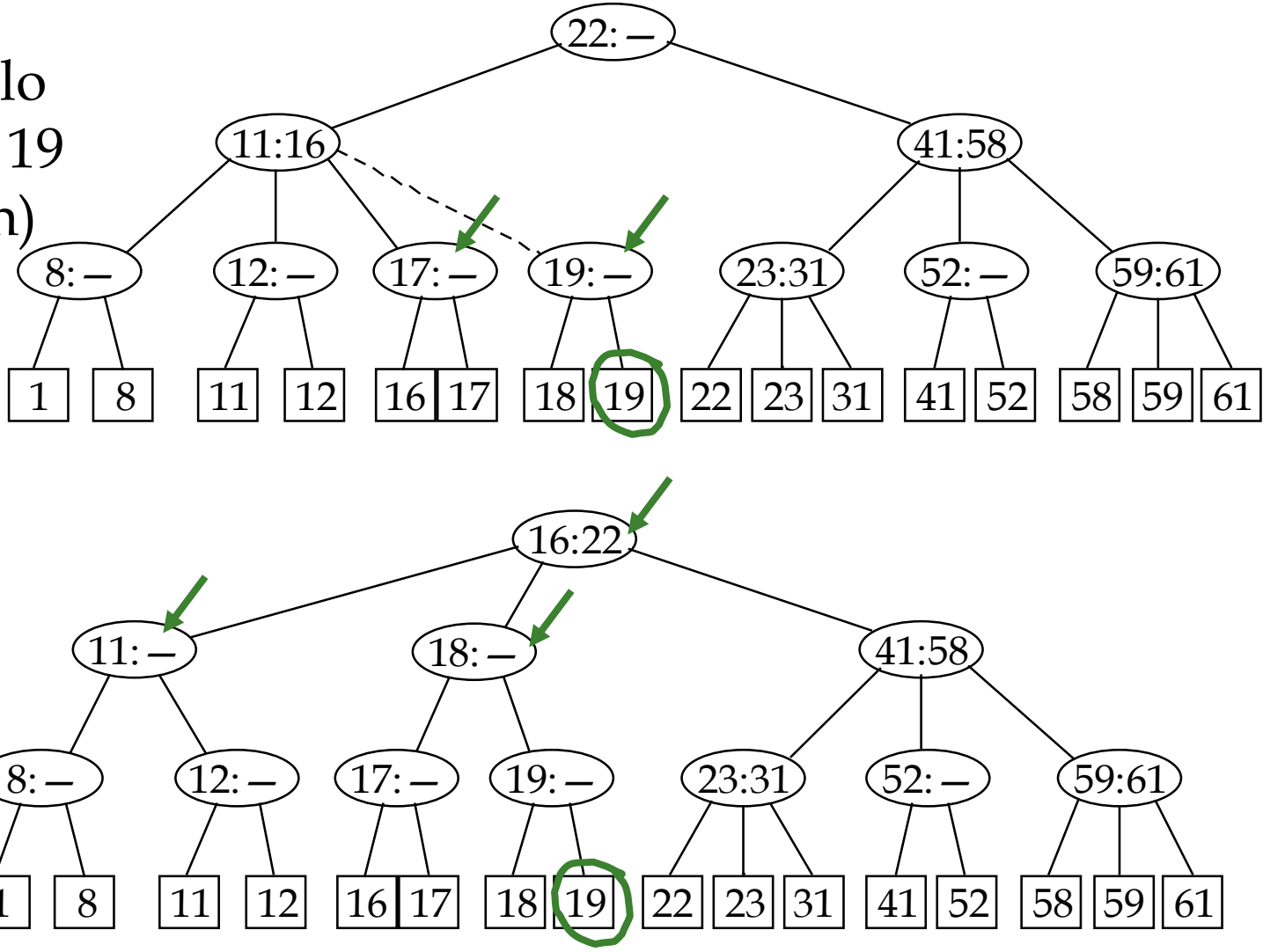
Ejemplo (de árbol B): árboles 2-3

Tercer ejemplo de inserción: 19



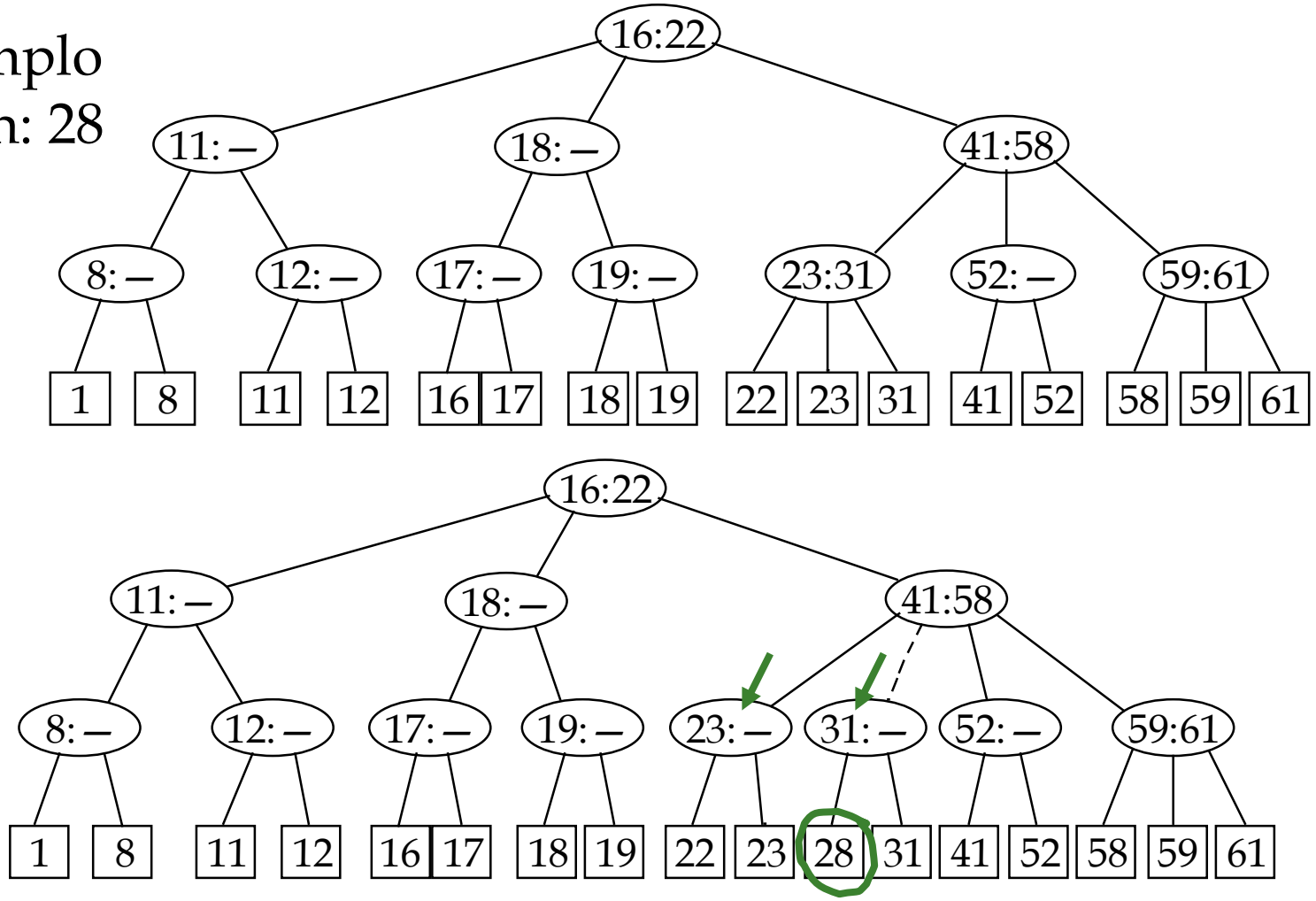
Ejemplo (de árbol B): árboles 2-3

Tercer ejemplo de inserción: 19 (continuación)



