Algoritmos y Estructuras de Datos

Arboles Binarios

Resultados Esperados del Aprendizaje

El propósito general de este módulo es introducir los árboles como estructura de datos intrínsecamente recursiva, por medio del caso más generalizado, esto es, los árboles binarios. Se presentarán las características principales de la estructura y los algoritmos y operaciones fundamentales, buscando la aplicación inmediata en problemas reales.

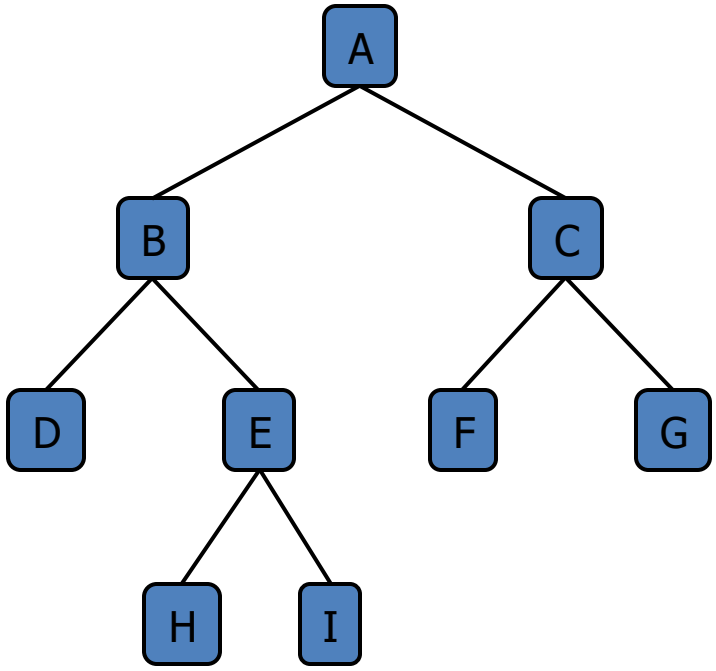
Al finalizar exitosamente esta unidad de aprendizaje serás capaz de:

- Utilizar árboles binarios para representar conjuntos de datos dinámicos en los cuales sea necesario agregar, eliminar, buscar y procesar información.
- Implementar árboles binarios balanceados cuando se requiera asegurar un orden del tiempo de ejecución logarítmico.
- Calcular el orden del tiempo de ejecución de las operaciones propias de los árboles binarios.
- Expresar procesos y algoritmos sencillos de árboles binarios en lenguaje natural, llevándolos luego a seudocódigo para finalmente implementarlos en JAVA.
- Desarrollar y programar aplicaciones sencillas que utilicen árboles binarios como repositorios de información en los cuales se desee agregar, eliminar o buscar datos.
- Desarrollar programas en JAVA más robustos y correctos, al mejorar la capacidad de diseño y desarrollo de Casos de Prueba (Test de Unidad).

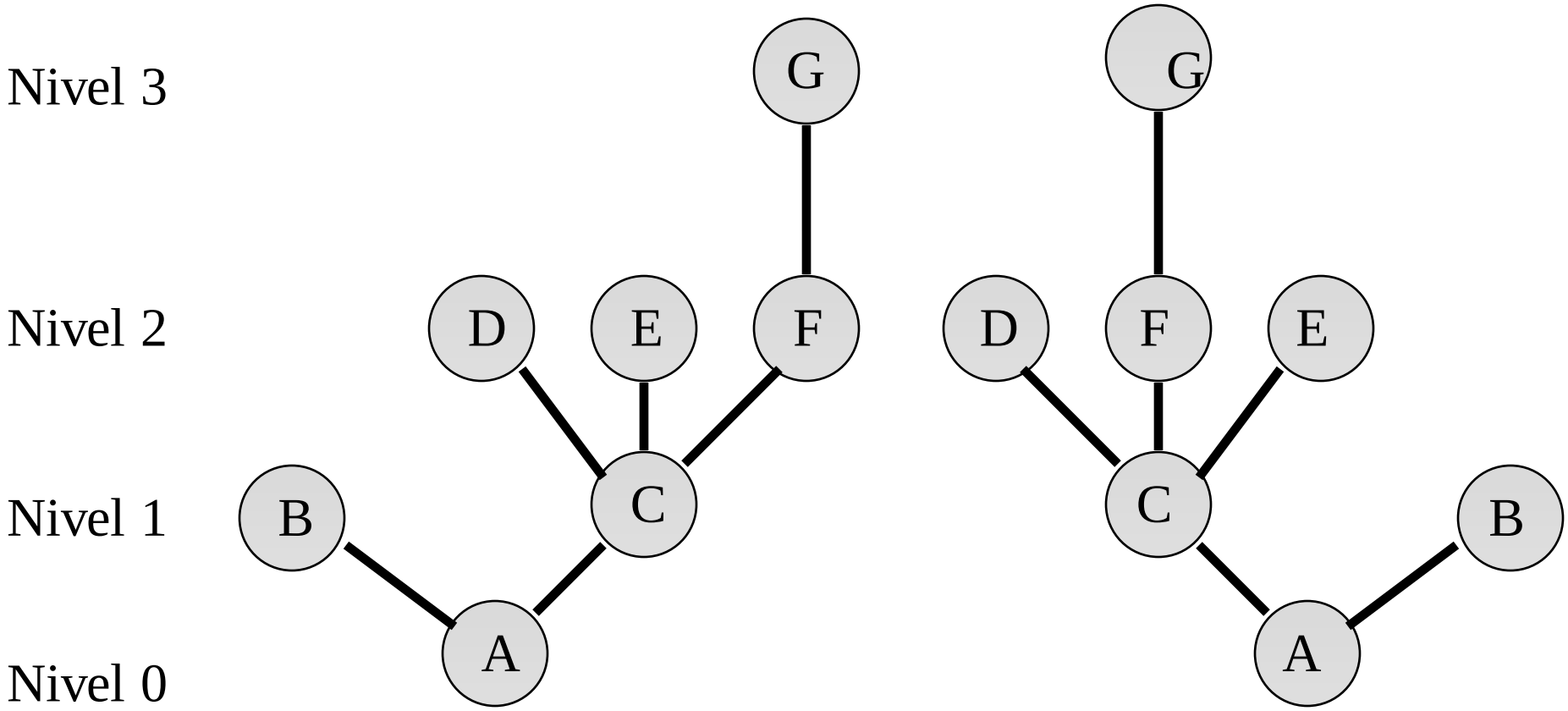
Arboles Binarios

- Un Arbol Binario es un Arbol con las siguientes propiedades:
  - Cada nodo interno tiene como máximo dos hijos (exactamente **dos** para árboles binarios completos)
  - Los hijos de un nodo son un par ordenado
- Los hijos de un nodo interno se denominan Hijo Izquierdo e Hijo Derecho
- Definiciones alternativas:
  - Un árbol con un sólo nodo, o un árbol cuya raíz tiene un par ordenado de hijos, cada uno de los cuales es a su vez un árbol binario
  - Conjunto finito de nodos, que puede estar vacío o consistir de una raíz y dos árboles binarios disjuntos, llamados subárbol izquierdo y derecho de la raíz.

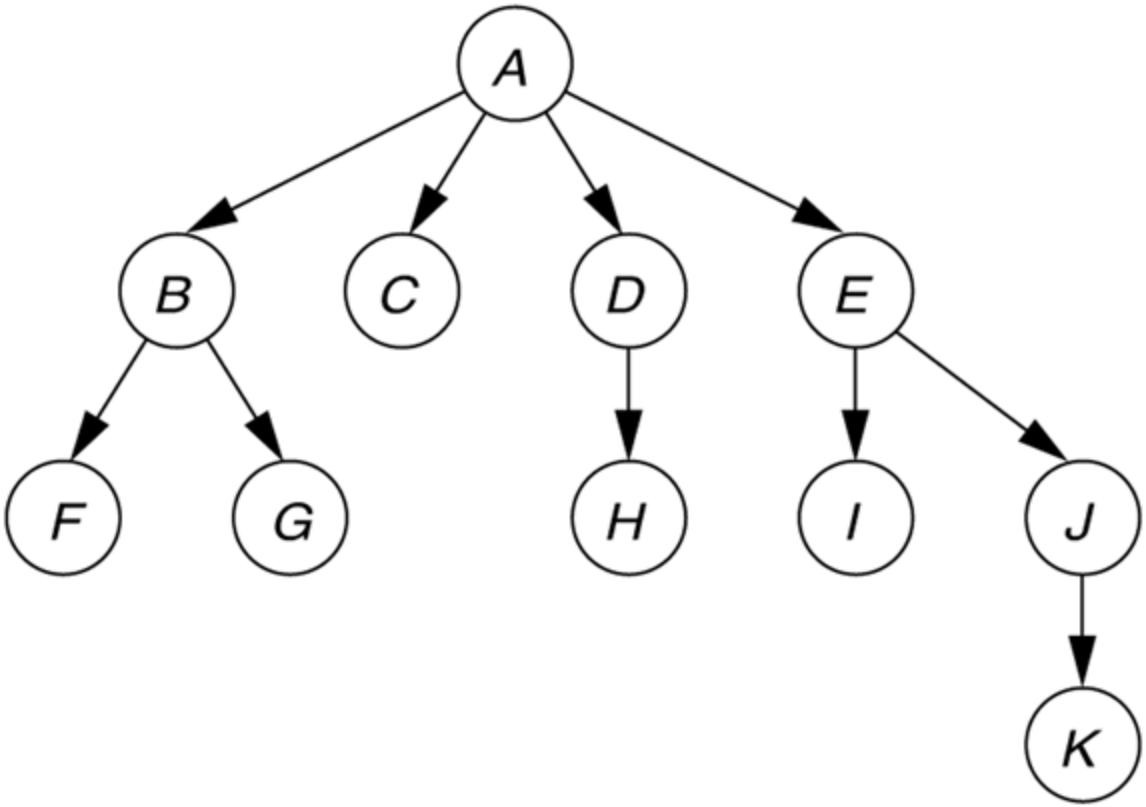
- **Aplicaciones:**
  - Expresiones aritméticas
  - Procesos de decisión
  - Búsqueda



ARBOLES



Altura y profundidad o nivel



Nodo	Altura	Nivel
A	3	0
B	1	1
C	0	1
D	1	1
E	2	1
F	0	2
G	0	2
H	0	2
I	0	2
J	1	2
K	0	3

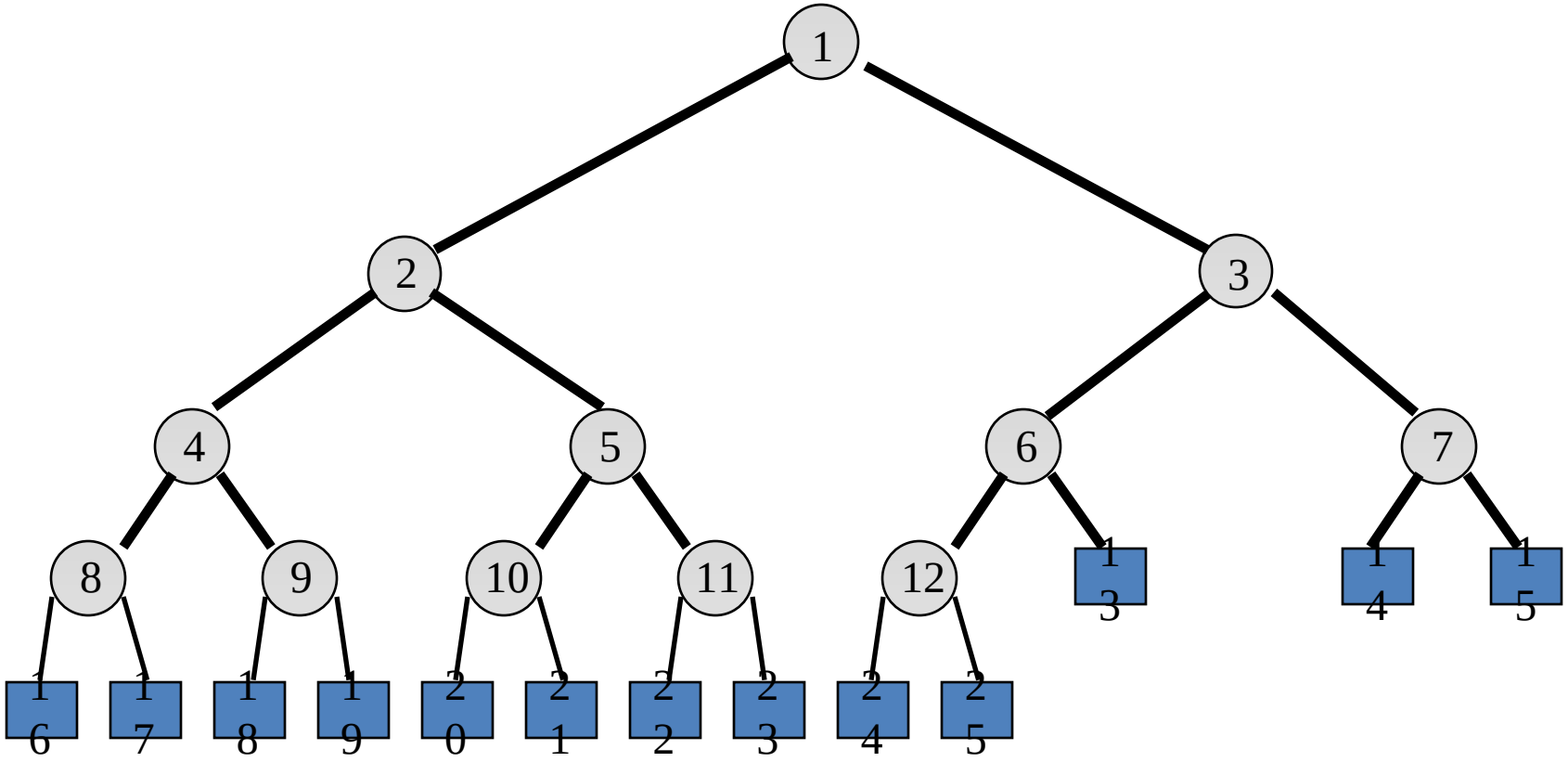
ARBOLES: Representación

- La raíz arriba, las hojas abajo.
- Se dice que cada raíz es el “padre” de las raíces de sus subárboles, los cuales son llamados “hermanos”.
- Los subárboles son llamados “hijos” de su “padre”.
- La raíz del árbol total no tiene padre.
- Ancestros y descendientes.

