



Universidad Nacional Mayor de San Marcos
Universidad del Perú. Decana de América

Estructura de Datos

Semana 11



Universidad Nacional Mayor de San Marcos
Universidad del Perú. Decana de América

Logro de la sesión

Al finalizar la sesión, el estudiante:

- **Identifica, analiza y resuelve problemas algorítmicos que usan el tipo abstracto de datos: árboles balanceados desarrollando funciones/métodos que usen esas estructuras.**

Estructuras de datos no lineales

Arboles Balanceados

01

Agenda

- Árboles 2-3
- Árboles B
- Código de Huffman

Árboles 2-3



Tabla de símbolos: Revisión

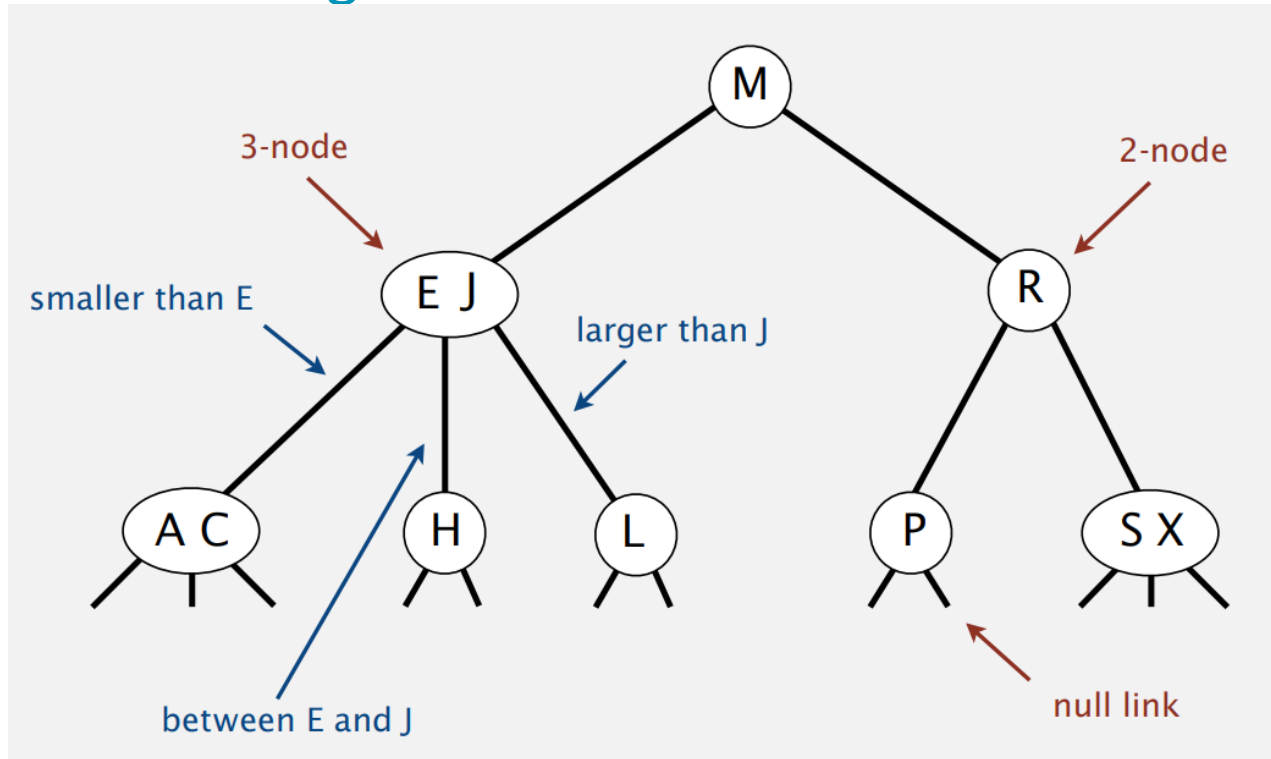
Implementación	Garantía			Caso promedio			Operaciones sobre claves	Soporta operaciones eficientes
	Búsqueda	Inserción	Delete	Acierto en búsqueda	Inserción	Delete		
Lista enlazada desordenada con búsqueda secuencial	N	N	N	N/2	N	N/2	equals()	No
Arreglo ordenado con búsqueda binaria	Log N	N	N	Log N	N/2	N/2	compareTo()	Si
BST	N	N	N	1.39 Log N	1.39 Log N	\sqrt{N}	compareTo()	Si
Meta	Log N	Log N	Log N	Log N	Log N	Log N	compareTo()	Si

Reto: Garantizar rendimiento.

Árboles 2-3

- **Árbol triario ordenado balanceado**, permite 1 o 2 claves por nodo:
 - 2-Nodo: Una clave, dos hijos
 - 3-Nodo: Dos claves, tres hijos.
- **Orden simétrico**: El recorrido Inorder produce claves en orden ascendente
- **Balance perfecto**: Cada ruta desde la raíz hasta el enlace nulo tiene la misma longitud

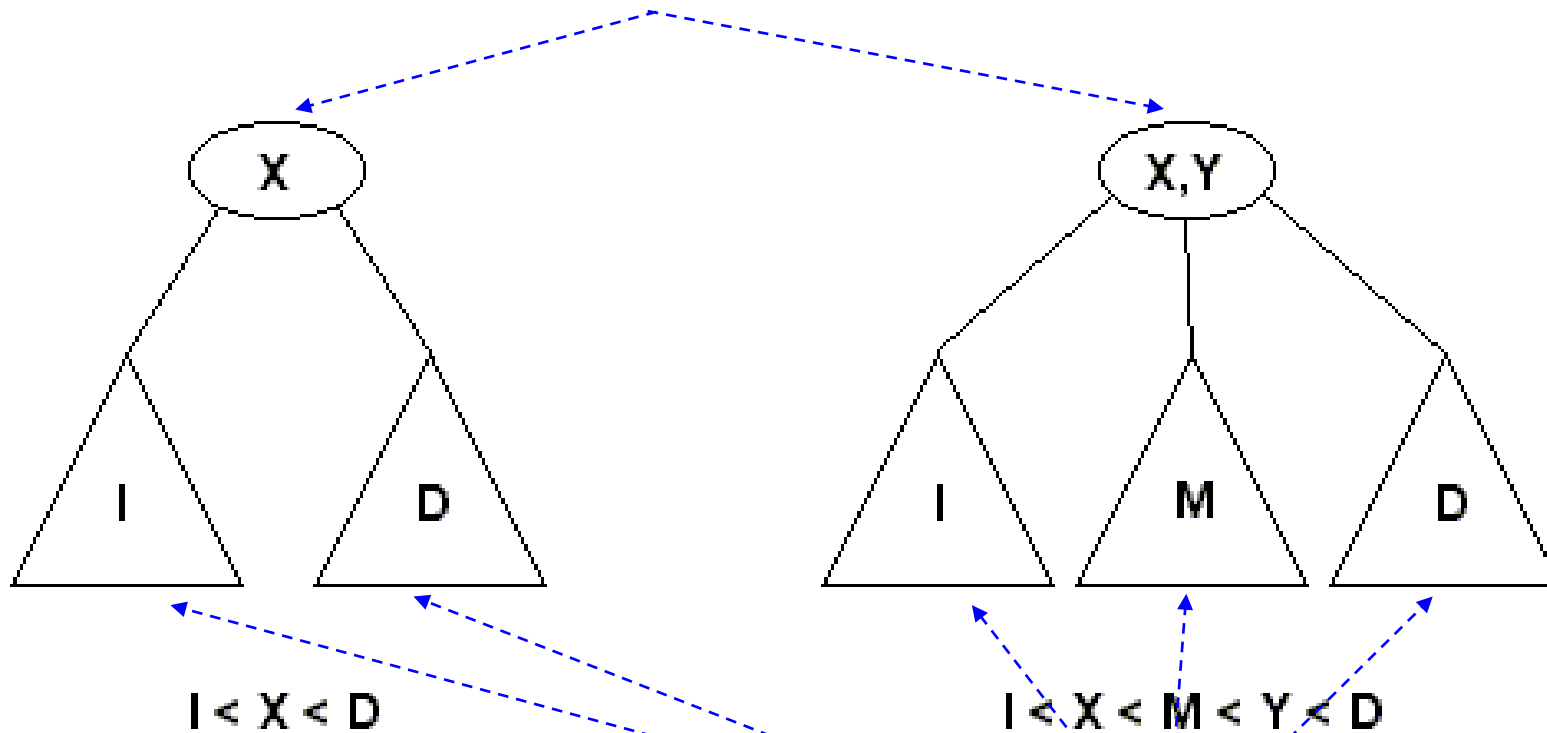
¿Cómo mantenerlo?



- Todos los nodos pueden tener hasta 2 elementos.
- Un nodo interno puede tener 2 ó 3 hijos, dependiendo de cuántos elementos posea el nodo:
 - Si hay 1 elemento en el nodo, debe tener 2 hijos
 - Si hay 2 elementos en el nodo, debe tener 3 hijos

Árbol 2-3

Cada nodo tiene hasta 2 elementos

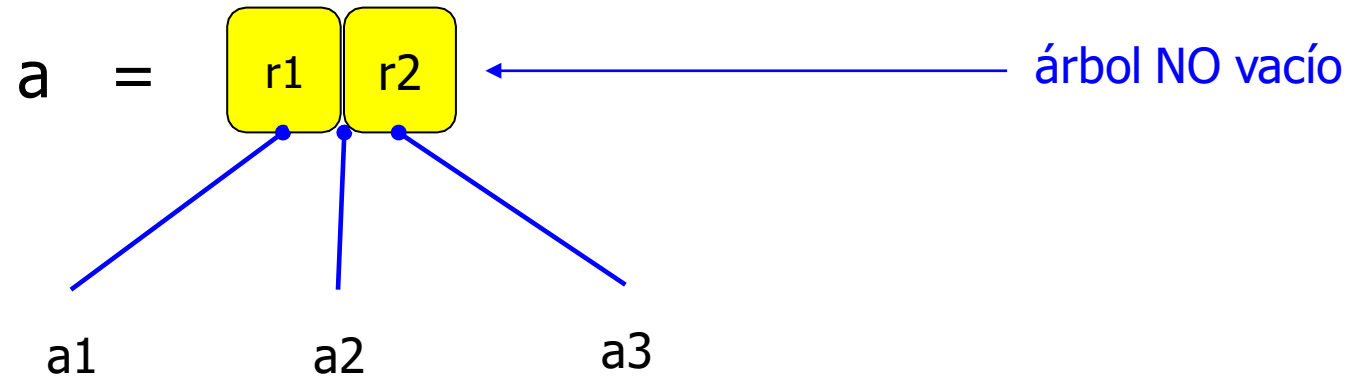


Cada nodo interno puede tener 2 o 3 hijos (dependiendo de cuántos elementos posea el nodo)

Árbol 2-3

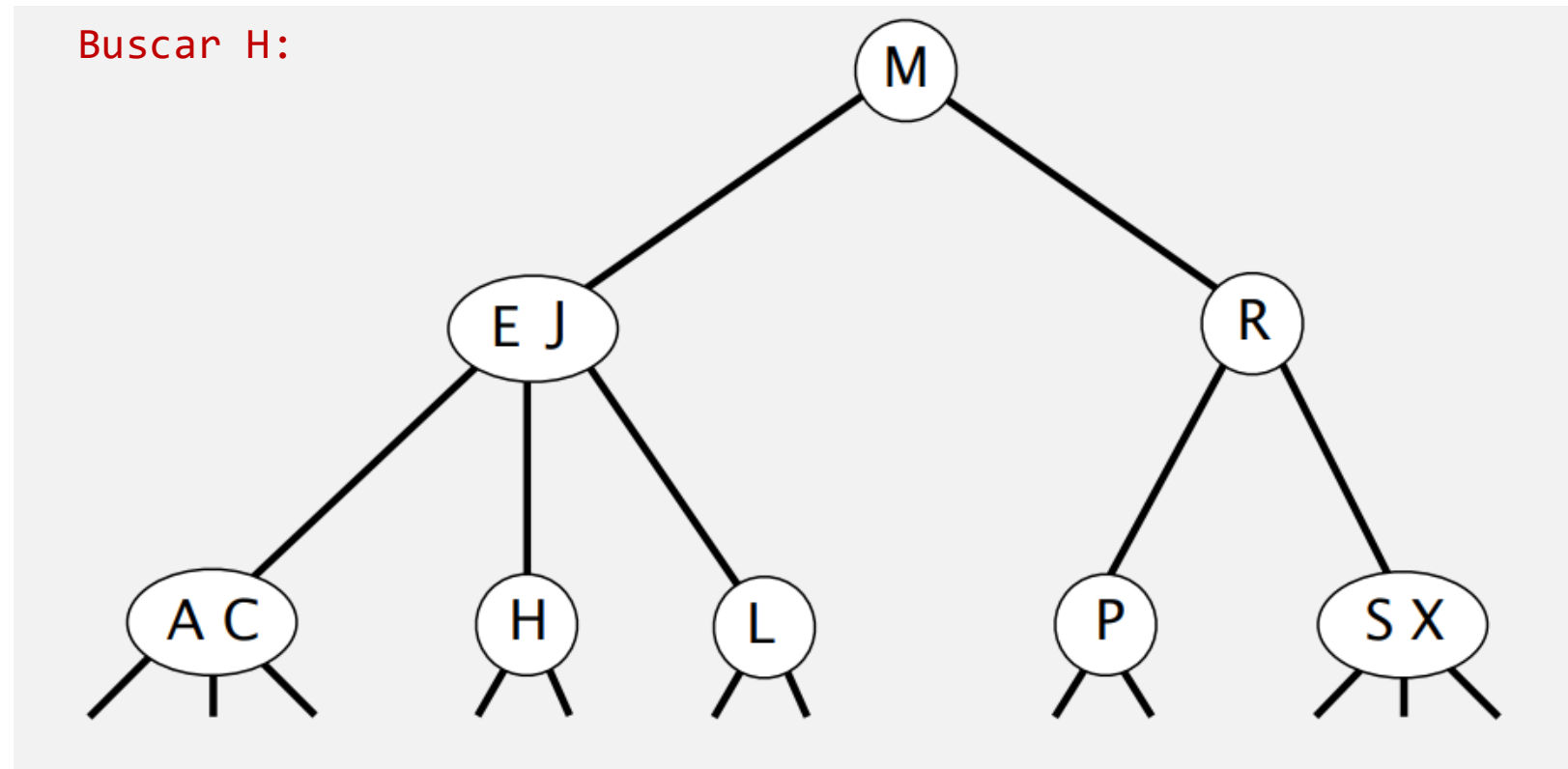
- Formalismo abstracto

$a = \Delta$ ← árbol vacío



Árboles 2-3

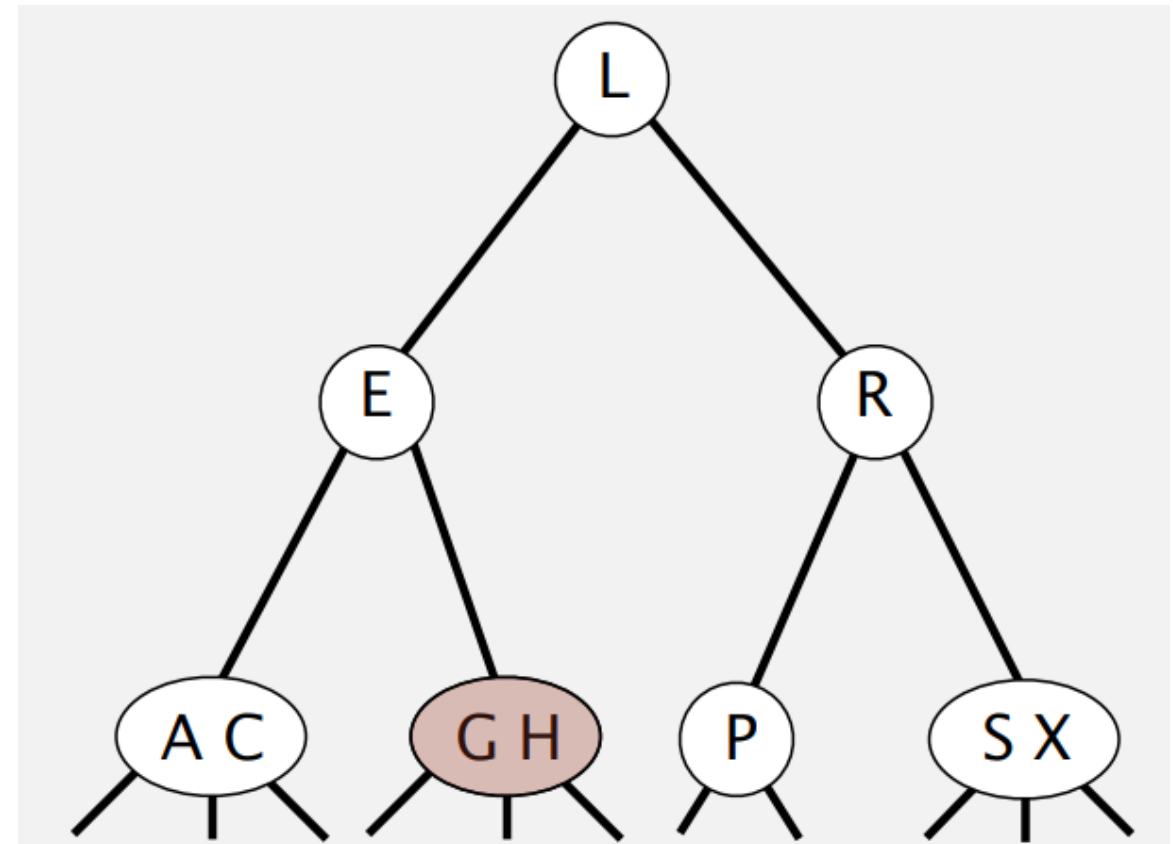
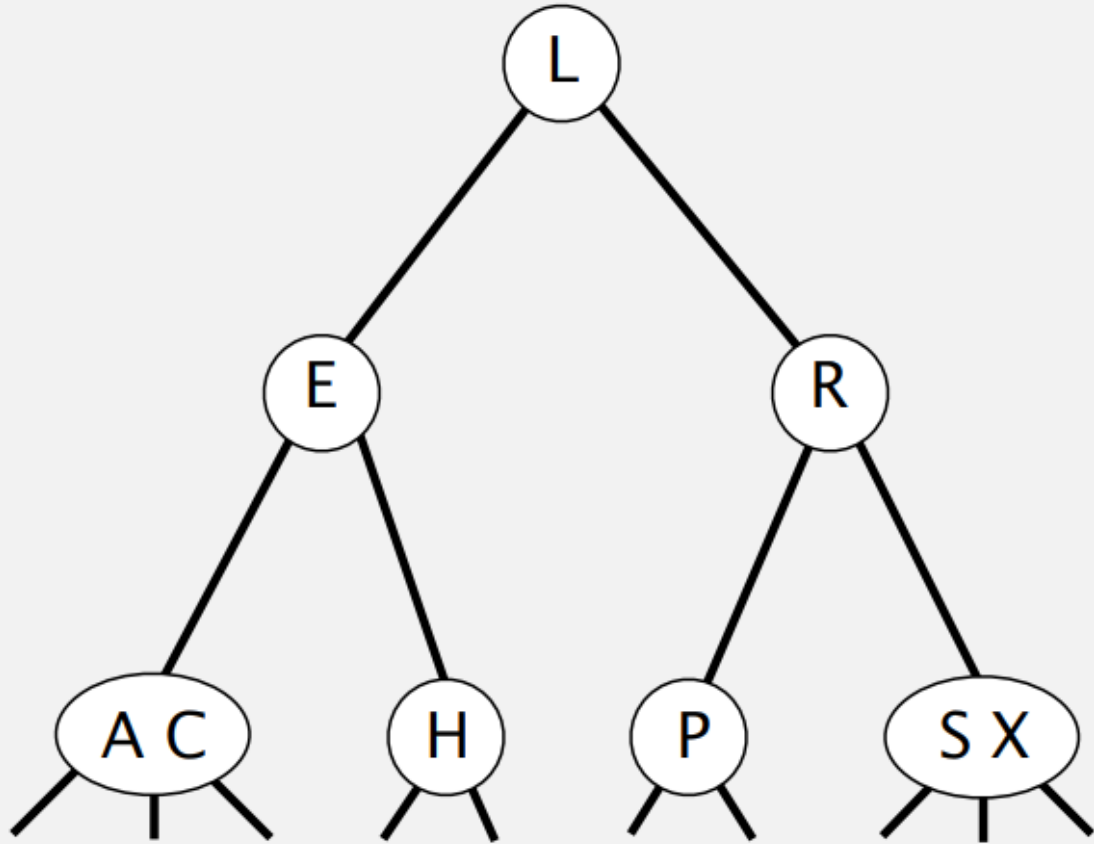
- **Búsqueda:**
 - Compare la clave de búsqueda con las claves en el nodo
 - Encuentra el intervalo que contiene la clave de búsqueda
 - Seguir enlace asociado (recursivamente)



Árboles 2-3

- Inserción en un 2-Nodo en la parte inferior (nodo hoja):
 - Agregue una nueva clave al 2-nodo para crear un 3-nodo

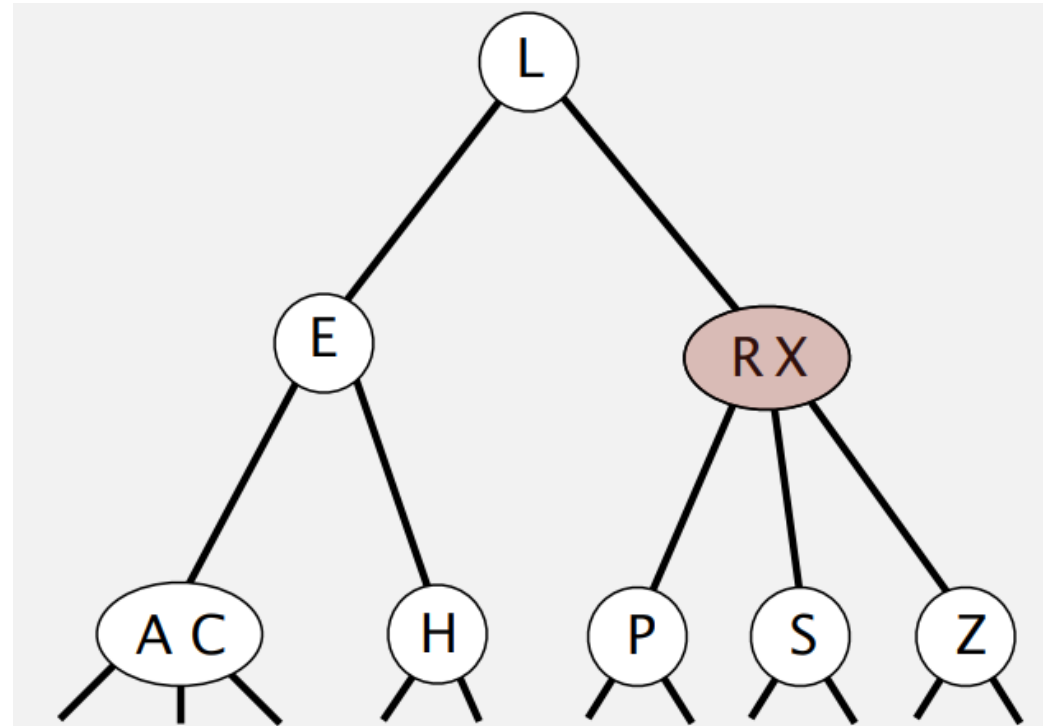
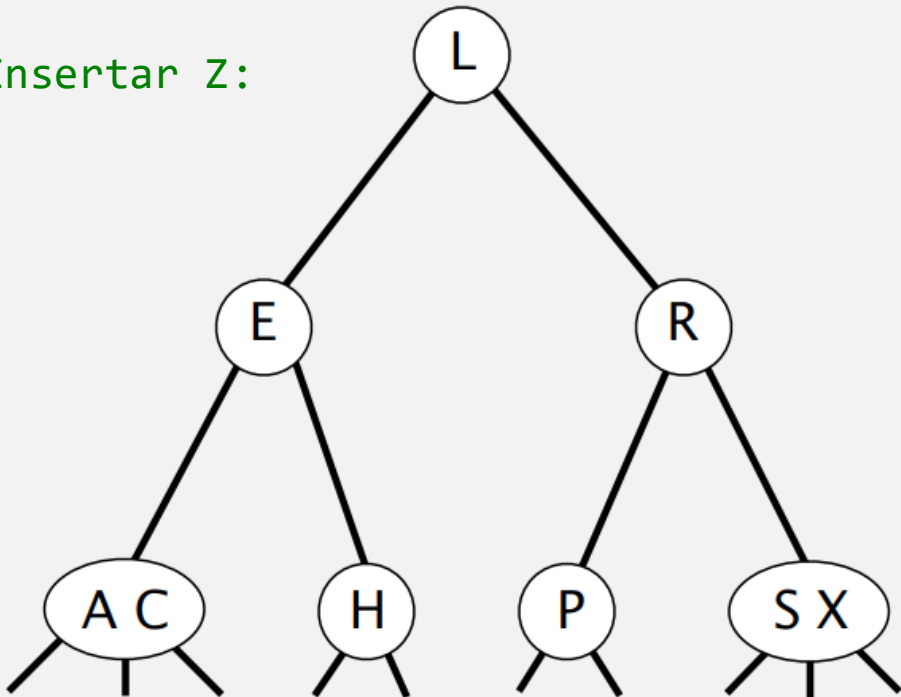
Insertar G:



Árboles 2-3

- Inserción en un 3-Nodo en la parte inferior (nodo hoja):
 - Agregue una nueva clave al 3-Nodo para crear un 4-Nodo temporal
 - Mover la clave central del 4-Nodo al padre
 - Repita hacia arriba del árbol, según sea necesario
 - Si se llega a la raíz y se convierte en un 4-Nodo, divídalo en tres 2-Nodo

Insertar Z:



Árbol 2-3: Inserción

- El crecimiento NO se hace a nivel de las hojas
 - Aunque la inserción se sigue haciendo en las hojas
- El crecimiento se hace a nivel de la raíz
 - Todas las hojas se deben mantener siempre en el mismo nivel

Demo Operaciones en un Árbol 2-3

Árbol 2-3: Inserción

- 1 Localizar la hoja en la cual se debe agregar el elemento
- 2 Insertar
 - **Caso1:** Existe espacio en el nodo -> la estructura del árbol NO se altera

Situación inicial	Situación final				
<table border="1"><tr><td>r1</td><td></td></tr></table> , $r1 < elem$	r1		<table border="1"><tr><td>r1</td><td>elem</td></tr></table>	r1	elem
r1					
r1	elem				
<table border="1"><tr><td>r1</td><td></td></tr></table> , $r1 > elem$	r1		<table border="1"><tr><td>elem</td><td>r1</td></tr></table>	elem	r1
r1					
elem	r1				

Árbol 2-3: Inserción

- **Caso 2:** El nodo está lleno. Se debe modificar la estructura del árbol:

El nodo se parte en dos nodos del mismo nivel

Los tres elementos (dos elementos del nodo y el nuevo elemento) se reparten de la siguiente manera:


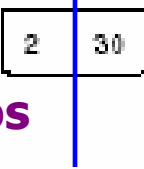
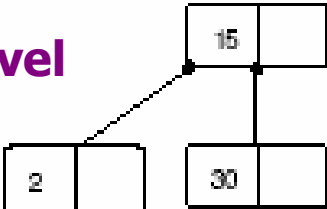
Situación inicial	Situación intermedia						
<table><tr><td>r1</td><td>r2</td></tr></table> , elem > r2	r1	r2	<table><tr><td>r1</td><td></td><td>elem</td><td></td></tr></table> , sube r2	r1		elem	
r1	r2						
r1		elem					
<table><tr><td>r1</td><td>r2</td></tr></table> , elem < r1	r1	r2	<table><tr><td>elem</td><td></td><td>r2</td><td></td></tr></table> , sube r1	elem		r2	
r1	r2						
elem		r2					
<table><tr><td>r1</td><td>r2</td></tr></table> , r1 < elem < r2	r1	r2	<table><tr><td>r1</td><td></td><td>r2</td><td></td></tr></table> , sube elem	r1		r2	
r1	r2						
r1		r2					

Árbol 2-3: Inserción

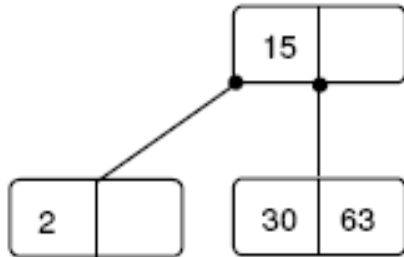
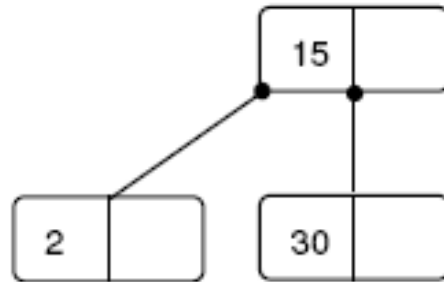
Situación inicial	Situación intermedia
<div><div>r1r2</div>, elem > r2</div>	<div>r1</div> <div>elem</div> , sube <div>r2</div>
<div>r1r2</div> , elem < r1	<div>elem</div> <div>r2</div> , sube <div>r1</div>
<div>r1r2</div> , r1 < elem < r2	<div>r1</div> <div>r2</div> , sube <div>elem</div>

- El elemento que no fue incluido en los dos nodos nuevos se sube en la estructura y se inserta en su padre.
- Se repite el proceso hacia arriba:
 - Al partir en dos el nodo se está generando un nuevo subárbol que puede generar que los ancestros se tengan que partir a su vez para poderlo incluir.