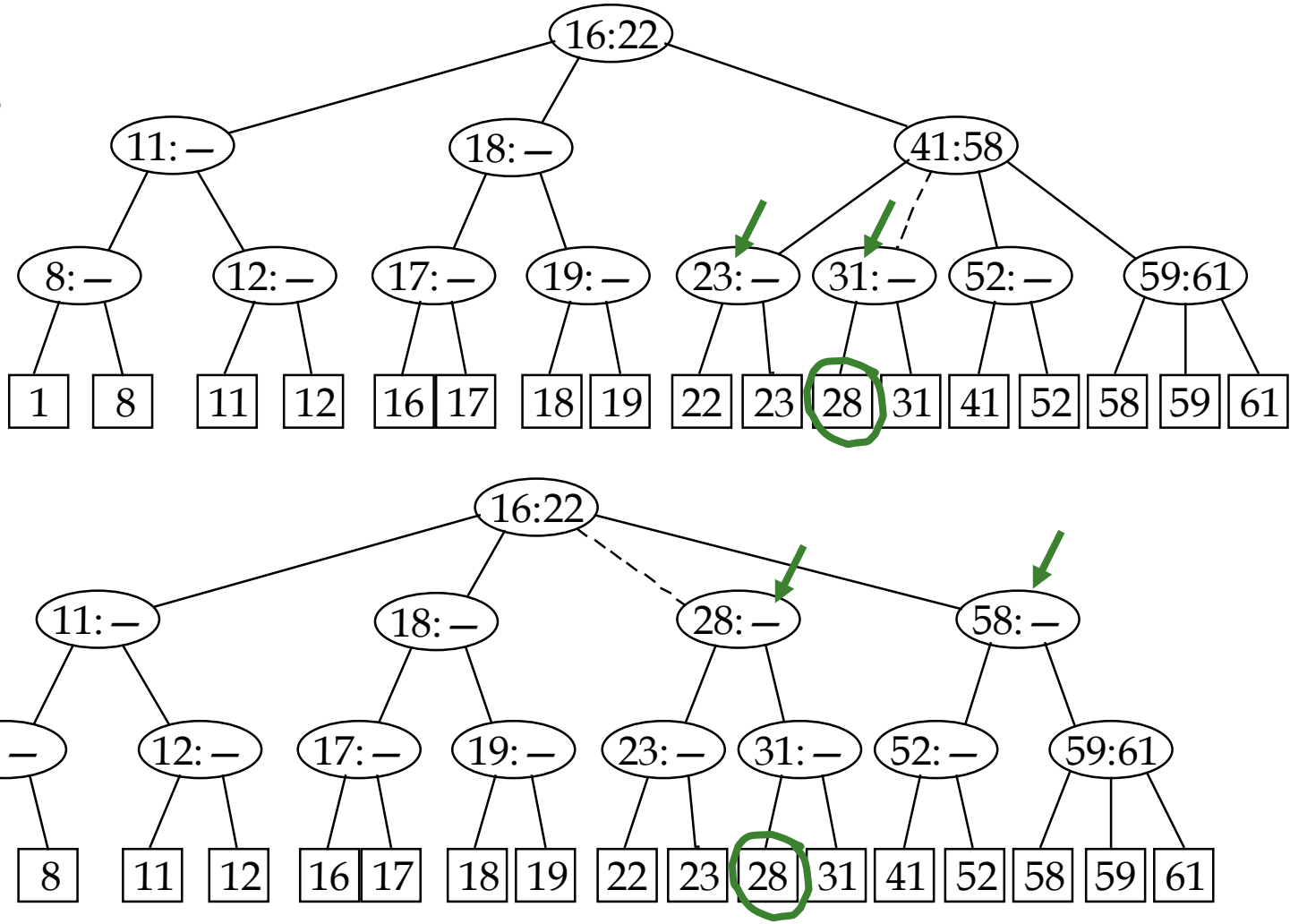
Ejemplo (de árbol B): árboles 2-3

Cuarto ejemplo de inserción: 28



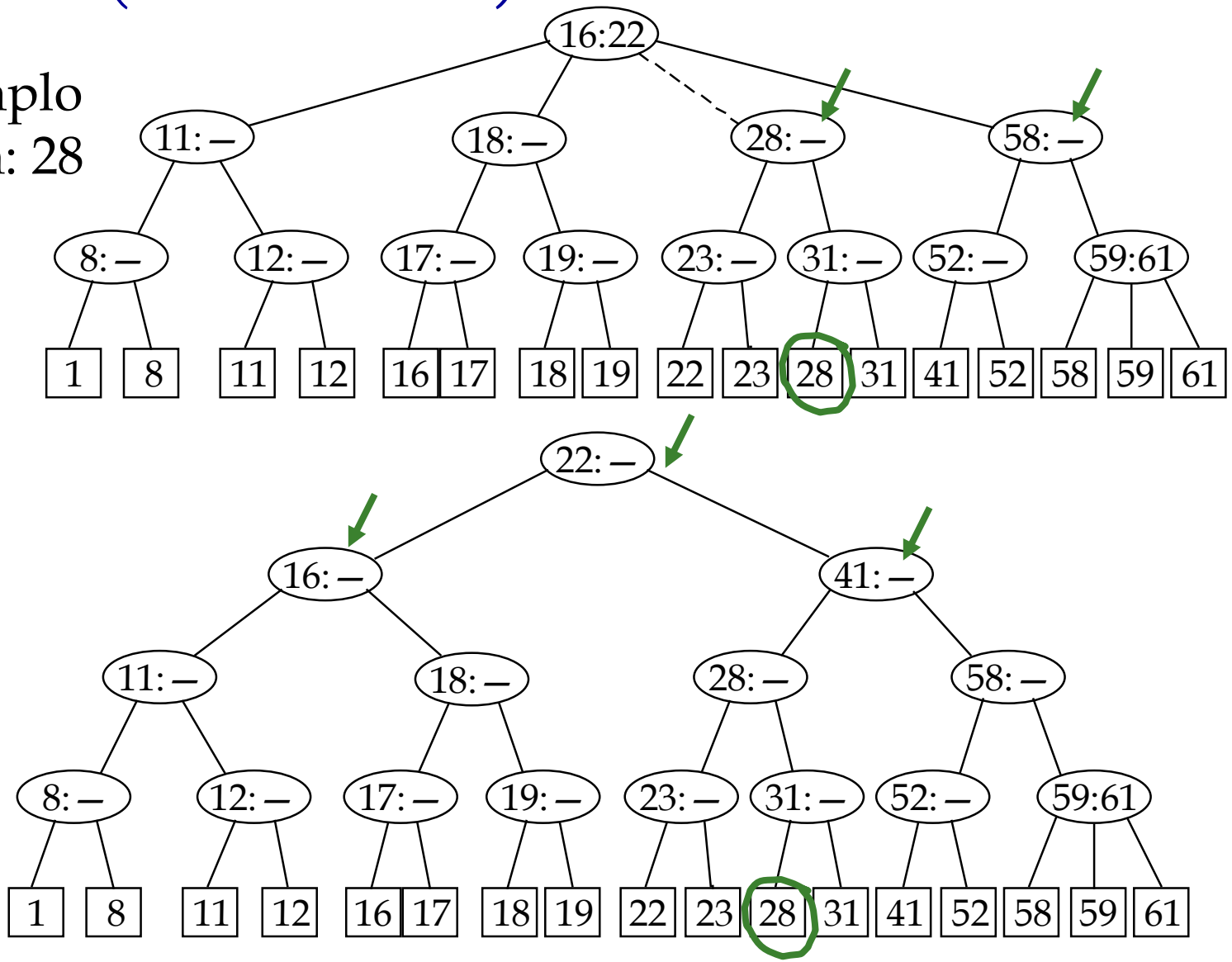
Ejemplo (de árbol B): árboles 2-3

Cuarto ejemplo de inserción: 28 (continuación)



Ejemplo (de árbol B): árboles 2-3

Cuarto ejemplo de inserción: 28 (final)



Estructura de datos para árboles 2-3

```
tipos
  tipo_elmto = registro
                 clave:tipo_clave;
                 ... {el resto de campos necesarios}
               freg;
  tipos_nodo = (hoja,interior);
  diccionario = ↑nodo_dos_tres;
  nodo_dos_tres =  registro
      clase:tipos_nodo;
        {el siguiente campo sólo se usa si clase=hoja}
      elmto:tipo_elmto;
        {los siguientes campos sólo se usan si clase=interior}
      primer_hijo,segundo_hijo,tercer_hijo:diccionario;
      menor_de_segundo,menor_de_tercero:tipo_clave
    freg
```

Inserción en árboles 2-3

```
procedimiento inserta(ent x:tipo_elmto; e/s S:diccionario)
  variables
    pt_atrás:diccionario; {punt. al nuevo nodo devuelto por inserta1}
    menor_atrás:tipo_clave; {valor mín. en el subárbol de pt_atrás}
    guardaS:diccionario {para almacenar una copia temporal de S}
principio
    {Debe incluirse AQUÍ un código de inserción apropiado para los casos