Lleno y completo

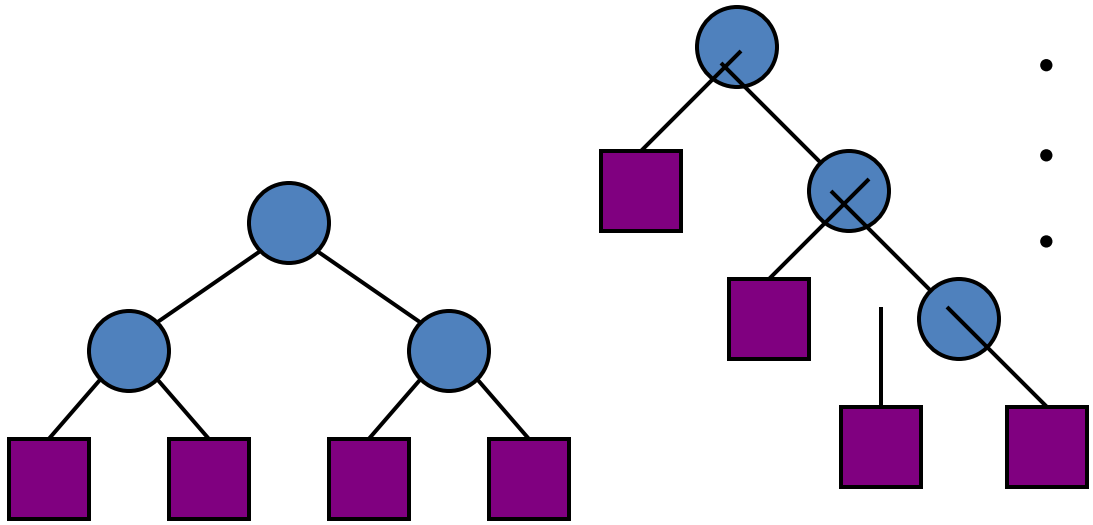
- Arbol binario Lleno
  - Todos los nodos son hoja o tienen los dos hijos
- Arbol binario completo
  - Todos los niveles, excepto eventualmente el ultimo, tienen todos los nodos
  - El ultimo siempre estará lleno de izquierda a derecha

ARBOL BINARIO COMPLETO



Propiedades de los Arboles Binarios llenos

- **Notación**
  - n número de nodos
  - e número de nodos externos
  - i número de nodos internos
  - h altura



- **Propiedades:**
  - $e = i + 1$
  - $n = 2e - 1$
  - $h \leq i$
  - $h \leq (n - 1)/2$
  - $e \leq 2^h$
  - $h \geq \log_2 e$
  - $h \geq \log_2 (n + 1) - 1$

Arboles y recursividad

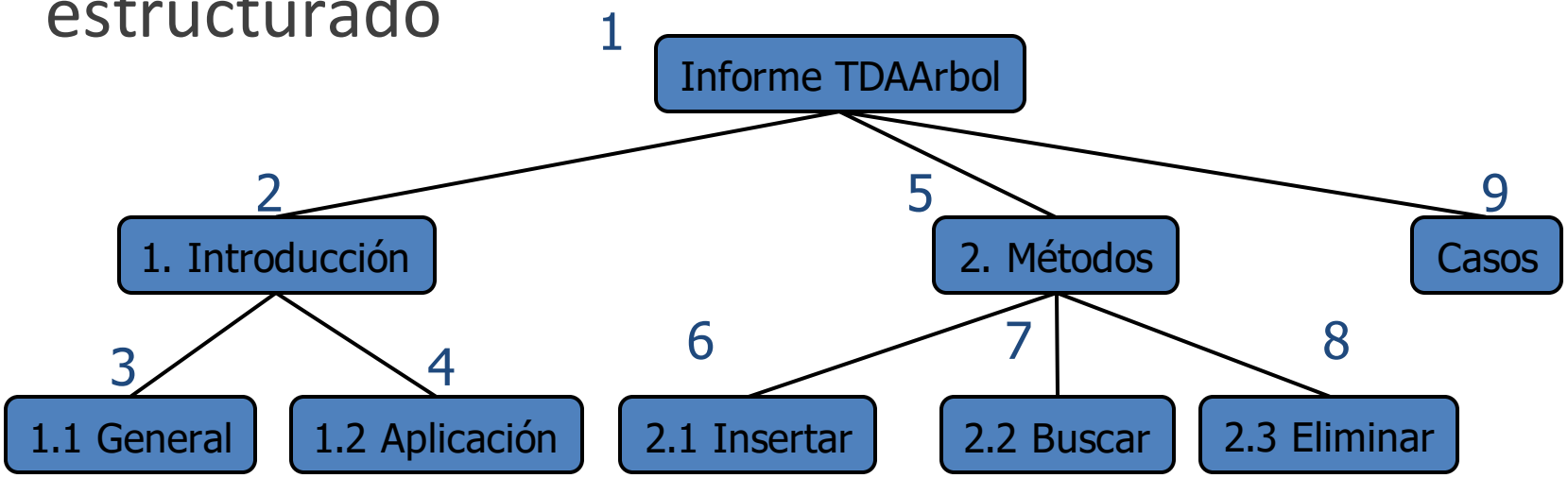
Operaciones en arboles binarios

- Operaciones básicas.
  - Inserción, búsqueda, eliminación
- Recorridos
  - Preorden
  - Postorden
  - inorden

Recorrida en Preorden

- En un recorrido en preorden, un nodo es visitado antes que sus descendientes
- Aplicación: imprimir un documento estructurado

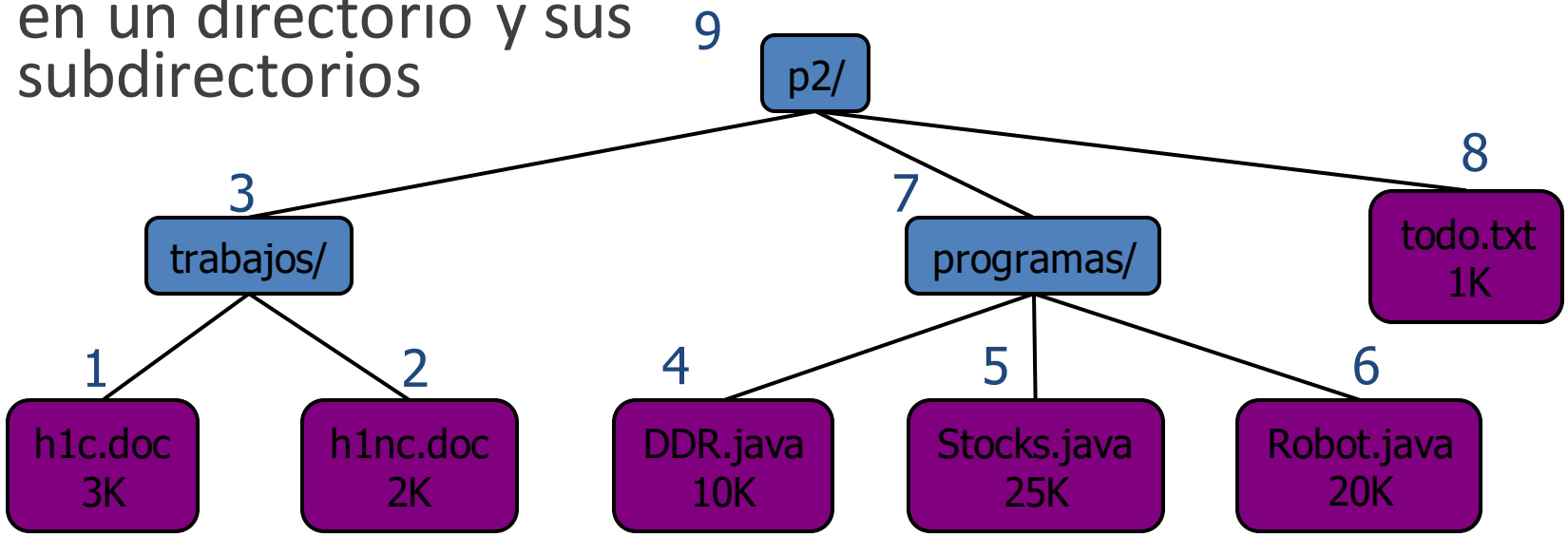
Algoritmo ***preOrden*****(*****v*****)**
  visitar**(*****v*****)**
  Para cada hijo ***w*** de ***v***
    preorden **(*****w*****)**



Recorrida en Postorden

- En un recorrido en postorden, un nodo es visitado después de sus descendientes
- Aplicación: calcular el espacio usado por archivos en un directorio y sus subdirectorios

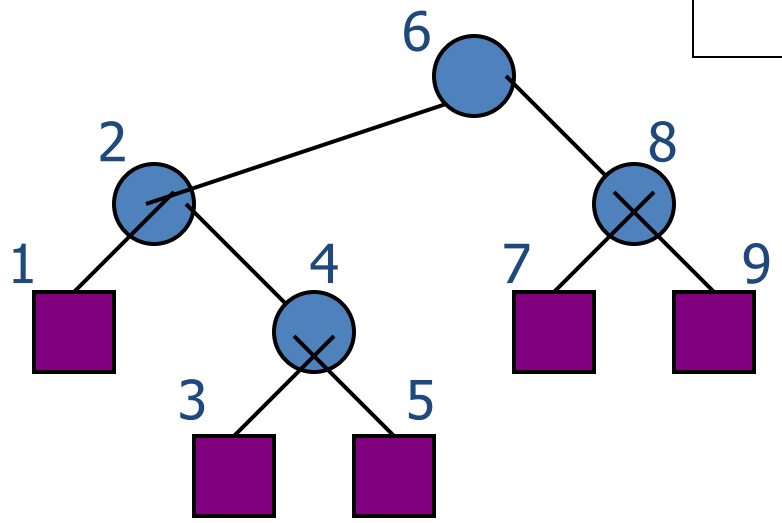
Algoritmo *postOrden*(*v*)
  Para cada hijo *w* de *v*
    postOrden (*w*)
  visitar(*v*)