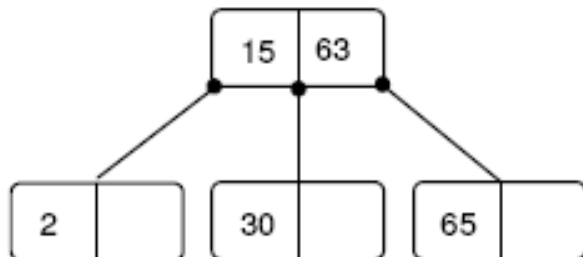
Árbol 2-3 : Inserción

	Insertar el elemento 30: se crea una hoja y se coloca el elemento como raíz izquierda
<p>Se parte en dos</p> 	Insertar el elemento 2: corresponde al caso 1. Se mueve a la derecha la raíz izquierda para dar cabida al nuevo elemento.
<p>Se crea un nivel</p> 	Insertar el elemento 15: corresponde al caso 2. Encuentra una hoja llena. La parte en dos nodos e inserta en el padre el elemento que se encuentre en la mitad de los tres ($2 < 15 < 30$). Como el padre es vacío, se crea un nuevo nivel, se coloca el elemento como la raíz izquierda del nodo, y se le asocian los dos nodos que se acaban de partir.

Árbol 2-3 : Inserción

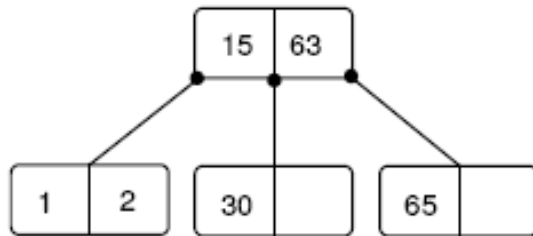
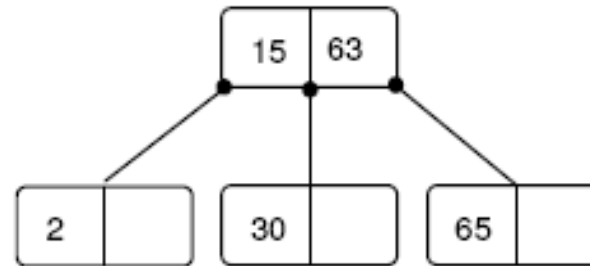


Insertar el elemento 63: corresponde al caso 1

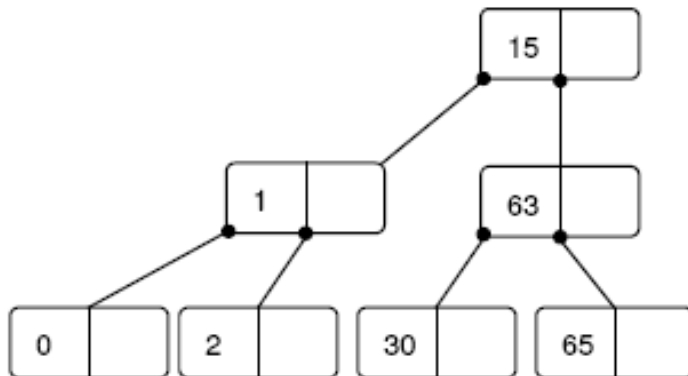


Insertar el elemento 65: corresponde al caso 2. Se parte la hoja y sube al padre el elemento de la mitad (63). Al insertar dicho valor en el padre se trata como el caso 1, porque en el nodo hay espacio.

Árbol 2-3 : Inserción

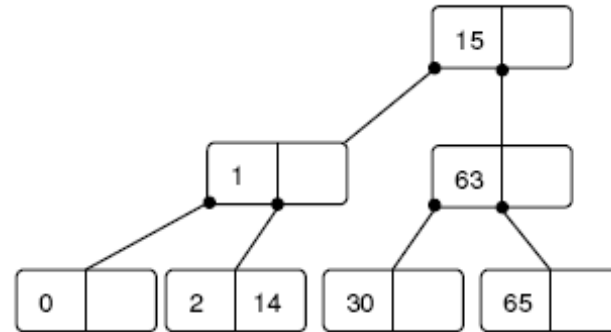
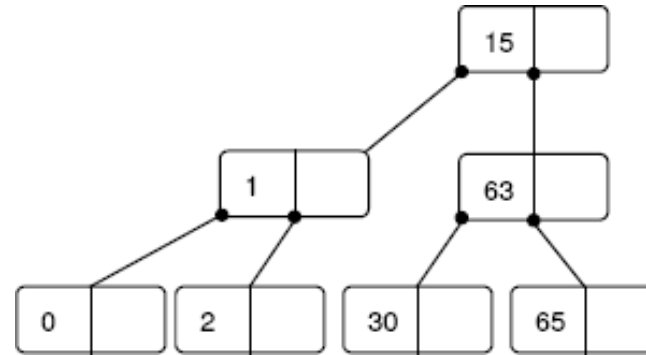


Insertar el elemento 1: corresponde al caso 1.

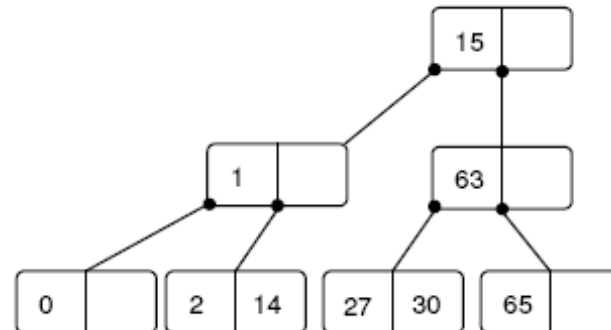


Insertar el elemento 0: corresponde al caso 2. Se parte la hoja [1, 2], se colocan allí los elementos 0 y 2, y sube el valor 1 a su padre. Como el nodo del padre [15, 63] está lleno también se debe partir, dejando en ese nivel los elementos 1 y 63, y subiendo el 15.

Árbol 2-3 : Inserción

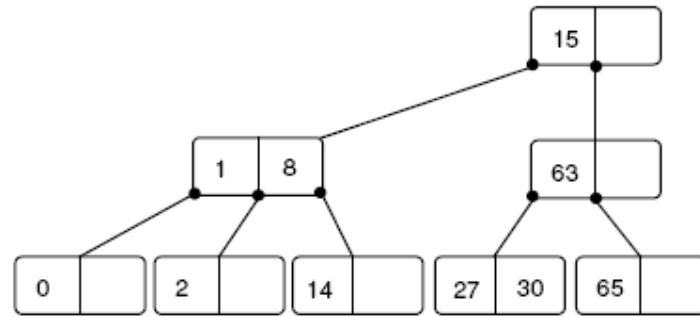
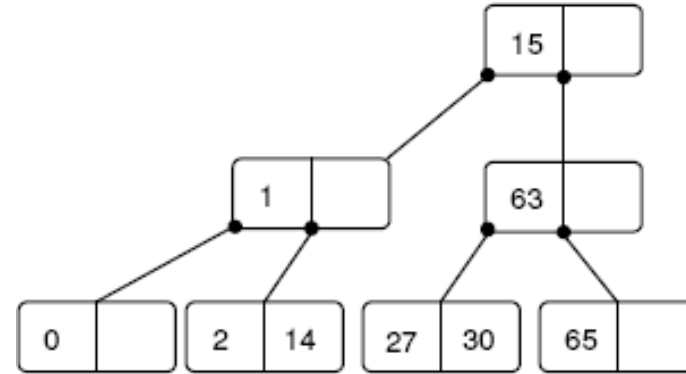


Insertar el elemento 14: corresponde al caso 1.

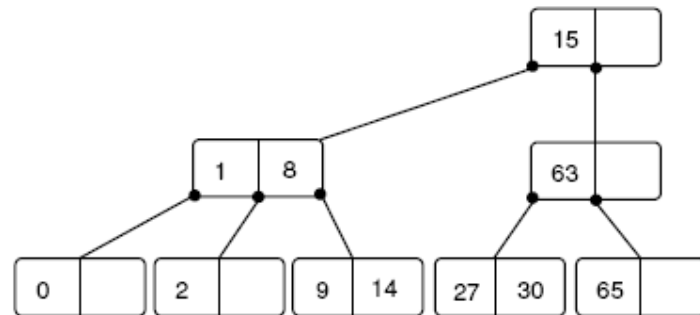


Insertar el elemento 27: corresponde al caso 1.

Árbol 2-3 : Inserción

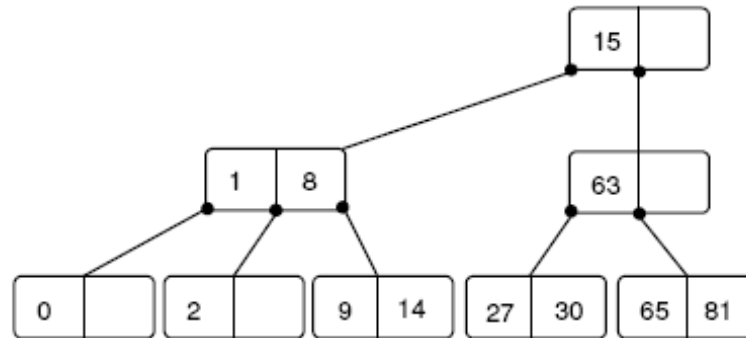
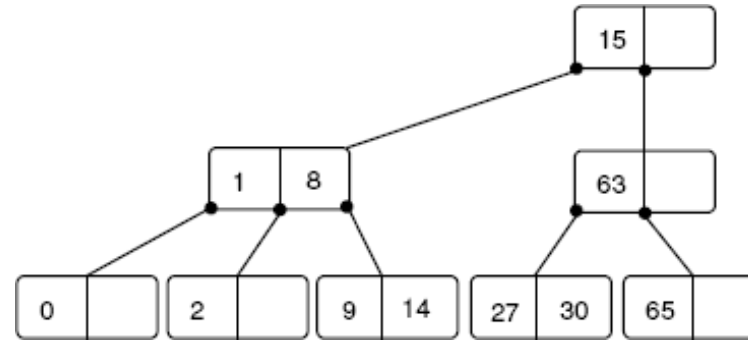


Insertar el elemento 8: corresponde al caso 2. Se parte el nodo [2, 14] y sube el 8. Allí encuentra espacio y se coloca como raíz derecha.

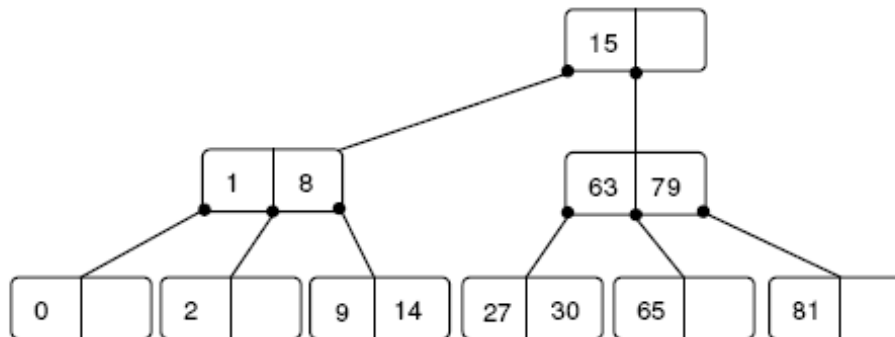


Insertar el elemento 9: corresponde al caso 1.

Árbol 2-3 : Inserción

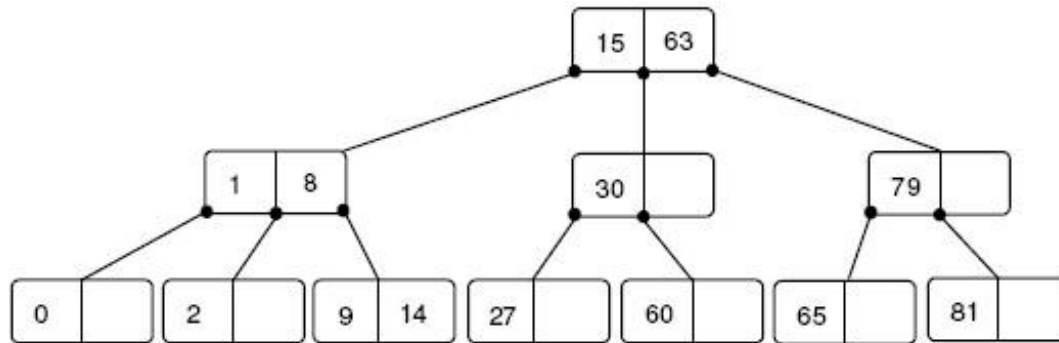
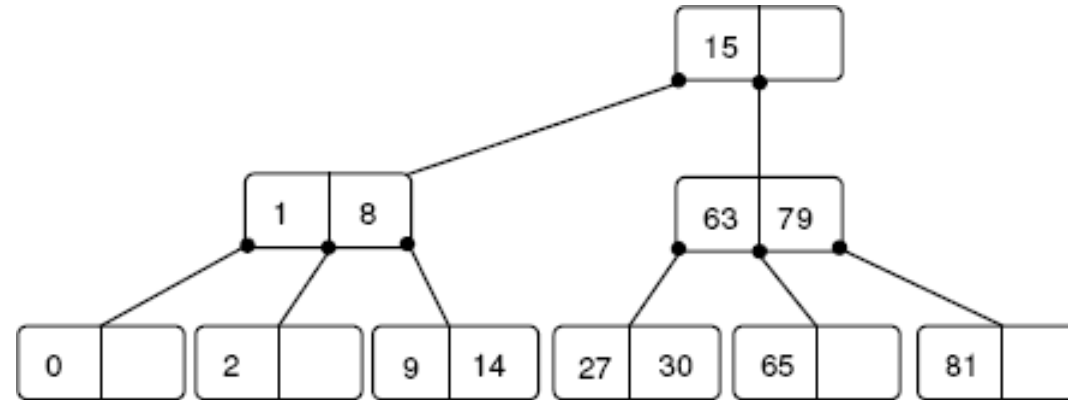


Insertar el elemento 81: corresponde al caso 1.



Insertar el elemento 79: corresponde al caso 2. Se parte el nodo [65, 81] y sube el elemento 79.

Árbol 2-3 : Inserción



Insertar el elemento 60: corresponde al caso 2. Se parte el nodo [27, 30], se incluye el 60 y sube el elemento 30. Como su padre está lleno se parte en los nodos [30] y [79], y sube el elemento 63. Este elemento se sitúa en la raíz derecha del árbol, donde hay espacio libre.

Árbol 2-3 : Inserción

insert S



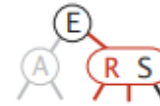
E



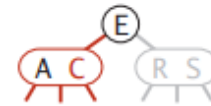
A



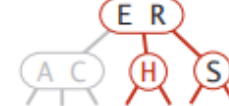
R



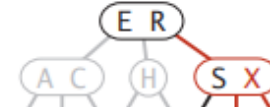
C



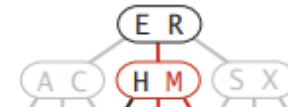
H



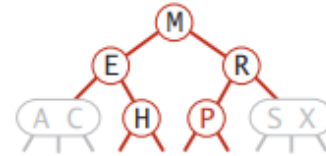
X



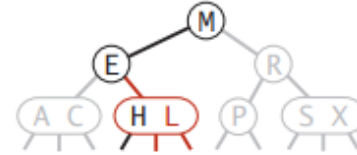
M



P



L



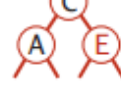
insert A



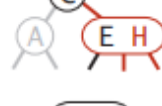
C



E



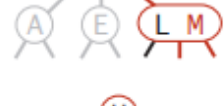
H



L



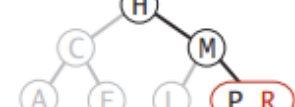
M



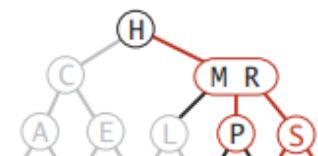
P



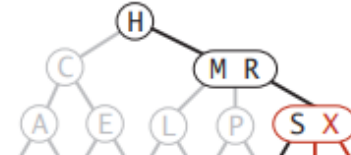
R



S



X



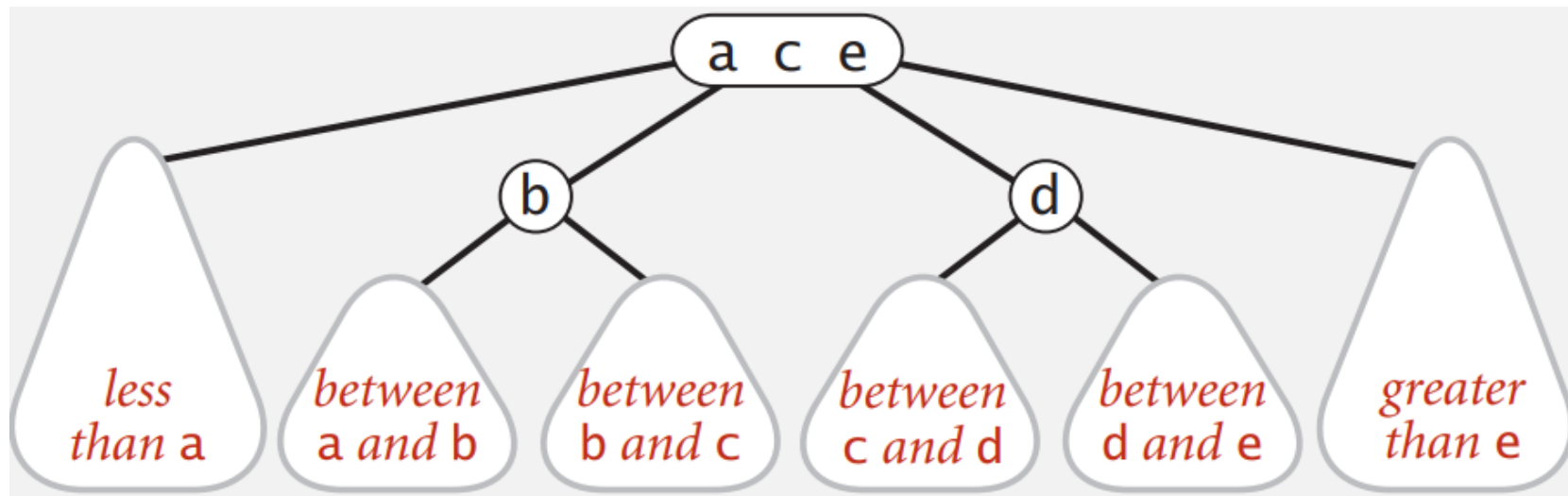
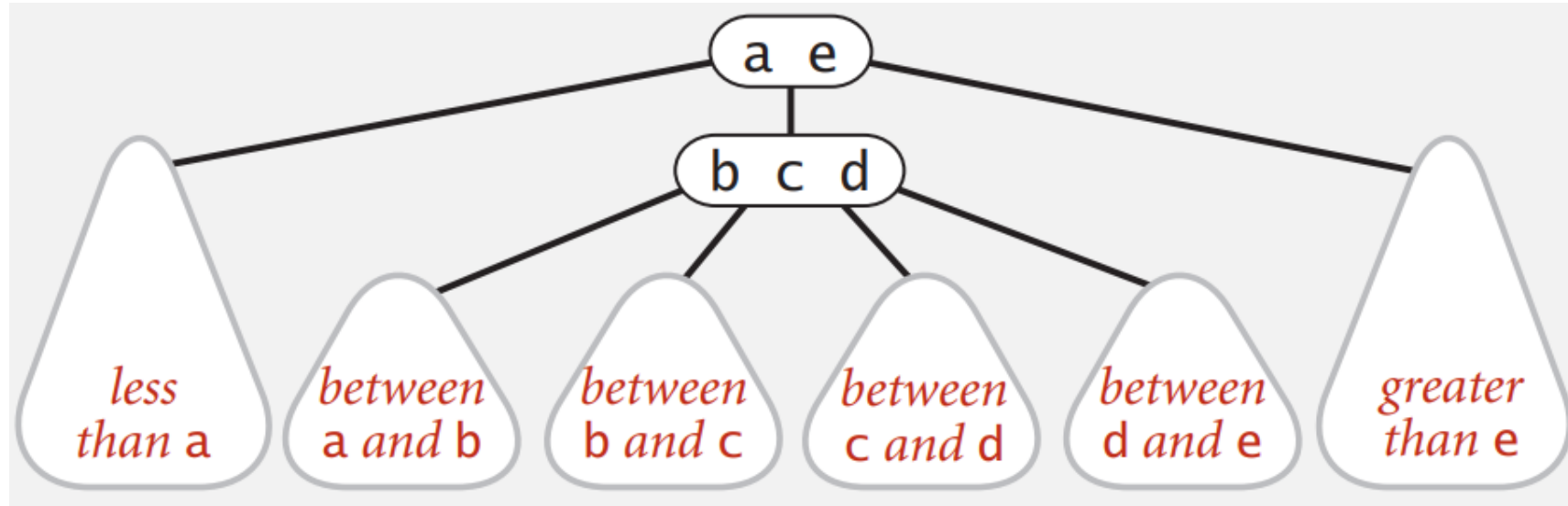
Ejercicio

- Muestre el proceso de inserción en un árbol 2-3 de la siguiente secuencia de valores:

25 – 86 – 34 – 23 – 4 – 98 – 12 – 56 – 74 – 77 - 80

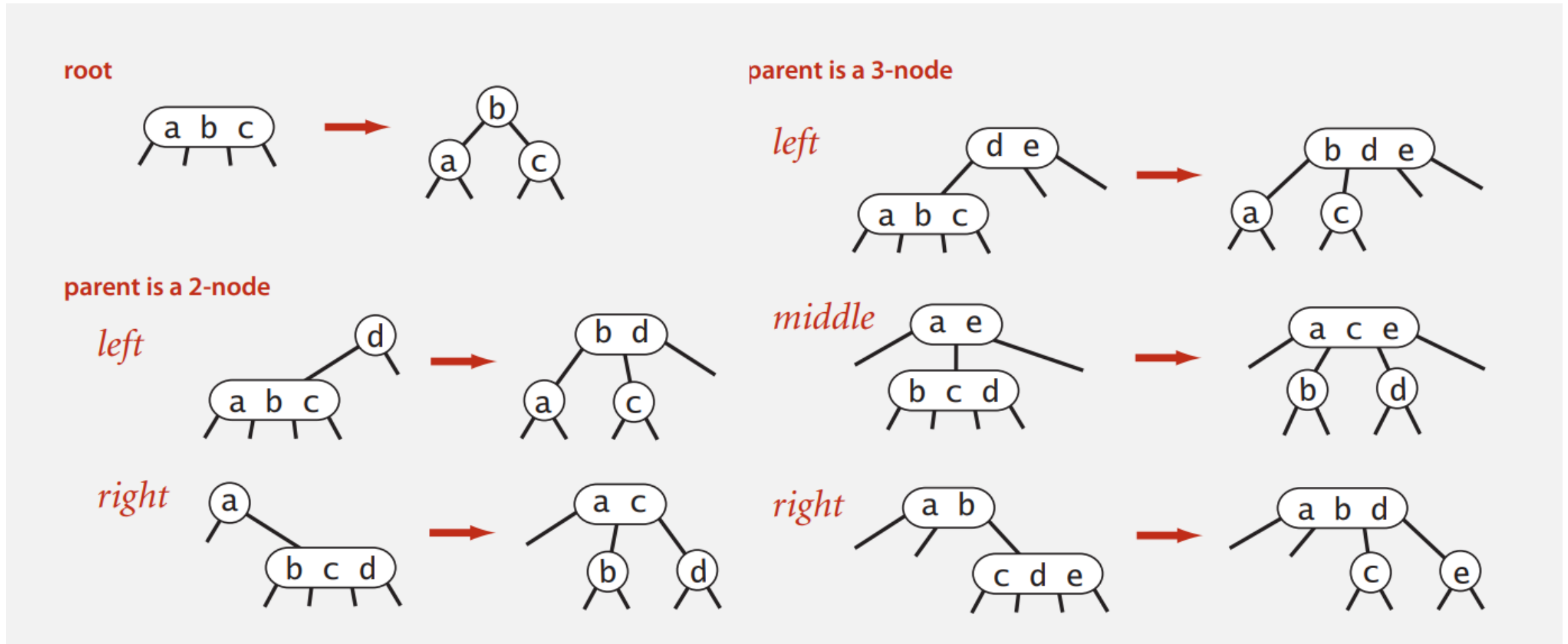
Árboles 2-3

- Dividir un 4-Nodo es una transformación local : Requiere un número constante de operaciones.



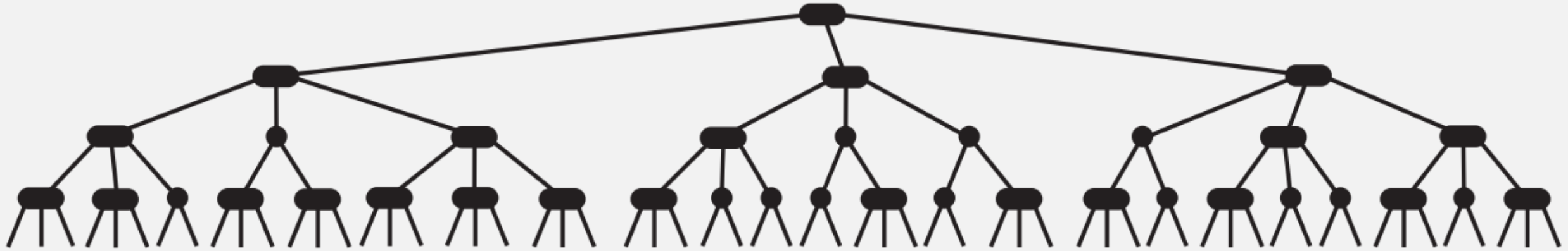
Propiedades globales de un Árbol 2-3

Invariantes. Mantiene el orden simétrico y el equilibrio perfecto.
Cada transformación mantiene un orden simétrico y un equilibrio perfecto.



Árbol 2-3 Rendimiento

Balance perfecto: Cada ruta desde la raíz hasta el enlace nulo (null) tiene la misma longitud.