     particulares: si S está vacío, o si tiene un solo nodo (hoja)}
      ...
    insertaRec(S,x,pt_atrás,menor_atrás);

    si pt_atrás≠nil entonces {se ha creado una nueva rama hermana de la actual raíz !}
      {crea la raíz nueva; sus hijos están ahora apuntados por S y pt_atrás}
      guardaS:=S;
      nuevoDato(S);
      S↑.primer_hijo:=guardaS;
      S↑.segundo_hijo:=pt_atrás;
      S↑.menor_de_segundo:=menor_atrás;
      S↑.tercer_hijo:=nil
    fsi
fin
```

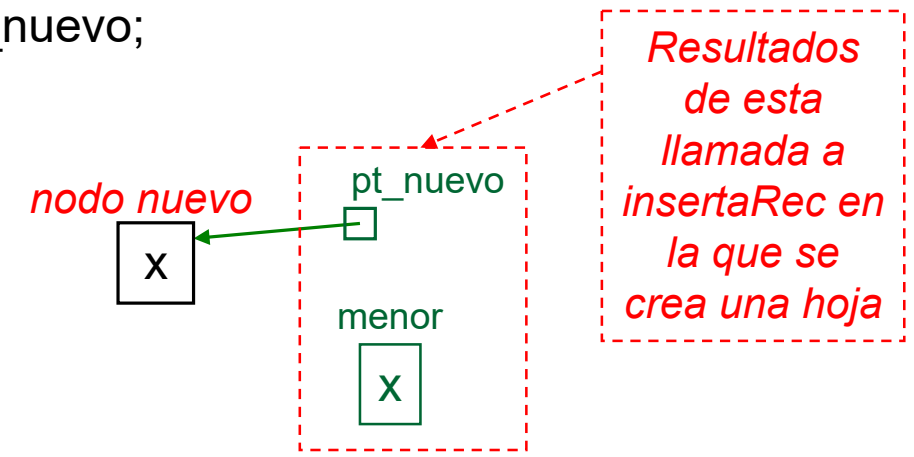
En este caso el árbol crece

Inserción en árboles 2-3

Primera aproximación

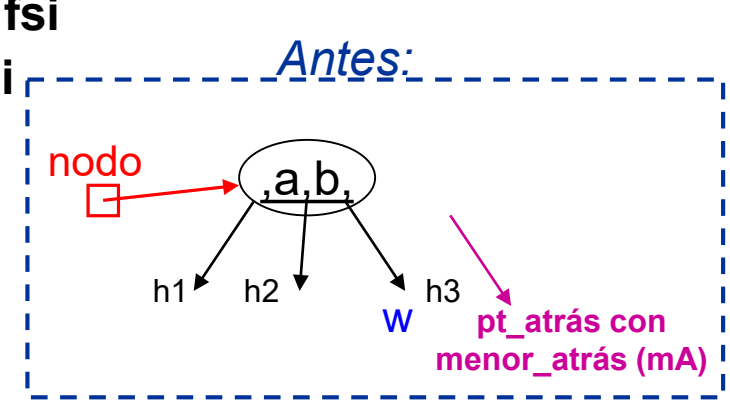
```
procedimiento insertaRec(e/s nodo:diccionario;
                         ent x:tipo_elmto;  {x se insertará en el subárbol de nodo}
                         sal pt_nuevo:diccionario;
                                  {puntero al nodo recién creado a la derecha de nodo}
                         sal menor:tipo_clave)
                                {elmto más pequeño del subárbol al que apunta pt_nuevo}
principio
  pt_nuevo:=nil;
  si nodo es una hoja entonces
    si x no es el elmto que está en nodo entonces   {creación de un hoja}
      crea un nodo nuevo apuntado por pt_nuevo;
      pone x en el nodo nuevo;
      menor:=x.clave
    fsi

 { sino … nodo es un nodo interno}
  ...
```