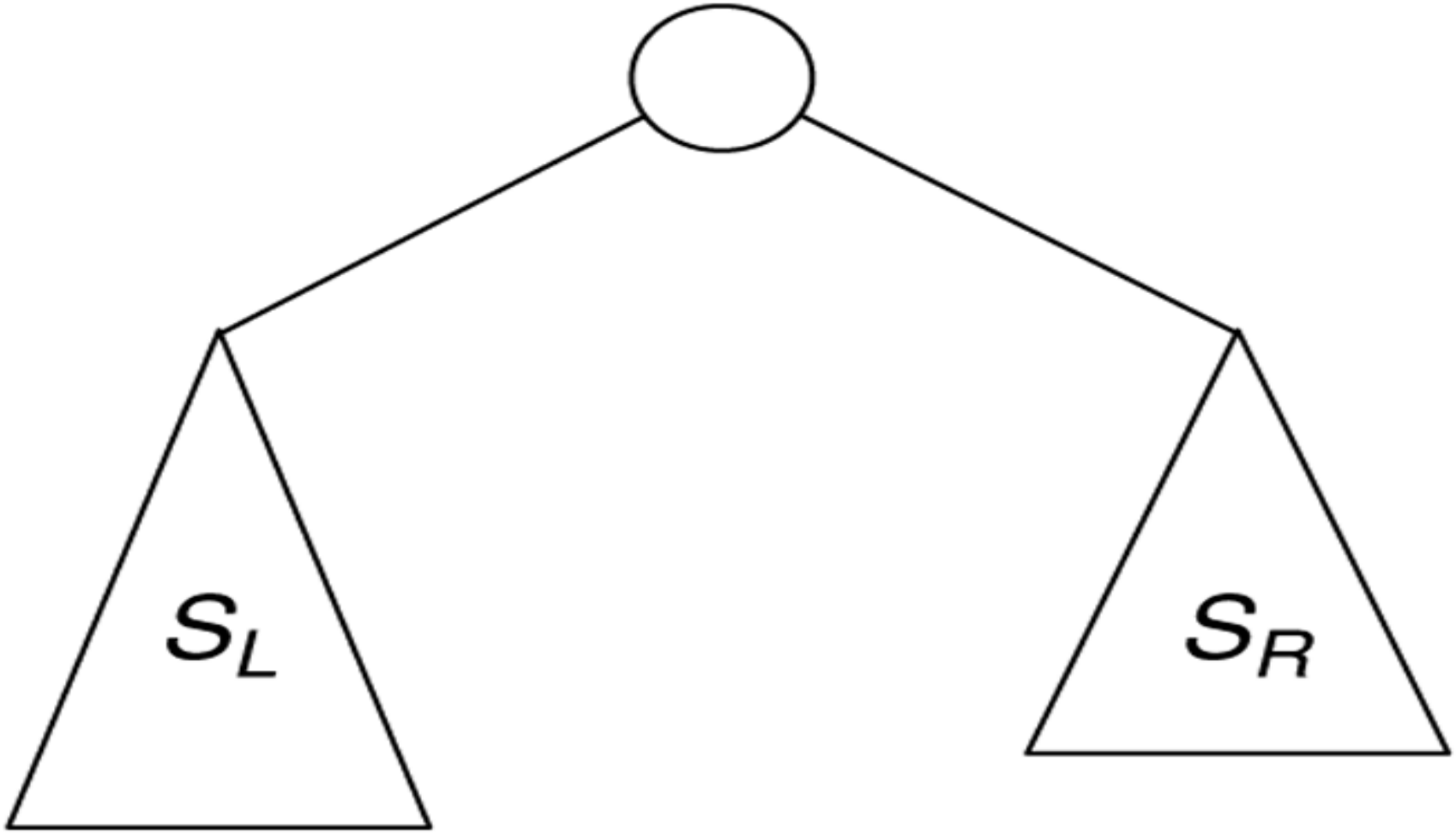
Recorrido en Inorden de un árbol binario

- En un recorrido en inorden el nodo es visitado después que su subárbol izquierdo y antes que su subárbol derecho

Algoritmo *TNodoAB.InOrden*
Si *tiene HijoIzquierdo*
HijoIzquierdo.inOrden
visitar(v)
Si *tiene HijoDerecho*
HijoDerecho. inOrden

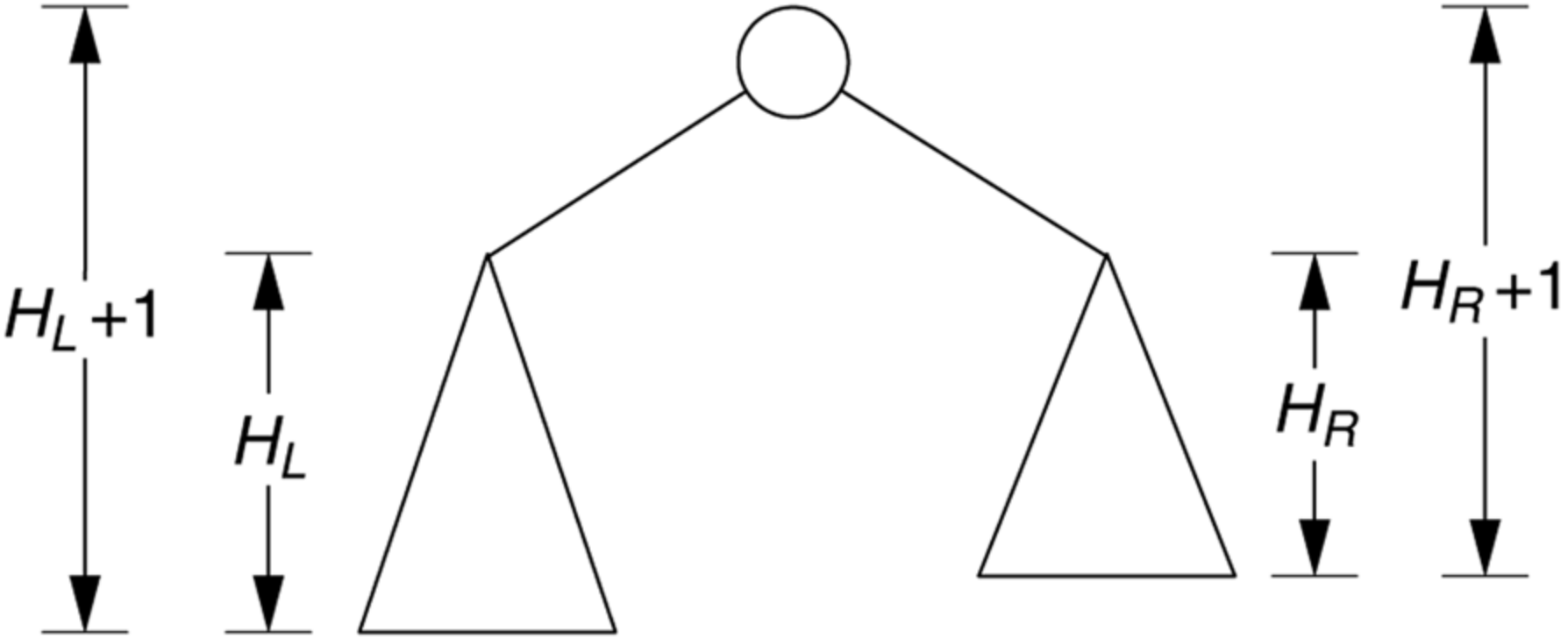


Vista recursiva usada para calcular el tamaño de un árbol $ST = SL + SR + 1$



Vista recursiva usada para calcular la altura de un nodo:

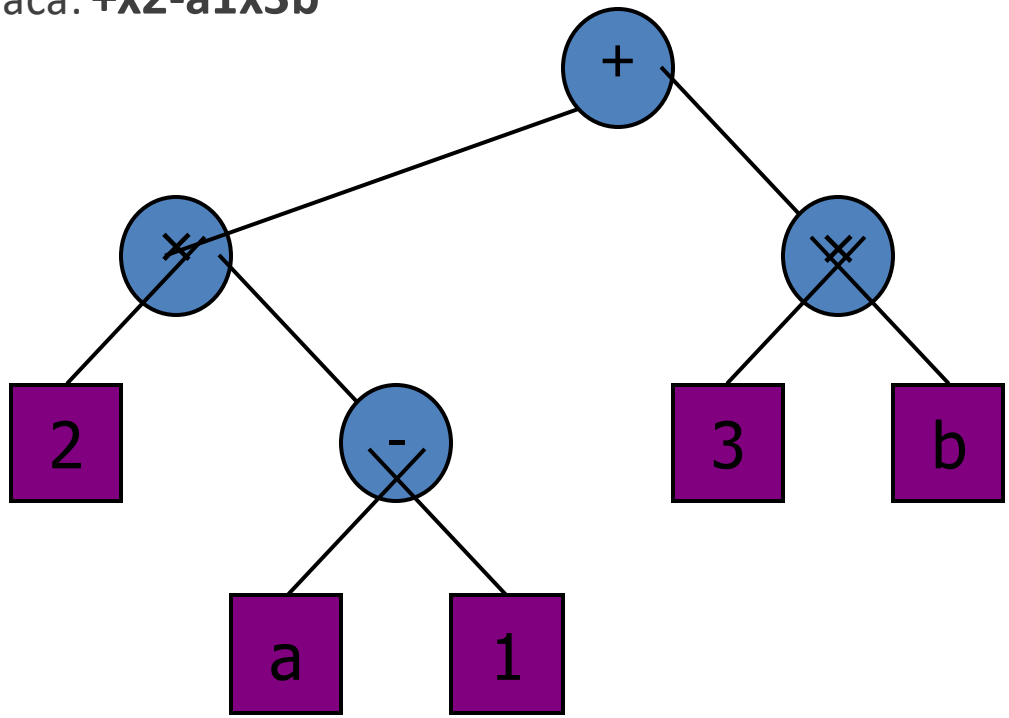
$$HT = \text{Max}\,(HL + 1, HR + 1)$$



Arbol de Expresión Aritmética

- Arbol binario asociado con una expresión aritmética:
  - Nodos internos: **operadores**
  - Nodos externos: **operandos**

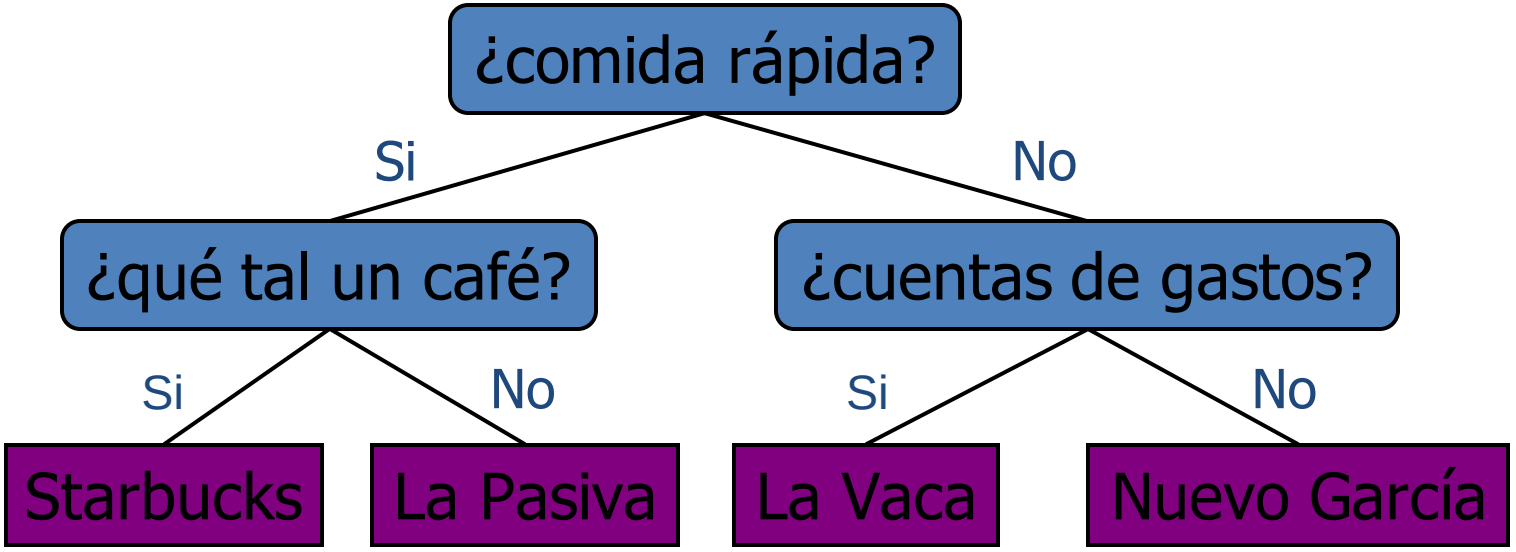
- Ejemplo: árbol de expresión aritmética para la expresión (2 × (a - 1) + (3 × b))
- Notación polaca: **+x2-a1x3b**



Aplicación de Arbol Binario: Arbol de Decisión

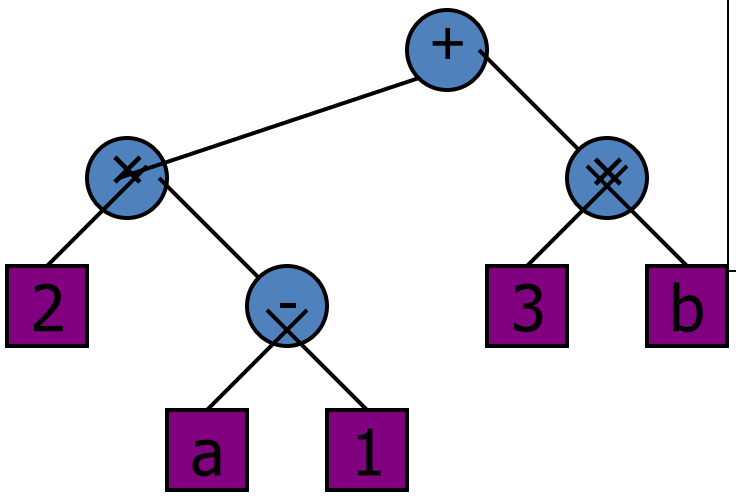
- Arbol binario asociado con un proceso de decisión
  - Nodos internos : preguntas con respuestas si / no
  - Nodos externos : decisiones