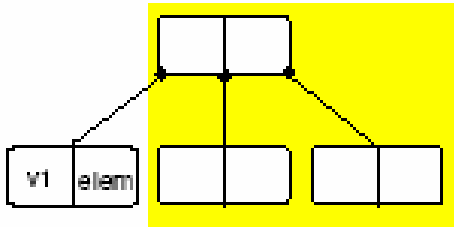
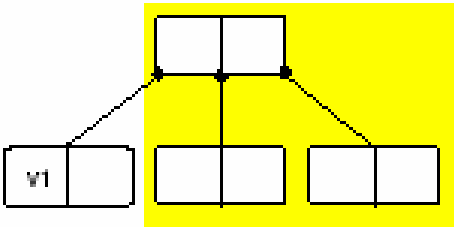
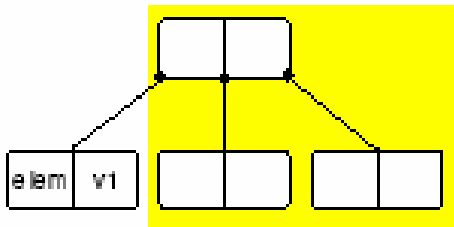
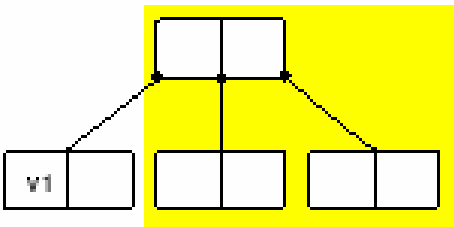
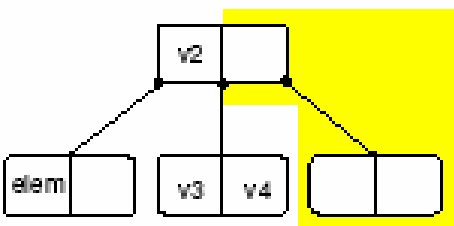
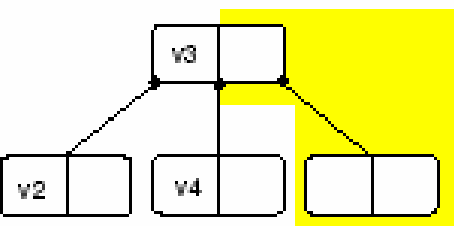


Árbol típico 2-3 construido a partir de claves aleatorias

- **Altura del árbol:**
 - Peor caso: $\log_2 N$ (todos 2-nodo)
 - Mejor caso: $\log_3 N \sim 0.631 \log N$ (todos 3-nodo)
 - Entre 12 y 20 para un millón de nodos.
 - Entre 18 y 30 para mil millones de nodos

Árbol 2-3: Eliminación

• Caso 1: El elemento está en una hoja

Caso	Situación inicial	Solución
A		
B		
C		

En el caso C, se hace redistribución, subo la clave más chica del hermano derecho y bajo el padre que separa ambos

Árbol 2-3 : Eliminación

• Caso 1: El elemento está en una hoja

Redistribución:

- Subo la clave más chica del hermano derecho y bajo el padre que separa ambos.

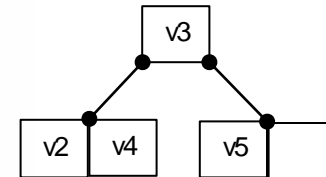
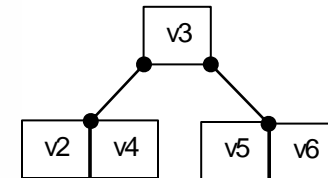
- Subo la clave mas grande del hermano izquierdo y bajo el padre que separa ambos.

Mezcla o

concatenación:

- Uno al nodo afectado con algún hermano y el padre que separa ambos

D		
E		
F		



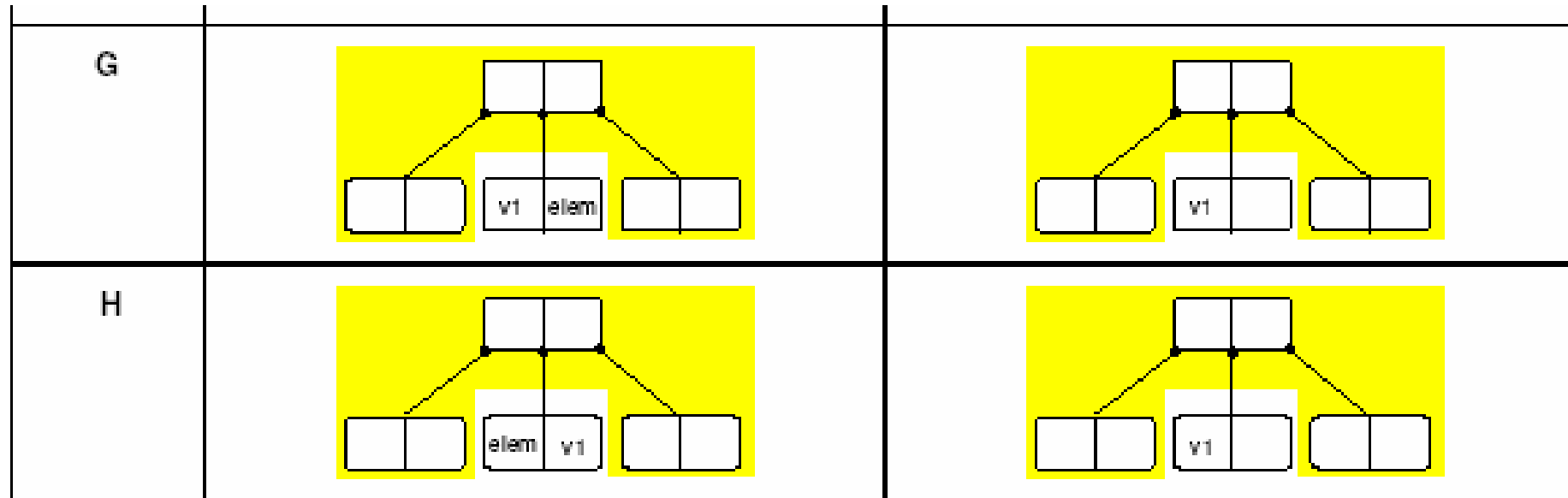
En el caso D, la redistribución se hace dos veces, o se puede solo aplicar mezcla

En el caso E también se hace la redistribución y mezcla, o se puede solo hacer mezcla

En el caso F se mezcla o concatena: Uno al nodo afectado con algún hermano y el padre que separa ambos

Árbol 2-3 : Eliminación

- **Caso1:** El elemento está en una hoja



Árbol 2-3 : Eliminación

• Caso1: El elemento está en una hoja

Redistribución:

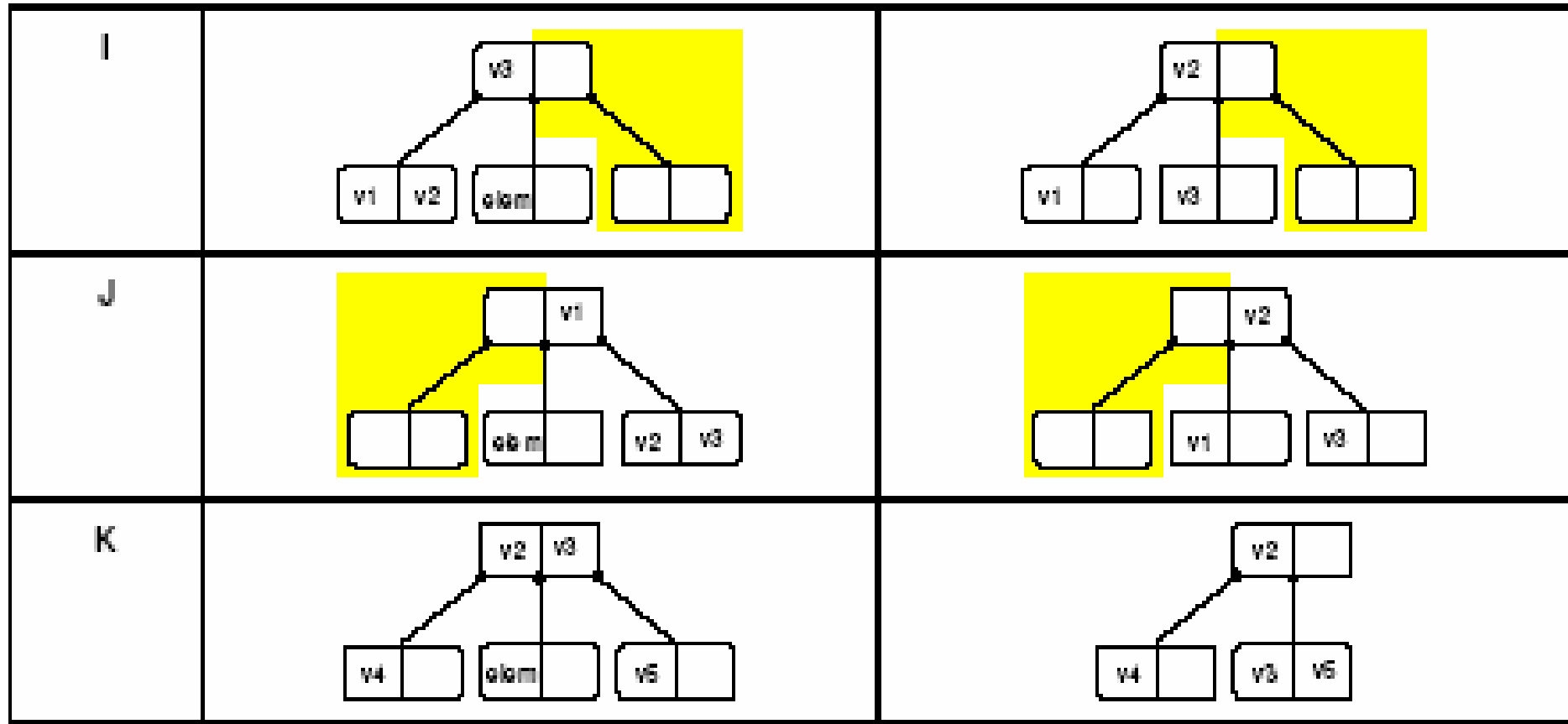
- Subo la clave más chica del hermano derecho y bajo el padre que separa ambos.

- Subo la clave mas grande del hermano izquierdo y bajo el padre que separa ambos.

Mezcla o

concatenación:

- Uno al nodo afectado con algún hermano y el padre que separa ambos



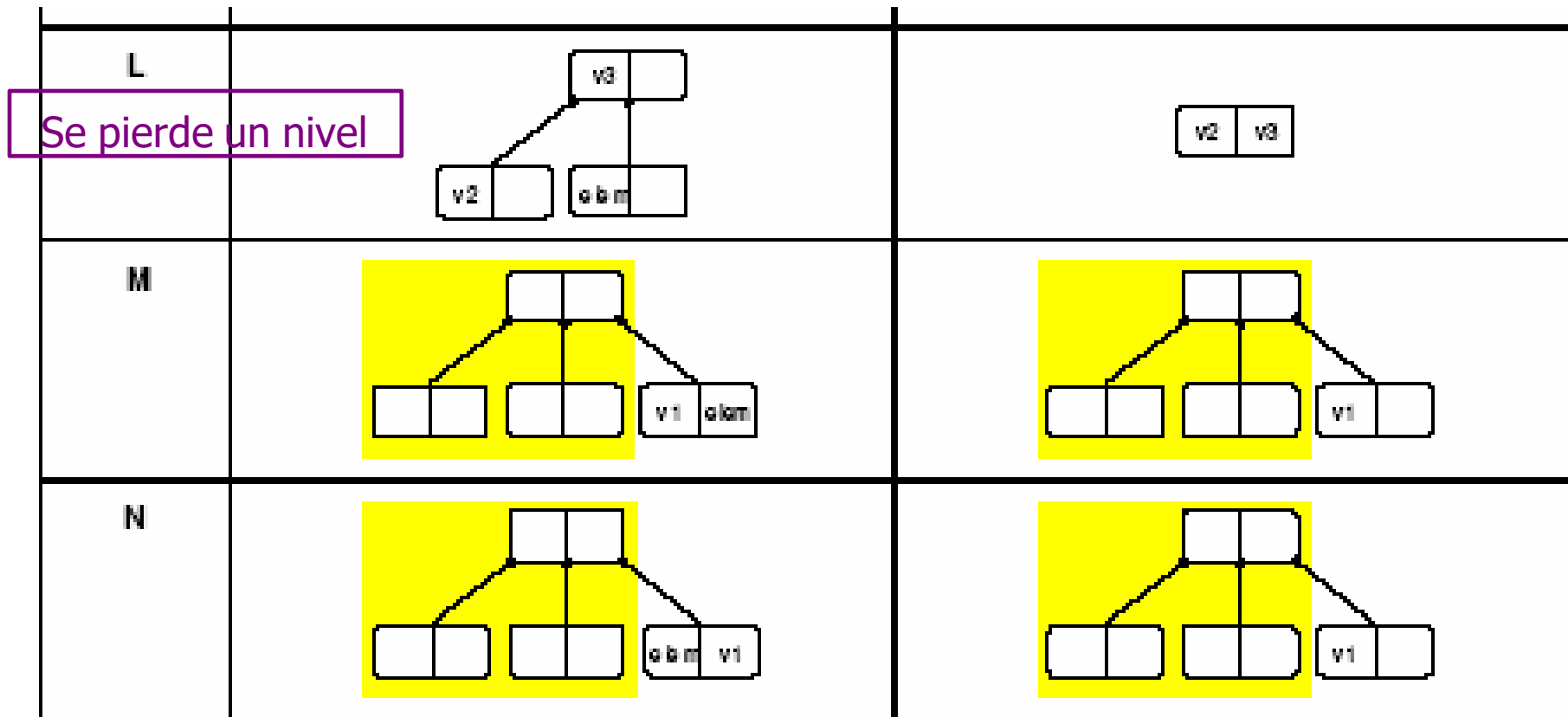
En el caso I, hago redistribución con la izquierda, subo la clave más grande del hermano izquierdo y bajo el padre que separa ambos

En el caso J, hago redistribución, subo la clave más chica del hermano derecho y bajo el padre que separa ambos

En el caso K se aplica mezcla, uno al nodo afectado con algún hermano (derecho) y el padre que separa ambos

Árbol 2-3 : Eliminación

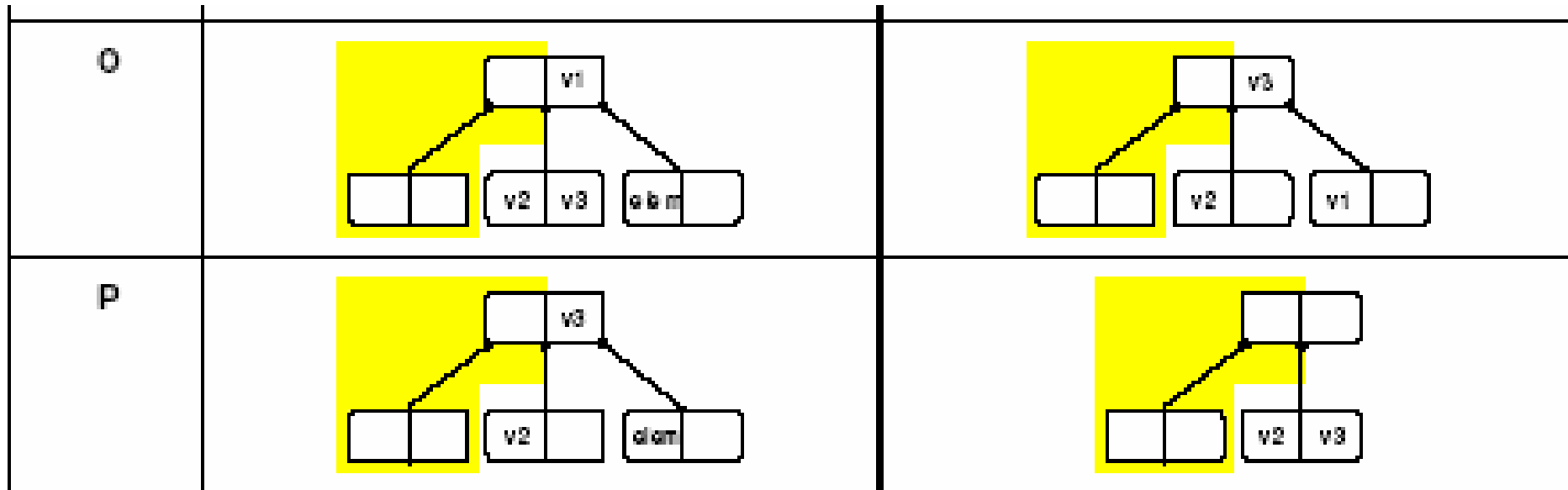
- **Caso1:** El elemento está en una hoja



En el caso L se mezcla o concatena: Uno al nodo afectado con algún hermano (izquierdo) y el padre que separa ambos

Árbol 2-3 : Eliminación

- **Caso1:** El elemento está en una hoja



En el caso O, subo la clave más grande del hermano izquierdo y bajo el padre que separa ambos

En el caso P, uno al nodo afectado con algún hermano y el padre que separa ambos

Árbol 2-3 : Eliminación

- **Caso 2:** El elemento no está en una hoja
 - Se busca un valor que se encuentre en una hoja y que pueda reemplazar el valor. (Como está ordenado, el candidato sería el menor del subárbol derecho)

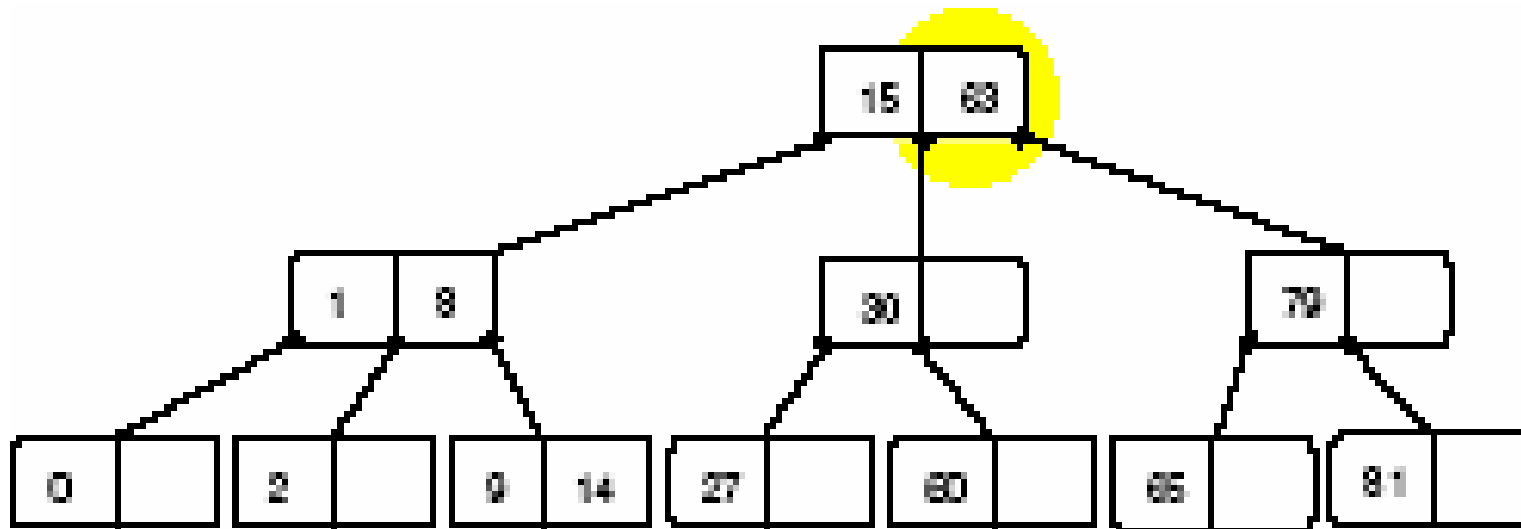
Árbol 2-3 : Eliminación

- **Caso2:** El elemento no está en una hoja

	Situación inicial	Situación intermedia: Eliminar v2 de una hoja (Caso 1)
Q		
R		

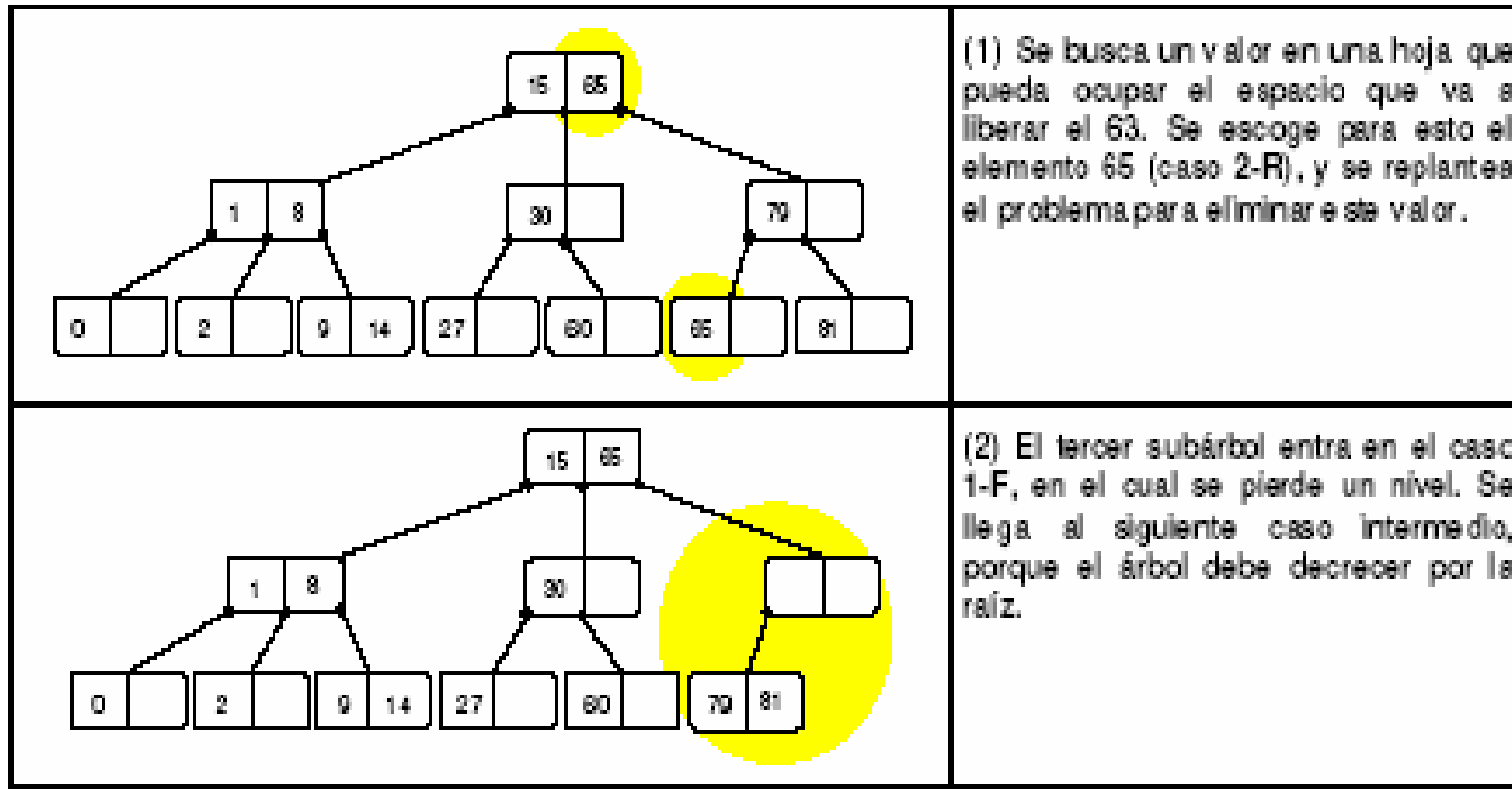
Árbol 2-3 : Eliminación

- Ejemplo: Eliminar el 63



Árbol 2-3 : Eliminación

- Ejemplo: Eliminar el 63



Árbol 2-3 : Eliminación

- Ejemplo: Eliminar el 63

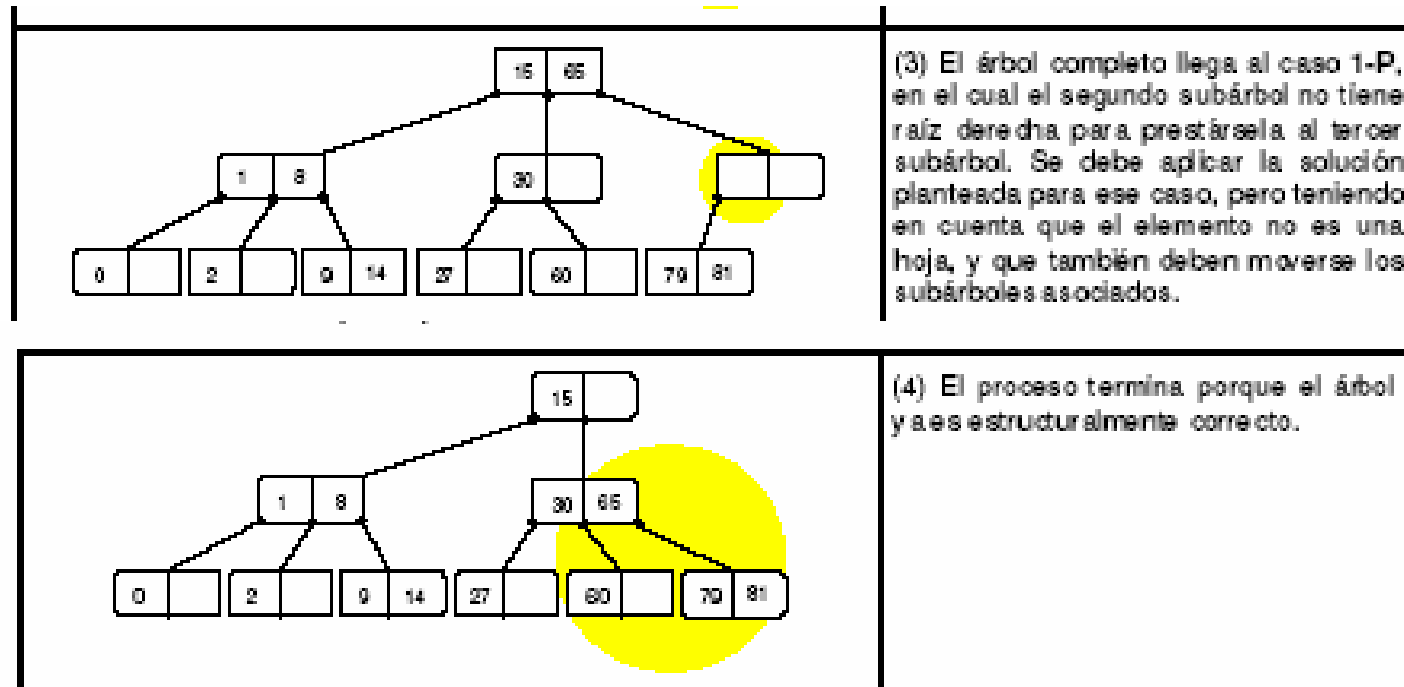


Tabla de símbolos: Resumen

Implementación	Garantía			Caso promedio			Operaciones sobre claves	Soporta operaciones eficientes
	Búsqueda	Inserción	Delete	Acierto en búsqueda	Inserción	Delete		
Lista enlazada desordenada con búsqueda secuencial	N	N	N	N/2	N	N/2	equals()	No
Arreglo ordenado con búsqueda binaria	Log N	N	N	Log N	N/2	N/2	compareTo()	Si
BST	N	N	N	1.39 Log N	1.39 Log N	\sqrt{N}	compareTo()	Si
Meta	c Log N	c Log N	c Log N	c Log N	c Log N	c Log N	compareTo()	Si

constante c depende de la implementación



Árboles 2-3 Implementación

- La implementación directa es complicada porque:
 - El mantenimiento de múltiples tipos de nodos es engorroso
 - Necesita múltiples comparaciones para moverse hacia abajo del árbol
 - Necesita volver a subir en el árbol para dividir los 4-nodos
 - Gran cantidad de casos para dividir

```
public void put(Key key, Value val)
{
    Node x = root;
    while (x.getTheCorrectChild(key) != null)
    {
        x = x.getTheCorrectChildKey();
        if (x.is4Node()) x.split();
    }
    if (x.is2Node()) x.make3Node(key, val);
    else if (x.is3Node()) x.make4Node(key, val);
}
```

Código fantasía

Árboles 2-3 Implementación

```
public class Arbol2_3<T extends Comparable<? super T>>
{
    // -----
    // Atributos
    // -----
    /**
     * Raíz del árbol 2-3
     */
    private Nodo2_3<T> raiz;
```

```
public class Nodo2_3<T extends Comparable<? super T>>
{
    // -----
    // Atributos
    // -----
    /**
     * Primera raíz del nodo
     */
    private T r1;

    /**
     * Segunda raíz del nodo
     */
    private T r2;

    /**
     * Subárbol 1
     */
    private Nodo2_3<T> h1;

    /**
     * Subárbol 2
     */
    private Nodo2_3<T> h2;

    /**
     * Subárbol 3
     */
    private Nodo2_3<T> h3;
```

Árboles 2-3 Implementación

Desarrolle el método que permite buscar un elemento en un árbol 2-3

```
public T buscar( T modelo )  
{  
    return ( raiz != null ) ? raiz.buscar( modelo ) : null;  
}
```

```
...  
public T buscar( T modelo )  
{  
    ...  
    ...  
}
```

Arboles B: Motivación

- Los sistemas de almacenamiento masivo suelen tener un **tiempo de acceso** mucho mayor que el **tiempo de transferencia**: La localización de un elemento es mucho más costosa que la lectura secuencial de datos, una vez localizados.
- Esto se aplica sobre todo a discos duros, pero también, aunque en menor medida, a memorias de estado sólido (flash) e incluso a memorias volátiles.
- Esto supone un problema para estructuras enlazadas, como los árboles BST, donde las operaciones acceden a bastantes nodos de pequeño tamaño.
- Para grandes volúmenes de datos, sería conveniente **reducir el número de accesos**, a cambio de que esos accesos contuvieran **elementos de mayor tamaño**.