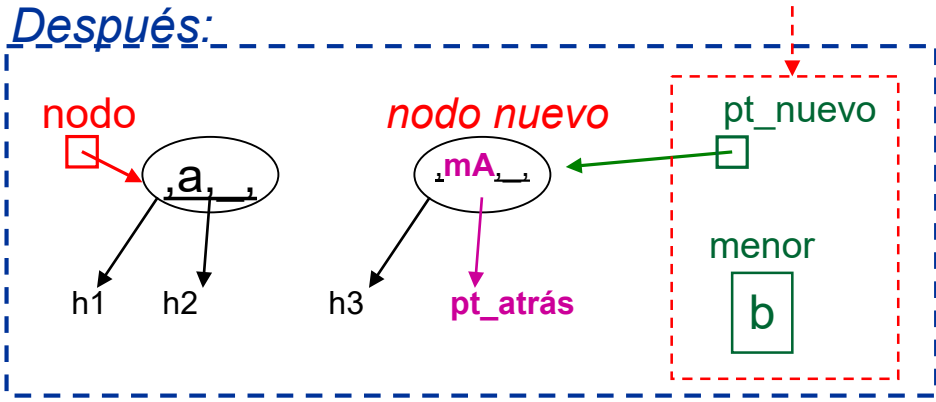
para el caso de crear una nueva rama



sino *{nodo es un nodo interno}*
sea w el hijo de nodo a cuyo subárbol pertenece x;
insertaRec(w,x,pt_atrás,menor_atrás);
si pt_atrás↑ ≠ nil **entonces** *{creada una nueva rama que debe ir a la derecha de w}*
inserta el puntero pt_atrás entre los hijos de nodo, a la derecha de w;
{el nodo queda arreglado: claves e hijos bien colocados, salvo si: }
si nodo tiene cuatro hijos **entonces**
{debe dividirse, y distribuir sus hijos y claves con un nuevo nodo que deberá quedar a su derecha:}
crea un nodo nuevo apuntado por pt_nuevo;
da al nuevo nodo los hijos 3º y 4º de nodo;
ajusta menor_de_segundo y menor_de_tercero en nodo en nodo nuevo;
asigna a menor la menor clave entre los hijos del nodo nuevo
fsi
fsi
fsi
fin

Resultados de esta llamada a insertaRec





w puede ser cualquiera de los 3 hijos, y el nuevo hijo tendrá que ir a su derecha

Inserción en árboles 2-3

Detalles…

```
procedimiento insertaRec(e/s nodo:diccionario;
        ent x:tipo_elmto;  {x se insertará en el subárbol de nodo}
        sal pt_nuevo:diccionario;
        {puntero al nodo recién creado a la dcha. de nodo}
        sal menor:tipo_clave)
                {elmto más pequeño del subárbol al que apunta pt_nuevo}

variables
   pt_atrás:diccionario;
   menor_atrás:tipo_clave;
   hijo:1..3; {indica qué hijo de nodo se sigue en la
              llamada recursiva (relacionado con la
              variable w del esquema anterior)}
   w:diccionario {puntero al hijo}
```

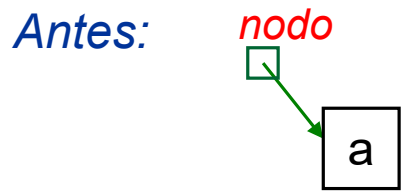
Inserción en árboles 2-3

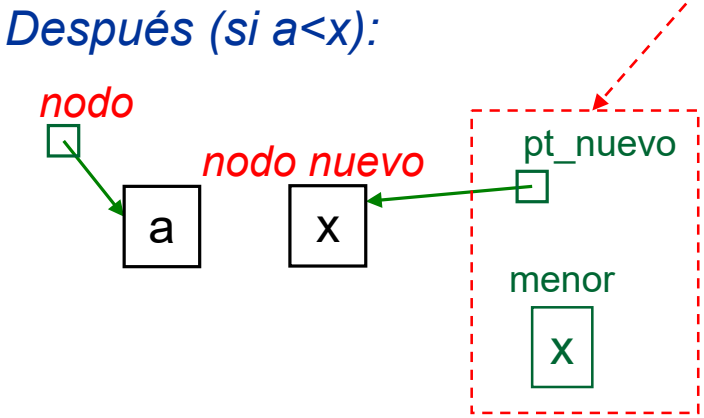
```
principio
   pt_nuevo:=nil;
   si nodo↑.clase=hoja entonces
        si nodo↑.elmto.clave↑≠x.clave entonces
             {crea una hoja nueva que contiene x.clave y "devuelve" este nodo}
             nuevoDato(pt_nuevo);
             pt_nuevo↑.clase:=hoja;
             si nodo↑.elmto.clave<x.clave entonces
                {pone x en el nuevo nodo a la dcha del nodo actual}
                pt_nuevo↑.elmto:=x;
                menor:=x.clave
             sino {x está a la izq del elemento en el nodo actual}
               pt_nuevo↑.elmto:=nodo↑.elmto;
               nodo↑.elmto:=x;
               menor:=pt_nuevo↑.elmto.clave
             fsi
        fsi
 { sino    nodo es un nodo interno}
...
```

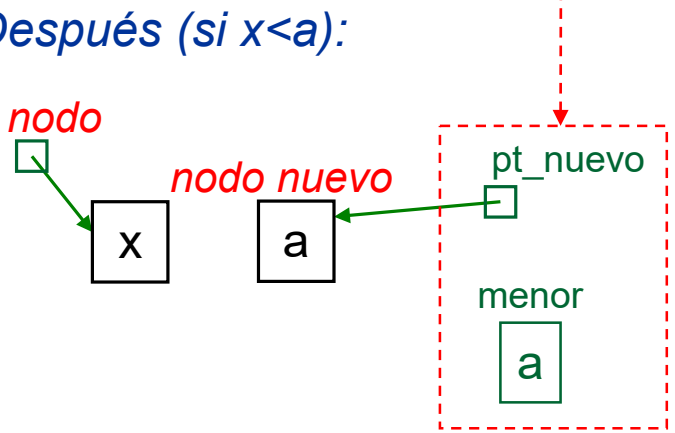
¿QUE HACE ESTO?