- Ejemplo: decisión de cena



Impresión de expresiones aritméticas

- Especialización del recorrido en inorden
  - Imprimir el operando u operador al visitar el nodo
  - Imprimir "(" antes de recorrer el subárbol izquierdo
  - Imprimir ")" después de recorrer el subárbol derecho



Algoritmo
TNodoAB.printExpression

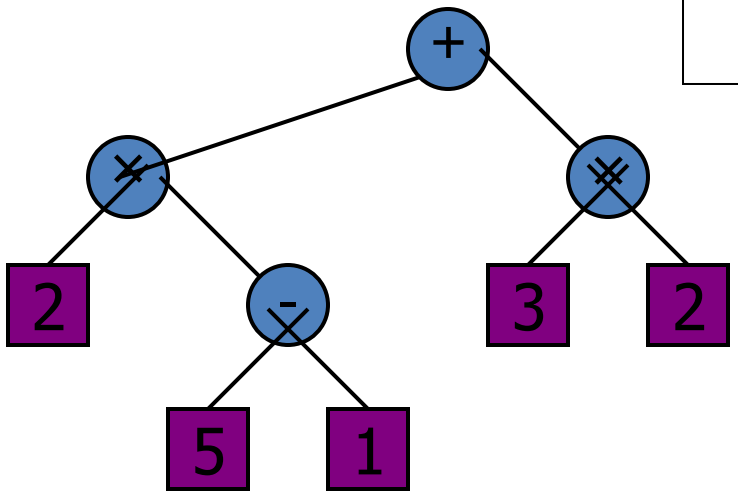
Si *tieneHijoIzquierdo*
imprimir("(")
HijoIzquierdo.printExpression
imprimir
Si *tieneHijoDerecho*
HijoDerecho .printExpression
imprimir(")")

((2 × (a - 1)) + (3 × b))

Evaluar expresiones aritméticas

- Especialización del recorrido en postorden
  - Método recursivo que retorna el valor de un subárbol
  - Al visitar un nodo interno, combina los valores de los subarboles

Algoritmo *TNodoAB.evalExpr*
  Si *esHoja*
    Devolver *elemento*
  sino
    x ← *HijoIzquierdo.evalExpr*
    y ← *HijoDerecho.evalExpr*
    ◊ ← operador contenido
    devolver *x* ◊ *y*



Ejercicios Arboles Binarios

- Define árbol binario. Dá un ejemplo de organización usando esta estructura.
- Representa la siguiente expresión aritmética utilizando un árbol binario:

  a - (b * (c + d / (f + g)) + h * (i - j * (k + l)))

  y dá un algoritmo para, utilizando este árbol, evaluar la expresión cuando las variables toman valores.

- El recorrido en preorden de un cierto árbol binario produce

  ADFGHKLPQRWZ

  y el recorrido en inorden produce

  GFHKDLAWRQPZ

  Dibuje el árbol binario correspondiente.

Ejercicios Arboles Binarios

Sean p(x), s(x) e i(x) las posiciones del nodo x en preorden, postorden e inorden respectivamente.

- **Marque en el cuadro siguiente las posiciones que pueden ser ciertas simultáneamente.**

	i(n) < i(m)	*s(n) < s(m)*	*p(n) < p(m)*
n* es ancestro de *m			
n* es descendiente de *m			
n* está a la izquierda de *m			
n* está a la derecha de *m			

Ejercicios Arboles Binarios

- Escriba un algoritmo para contar las hojas de un árbol binario
- Dado un árbol binario de elementos de tipo entero, escriba un algoritmo que calcule la suma de todos los elementos.
- Escribir un algoritmo que determine la cantidad de nodos que se encuentran en un árbol binario en un nivel n

ARBOL BINARIO DE BUSQUEDA

- Se dice que un árbol binario es de búsqueda si está organizado de forma que:
  - para cada nodo t_i todas las claves del subárbol izquierdo de t_i son menores que la clave de t_i y todas las claves del árbol derecho son mayores.
- Si el árbol tiene n nodos y está balanceado, su altura será de log(n)
- Una búsqueda en este caso puede tomar log(n) comparaciones o menos.

Árbol binario de búsqueda

TDA ArbolBinario

Raiz : ElementoArbolBinario

Primitivas

Buscar (UnaEtiquetaqueta) : ElementoArbolBinario...

....

ArbolBinario.Buscar(UnaEtiquetaqueta)

COM

```
resultado = nulo
si Raiz <> nulo ENTONCES
    resultado = Raiz.Buscar(UnaEtiquetaqueta)
Devolver resultado
```

FIN

Arbol binario de búsqueda

TDA ElementoArbolBinario

Etiqueta : TipoEtiqueta

HijoIzquierdo,HijoDerecho : ElementoArbolBinario

Operaciones

Insertar(unElementoArbolBinario)

Buscar(UnaEtiquetaqueta)

Eliminar(UnaEtiquetaqueta)

...

TArbolBB

```
public class TArbolBB<T> {

TElementoAB<T> raiz;
public TArbolBB () {…} // a Implementar

public boolean esVacio() {…} // a Implementar

public boolean insertar(TElementoAB unElemento) {…}

public TElementoAB buscar (Comparable UnaEtiquetaqueta) {…}

public void preOrden() {…} // a Implementar

public void inOrden() {…}

public void postOrden() {…} // a Implementar
}
```

TElementoAB

```
public class TElementoAB <T>{
   Comparable etiqueta;
   TElementoAB<T> hijoIzq;
   TElementoAB <T> hijoDer;
   <T> datos;

// a implementar
public TElementoAB <T> (Comparable UnaEtiquetaqueta, <T> unosDatos) {…}

public boolean insertar(TElementoAB<T> unElemento) {…}

public TElementoAB <T> buscar(Comparable UnaEtiquetaqueta) {…}

public void preOrden() {…}

public void inOrden() {…}

public void postOrden() {...}
}
```

Arbol binario de busqueda

De TElementoAB
Buscar(UnaEtiquetaqueta) : TElementoABinario

```
COM
 RESULTADO = nulo
  SI UnaEtiquetaqueta  = Etiqueta ENTONCES
      RESULTADO = THIS
   SINO
    SI UnaEtiquetaqueta  < Etiqueta ENTONCES
            SI HijoIzquierdo <> nulo ENTONCES
                        RESULTADO = HijoIzquierdo.Buscar(UnaEtiquetaqueta)
            FINSI
      SINO
            SI HijoDerecho <> nulo ENTONCES
                        RESULTADO = HijoDerecho.Buscar(UnaEtiquetaqueta)
      FINSI
      FINSI
  FINSI
  devolver  RESULTADO
FIN
```

TArbolBB -Buscar una Etiqueta

```
// de TArbolBB
public TElementoAB buscar (Comparable
UnaEtiquetaqueta) {
   if (esVacio()) {
        return null;
   } else {
        return raiz.buscar(UnaEtiquetaqueta);
    }
}
```

TElementoAB-Buscar Etiqueta

```
public TElementoAB buscar(Comparable UnaEtiquetaqueta) {
   if (UnaEtiquetaqueta.compareTo(etiqueta) == 0) {
        return this;
   } else {
        if (UnaEtiquetaqueta.compareTo(etiqueta) < 0) {
            if (hijoIzq != null) {
                return hijoIzq.buscar(UnaEtiquetaqueta);
            } else {
                return null;
            }
        } else {
            if (UnaEtiquetaqueta.compareTo(etiqueta) > 0) {
                if (hijoDer != null) {
                    return hijoDer.buscar(UnaEtiquetaqueta);
                } else {
                    return null;
                }
            } else {
                return null;
            }
  }
}
}
```

Inserción en árboles binarios

De ArbolBinario

Insertar(unElementoArbolBinario)

COM

```
 SI Raiz = nulo ENTONCES
        Raiz = unElementoArbolBinario
 SINO
   Raiz.Insertar(unElementoArbolBinario)
```

FIN

Inserción en árboles binarios

De TElementoAB
Insertar(UnElementoArbolBinario)
COM
 SI Etiqueta = unElementoArbolBinario.Etiqueta ENTONCES
 SALIR // ya está en el árbol
 FINSI
 SI unElementoArbolBinario.Etiqueta < Etiqueta ENTONCES
 SI HijoIzquierdo = nulo ENTONCES
 HijoIzquierdo ← unElementoArbolBinario
 SINO HijoIzquierdo.Insertar(unElementoArbolBinario)
 FINSI
 SINO
 SI HijoDerecho = nulo ENTONCES
 HijoDerecho ← unElementoArbolBinario
 SINO HijoDerecho.Insertar(unElementoArbolBinario)
 FINSI
 FINSI
FIN

Implementación de Recorridos ABB

- Preorden
- Inorden
- Postorden

TArbolBB - Inorden

```
public String inOrden(){
    if (raiz == null) return "arbol vacio";
        else return raiz.inOrden();
}
```

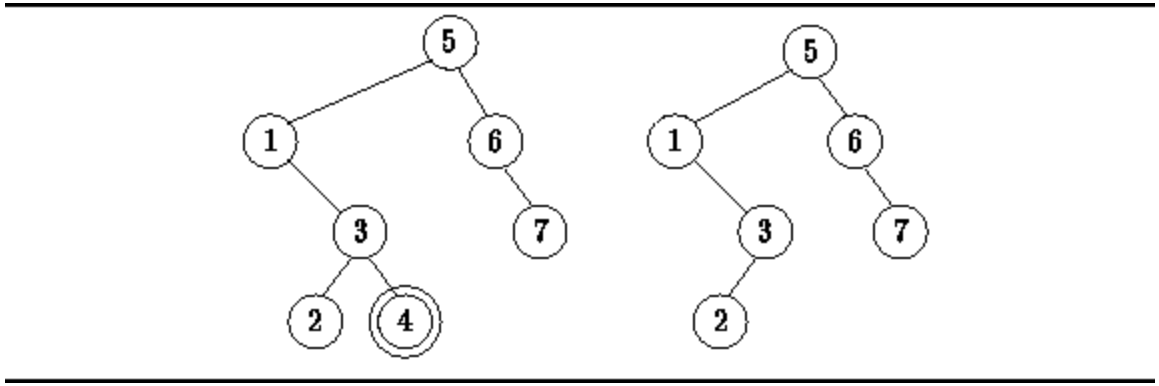
TElementoAB - Inorden

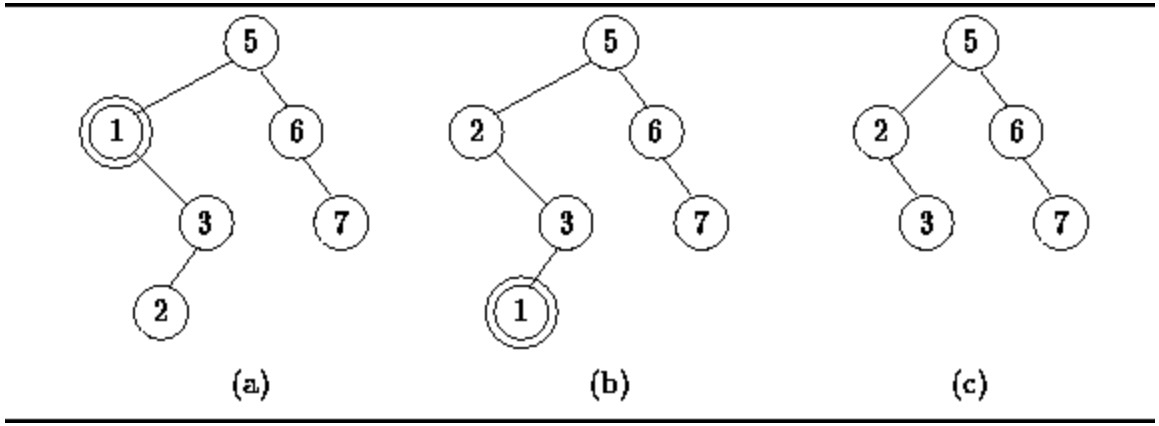
```
public String inOrden() {
    String tempStr = "";
    if (hijoIzq != null) {
        tempStr = hijoIzq.inOrden();
    }
    tempStr = tempStr + imprimir();
    if (hijoDer != null) {
        tempStr = tempStr + hijoDer.inOrden();
    }
    return tempStr;
}
```