Modelo de un sistemas de archivos

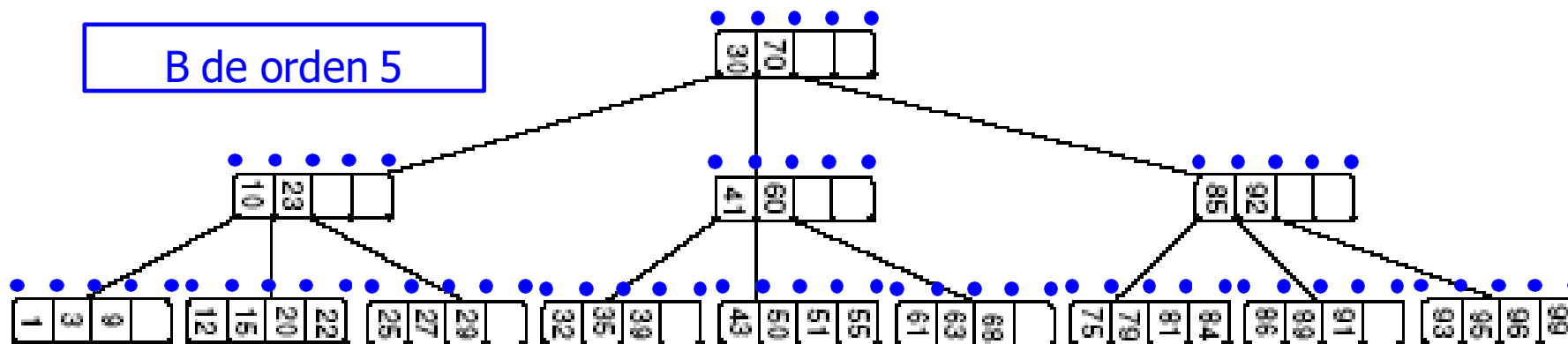
- **Página:** Bloque contiguo de datos (p. ej., un archivo o fragmento de 4096 bytes).
- **Sondeo:** Primer acceso a una página (por ejemplo, de disco a memoria).



- **Propiedad:** El tiempo requerido para un sondeo es mucho mayor que el tiempo para acceder a los datos dentro de una página.
- **Modelo de costos:** Número de Sondeos.
- **Meta:** Acceda a los datos utilizando un número mínimo de sondeos

Árboles B

- Árbol n-ario ordenado y balanceado
- Generalización de un árbol 2-3, Un árbol 2-3 es un árbol B de orden 3
- Para un árbol B de orden n :
 - Cada nodo tiene como mucho $n-1$ elementos y n hijos (subárboles B asociados no vacíos)
 - La raíz es una hoja o tiene al menos 2 hijos;
 - Todas las hojas se encuentran al mismo nivel
 - Todos los nodos internos, excepto la raíz y las hojas, tienen por lo menos $\lceil n/2 \rceil$ hijos. ¿La cantidad mínima de claves? $\lceil n/2 \rceil - 1$
 - Un nodo con q hijos contiene $q-1$ claves



Arboles B

- Si hay M hojas y las hojas están en el nivel L :
 - El nº de nodos en los niveles $1, 2, 3, \dots$ es por lo menos $2, 2 \lceil n/2 \rceil, 2 \lceil n/2 \rceil^2, \dots, 2 \lceil n/2 \rceil^{L-1}$
- Por lo tanto: $M \geq 2 \lceil n/2 \rceil^{L-1}$
- Es decir: $L \leq 1 + \log_{\lceil n/2 \rceil}(M/2)$
- Entonces, la altura está acotada por el logaritmo del nº de claves.

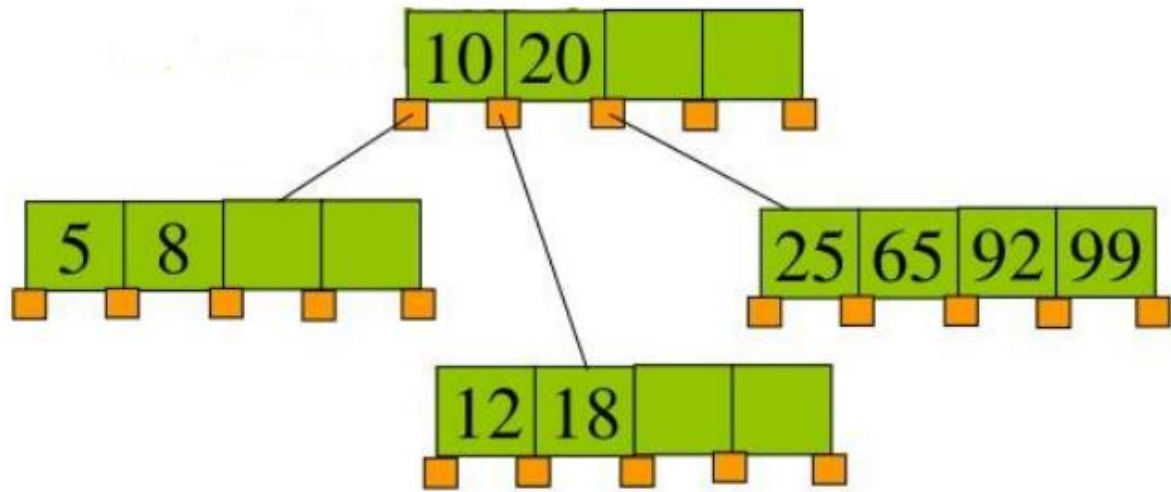


Demo Operaciones en un Árbol B

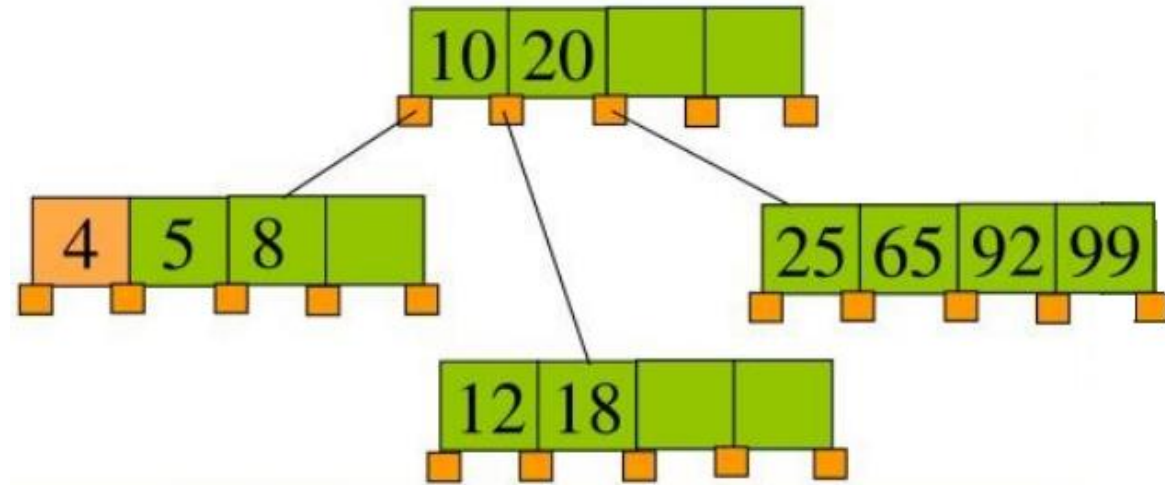
Árboles B - Inserción

- Buscar el nodo hoja donde se debería agregar el elemento
- Si hay espacio disponible en el nodo, agregar el elemento y terminar
- Si el nodo hoja **NO** tiene capacidad de almacenar el elemento, se deberá crear un nuevo nodo al mismo nivel de la hoja y distribuir a los $2k+1$ elementos de la siguiente forma:
 - El nuevo nodo recibe a los “k” elementos más grandes
 - El nodo existente se queda con los “k” elementos más pequeños
 - El elemento medio (1) se insertará en el nodo padre siguiendo la misma lógica de inserción. En caso de **NO** haber nodo padre, se creará un nuevo nodo que pasará a ser la nueva raíz

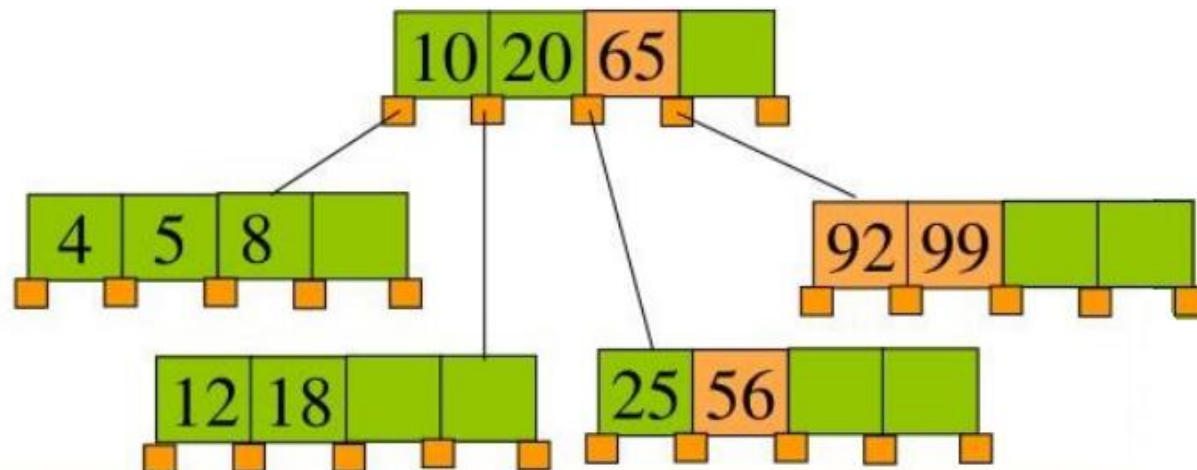
Árbol B: Inserción



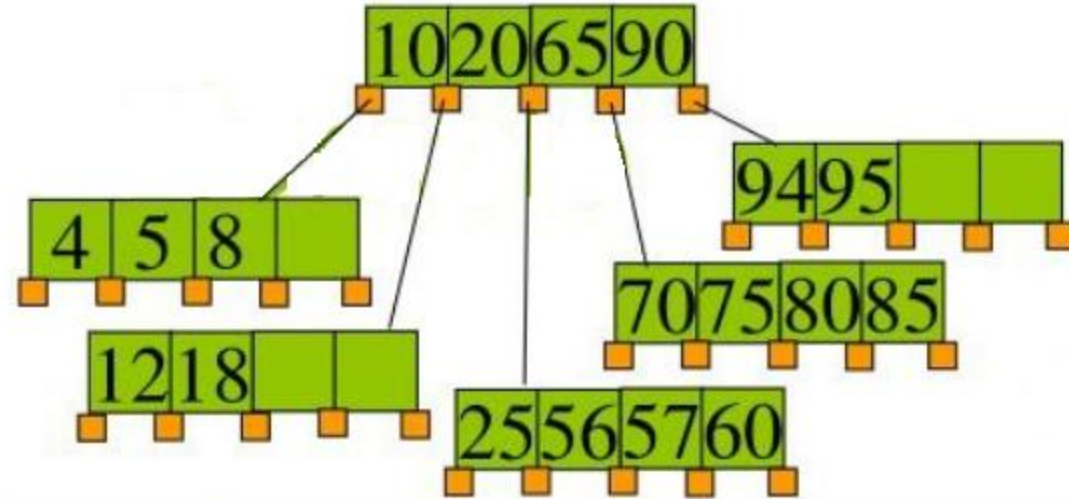
Agregar 4



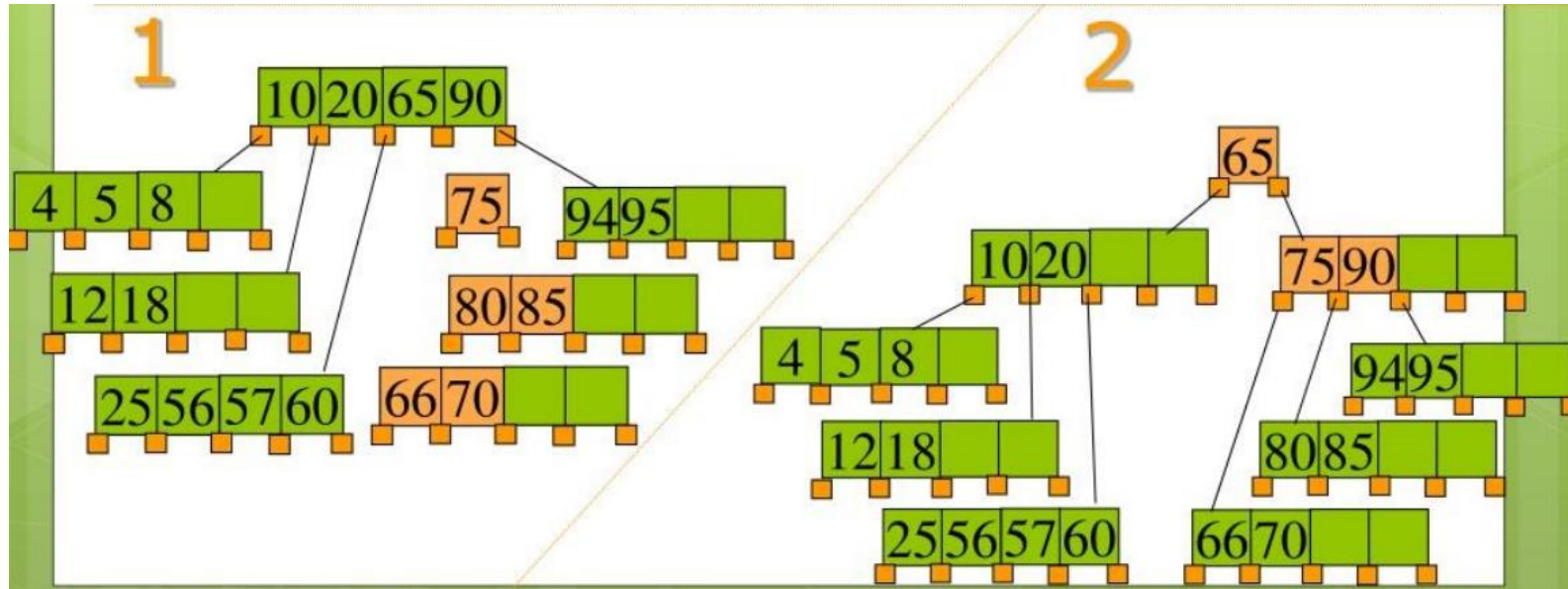
Agregar 56



Árbol B: Inserción



Agregar 66



Ejercicio

- Muestre el proceso de inserción en un árbol B (de orden 5) de la siguiente secuencia de valores:

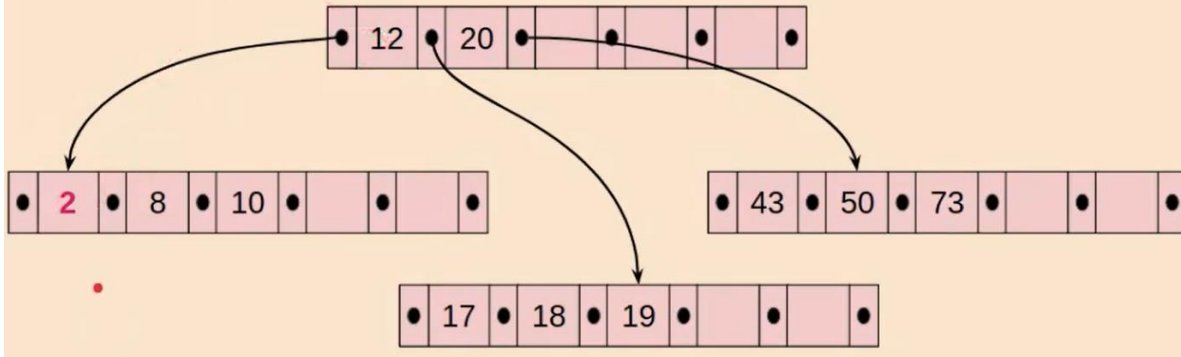
6-11-5-4-8-9-12-21-14-10-19-28-3-17-32-15-16-26-27

Árboles B - Eliminación

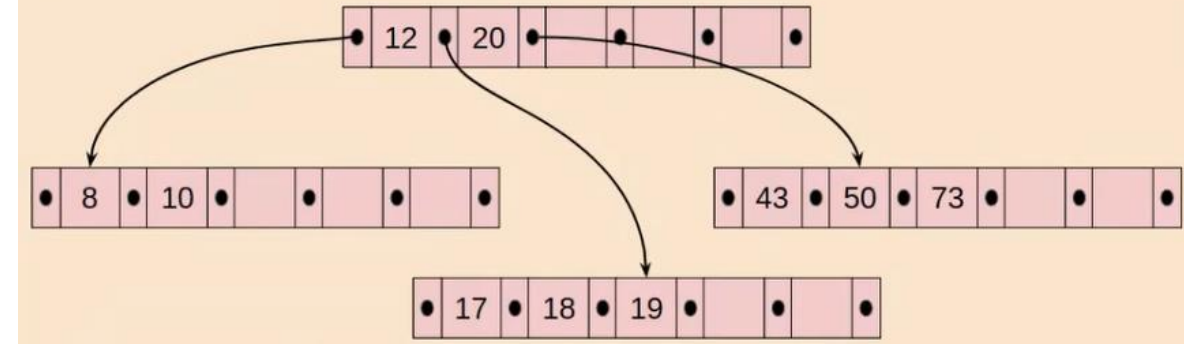
- Buscar el elemento a borrar
- Si el elemento a borrar está en un nodo hoja, antes de borrar se verifica si el nodo cumplirá con la propiedad de contener el mínimo número de claves, si es así se borra y termina el proceso. Sino se debe redistribuir bajo cualquiera de los sgtes criterios:
 - Subo la clave más chica del hermano derecho y bajo el padre que separa ambos
 - Subo la clave más grande del hermano izquierdo y bajo el padre que separa ambos
 - Si no es factible la redistribución se aplica la concatenación, donde se une al nodo afectado con algún hermano y el padre que separa ambos
- Si el elemento a borrar no se encuentra en una hoja, similar a un árbol binario de búsqueda (BST), se buscará al sustituto más apropiado. El sustituto será:
 - El último elemento de la hoja más derecha del subárbol izquierdo del nodo actual (el mayor de los menores) - intercambio el elemento con el inmediato inferior
 - El primer elemento de la hoja más izquierda del subárbol derecho del nodo actual (el menor de los mayores)- intercambio el elemento con el inmediato superior

Árbol B: Eliminación

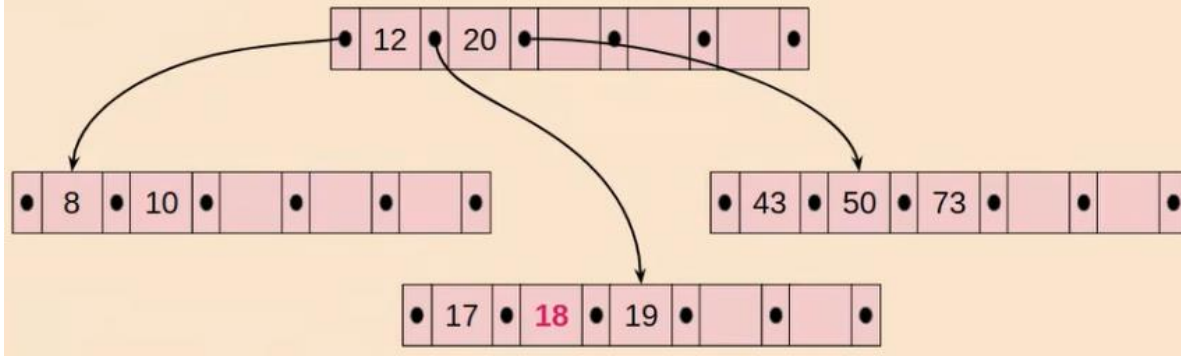
Numero a eliminar: 2



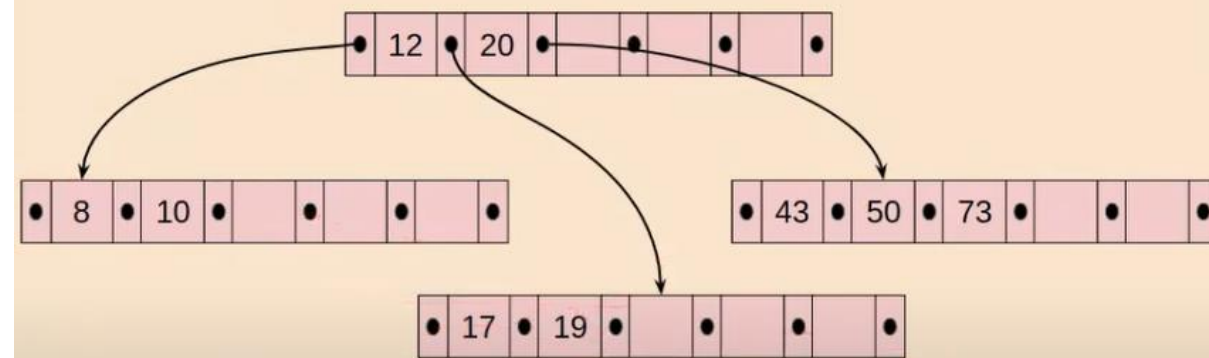
Numero a eliminar: 2



Numero a eliminar: 18

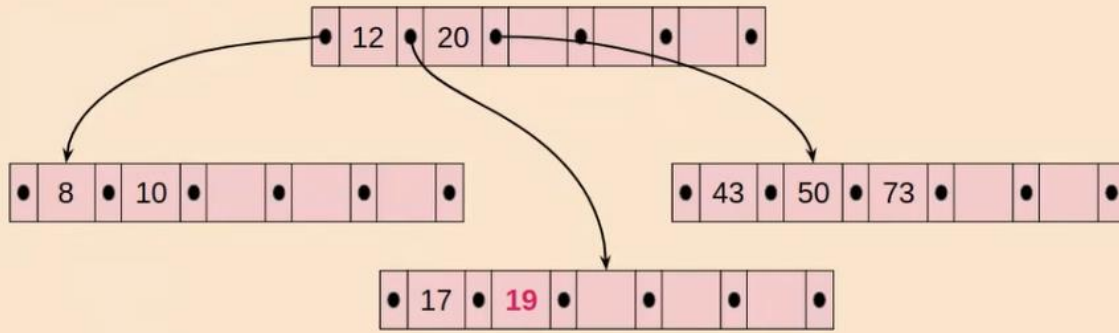


Numero a eliminar: 18

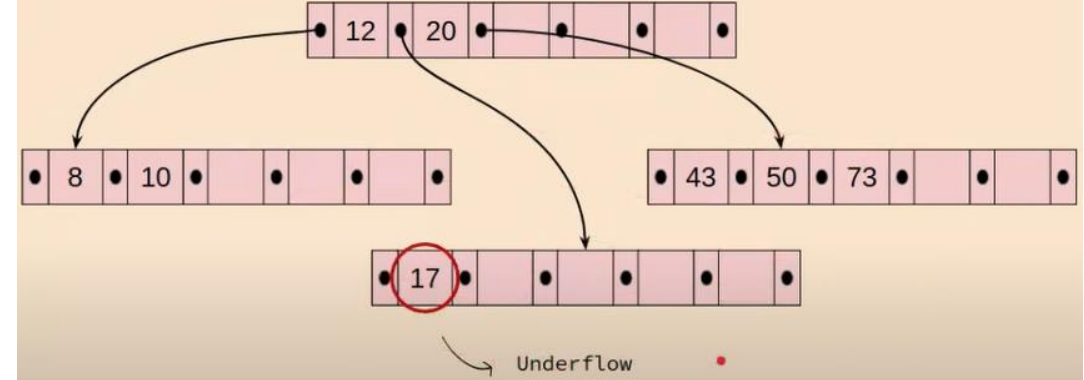


Árbol B: Eliminación

Numero a eliminar: 19



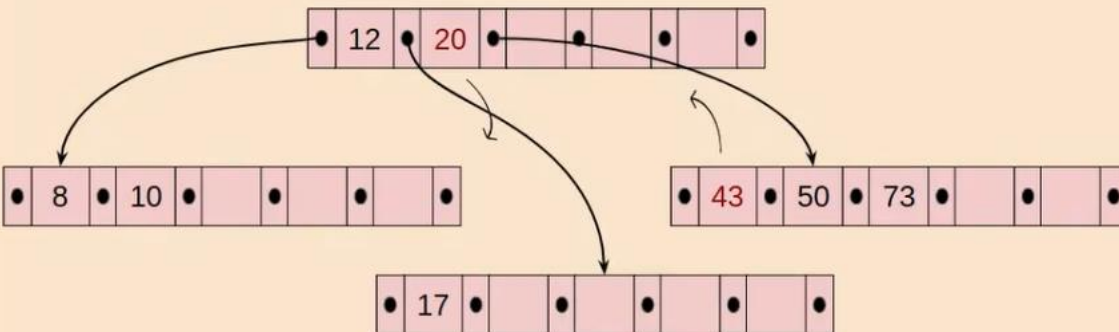
Numero a eliminar: 19



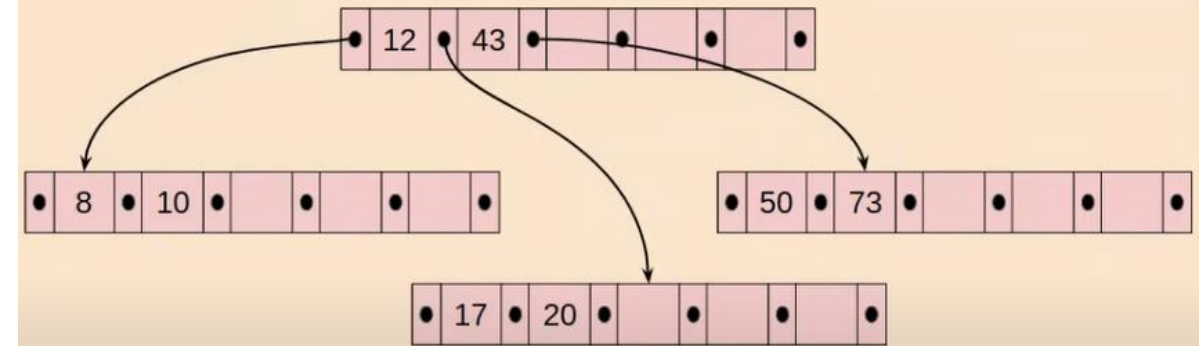
REDISTRIBUCIÓN:

- subo clave mas chica del hermano derecho y bajo el padre que separa ambos
- subo clave mas grande del hermano izquierdo y bajo el padre que separa ambos

Numero a eliminar: 19

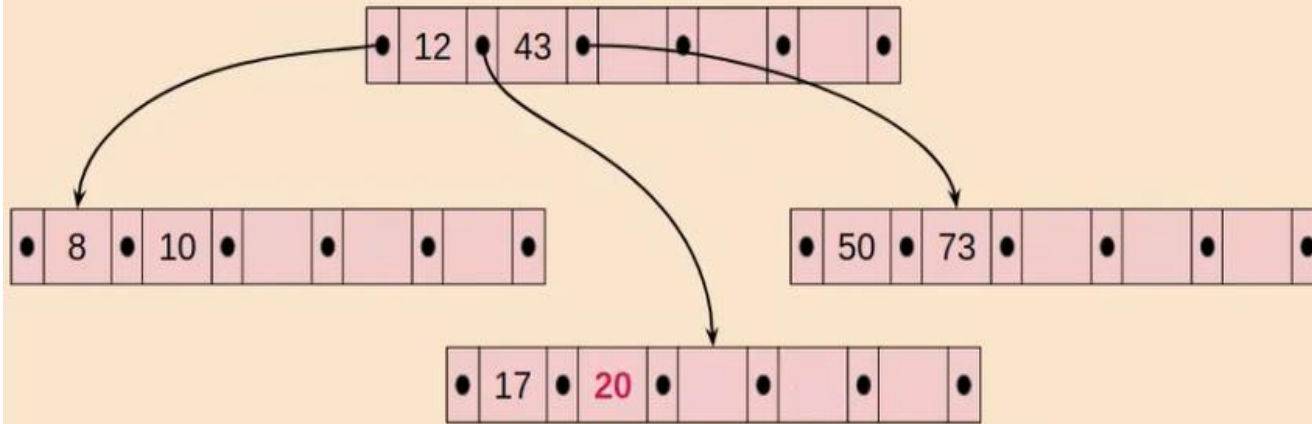


Numero a eliminar: 19

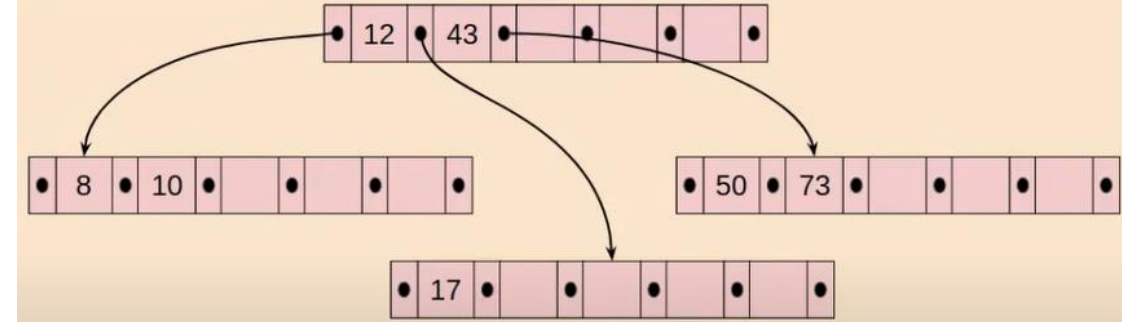


Árbol B: Eliminación

Numero a eliminar: 20



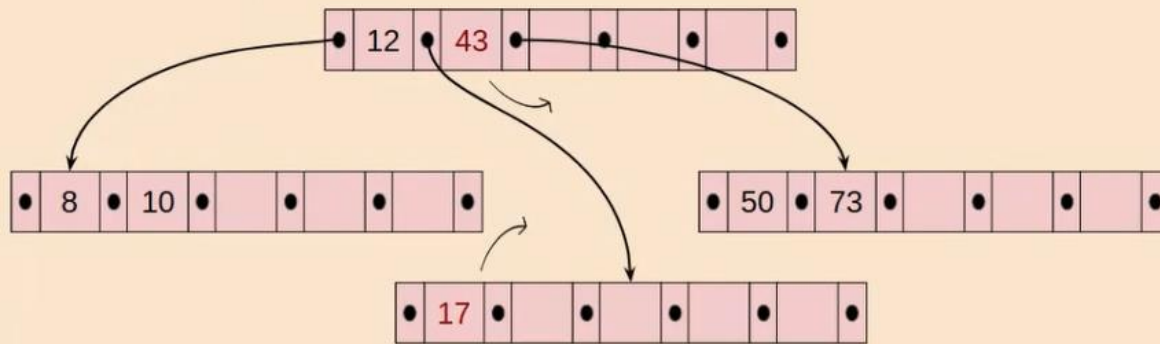
Numero a eliminar: 20



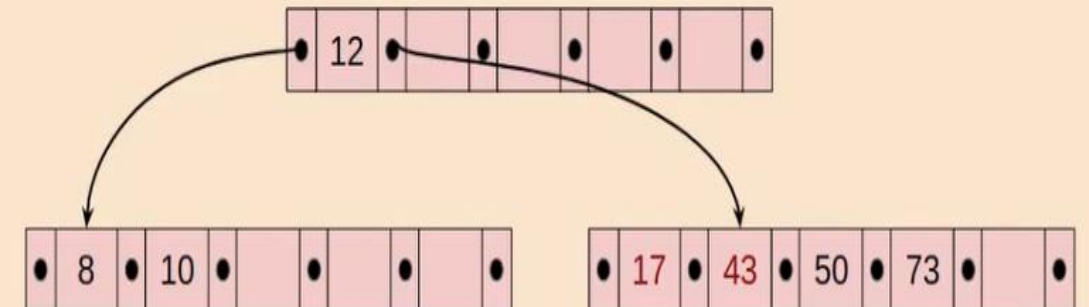
CONCATENACIÓN:

- uno al nodo afectado con algún hermano y el padre que separa ambos

Numero a eliminar: 20

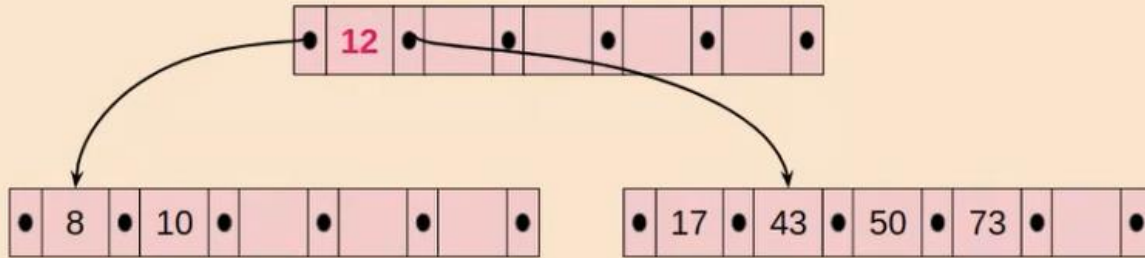


Numero a eliminar: 20

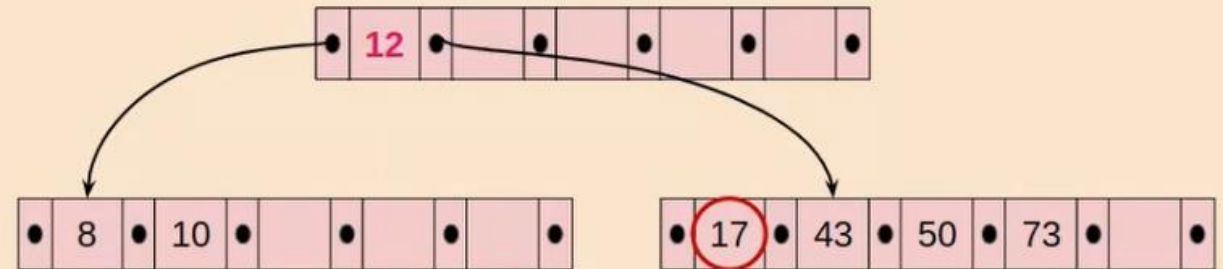


Árbol B: Eliminación

Numero a eliminar: 12

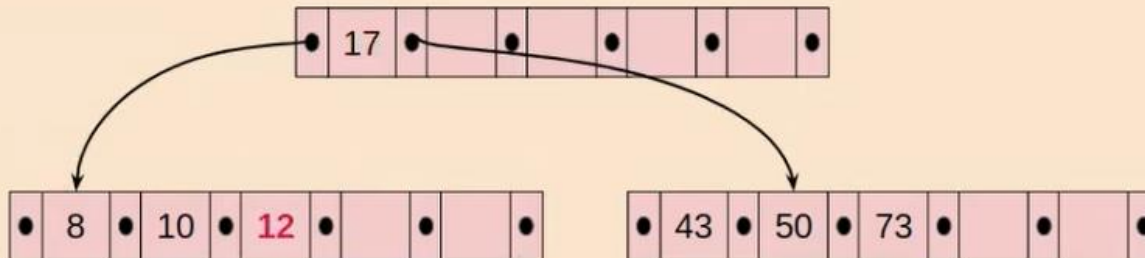


Numero a eliminar: 12

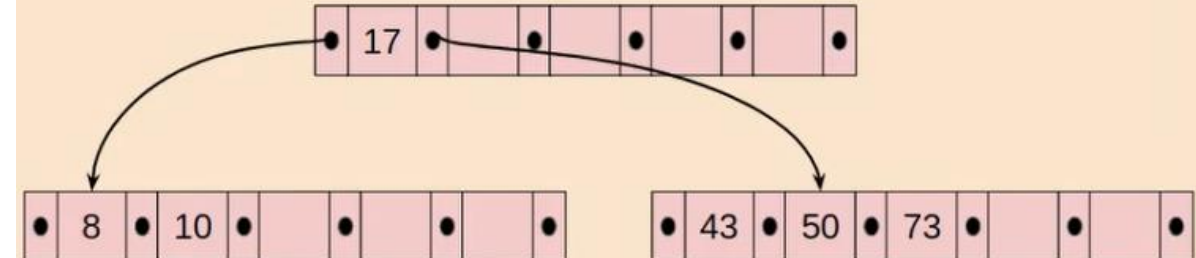


- intercambio el elemento con el inmediato superior
- intercambio el elemento con el inmediato inferior

Numero a eliminar: 12

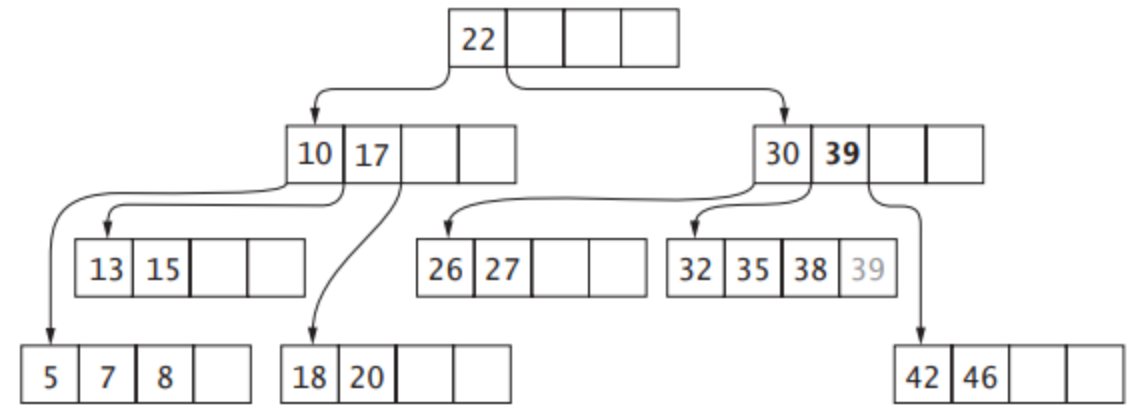
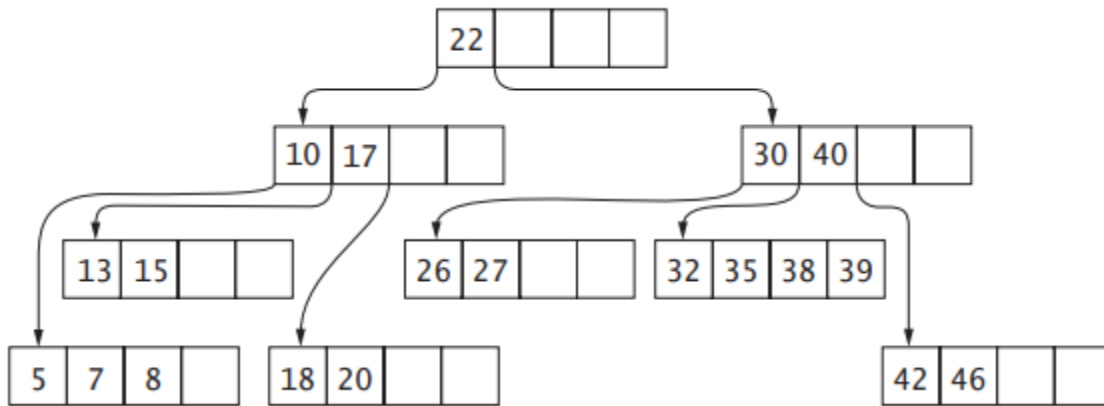


Numero a eliminar: 12



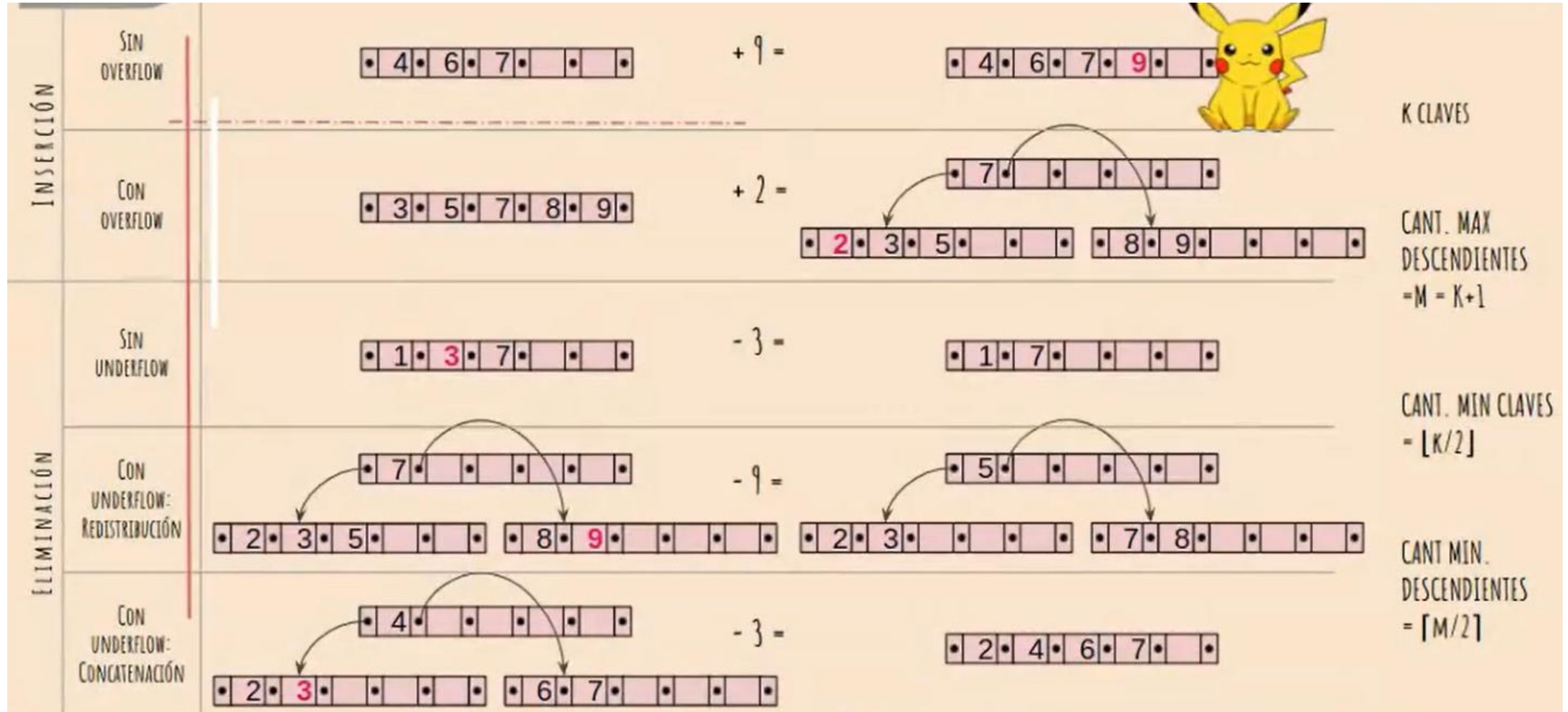
Árbol B: Eliminación

Remover 40



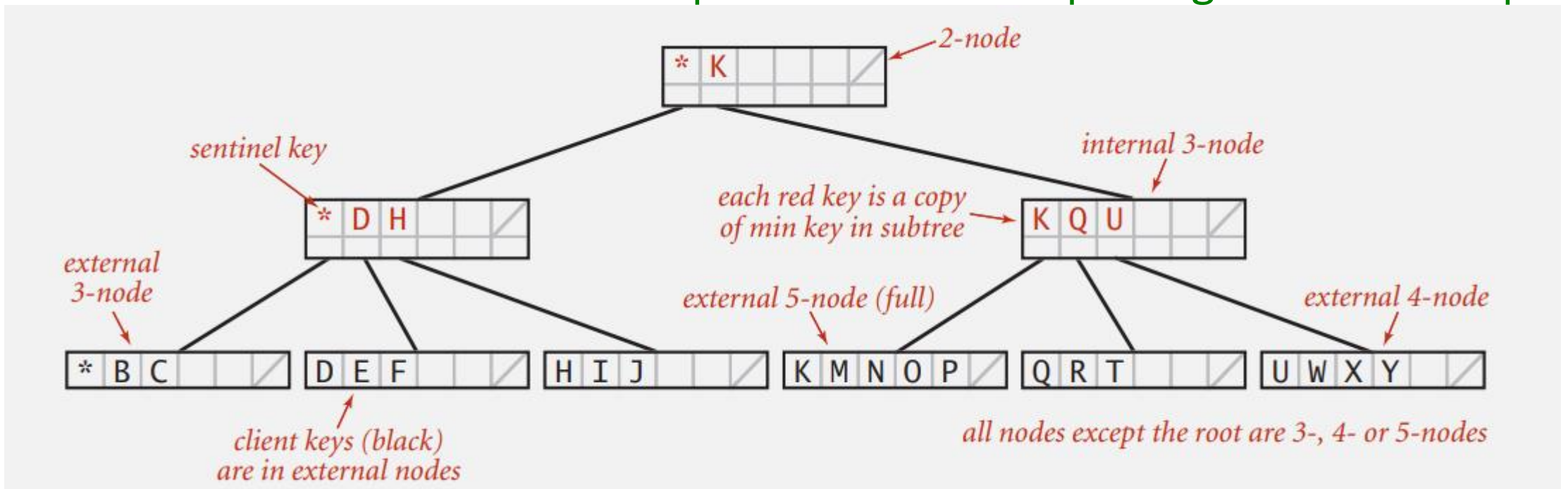
- intercambio el elemento con el inmediato superior
- intercambio el elemento con el inmediato inferior

Árbol B



Árboles B (Bayer-McCreight, 1972)

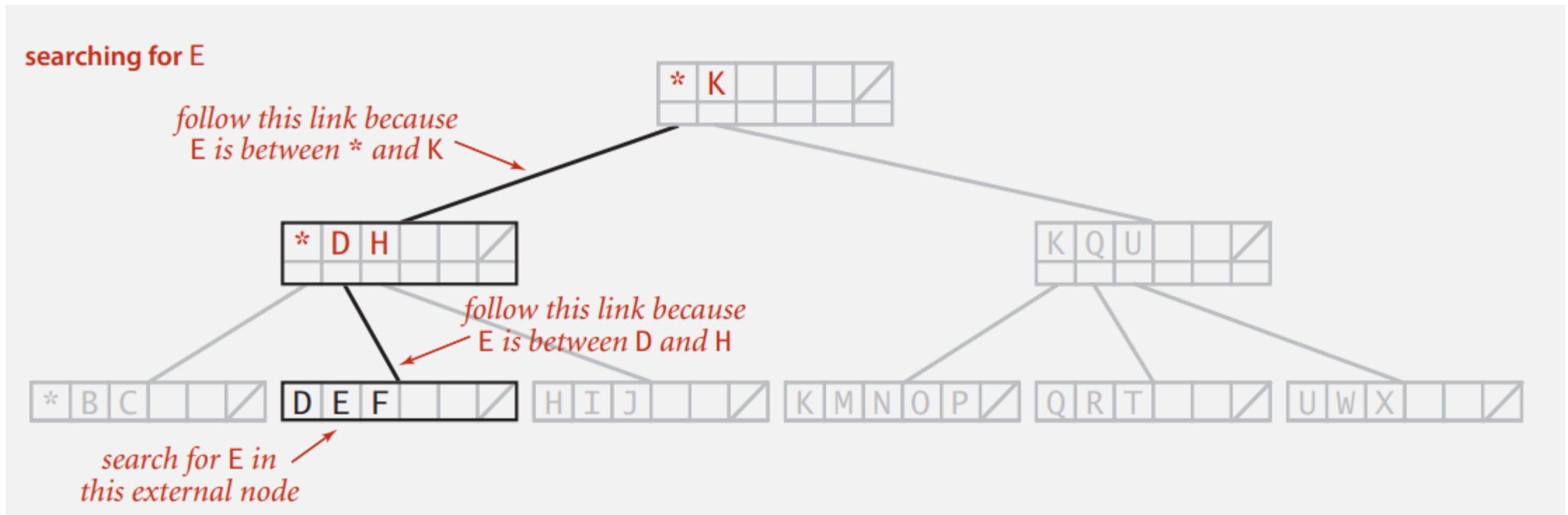
- Árboles - B: Generalice los árboles 2-3 permitiendo hasta $M-1$ claves por nodo
elija M lo más grande posible para que M enlaces quepan en una página, por ejemplo, $M = 1024$
- Al menos 2 hijos en la raíz
- Al menos $M/2$ hijos en otros nodos
- Los nodos internos contiene claves de clientes
- Los nodos internos contienen copias de claves para guiar la búsqueda



Anatomía de un árbol B ($M=6$)

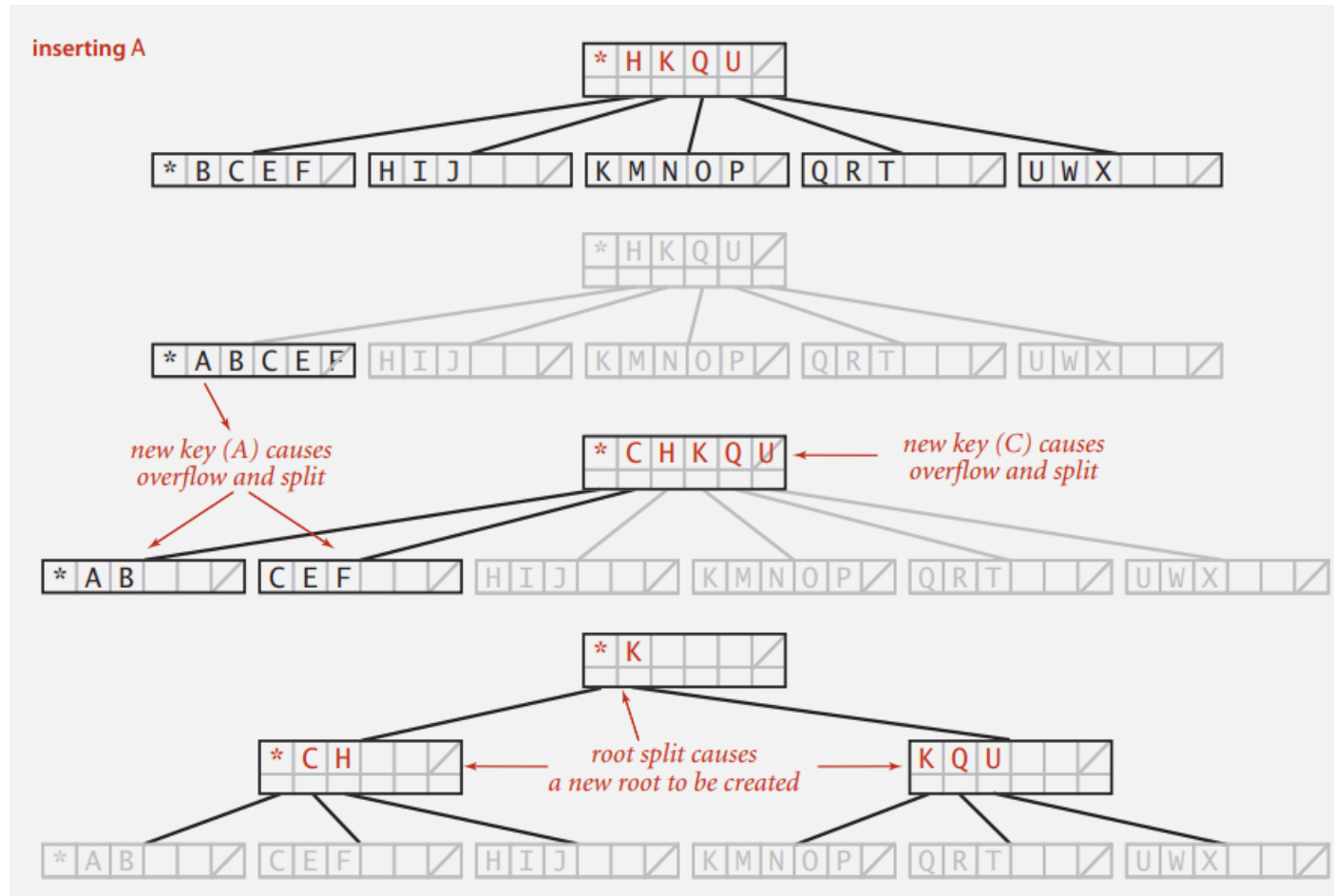
Búsqueda en un árbol B

- Comience en la raíz
- Encuentre el intervalo para la búsqueda de la clave y tome el enlace correspondiente
- La búsqueda termina en un nodo externo




Árboles B Inserción

- Buscar donde iría la nueva clave
- Insertar en la parte inferior (Hoja)
- Dividir los nodos con M pares clave-enlace hacia arriba del árbol



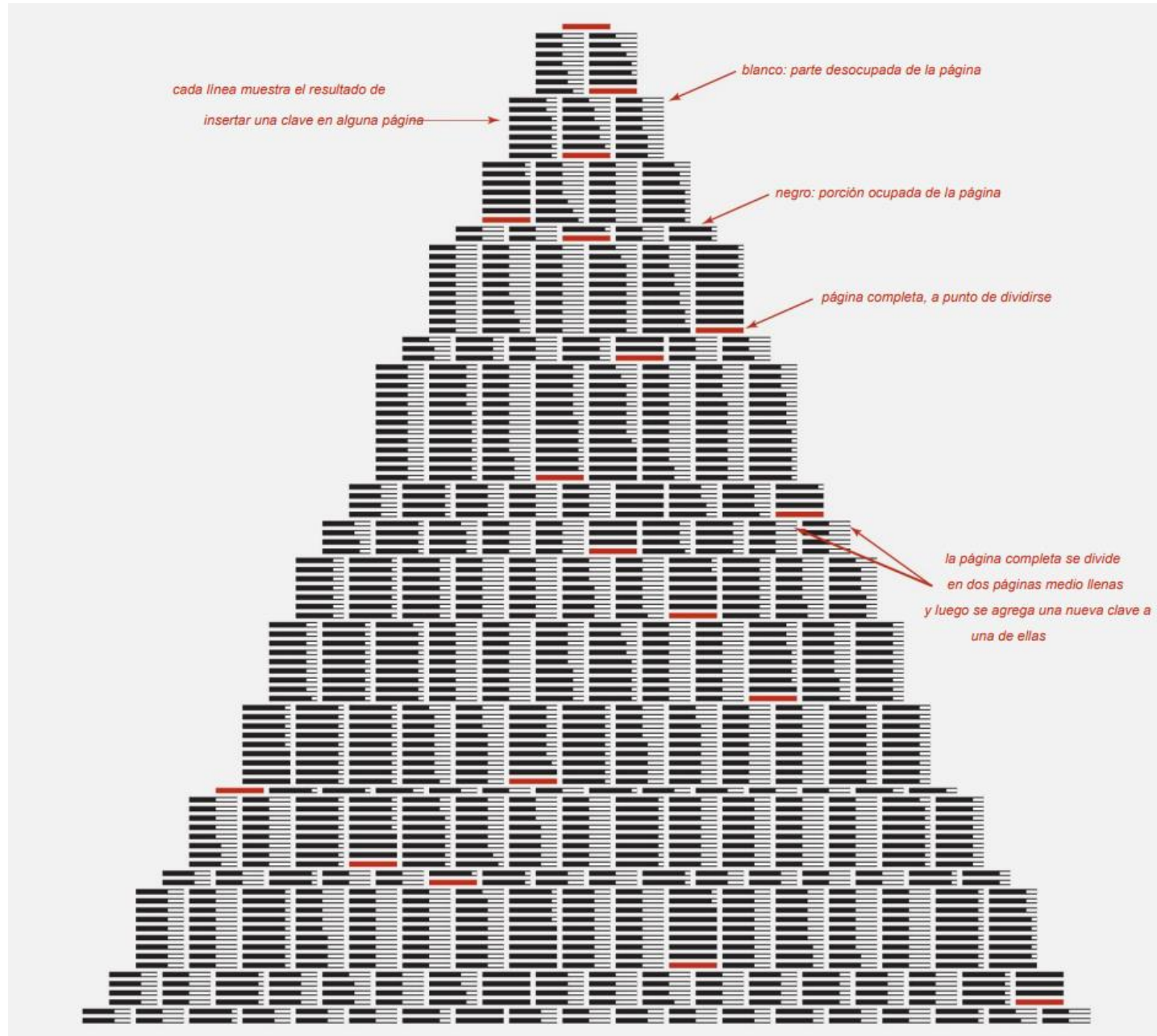
Insertar una nueva clave en un árbol B

Balance en un Árbol B

- **Proposición:** Una búsqueda o una inserción en un árbol B de orden M con N claves requiere sondeos entre $\log_{M-1}N$ y $\log_{M/2}N$
- **Pf:** Todos los nodos (además de la raíz) tiene entre $M/2$ y $M-1$ enlaces
- **En la práctica:** Número de sondeos es como máximo 4 

$M = 1024; N = 62 \text{ billones}$
 $\log_{M/2} N \leq 4$
- **Optimización:** Mantenga siempre la página raíz en memoria
- Los árboles B (y variantes) se utilizan ampliamente para sistemas de archivos y bases de datos.
- Windows: NTFS.
- Mac: HFS, HFS+.
- Linux: ReiserFS, XFS, Ext3FS, JFS.
- Bases de datos: ORACLE, DB2, INGRES, SQL, PostgreSQL

Construyendo un gran árbol B



Ejercicio

- Dada la secuencia de claves enteras:
190,57,89,90,121,170,35,48, 91,22,126,132 y 80;
Dibuje el árbol B de orden 5 cuya raíz es R, que se corresponde con dichas claves.
- En el árbol R del problema anterior, elimine la clave 91 y dibuje el árbol resultante. Elimine ahora la clave 48. Dibuje el árbol resultante, ¿ha habido reducción en el número de nodos?