Inserción en árboles 2-3

```
{...
    si nodo↑.clase=hoja entonces
        si nodo↑.elmto.clave↑≠x.clave entonces
            {crea una hoja nueva que contiene x.clave y "devuelve" este nodo}
            nuevoDato(pt_nuevo);
            pt_nuevo↑.clase:=hoja;  ...
        ¿QUE HACE el trozo de código de la transparencia anterior?
        }
```







Inserción en árboles 2-3

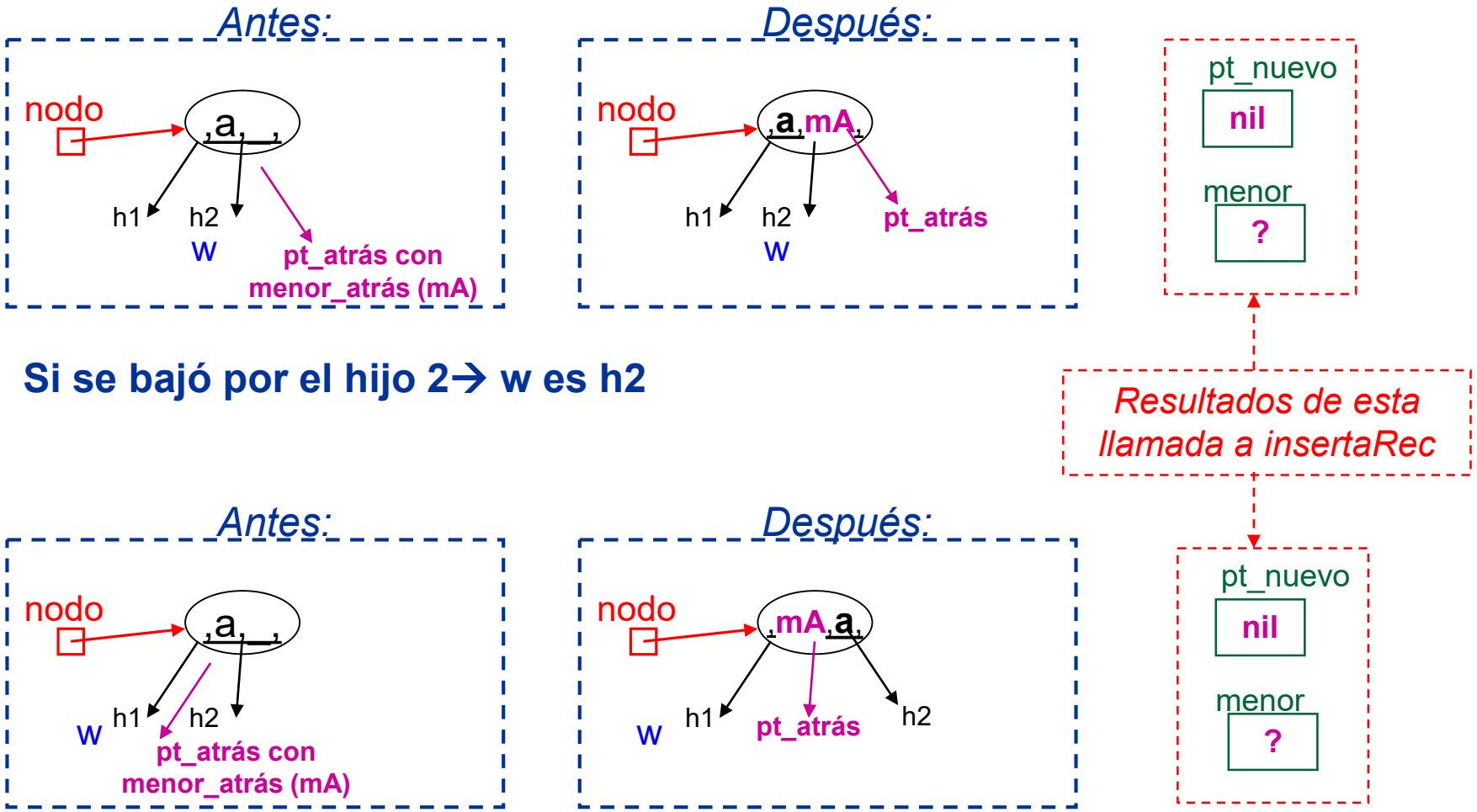
```
...
 sino {nodo es un nodo interno}
    {selecciona el hijo de nodo que se debe seguir}
    si x.clave<nodo↑.menor_de_segundo entonces
        hijo:=1;   w:=nodo↑.primer_hijo
    sino
       si (nodo↑.tercer_hijo=nil) or (x.clave<nodo↑.menor_de_tercero)
          entonces
           {x está en el segundo subárbol}
           hijo:=2; w:=nodo↑.segundo_hijo
       sino {x está en el tercer subárbol}
           hijo:=3; w:=nodo↑.tercer_hijo
       fsi
    fsi;
  {en la variable hijo recordamos por cuál de los hijos vamos a bajar (w)}
...
```

Inserción en árboles 2-3

```
insertaRec(w,x,pt_atrás,menor_atrás);

si pt_atrás≠nil entonces
    {se ha creado una nueva rama que debe quedar a la derecha de w:}

    {debe insertarse un nuevo hijo de nodo, y antes tenía 2 o 3 hijos}
    si nodo↑.tercer_hijo=nil entonces {nodo tiene 2 hijos:}
          si hijo=2 entonces {si se bajó por su hijo 2, el nuevo será el hijo 3}
               nodo↑.tercer_hijo:=pt_atrás;
               nodo↑.menor_de_tercero:=menor_atrás
          sino {si se bajó por su hijo 1:
                  el que antes era el hijo 2 será el hijo 3, y el nuevo será el hijo 2}
              nodo↑.tercer_hijo:=nodo↑.segundo_hijo;
              nodo↑.menor_de_tercero:=nodo↑.menor_de_segundo;
              nodo↑.segundo_hijo:=pt_atrás;
              nodo↑.menor_de_segundo:=menor_atrás
          fsi
   {sino    ....  nodo ya tiene tres hijos}
...
```

Inserción en árboles 2-3

Detalle casos de nodo con 2 hijos:



Si se bajó por el hijo 2→ w es h2

Si se bajó por el hijo 1→ w es h1

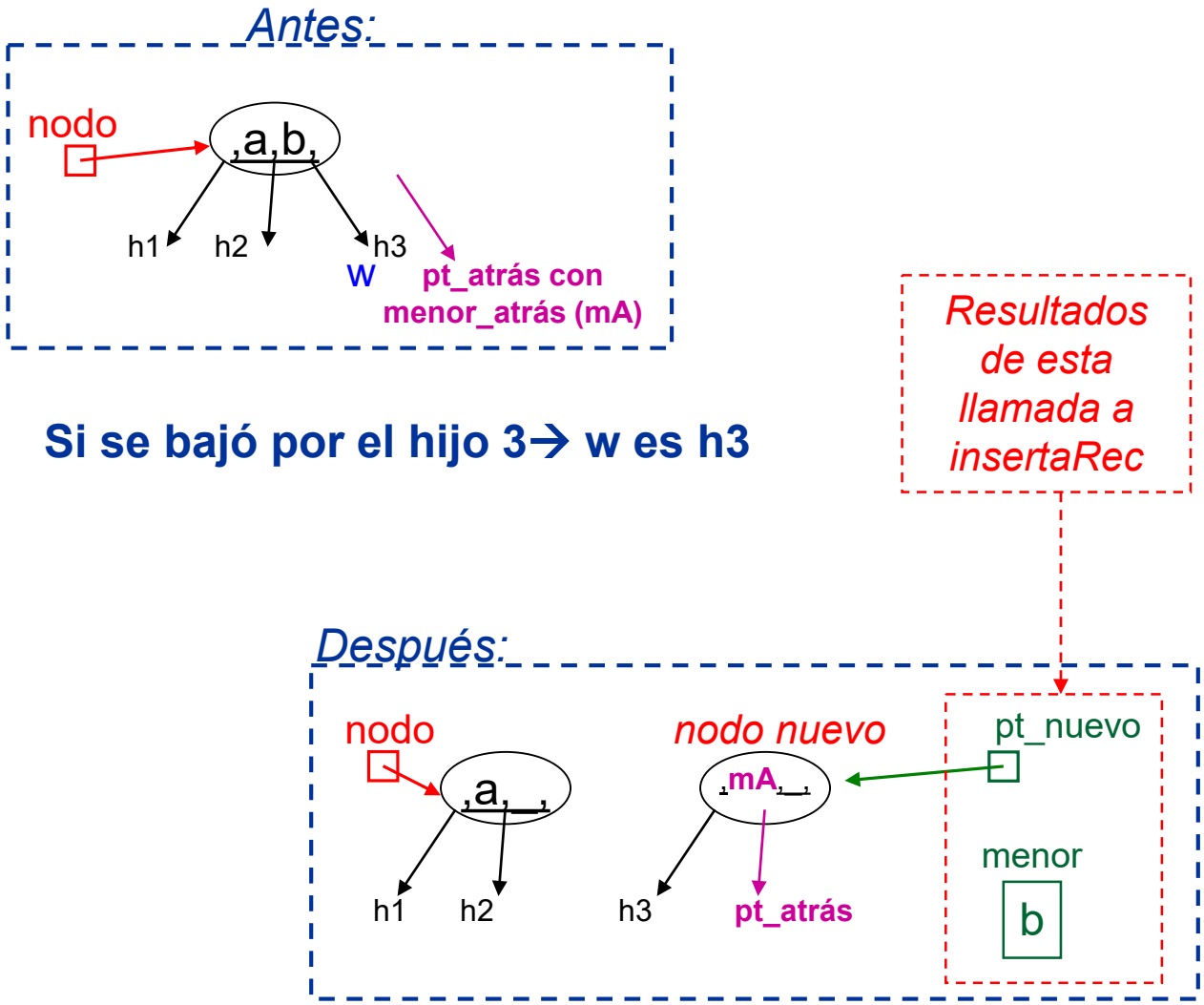
```
sino {nodo ya tiene tres hijos, con el nuevo ya son cuatro:}

   {creamos un nuevo nodo (pt_nuevo):}
   nuevoDato(pt_nuevo); pt_nuevo↑.clase:=interior;
  {y redistribuimos hijos y claves:}

   si hijo=3 entonces {si se bajó por su hijo 3, el nuevo será el hijo 4}
      {el tercer hijo y pt_atrás se convierten en 1er y 2º hijo del nuevo nodo}
      pt_nuevo↑.primer_hijo:=nodo↑.tercer_hijo;
      pt_nuevo↑.segundo_hijo:=pt_atrás;
      pt_nuevo↑.tercer_hijo:=nil;
      pt_nuevo↑.menor_de_segundo:=menor_atrás;
      {menor_de_tercero está indefinido para pt_nuevo}
      menor:=nodo↑.menor_de_tercero;
      nodo↑.tercer_hijo:=nil
   sino {si bajó por hijo≤2; el 3º hijo de nodo pasa a ser 2º hijo de pt_nuevo}
      pt_nuevo↑.segundo_hijo:=nodo↑.tercer_hijo;
      pt_nuevo↑.menor_de_segundo:=nodo↑.menor_de_tercer;
      pt_nuevo↑.tercer_hijo:=nil;
      nodo↑.tercer_hijo:=nil
   fsi;
...
```

Inserción en árboles 2-3

```
        si hijo=2 entonces
            {si se bajó por hijo 2; el 1er hijo de pt_nuevo será el recién creado, y
            el menor de toda su rama será el menor que está en el:}
            {pt_atrás se convierte en el 1º hijo de pt_nuevo}
            pt_nuevo↑.primer_hijo:=pt_atrás;
            menor:=menor_atrás
        fsi;
        si hijo=1 entonces
          {si se bajó por hijo 1; el 2º hijo pasa a ser el 1er hijo de pt_nuevo, y
            el recién creado será el segundo hijo del nodo original:}
            {el segundo hijo de nodo pasa a pt_nuevo;
             pt_atrás se convierte en el 2º hijo de nodo}
            pt_nuevo↑.primer_hijo:=nodo↑.segundo_hijo;
            menor:=nodo↑.menor_de_segundo;
            nodo↑.segundo_hijo:=pt_atrás;
            nodo↑.menor_de_segundo:=menor_atrás
         fsi
      fsi
    fsi
  fsi
fin
```