Búsqueda y eliminación en árboles binario de búsqueda

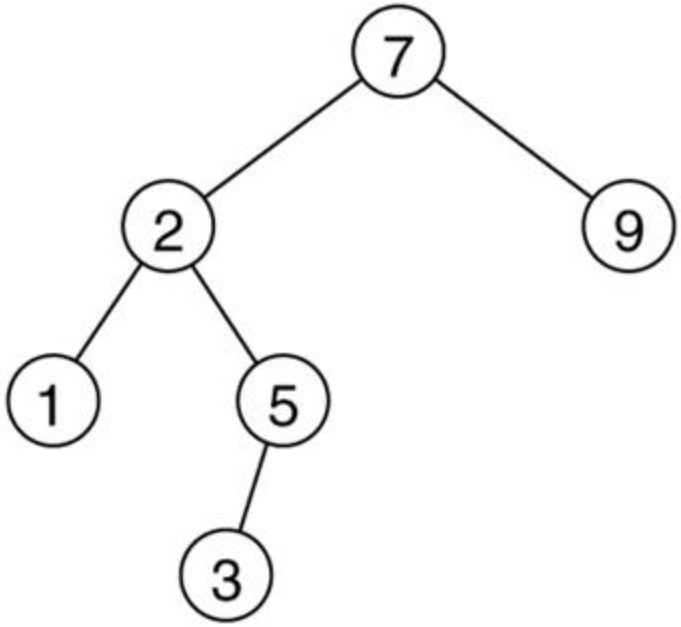
- Se busca el nodo con la etiqueta a eliminar.
- Si no existe un nodo con esa etiqueta, la eliminación es infructuosa.
- Si existe y es una hoja, se elimina el nodo directamente.
- Si no es una hoja, pero le falta alguno de sus dos sub árboles, la operación es sencilla, ya que basta con una reasignación de referencias.
- Si el nodo a eliminar es un nodo interno completo, deben sustituirse sus datos y etiqueta por el los del nodo de mayor etiqueta de su sub árbol izquierdo (su “inmediato anterior” en el orden lexicográfico dado).
- Luego se elimina ese elemento, que será una hoja, o le faltará el sub árbol derecho.

Eliminación

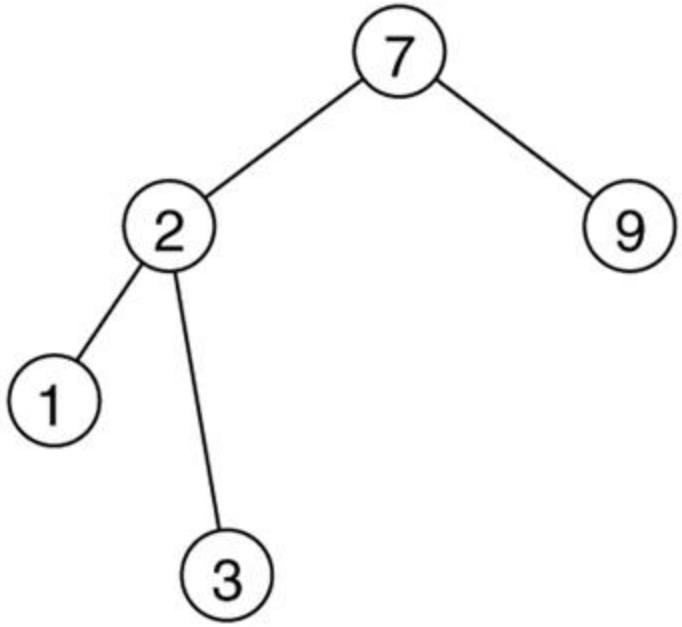




Eliminación del nodo 5 con 1 hijo: (a) antes y (b) después

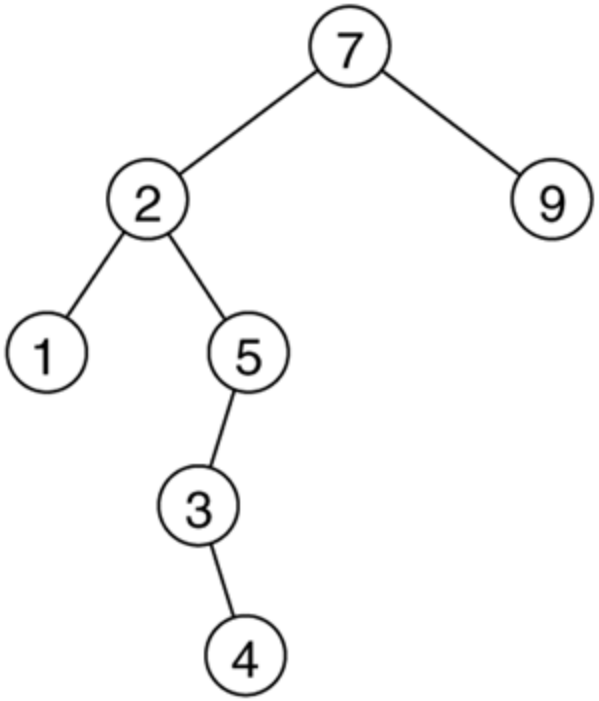


(a)

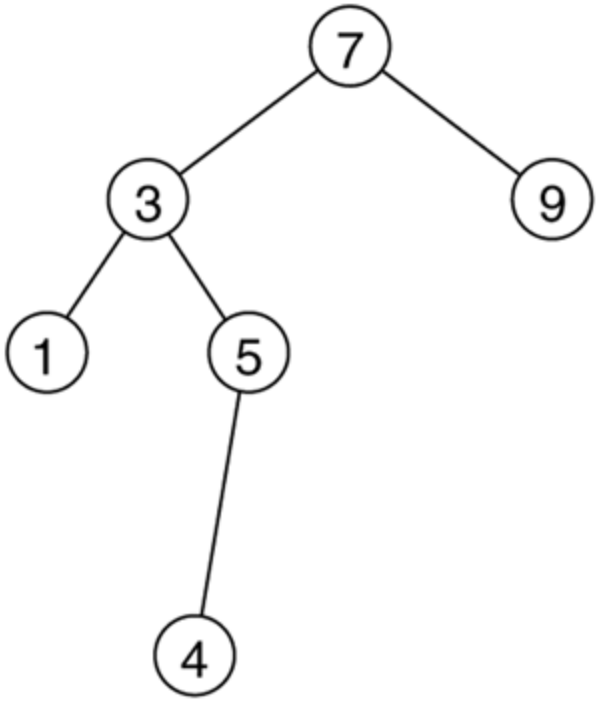


(b)

Eliminación del nodo 2 con dos hijos: (a) antes y (b) después



(a)



(b)

Búsqueda y eliminación en árboles binarios de búsqueda

ArbolBinario.Eliminar(UnaEtiqueta)

```
COM
  SI Raiz <> nulo ENTONCES
        Raiz ?  Raiz.Eliminar(UnaEtiqueta)
  SI NO
      mensaje “árbol vacío”
FIN
```

Búsqueda y Eliminación en árboles binarios de búsqueda

ElementoAB.Eliminar (UnaEtiqueta) : de Tipo TElementoAB

```
COM
(1) Si UnaEtiqueta < etiqueta entonces                    // si esta, está en el subárbol izquierdo
    Si hijoIzq <> nulo entonces
        hijoIzq ← hijoIzq.eliminar(UnaEtiqueta)           //actualiza el hijo, con el mismo u otro valor
    Finsi
    retornar (this)                                       //al padre le devuelve el mismo hijo
    Finsi

  (2) Si UnaEtiqueta > etiqueta entonces                  // si esta, está en el subárbol derecho
      Si hijoDer <> nulo entonces
        hijoDer ← hijoDer.eliminar(UnaEtiqueta)           //actualiza el hijo, con el mismo u otro valor
      Finsi
      retornar (this)                                     // al padre le devuelve el mismo hijo
    Finsi

   (3) retornar quitaElNodo()          // está, hay que eliminarlo
                                       // al padre le devuelve el nuevo hijo
Fin

// Cuando encuentra el nodo a eliminar llama, por claridad, al método que hace el trabajo
```

Búsqueda y Eliminación en árboles binarios de búsqueda

TElementoAB.quitaElNodo: de Tipo **TElementoAB**;

Comienzo

```
(1) Si hijoIzq = nulo entonces                  // le falta el hijo izquierdo o es hoja
        retornar hijoDer                        // puede retornar un nulo

(2) Si hijoDer = nulo entonces                  // le falta el hijo derecho
        retornar hijoIzq

(3) // es un nodo completo
        elHijo ← hijoIzq                        // va al subárbol izquierdo
        elPadre ← this
        mientras elHijo.hijoDer <> nulo hacer
            elPadre ← elHijo
            elHijo ← elHijo.hijoDer
        fin mientras                            // elHijo es el más a la derecha del subárbol izquierdo

       Si elPadre <> this entonces
          elPadre.hijoDer ← elHijo.hijoIzq
          elHijo.hijoIzq ← hijoIzq
        Finsi

        elHijo.hijoDer ← hijoDer
        retornar elHijo                         // elHijo quedara en lugar de this
```

Fin

Análisis de Búsqueda e Inserción en Árbol Binario de Búsqueda

- Los algoritmos de inserción y búsqueda analizados no controlan el balanceo del árbol.
- Se desconoce a priori la forma en que el árbol ha de crecer.
- Si el árbol estuviera balanceado, para encontrar una clave se precisarán aproximadamente log n comparaciones.
- El peor caso se da para n/2 comparaciones.
- El problema es encontrar la cantidad promedio de comparaciones.
- n claves, n! árboles correspondientes a n! permutaciones de claves.

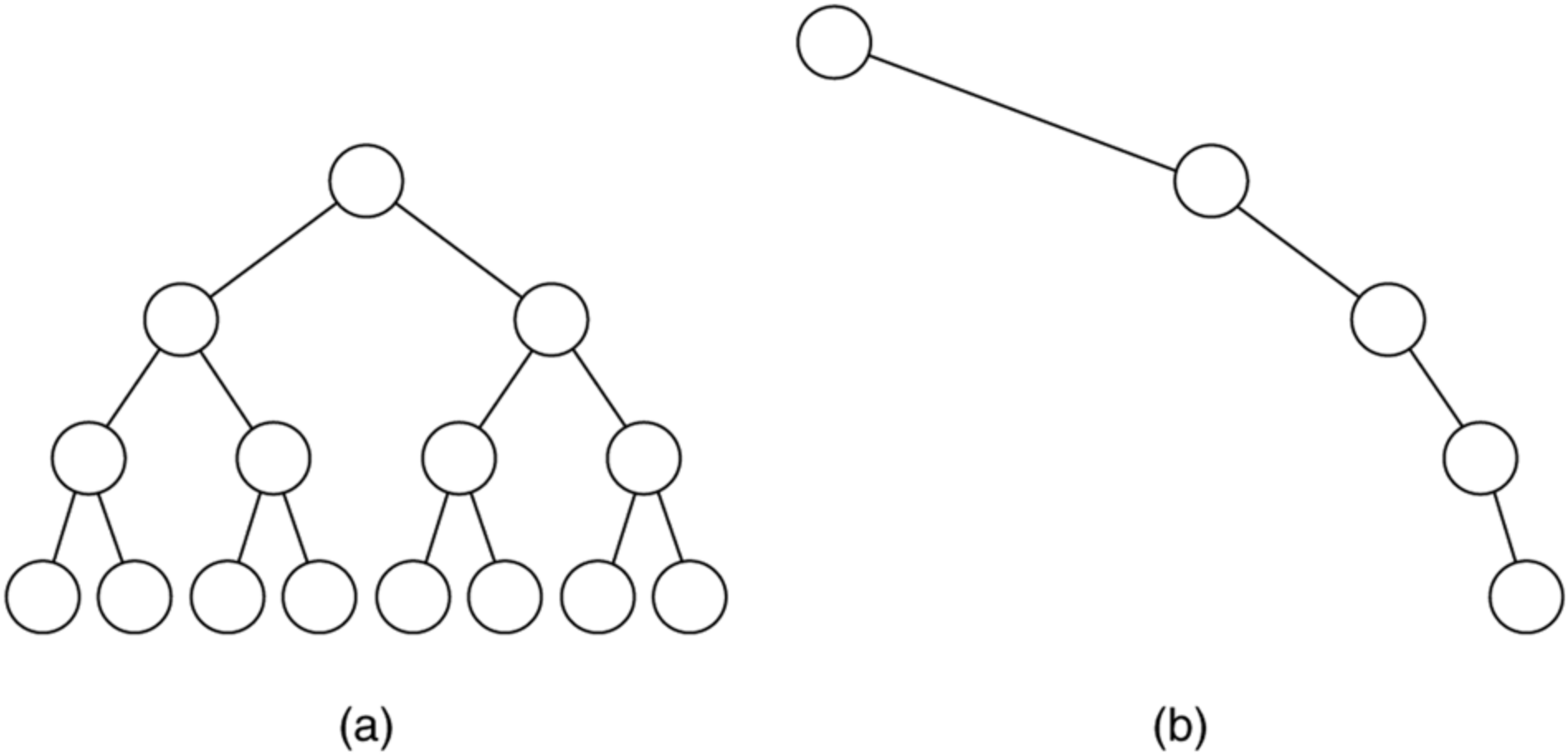
Ejercicios Árboles Binarios de Búsqueda

- ¿Qué es un árbol binario de búsqueda? Desarrolle el algoritmo de búsqueda correspondiente y evalúe su rendimiento.
- Desarrolle el algoritmo de eliminación en árbol binario de búsqueda y evalúe su rendimiento
- Muestre el resultado de insertar los elementos 3,1,4,6,9,2,5 y 7 en un árbol binario de búsqueda inicialmente vacío. Muestre después el resultado de eliminar la raíz