Compresión de Texto

Si el almacenamiento o el ancho de banda es escaso, ¿cómo podemos almacenar y transmitir datos de manera más eficiente?

La compresión es más útil para archivos grandes (por ejemplo, audio, gráficos, video y datos científicos)

Los archivos de texto suelen ser bastante pequeños, pero como ilustración, ¿podemos utilizar menos de 16 bits por carácter sin perder información?

Las técnicas de compresión de texto sin pérdida incluyen:

- Codificación por palabra clave (Keyword encoding)

- Codificación por longitud de secuencia (Run-length encoding)

- Códigos de Huffman (Huffman encoding)

Codificación por palabra Clave

Un método bastante sencillo de compresión de texto que reemplaza los patrones de texto utilizados frecuentemente con un único carácter especial, como por ejemplo:

WORD	SYMBOL
as	^
the	~
and	+
that	\$
must	&
well	%
these	#

Para descomprimir el documento, invierte el proceso: Sustituye los caracteres individuales por la palabra completa apropiada.

Codificación por palabra Clave

Dado el siguiente párrafo,

We hold these truths to be self-evident, that all men are created equal, that they are endowed by their Creator with certain unalienable Rights, that among these are Life, Liberty and the pursuit of Happiness. — That to secure these rights, Governments are instituted among Men, deriving their just powers from the consent of the governed, — That whenever any Form of Government becomes destructive of these ends, it is the Right of the People to alter or to abolish it, and to institute new Government, laying its foundation on such principles and organizing its powers in such form, as to them shall seem most likely to effect their Safety and Happiness.

Codificación por palabra Clave

El párrafo codificado es

We hold # truths to be self-evident, \$ all men are created equal, \$ ~y are endowed by ~ir Creator with certain unalienable Rights, \$ among # are Life, Liberty + ~ pursuit of Happiness. — \$ to secure # rights, Governments are instituted among Men, deriving ~ir just powers from ~ consent of ~ governed, — \$ whenever any Form of Government becomes destructive of # ends, it is ~ Right of ~ People to alter or to abolish it, + to institute new Government, laying its foundation on such principles + organizing its powers in such form, ^ to ~m shall seem most likely to effect ~ir Safety + Happiness.

Codificación por palabra Clave

¿Qué ahorramos?

Párrafo original

656 caracteres

Parámetro codificado

596 caracteres

Caracteres ahorrados

60 caracteres

Índice de compresión

$596/656 = 0,9085$

¿Podríamos usar esta tabla de sustitución para todo el texto?

Codificación por longitud de secuencia

En algunos tipos de archivos de datos, un único valor puede repetirse una y otra vez en una larga secuencia

Reemplazar una secuencia repetida con:

- un flag

- El valor repetido

- El número de repeticiones

*n8

- * es el flag

- n es el valor repetido

- 8 es el número de veces que n se repite

Codificación por longitud de secuencia

Texto Original

bbbbbbbbbjjjklqqqqqq+++++

Texto Codificado

*b8jjjkl*q6*+5 (*¿Por qué no se codifica j ? l ?*)

El índice de compresión es 15/25 or .6

Texto Codificado

*x4*p4l*k7

Texto Original

xxxxppppplkkkkkkk

Este tipo de repetición no ocurre en un texto en español o inglés; ¿Puede pensar en una situación en la que podría ocurrir?

Código Huffman

En un texto, los caracteres 'X' y 'z' ocurren con mucha menos frecuencia que 'e' y el carácter de espacio, por ejemplo.

¿Qué pasa si usamos menos bits para representar caracteres comunes a cambio de usar más bits para representar caracteres poco comunes?

Esta es la idea detrás de los códigos de prefijo, incluyendo los **códigos de Huffman**

Código Huffman

La codificación de Huffman es un ejemplo de codificación de prefijo: ninguna cadena de bits utilizada para representar un carácter es el prefijo de cualquier otra cadena de bits utilizada para representar un carácter.

Para decodificar :

Buscar la correspondencia o coincidencia de izquierda a derecha, bit a bit
Registrar la letra cuando se encuentra una coincidencia
Empezar de nuevo donde se detuvo, yendo de izquierda a derecha

Código Huffman

Huffman Code	Character
00	A
01	E
100	L
110	O
111	R
1010	B
1011	D

“ballboard” sería:

10100010
01001010
11000111
1011xxxx

Índice de compresión
 $4 \text{ bytes} / 18 \text{ bytes} = 0.222$
asumiendo 16-bit Unicode

Intente con “roadbed”

Nota: sólo se muestra la parte del código necesario para codificar "ballboard" y "roadbed". En el código completo, cada carácter tendría una codificación, y los caracteres más comunes tendrían las codificaciones más cortas.

Código Huffman

La técnica para determinar los códigos Huffman que representan los caracteres garantiza el prefijo apropiado

Dos tipos de código dependientes de donde viene la frecuencia:

General, basado en la frecuencia de uso de las letras en un idioma (Inglés, Español,....)

Especializado, basado en la frecuencia de uso de algún carácter en el propio texto, o tipos específicos de texto.

Código Huffman



Ejercicios!

Huffman Code	Character
00	A
01	E
100	L
110	O
111	R
1010	B
1011	D

Decodificar
1011111001010

HUFFMAN CODE	CHARACTER
00	A
11	E
010	T
0110	C
0111	L
1000	S
1011	R
10010	O
10011	I
101000	N
101001	F
101010	H
101011	D

Decodificar

- a. 1101110001011
- b. 0110101010100101011111000
- c. 101001001010000100010000101 00110110
- d. 101000100101010001000111010 00100011

Codificación Huffman

- Propuesto por el Dr. David A. Huffman
 - Clase de posgrado en 1951 en el MIT con Robert Fano
 - Trabajo final: demuestra los bits mínimos necesarios para la codificación binaria de datos.
 - *Un método para la construcción de códigos de redundancia mínima*
- Aplicable a muchas formas de transmisión de datos.
 - Nuestro ejemplo: archivos de texto
 - Todavía se utiliza en máquinas de fax, codificación de mp3, otros

El algoritmo básico

- La codificación de Huffman es una forma de codificación estadística.
 - No todos los caracteres aparecen con la misma frecuencia en los archivos de texto típicos. (también puede ser cierto al leer bytes sin formato)
 - Sin embargo, en ASCII a todos los caracteres se les asigna la misma cantidad de espacio.
 - 1 carácter = 1 byte, ya sea **e** o **x**
 - **codificación de ancho fijo**
-

El algoritmo básico

- ¿Algún ahorro al adaptar los códigos a la frecuencia del carácter?
- Las longitudes de las palabras de código ya no son fijas como ASCII o Unicode
- La longitud de las palabras de código varía y será más corta para los caracteres utilizados con más frecuencia.
- Los ejemplos utilizan caracteres para mayor claridad, pero en realidad solo leen bytes sin procesar del archivo.

El algoritmo básico

1. Escanee el archivo que desea comprimir y determine la frecuencia de todos los valores.
 2. Ordene o priorice valores según la frecuencia en el archivo.
 3. Cree un árbol de código de Huffman basado en valores priorizados.
 4. Realice un recorrido del árbol para determinar nuevos códigos para los valores.
 5. Escanee el archivo nuevamente para crear un archivo nuevo usando los nuevos códigos Huffman.
-

Construyendo un árbol

Escanea el texto original

- Considere el siguiente texto breve

Eerie eyes seen near lake.

(Ojos espeluznantes vistos cerca del lago)

- Determinar la frecuencia de todos los números (valores o en este caso caracteres) en el texto.

Construyendo un árbol

Escanea el texto original

Eerie eyes seen near lake.

(Ojos espeluznantes vistos cerca del lago.)

- ¿Qué caracteres están presentes?

E e r i space y s n a l k .

Construyendo un árbol

Escanea el texto original

Eerie eyes seen near lake.

(Ojos espeluznantes vistos cerca del lago.)

- ¿Cuál es la frecuencia de cada carácter en el texto?

Char	Freq.	Char	Freq.	Char	Freq.
E	1	y	1	k	1
e	8	s	2	.	1
r	2	n	2		
i	1	a	2		
space	4	l	1		

Construyendo un árbol

Priorizar valores desde un archivo

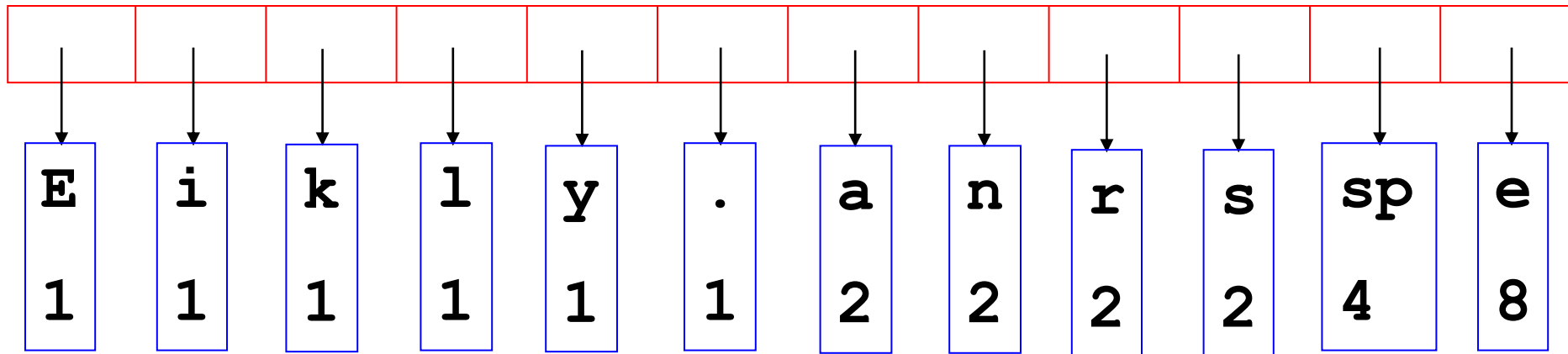
- Cree nodos de árbol binario con un valor y la frecuencia para cada valor.
- Colocar nodos en una cola de prioridad
 - Cuanto **menor sea** la frecuencia, **mayor** será la prioridad en la cola

Construyendo un árbol

- La cola después de poner en cola todos los nodos

front

back

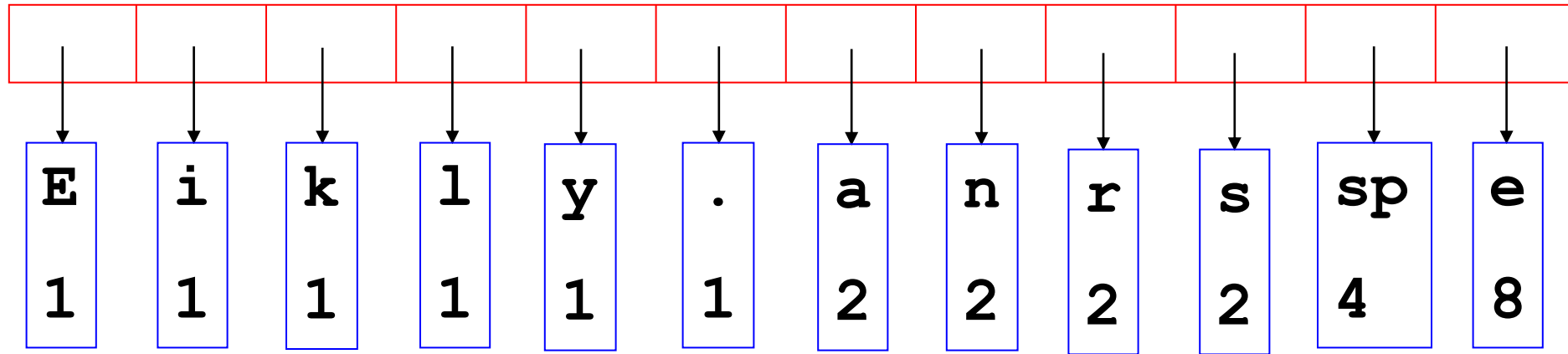


- Los punteros nulos no se muestran
- sp = space (espacio)

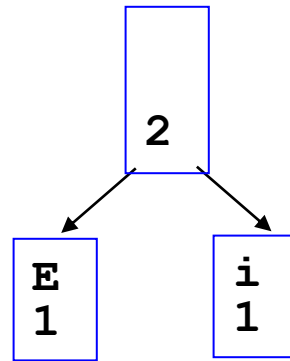
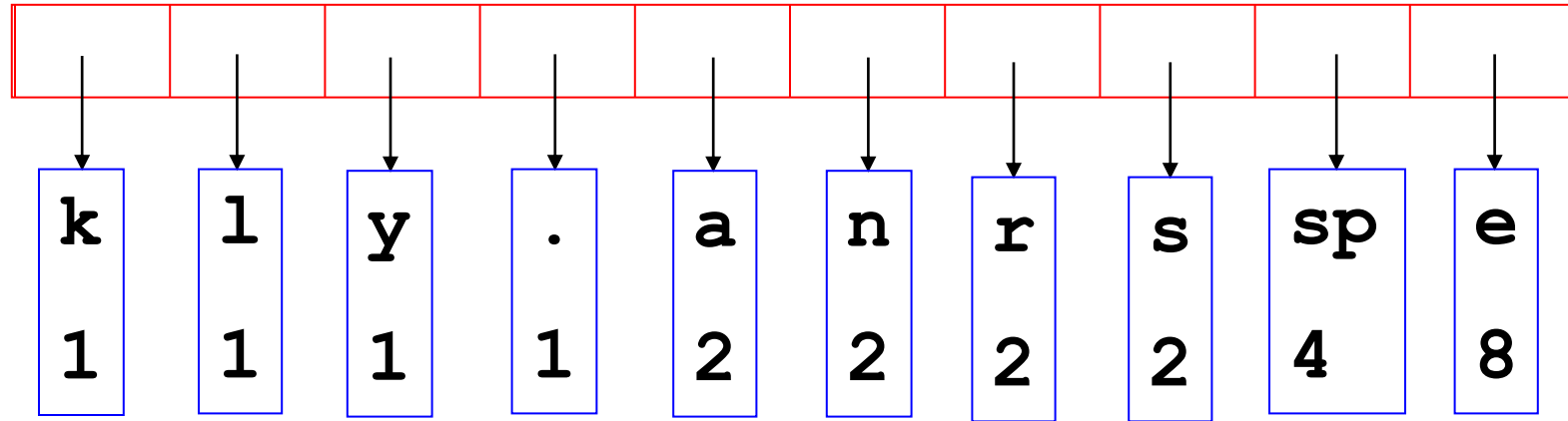
Construyendo un árbol

- Mientras que la cola de prioridad contiene dos o más nodos
 - Crear nuevo nodo
 - Quitar el nodo de la cola y convertirlo en hijo
 - Quitar de la cola el siguiente nodo y convertirlo como hijo
 - La frecuencia del nuevo nodo es igual a la suma de la frecuencia de los hijos izquierdo y derecho
 - El nuevo nodo no contiene valor
 - Volver a poner el nuevo nodo en la cola de prioridad