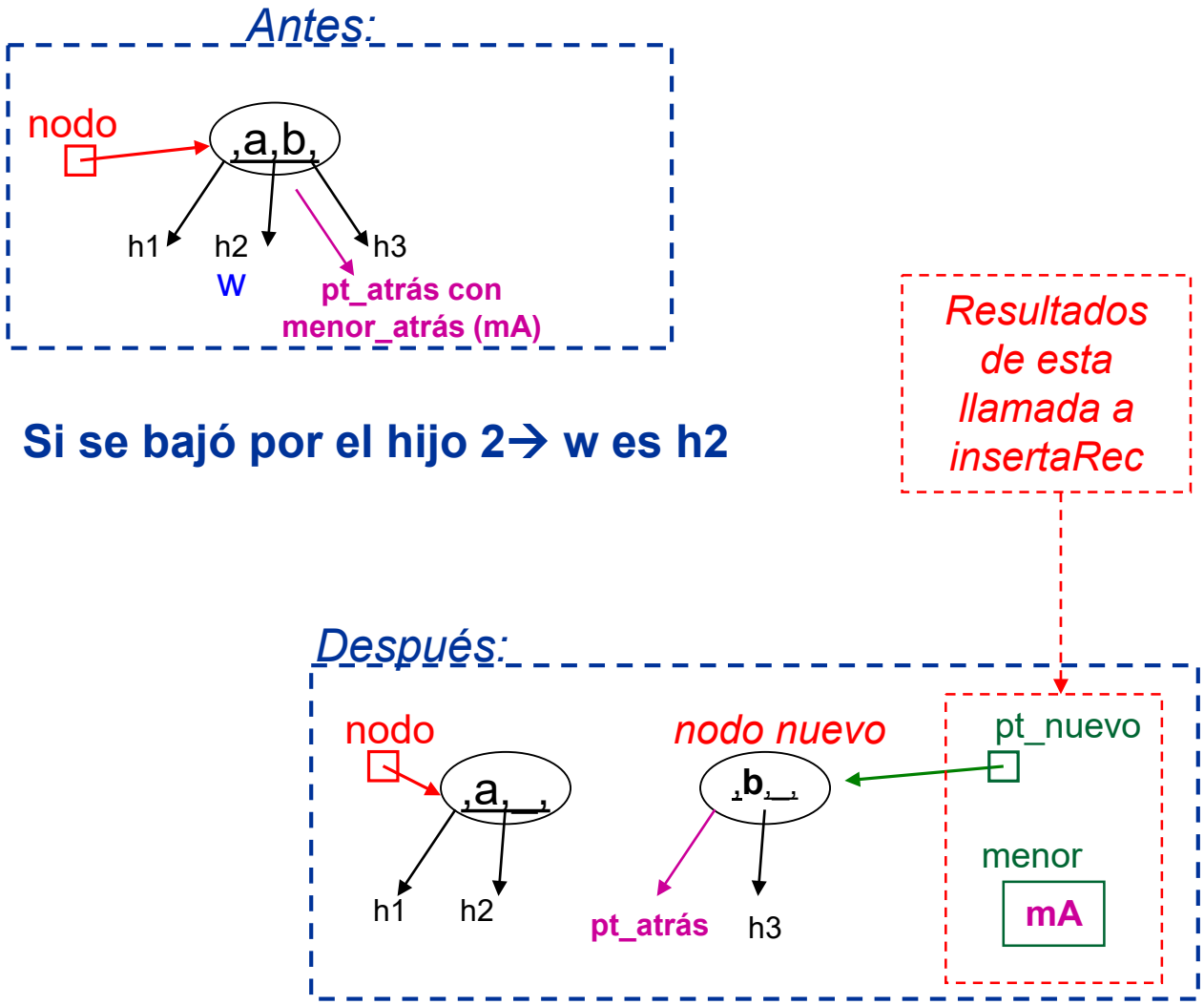
Inserción en árboles 2-3

Detalle casos de nodo con 3 hijos:



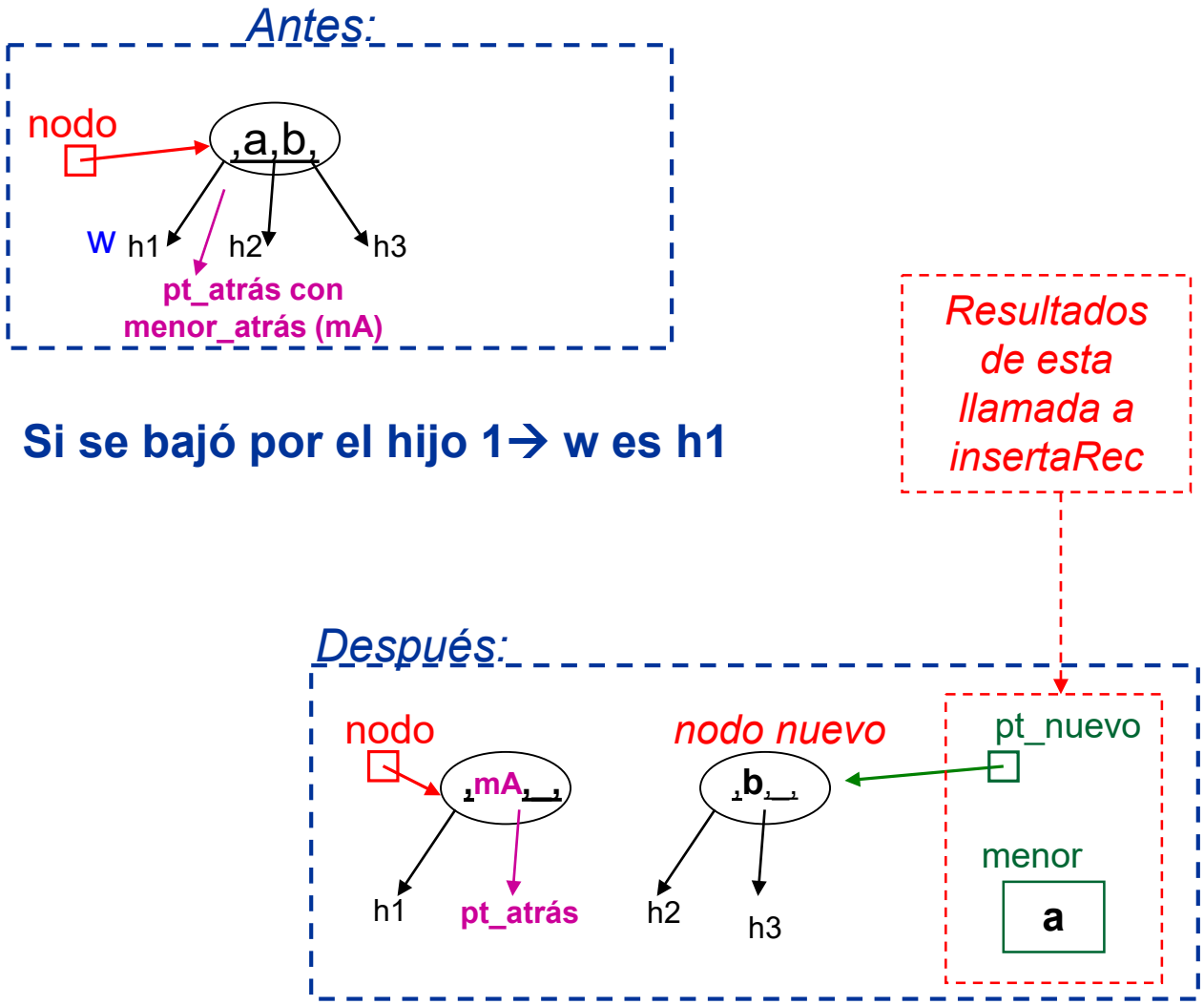
Inserción en árboles 2-3

Detalle casos de nodo con 3 hijos:



Inserción en árboles 2-3

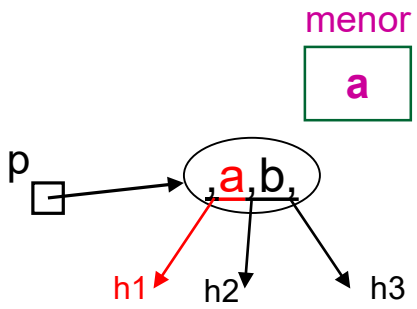
Detalle casos de nodo con 3 hijos:



Esquema del borrado en árboles 2-3

```
procedimiento borra(e/s d:diccionario; ent clave:tipo_clave)
  variables unHijo:booleano; pAux:ptNodo; menor:tipo_clave
principio

  si el diccionario tiene 0 ó 1 elemento entonces
      hacer el borrado de esos casos especiales

  sino {el diccionario tiene más de un elemento}
      borraRec(d,clave,unHijo,menor);
      si unHijo entonces {la raíz sólo tiene un hijo}
         eliminar la raíz y hacer que d sea el que ha quedado siendo
         el único hijo de d
      fsi
  fsi

fin
```

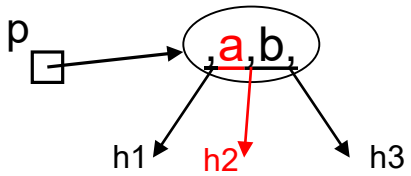
```
procedimiento borraRec(e/s p:ptNodo; ent clave:tipo_clave;
                       sal unHijo:booleano; sal menor:tipo_clave)
{borra 'clave' de 'p'; si 'p' se queda con un único hijo devuelve verdad
 en 'unHijo';  si 'clave' es la menor de p entonces 'menor' devuelve
 la siguiente clave mas pequeña de p}
  variables soloUno:booleano; w:ptNodo; hijo:natural
principio
   unHijo:=falso;
   si los hijos de p son hojas entonces
        si la clave a borrar es el primer hijo de p entonces
             se borra y se desplaza el segundo hacia la izquierda;
             se actualiza el valor de 'menor' con el nuevo primer hijo;
             si ahora p tiene un solo hijo entonces
                  unHijo:=verdad
             sino
                  se desplaza el tercero hacia su izquierda
             fsi
        { sino_si la clave a borrar es el segundo hijo de p entonces}
```



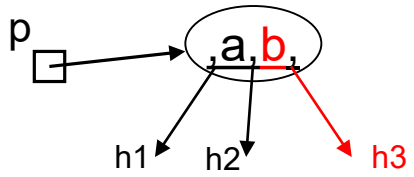
...

Esquema del borrado en árboles 2-3

sino_si la clave a borrar es el segundo hijo de p **entonces**
se borra;
si ahora p tiene un solo hijo **entonces**
unHijo:=verdad
sino
se desplaza el tercero hacia su izquierda con la clave
fsi



sino_si la clave a borrar es el tercer hijo de p **entonces**
se borra
fsi



sino *{los hijos de p no son hojas}*
{se selecciona el hijo de p por el que hay que bajar a borrar}
si puede estar en el primer subárbol **entonces**
hijo:=1; w:=primer hijo de p
sino_si puede estar en el segundo subárbol **entonces**
hijo:=2; w:=segundo hijo de p
sino *{puede estar en el tercer subárbol}*
hijo:=3; w:=tercer hijo de p
fsi; *{en la variable hijo recordamos por cuál de los hijos vamos a bajar (w)}*
...

Esquema del borrado en árboles 2-3

```
borraRec(w,clave,soloUno,menor);

si soloUno entonces
{arreglar los hijos de p para que ninguno tenga menos de dos hijos}
    si hijo=1 entonces {bajó por el primer hijo de p, y ahora sólo tiene un hijo}
        si el segundo hijo de p tiene tres hijos entonces
            se pasa el 1º hijo del 2º de p a 2º hijo del 1º de p
        sino {el segundo hijo de p tiene dos hijos}
            se pasa el hijo del 1º de p como 1º hijo del 2º de p
            y se elimina el primer hijo de p
            si ahora p sólo tiene un hijo entonces
                unHijo:=verdad
            fsi
        fsi

    {sino_si hijo=2 entonces el 2º hijo de p sólo tiene un hijo}
...
```

El hijo1 adopta a uno de los hijos de hijo2

El hijo1 de p desaparecerá, su único hijo es adoptado por hijo2 de p

sino_si hijo=2 **entonces** *{bajó por el 2º hijo de p, y ahora sólo tiene un hijo}*

El hijo2 adopta a uno de los hijos de hijo1

si el primer hijo de p tiene tres hijos **entonces**
    se pasa el 3º hijo del 1º de p como 1º hijo del 2º de p
    si se ha borrado la clave menor del 2º hijo de p **entonces**
        se actualiza con el valor 'menor' el campo adecuado
    sino *{no ha cambiado la clave menor de 2º hijo de p}*
        no se usa el valor 'menor'
    fsi

El hijo2 adopta a uno de los hijos del hijo3

sino_si el 3º hijo de p existe y tiene tres hijos **entonces**
    se pasa el 1º hijo del 3º de p como 2º del 2º de p
    si se ha borrado la menor clave del 2º hijo de p **entonces**
        se actualiza con el valor ´menor´ el campo adecuado
    fsi

El hijo2 de p desaparecerá, su único hijo es adoptado por hijo1 de p

sino *{ningún otro hijo de p tiene tres hijos}*
    el único hijo del 2º de p pasa a ser el 3º del 1º de p
    si se ha borrado la menor clave del 2º hijo de p **entonces**
        se actualiza con el valor 'menor' el campo adecuado
    sino
        no se usa el valor 'menor'
    fsi;
    el hijo 3º de p pasa a ser el 2º de p, si existe
    si p se ha quedado con un solo hijo **entonces**
        unHijo:=verdad
    fsi
fsi ...

```
        sino {bajó por el 3er hijo de p, y ahora sólo tiene un hijo}
            si el segundo hijo de p tiene tres hijos entonces
                se pasa el 3º hijo del 2º de p como 1º del 3º de p
                si se ha borrado la menor clave del 3º hijo de p entonces
                   se actualiza con el valor 'menor' el campo adecuado
                sino
                   no se usa el valor 'menor'
                fsi
            sino {el segundo hijo de p tiene dos hijos}
                   el único hijo del 3º de p pasa como 3º del 2º de p
                   si se ha borrado la menor clave del 3º hijo de p entonces
                      se actualiza con el valor 'menor' el campo adecuado
                   sino
                      no se usa el valor 'menor'
                   fsi
            fsi
        fsi
        sino {soloUno=falso; todos los hijos de p tienen 2 ó 3 hijos}
            cambiar, si hace falta, los campos que guardan la menor
            clave del segundo y del tercer subárbol de p
        fsi
    fsi
fin
```

El hijo3 adopta a uno de los hijos de hijo2

El hijo3 de p desaparecerá, su único hijo es adoptado por hijo2 de p