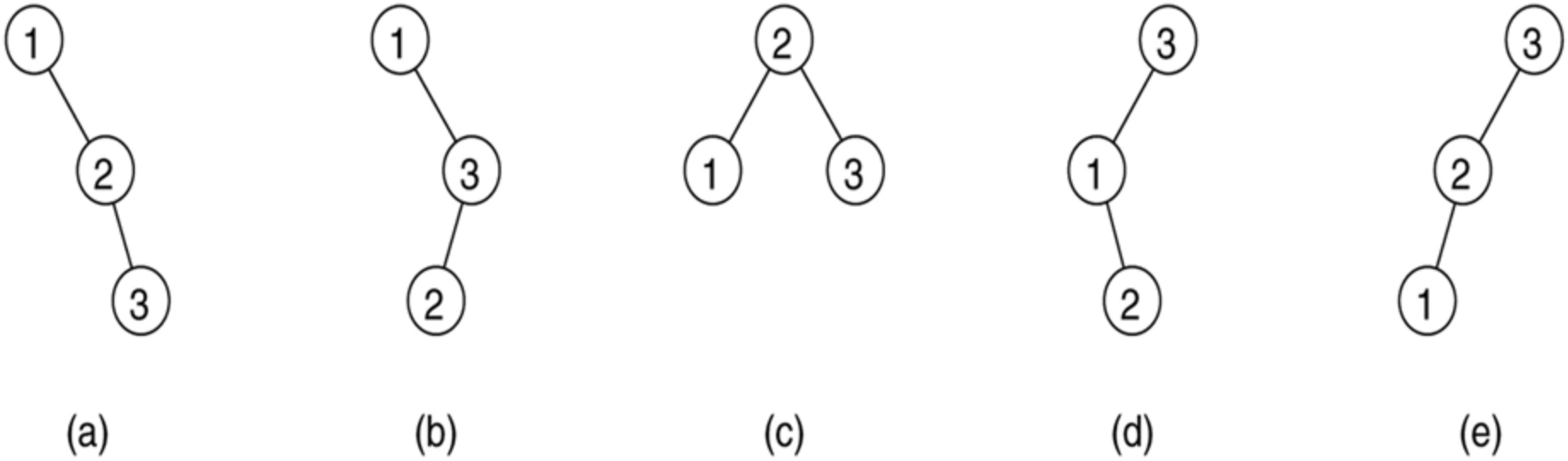
Arboles Balanceados

- Criterio de Adelson, Velskii y Landis:
  - Un árbol está balanceado si y sólo si para cada nodo las alturas de sus dos subárboles difieren a lo sumo en 1.
  - Asegura un **O log(n)** en el **peor caso** para las tres operaciones: **búsqueda**, **inserción** y **eliminación**.
  - El teorema de Adelson-Velski y Landis garantiza que un árbol balanceado nunca tendrá una altura mayor que el **45 %** respecto a su equivalente perfectamente balanceado.
  - Si la altura de un árbol balanceado de n nodos es hb(n), entonces
    - log(n+1) <= hb(n) <= 1.4404*log(n+2) -0.328

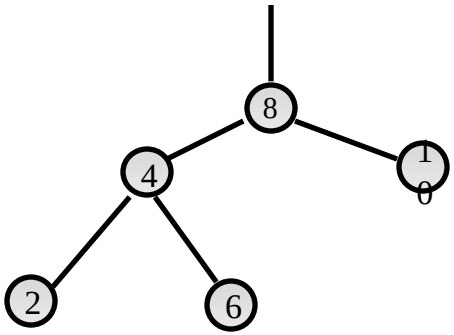
(a) Arbol balanceado altura log *N*;
(b) árbol no balanceado, altura *N* – 1.

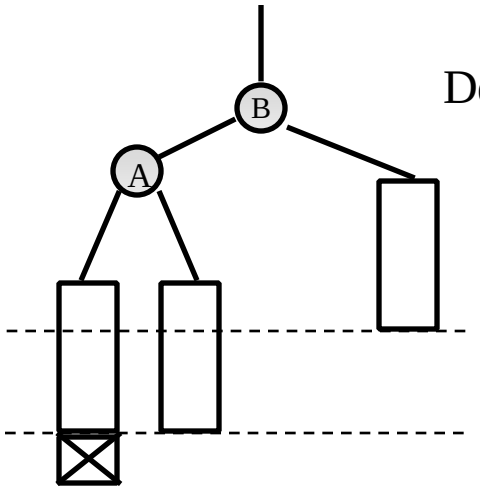


Arboles binarios de búsqueda posibles insertando las claves 1, 2 y 3 de diferentes maneras

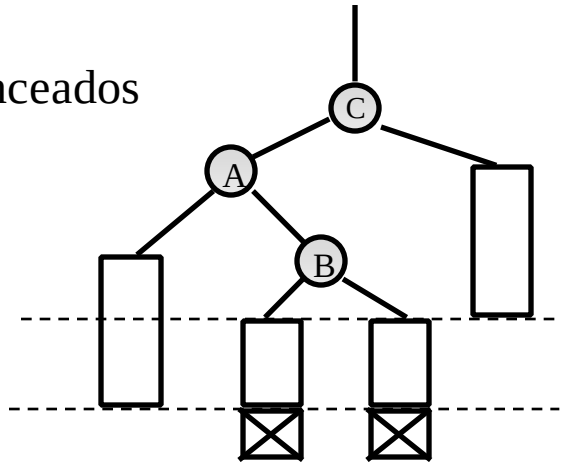


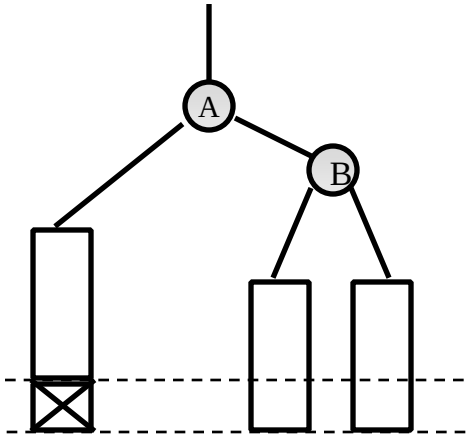
ARBOLES BALANCEADOS



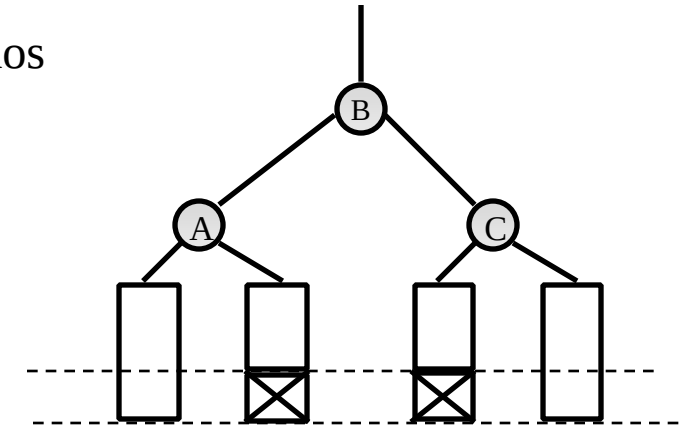


Desbalanceados





Balanceados



Arboles Balanceados

- Criterio de Adelson, Velskii y Landis:
  - Un árbol está balanceado si y sólo si para cada nodo las alturas de sus dos subárboles difieren a lo sumo en 1.
  - Asegura un O log(n) en el peor caso para las tres operaciones: búsqueda, inserción y eliminación.
  - Se demuestra con la ayuda de los árboles de Fibonacci.

AVL

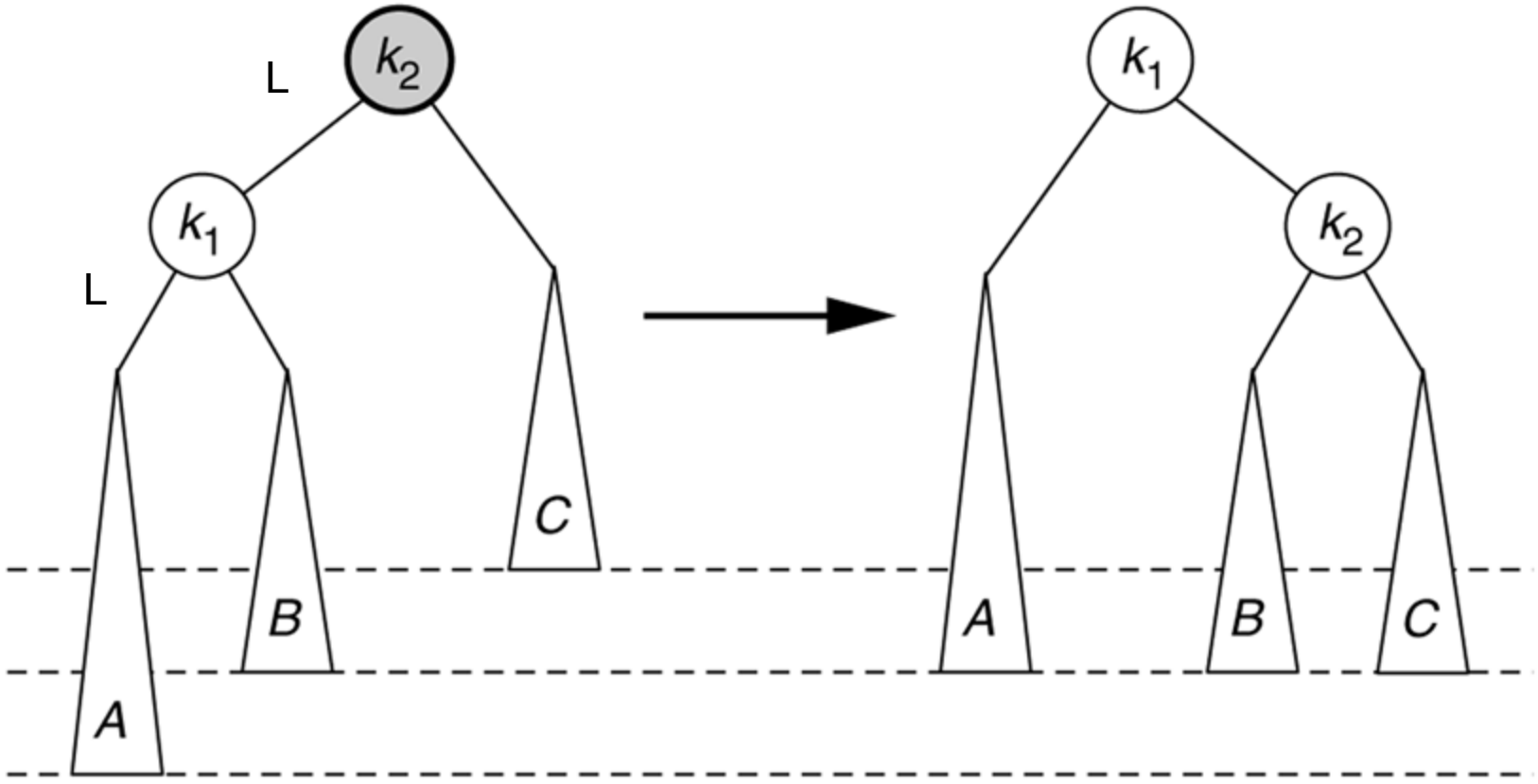
- Una inserción o eliminación puede destruir el balance
- Se debe revisar la condición de balance y corregir si es necesario
- Después de la Inserción, sólo los nodos que se encuentran en el camino desde el punto de inserción hasta la raíz pueden tener el balance alterado
- Si se repara correctamente el equilibrio del nodo desequilibrado más profundo, se recupera el equilibrio de todo el árbol