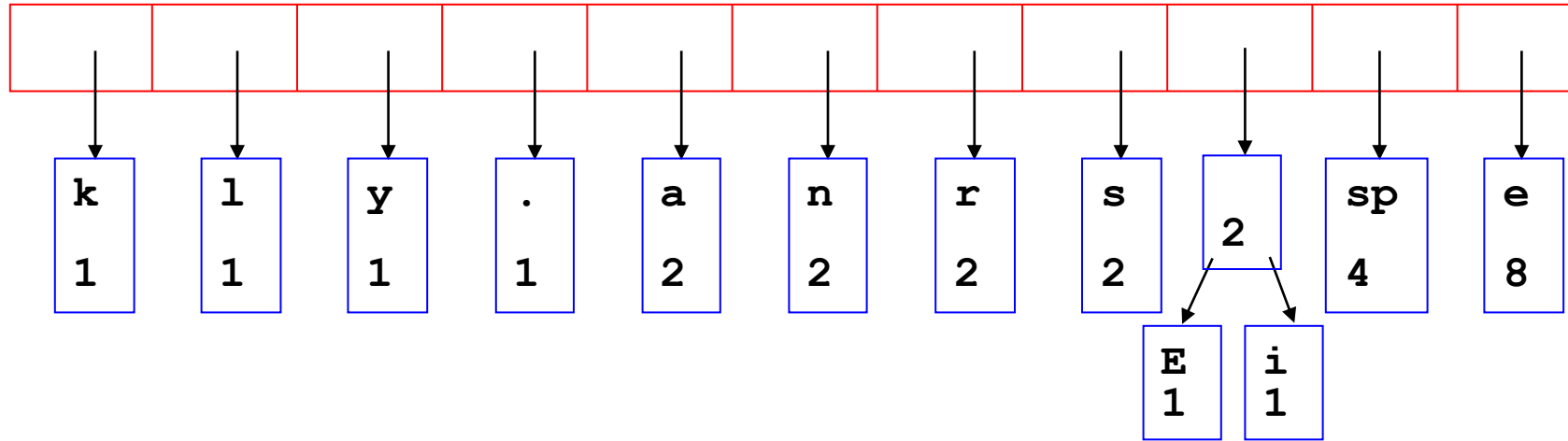
Construyendo un árbol



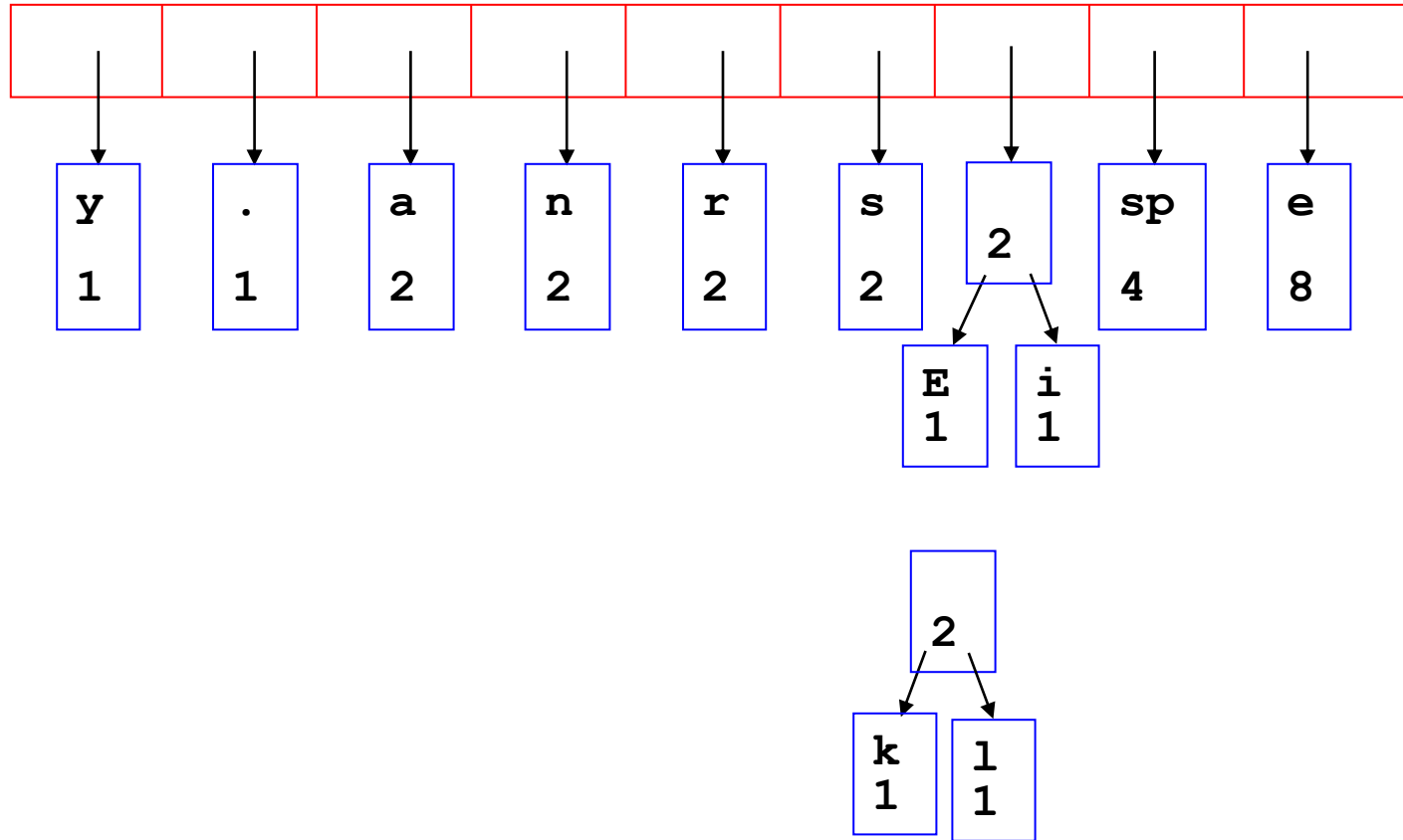
Construyendo un árbol



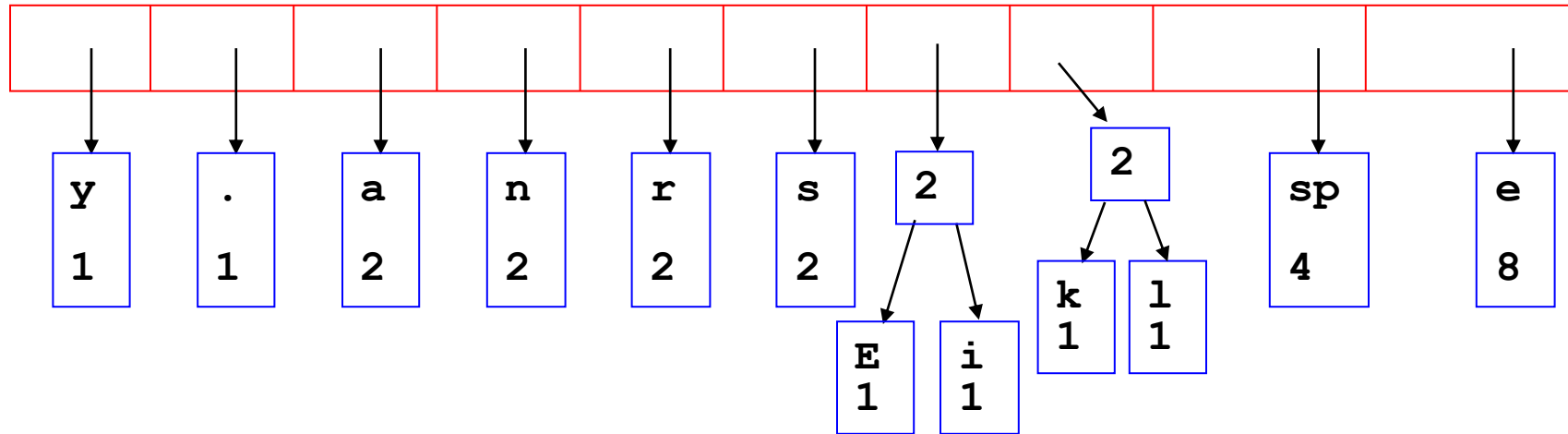
Construyendo un árbol



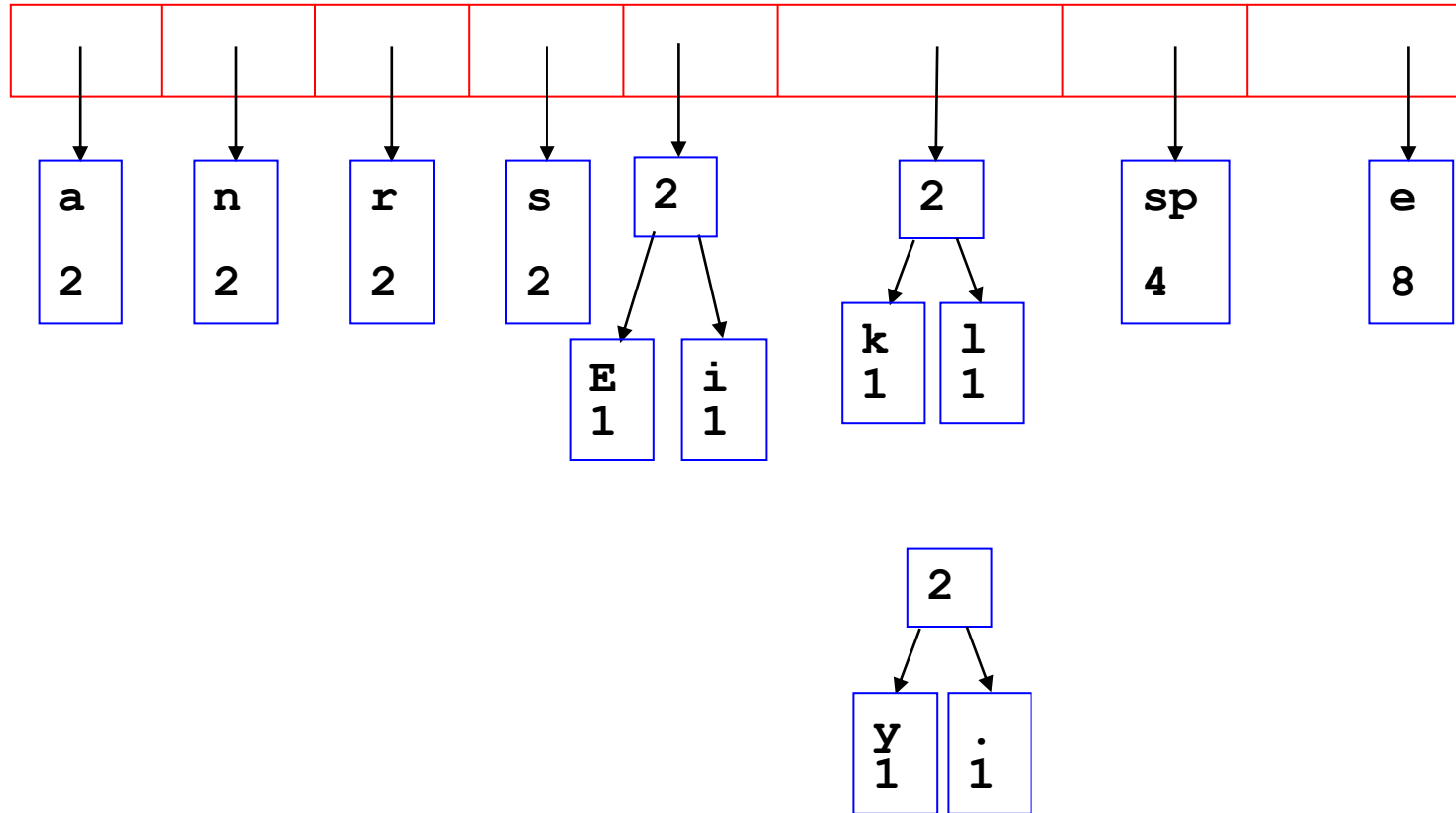
Construyendo un árbol



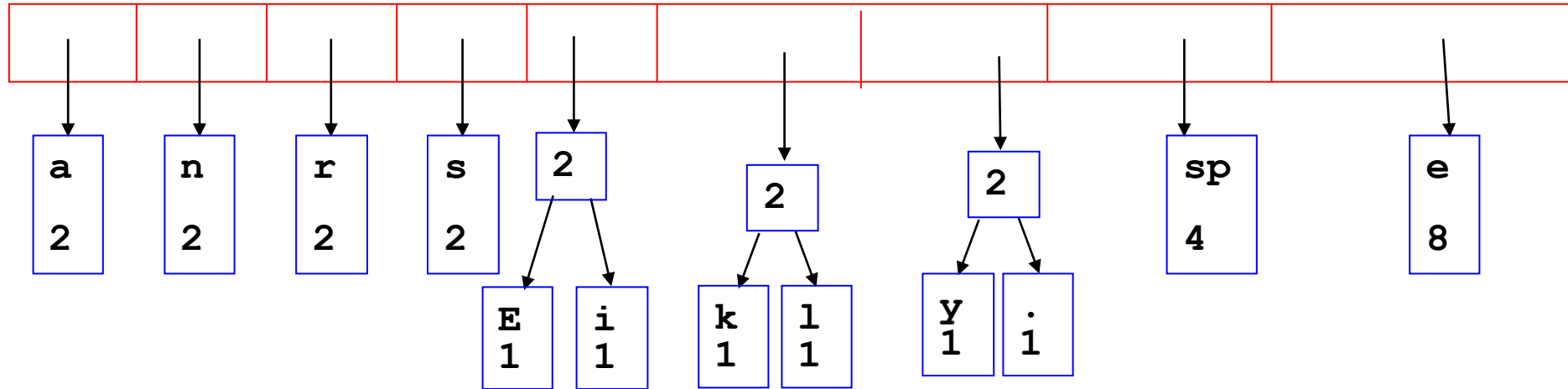
Construyendo un árbol



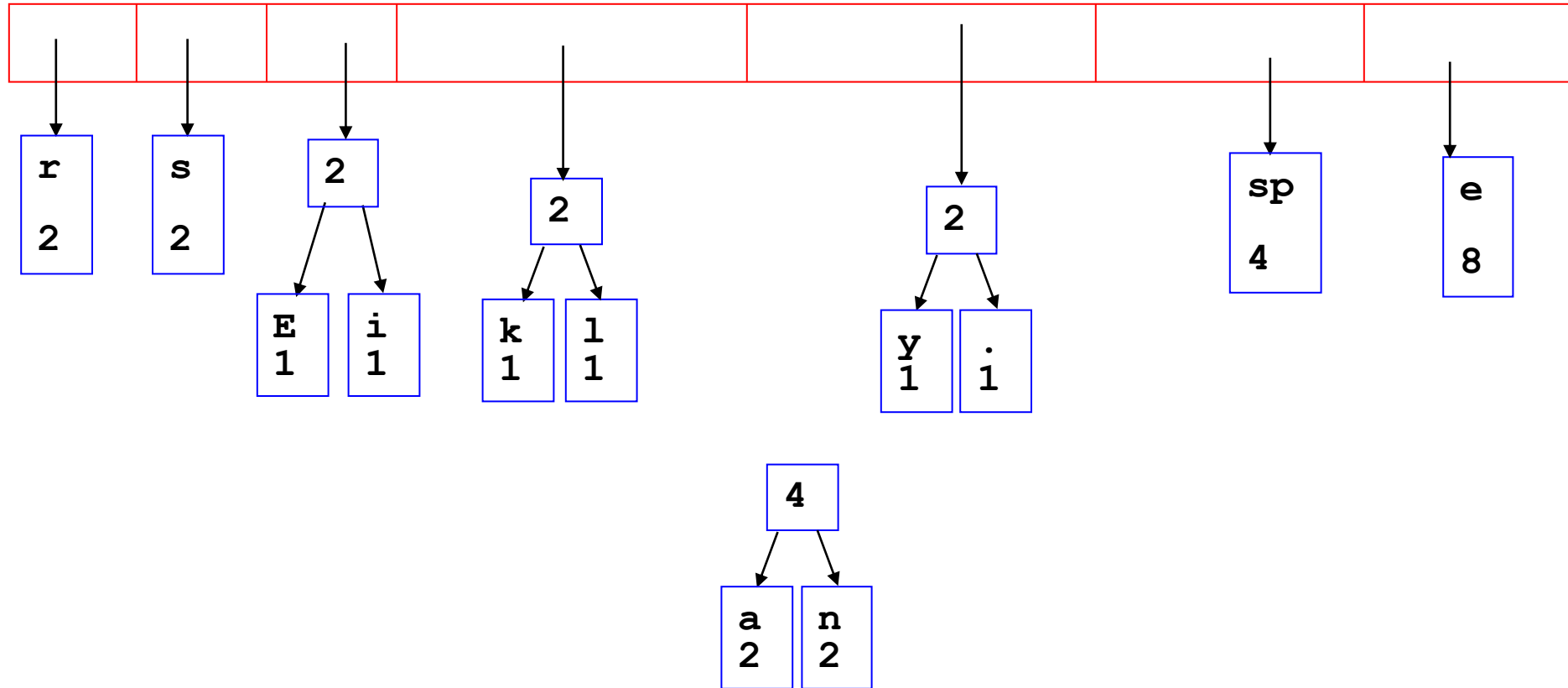
Construyendo un árbol



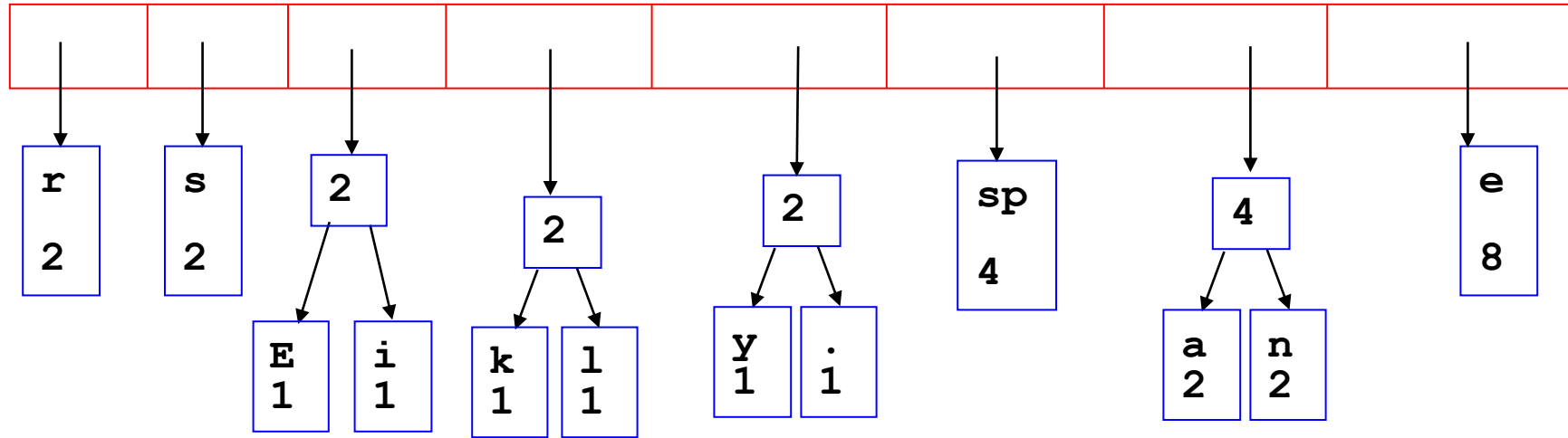
Construyendo un árbol



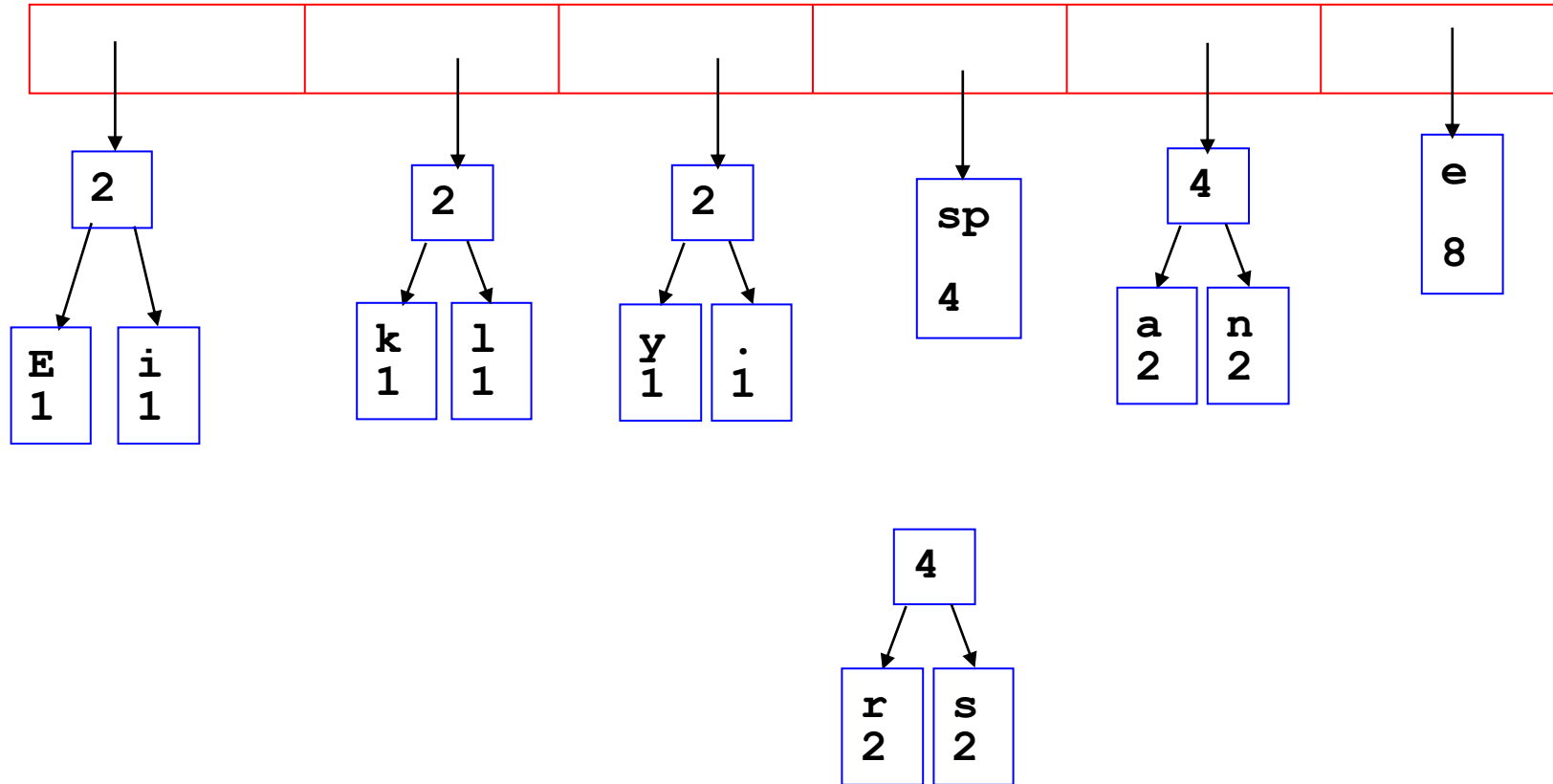
Construyendo un árbol



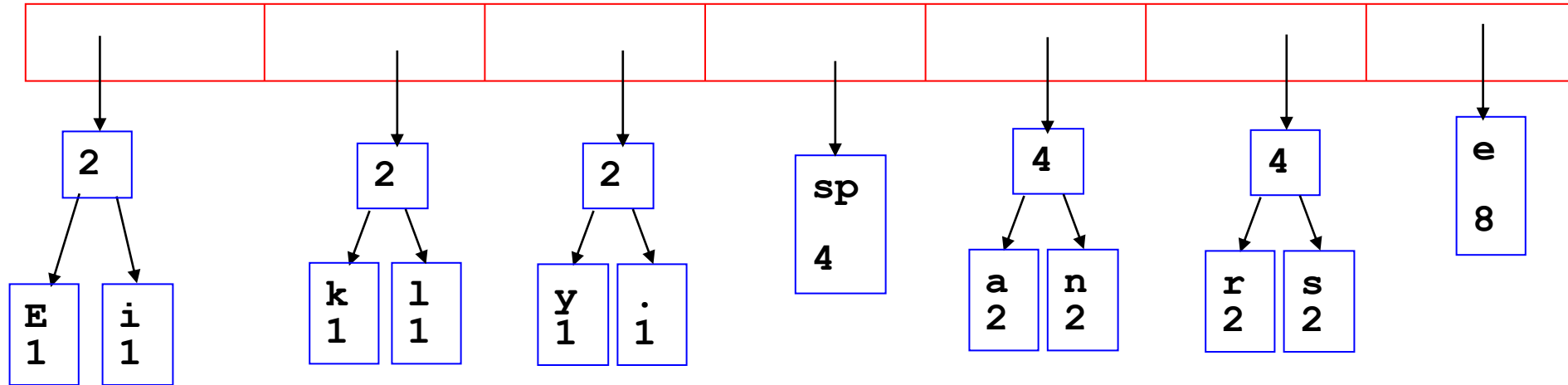
Construyendo un árbol



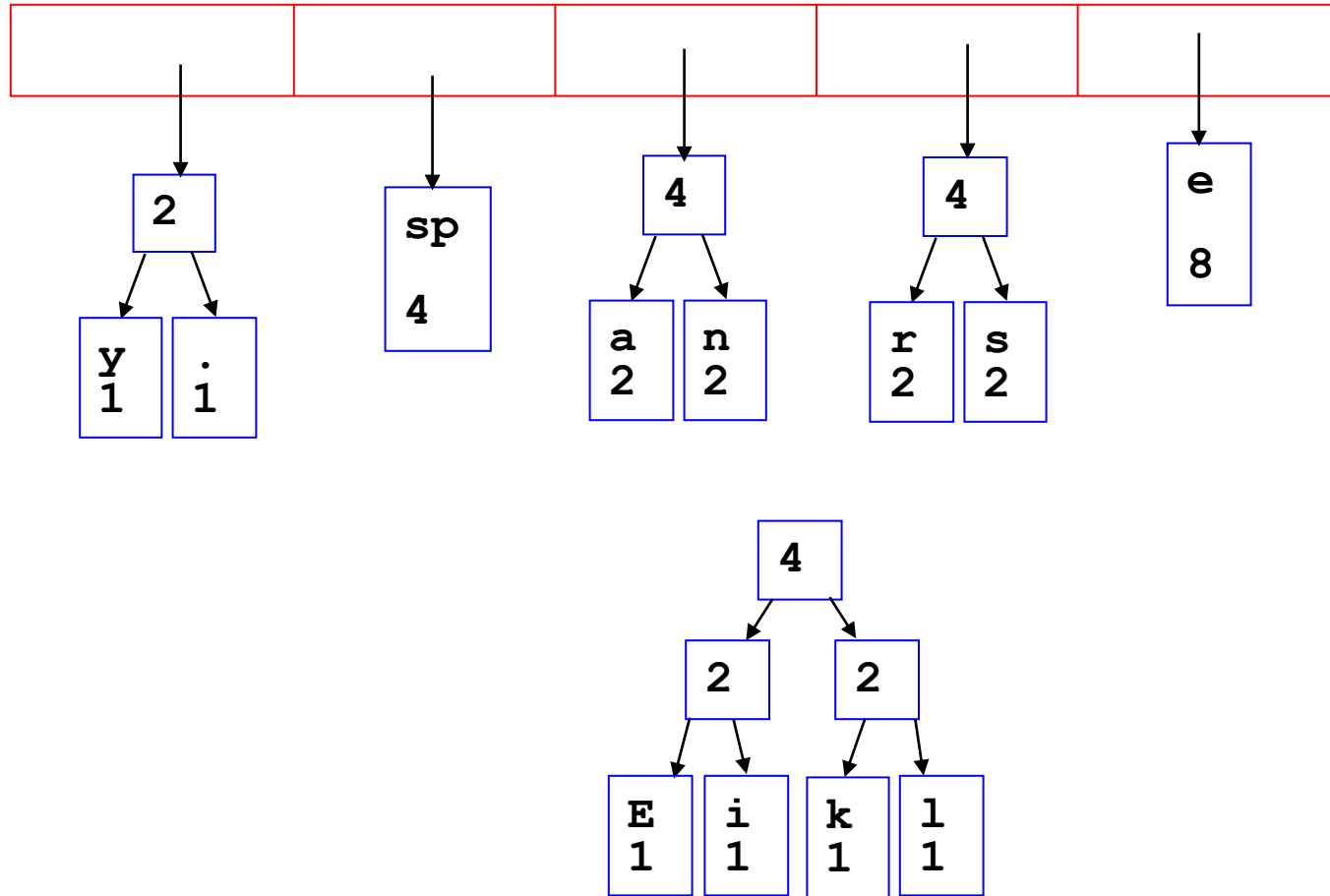
Construyendo un árbol



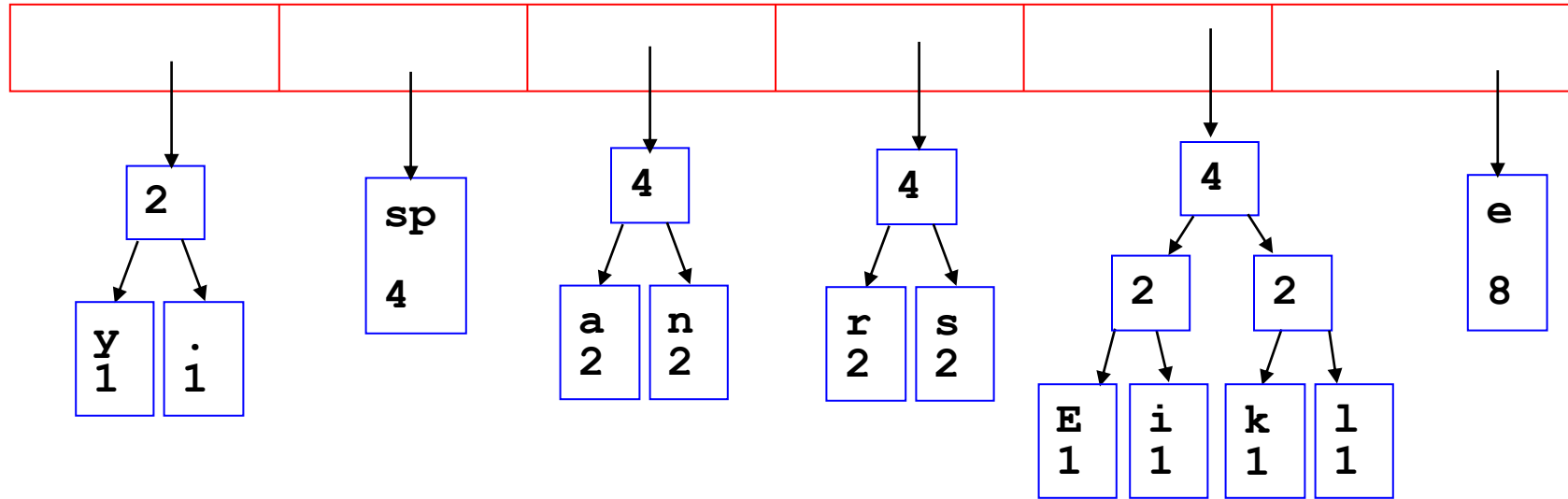
Construyendo un árbol



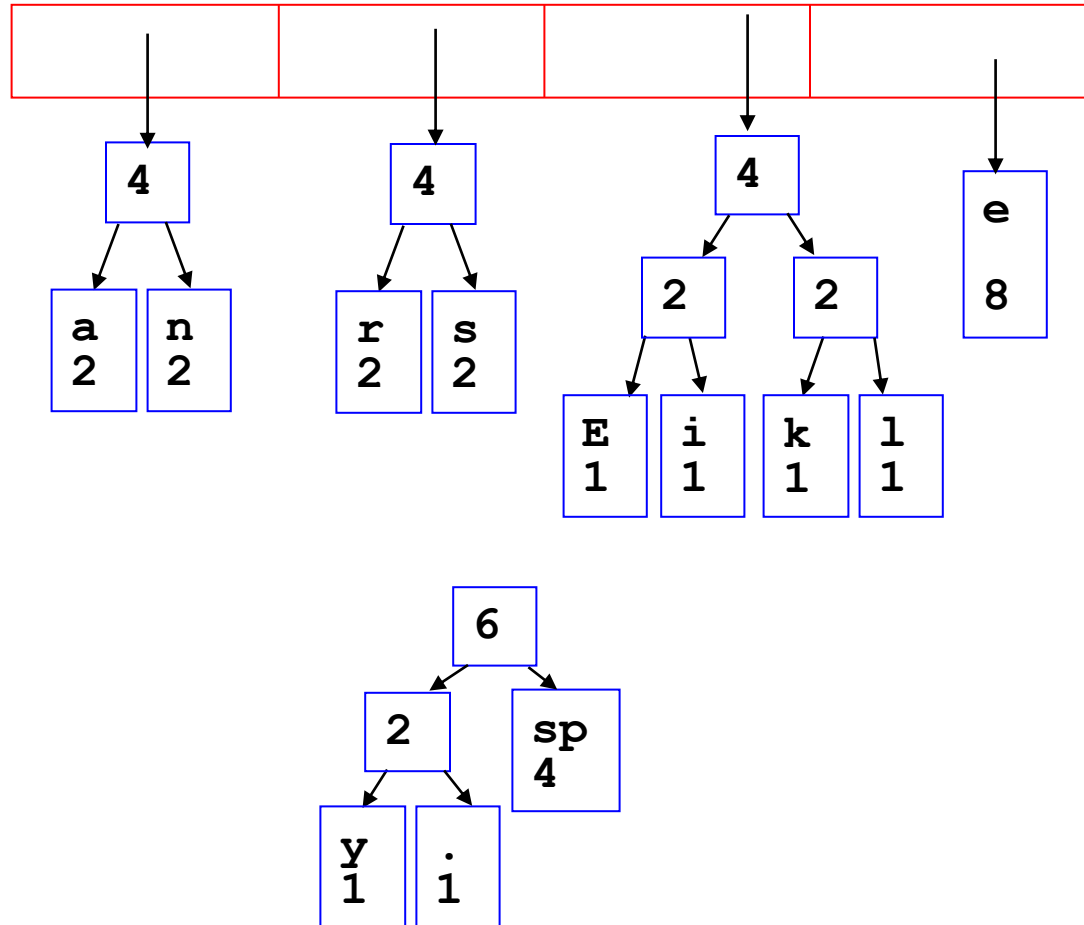
Construyendo un árbol



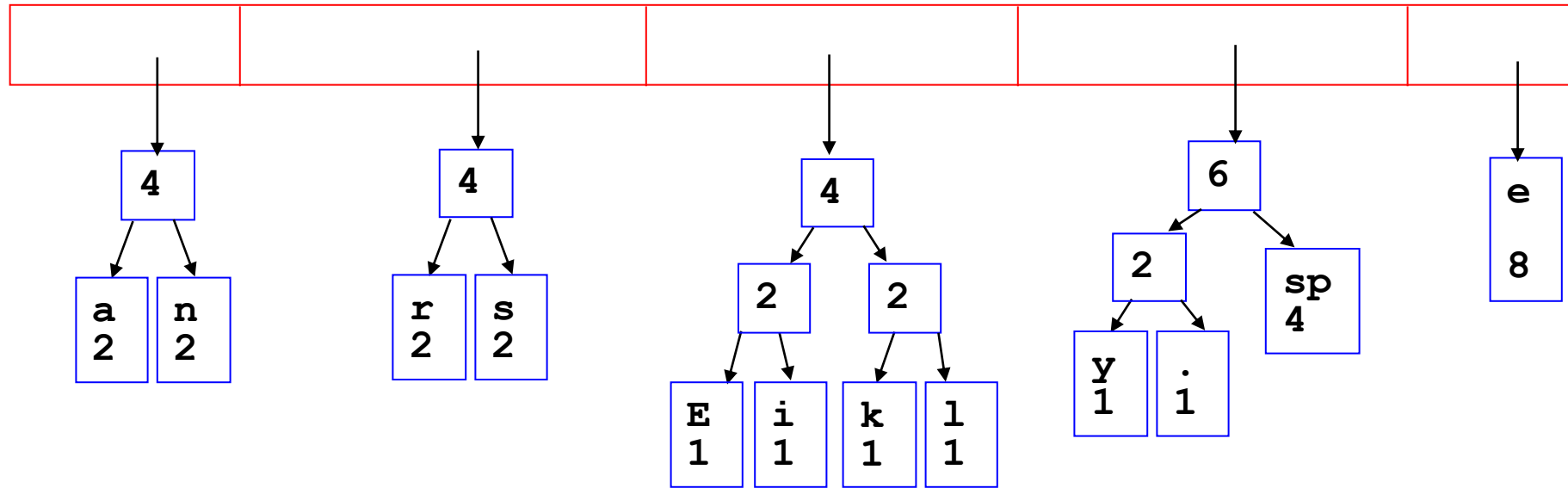
Construyendo un árbol



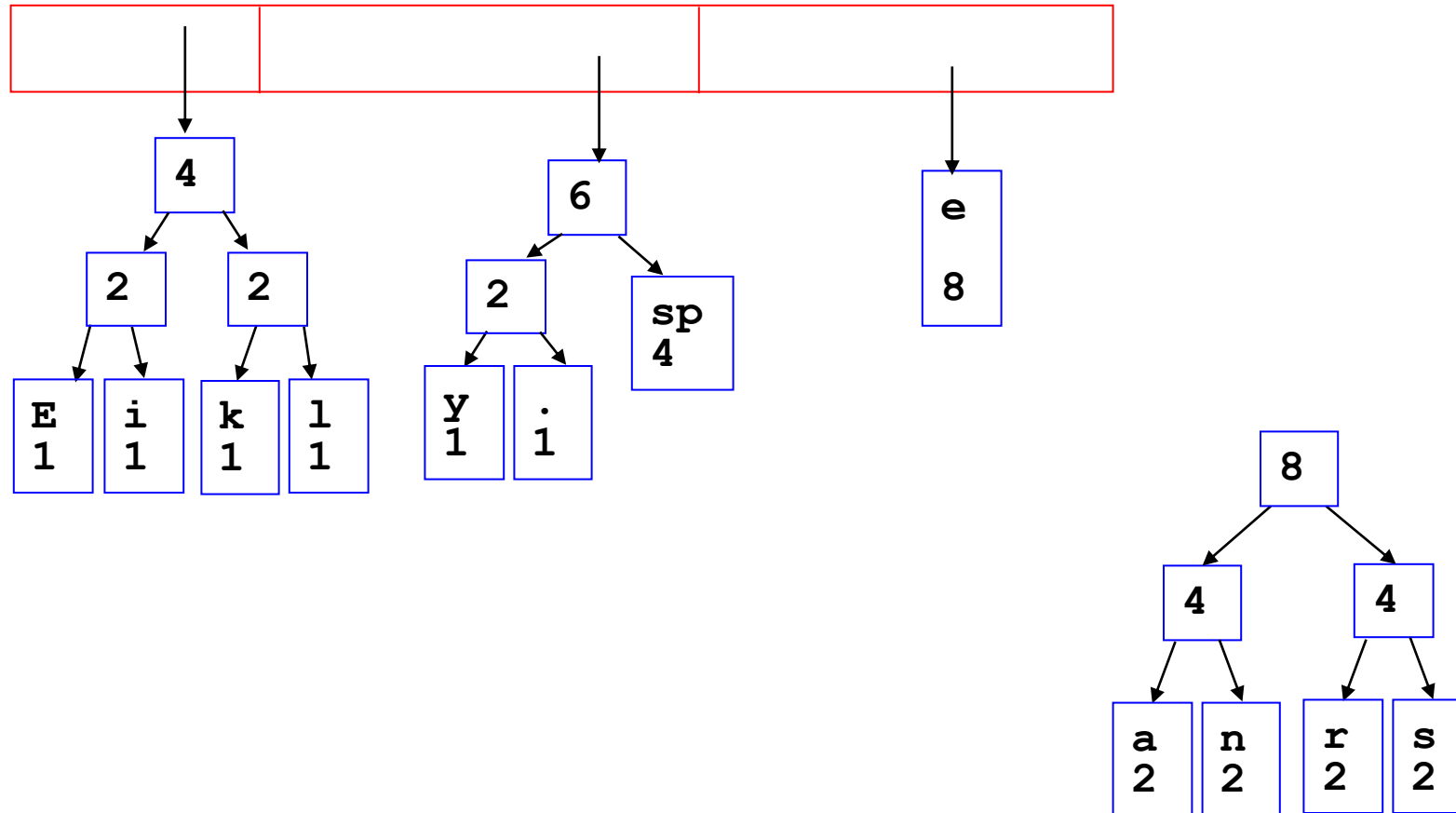
Construyendo un árbol



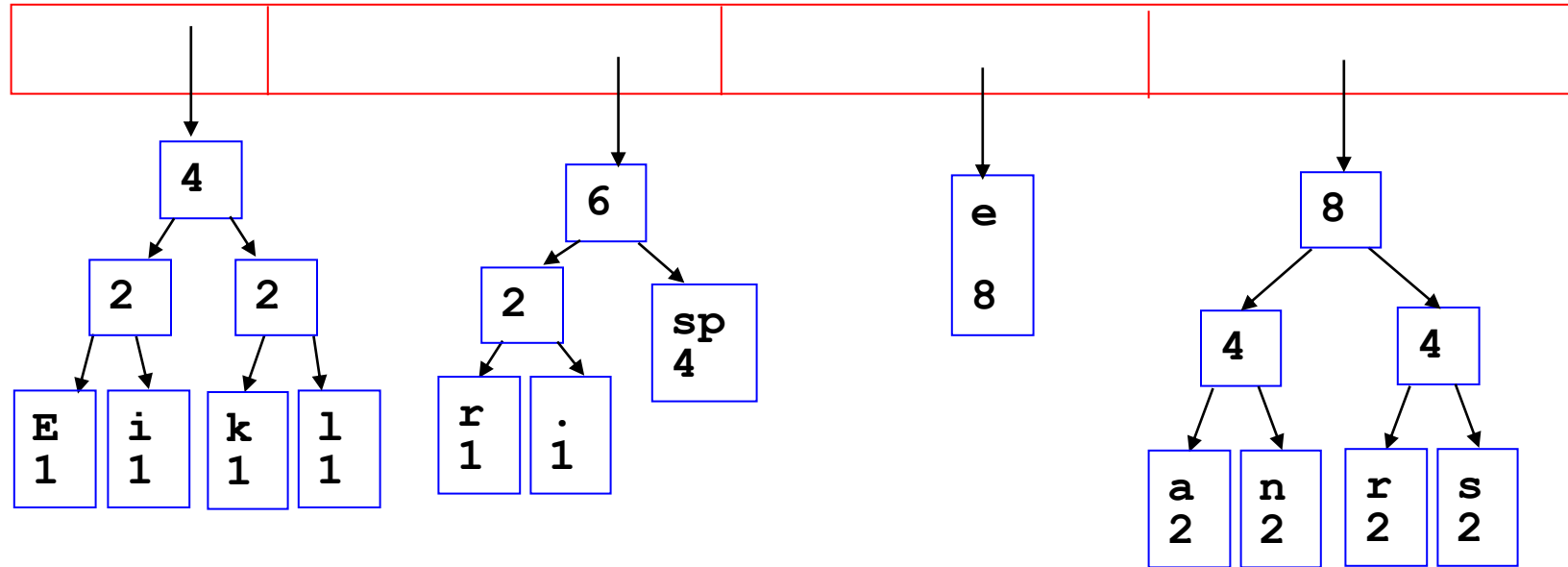
Construyendo un árbol



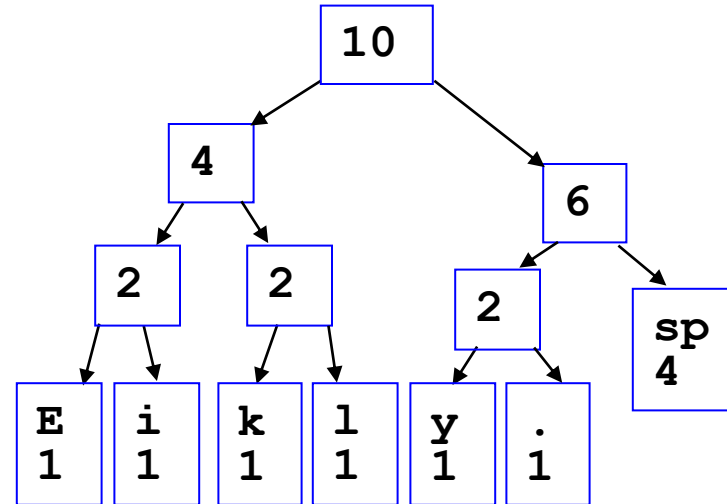
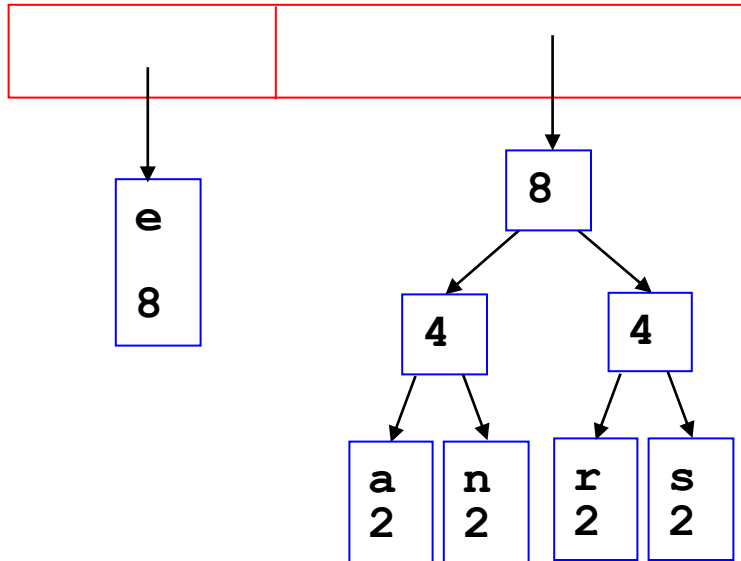
Construyendo un árbol