Diferentes condiciones de desequilibrio

- Supongamos que X es el nodo cuyo equilibrio queremos ajustar
- Se pueden dar cuatro casos:
  - Inserción en el subárbol izquierdo del hijo izquierdo de X
  - Inserción en el subárbol derecho del hijo izquierdo de X
  - Inserción en el subárbol izquierdo del hijo derecho de X
  - Inserción en el subárbol derecho del hijo derecho de X

- 1 y 4 son simétricos respecto a X
- 2 y 3 son simétricos respecto a X

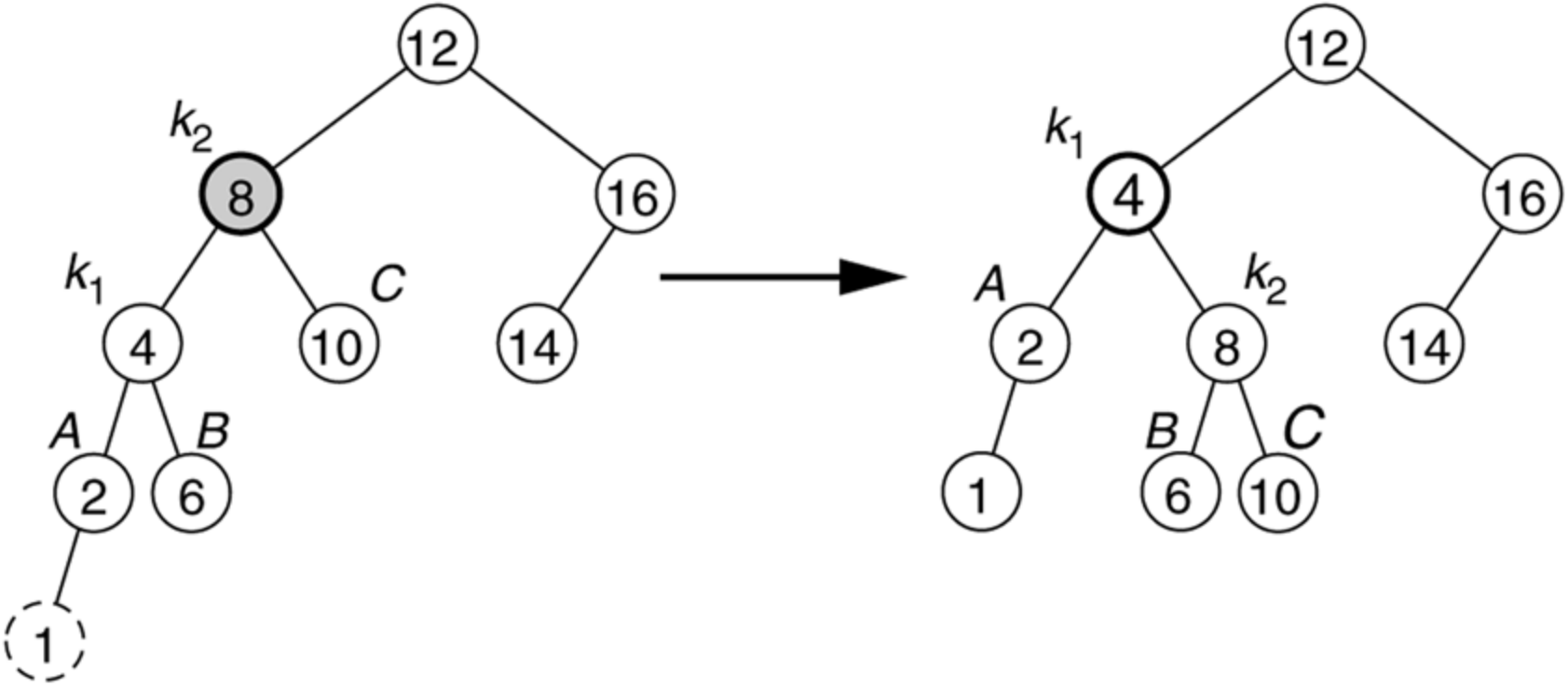
Rotación Simple para resolver el caso 1



a) Antes de la rotación

b)Después de la rotación

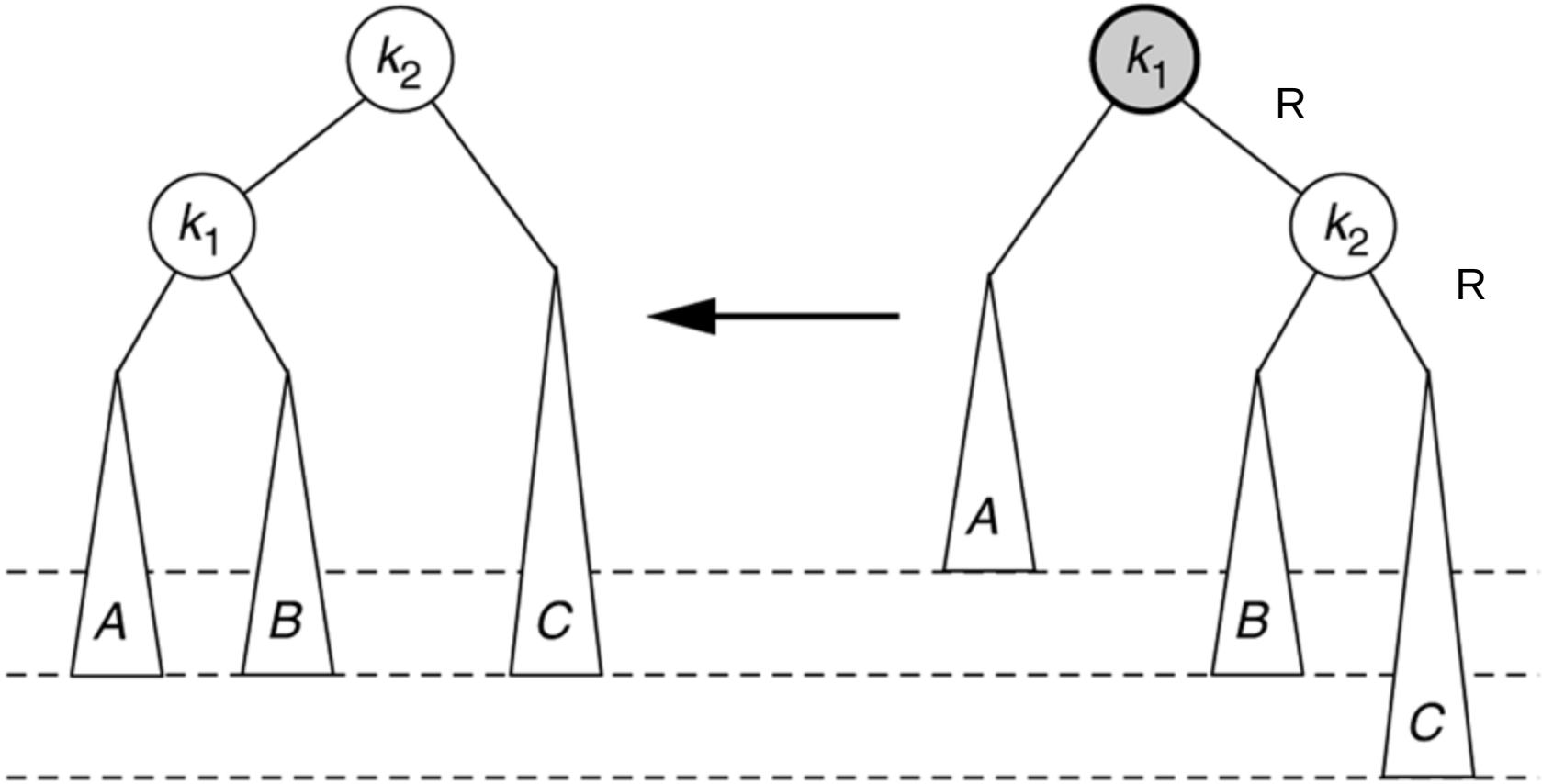
La rotación simple balancea el árbol después de la inserción del 1.



a) Antes de la rotación

b)Después de la rotación

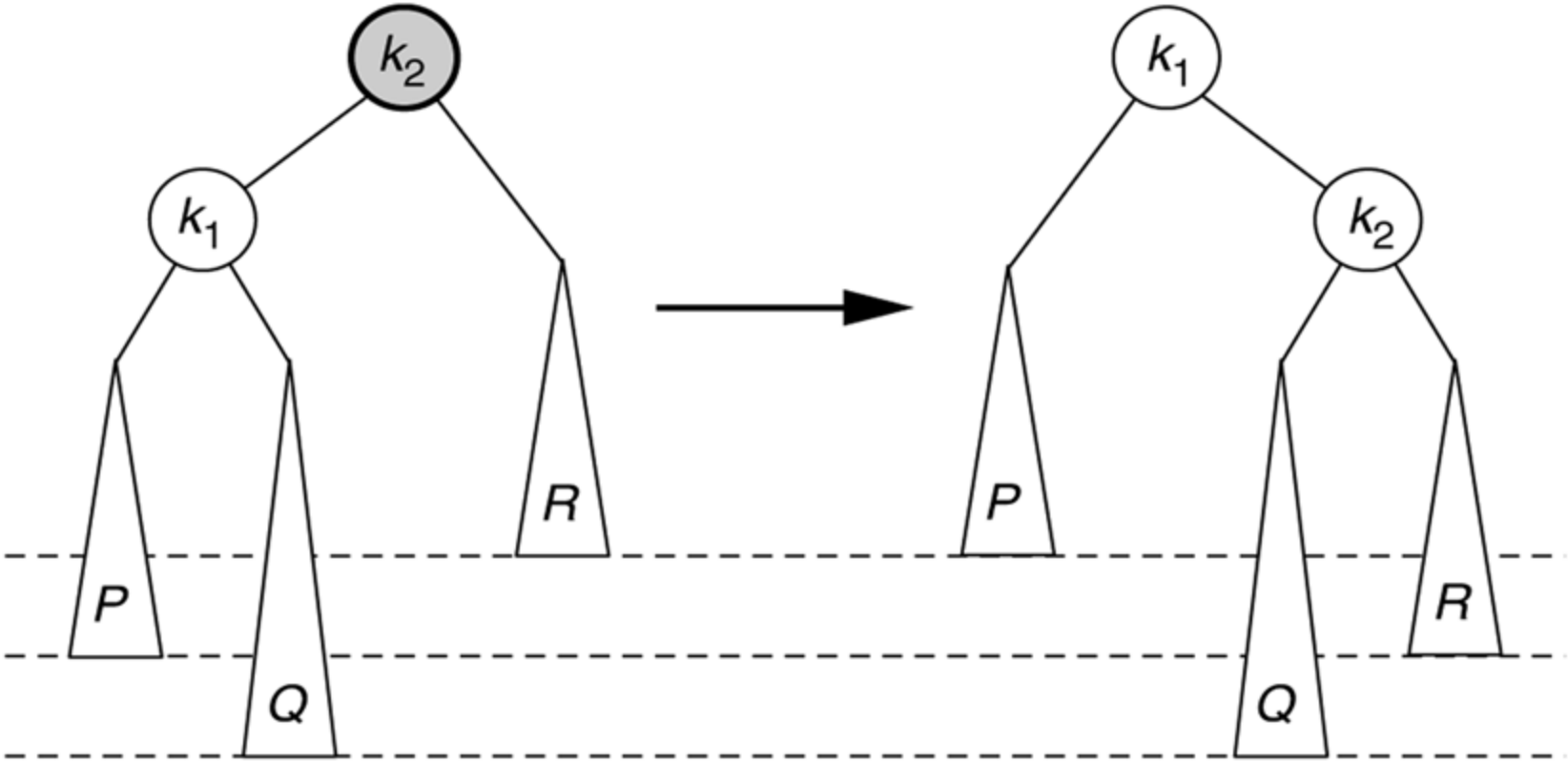
Rotación simple simétrica para resolver el caso 4



a) Antes de la rotación

b)Después de la rotación

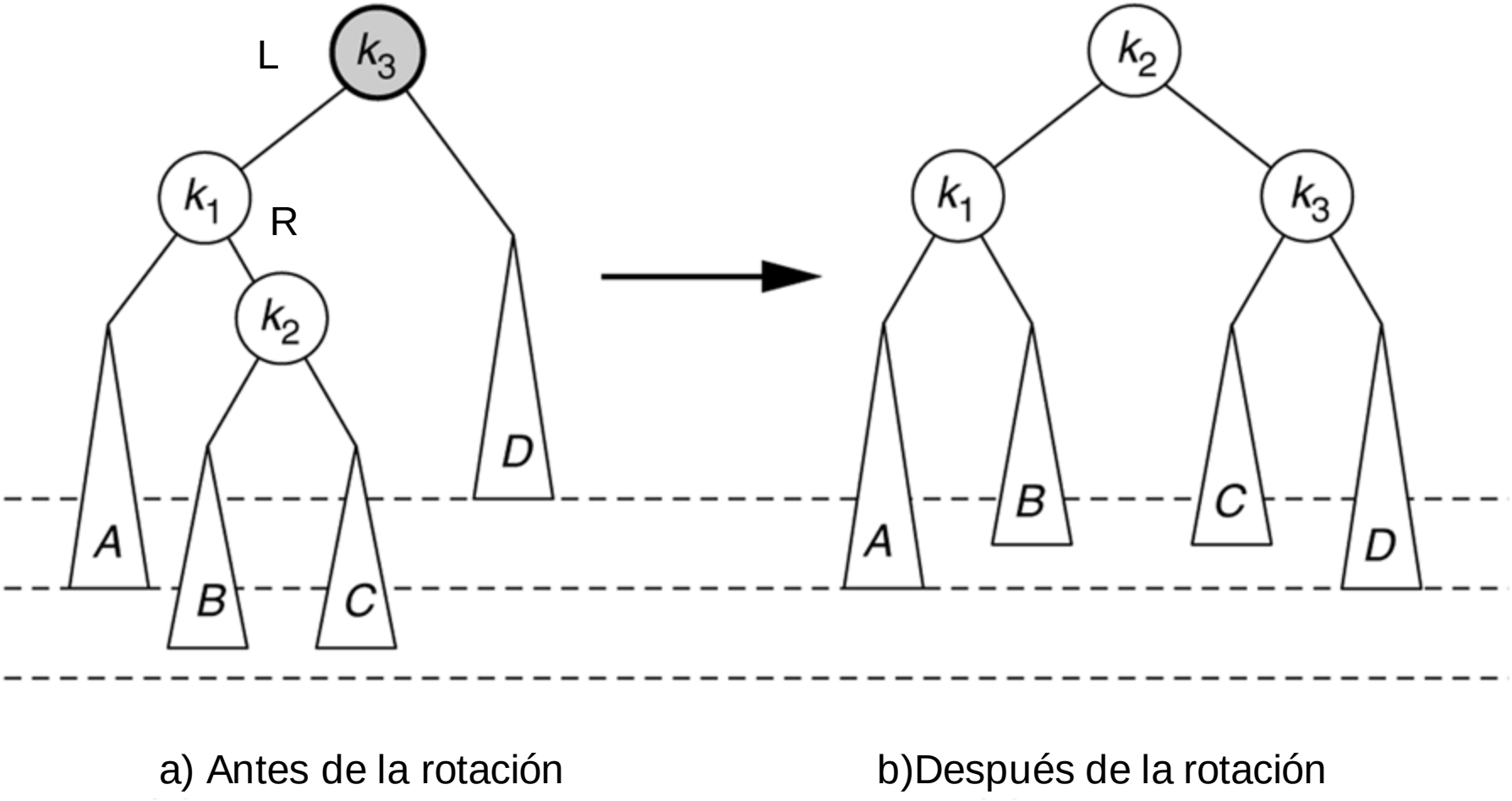
La rotación simple no resuelve el caso 2.



a) Antes de la rotación

b)Después de la rotación

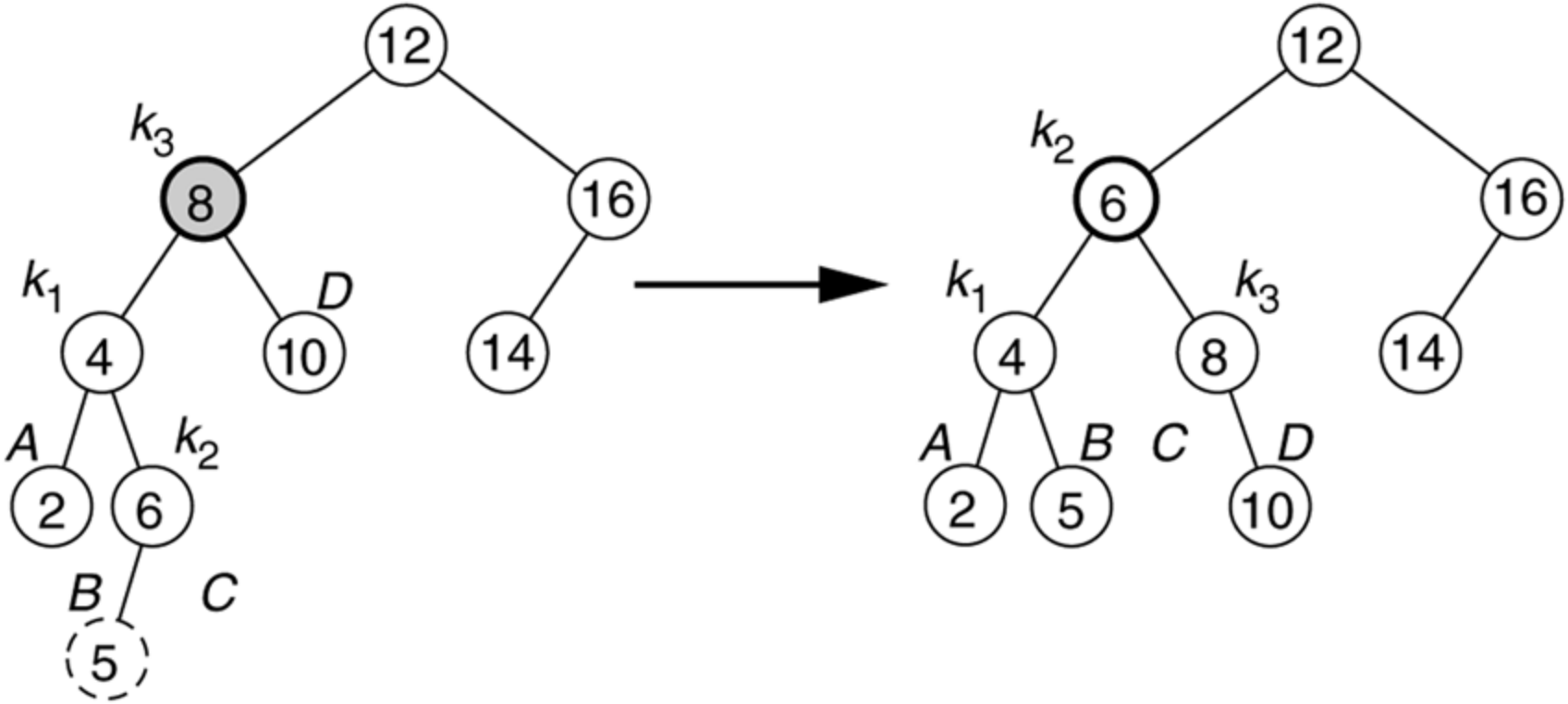
Rotación doble LR para resolver el caso 2



a) Antes de la rotación

b)Después de la rotación

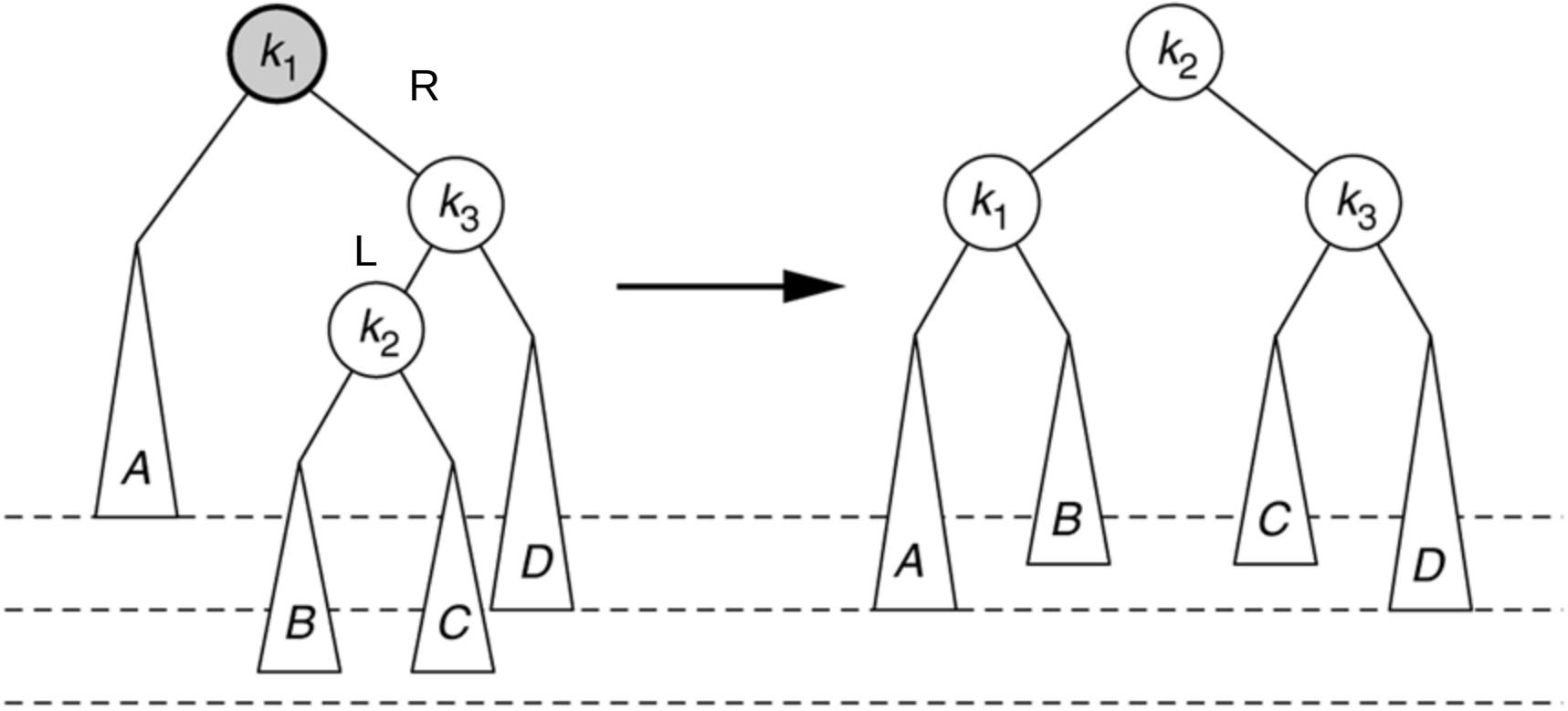
Rotación doble restablece el balance después de la inserción del 5.



a) Antes de la rotación

b)Después de la rotación

Rotación doble RL para resolver el caso 3.



a) Antes de la rotación

b)Después de la rotación

```
CASO 1 LL
TElementoAB rotacionLL ( TElementoAB k2 )
   {
 TElementoAB k1 = k2.hijoIzq;
       k2. hijoIzq = k1.hijoDer;
       k1. hijoDer = k2;
       return k1;
   }

 CASO 4 RR
TElementoAB rotacionRR(TElementoAB k1 )
   {
 TElementoAB k2 = k1.hijoDer;
       k1. hijoDer = k2.hijoIzq;
       k2.hijoIzq = k1;
       return k2;
   }
```

CASO 2 LR

```
TElementoAB rotacionLR ( TElementoAB k3 )
   {
      k3. hijoIzq = rotacionRR( k3. hijoIzq );
      return rotacionLL( k3 );
   }
```

CASO 3 RL

```
TElementoAB rotacionRL ( TElementoAB k1 )
   {
      k1. hijoDer = rotacionLL( k1. hijoDer );
      return rotacionRR( k1 );
   }
```

Ejercicios Arboles AVL

- **¿Qué es un Arbol Binario Balanceado – AVL? Describa las situaciones de desbalanceo posibles (ilustre con ejemplos) y qué operaciones se deben aplicar en cada caso para restituir la condición de AVL.**

- **Defina árbol balanceado. Describa una estructura de datos apta para implementarlo. Desarrolle el algoritmo de búsqueda e inserción en árbol balanceado. Ilustre el funcionamiento mediante un ejemplo. ¿Cuál es el orden del tiempo de ejecución del mismo?**

- **Insertar en un árbol AVL las siguientes claves, en el orden indicado,**
  8, 5, 2, 1, 20, 15, 16, 17,7
  mostrando la evolución del árbol. Luego eliminar las clave 15 y 7, mostrando los pasos.

- **Muestre el resultado de insertar en un árbol AVL las siguientes claves, en el orden indicado,**
  2,1,4,5,9,3,6, 7
  mostrando la evolución del árbol. Luego muestre el resultado de eliminar las clave 4, 5 y 9, mostrando la evolución

- **Insertar en un árbol AVL las siguientes claves, en el orden indicado**
  1,88,77,4,5,9,18,56,33,21,45,99,22
  mostrando la evolución del árbol. Luego eliminar las clave 18, 77, 1 y 45 **mostrando los pasos.**

Ejercicios Arboles AVL

- **Weiss – Cuestiones Breves**
  - 18.3 y 18.4
- **Weiss – Problemas teóricos**
  - 18.9
  - 18.10