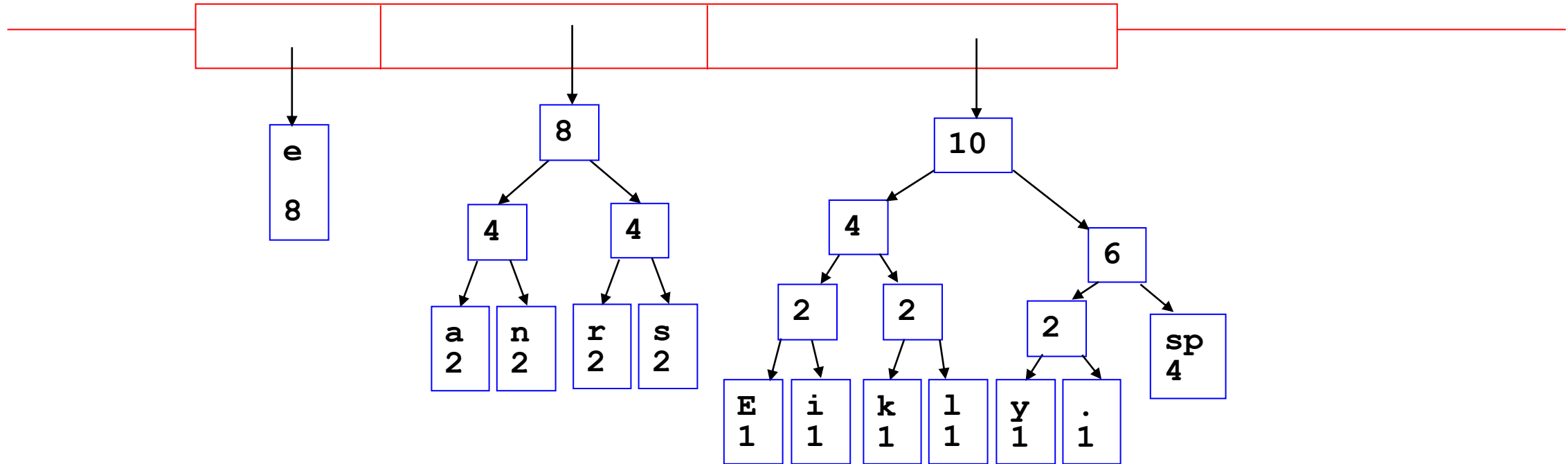
Construyendo un árbol



Construyendo un árbol



Construyendo un árbol



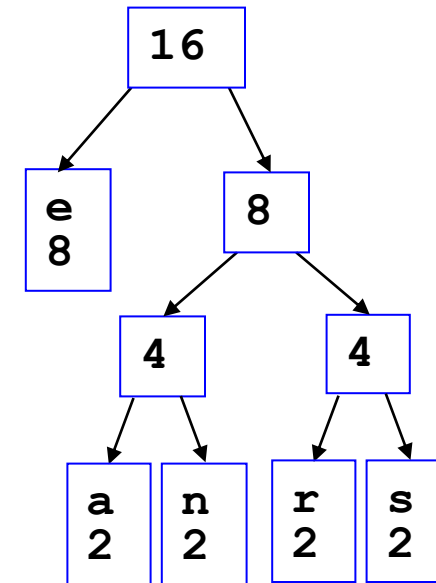
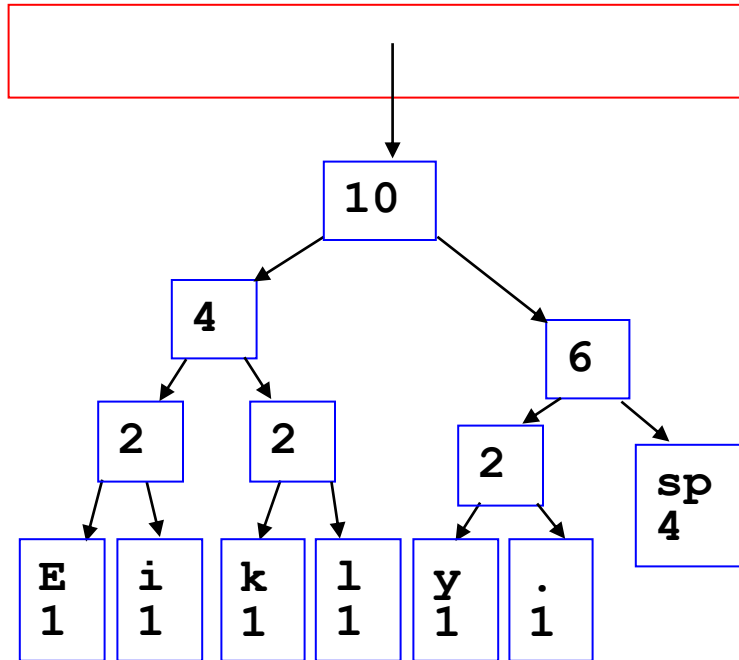
Pregunta: ¿Qué sucede con los valores de baja frecuencia en comparación con los valores de alta frecuencia?

A. Profundidad menor

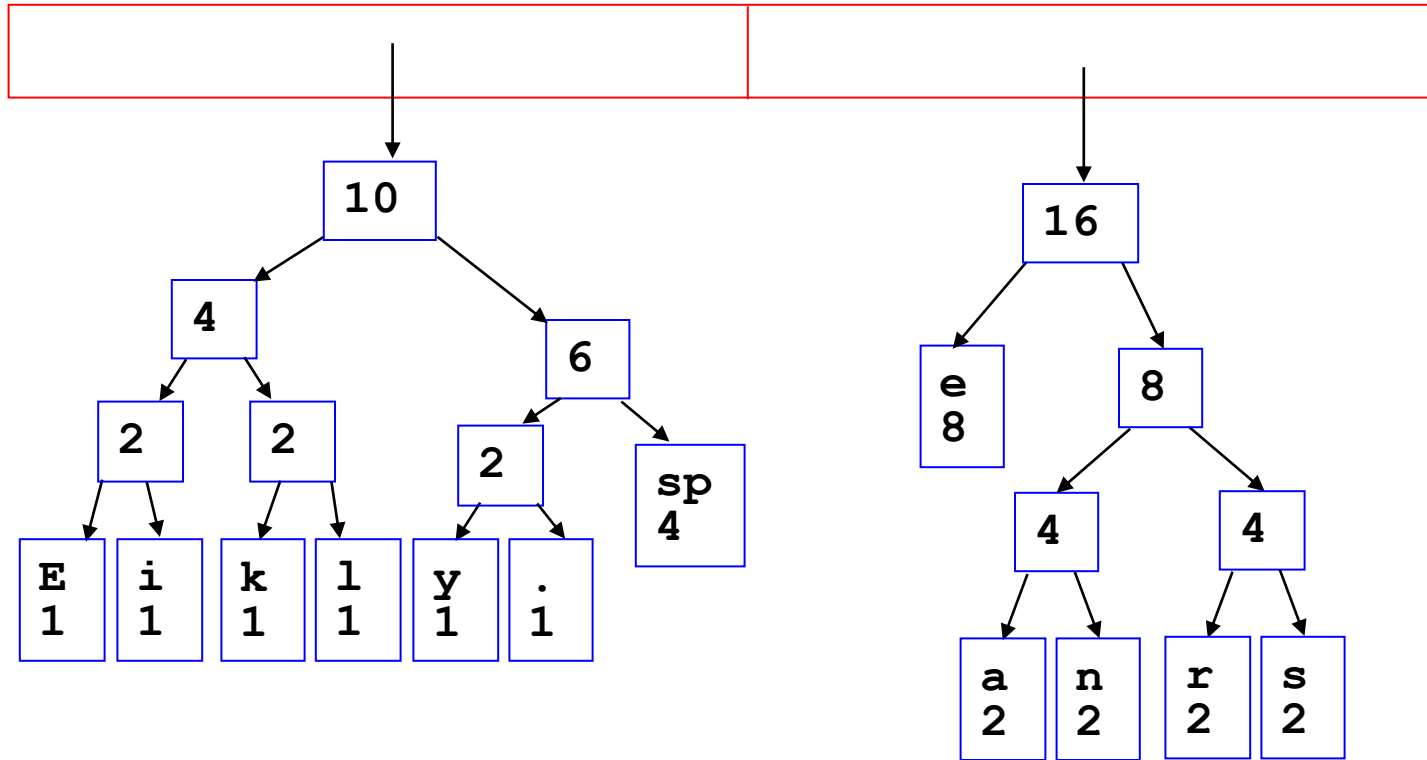
B. Profundidad mayor

C. Algo más

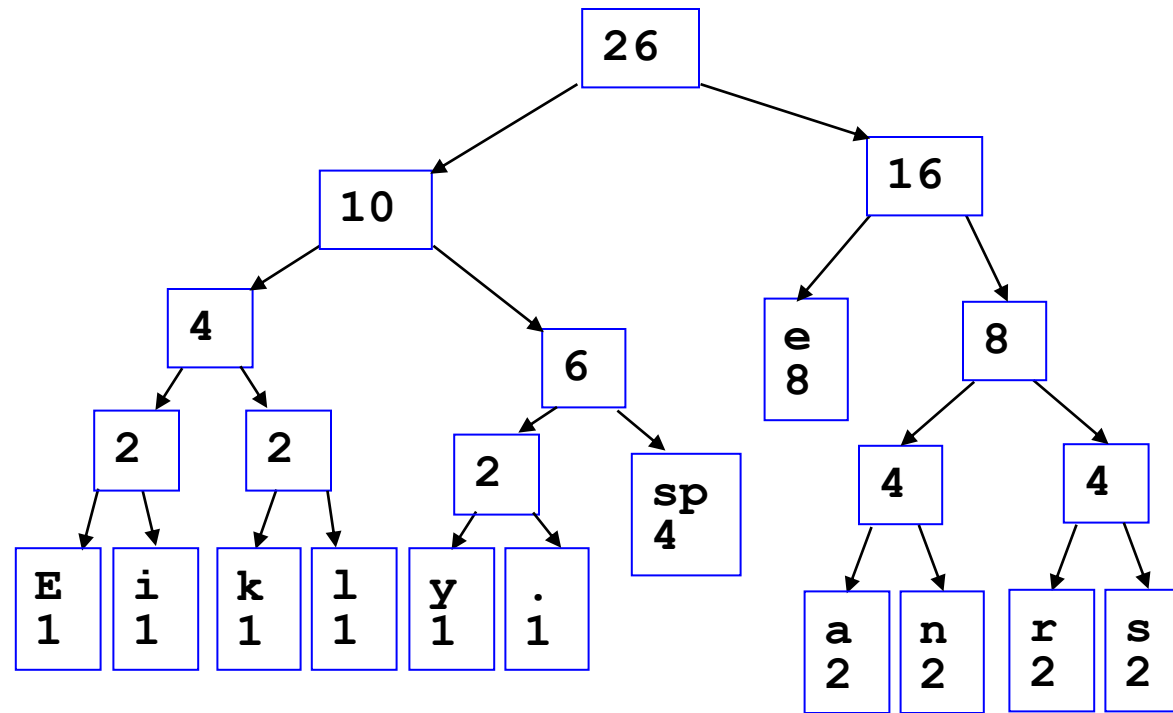
Construyendo un árbol



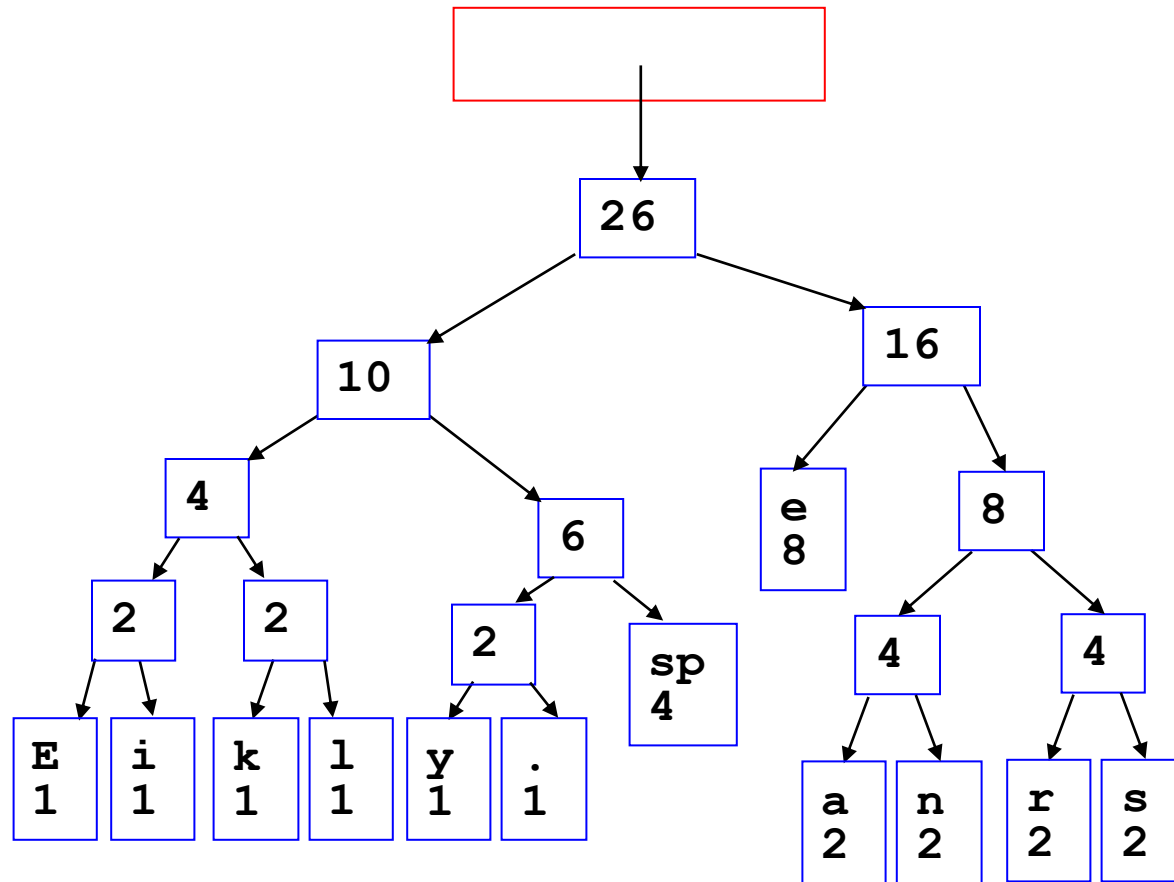
Construyendo un árbol



Construyendo un árbol



Construyendo un árbol



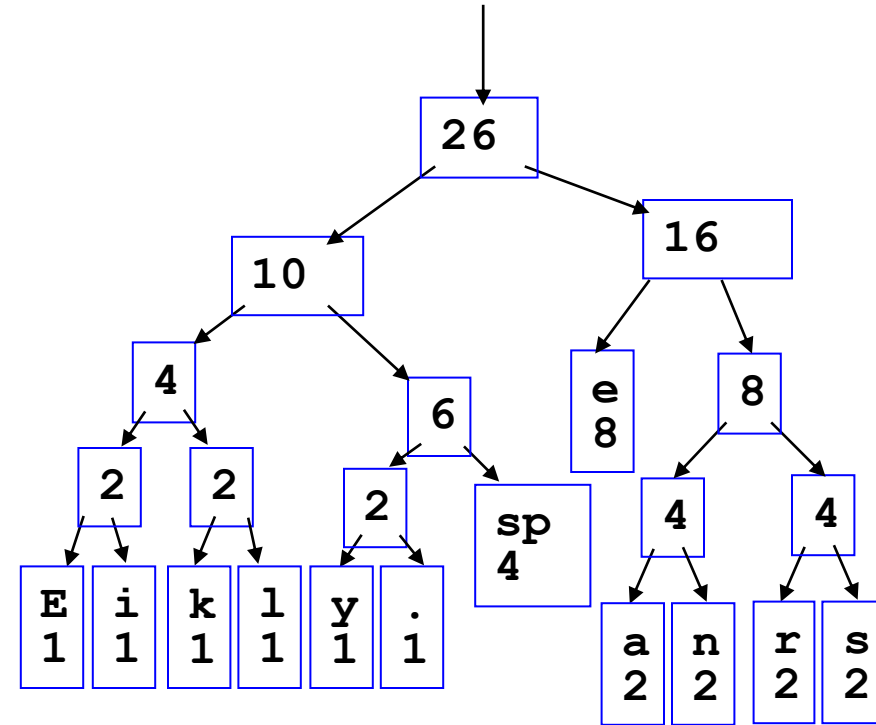
• Después de poner en cola este nodo, solo queda un nodo en la cola de prioridad.

Construyendo un árbol

Retire de la cola el único nodo que queda.

Este árbol contiene las nuevas palabras de código para cada carácter.

La frecuencia del nodo raíz debe ser igual al número de caracteres del texto.



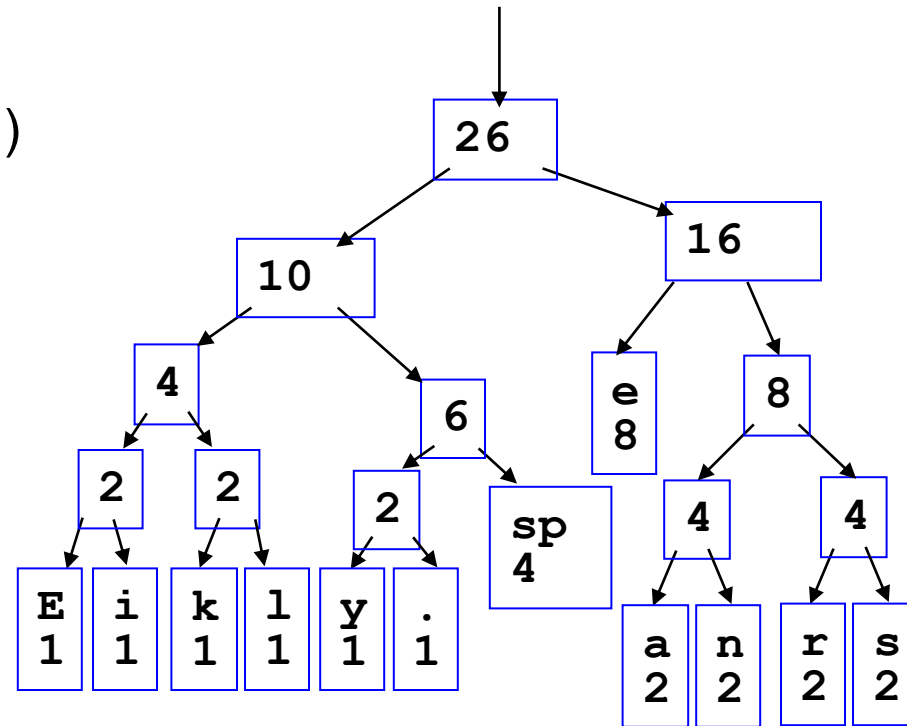
Eerie eyes seen near lake.

4 espacios, 26 caracteres en total

Codificación del Archivo

Recorrido del árbol para los códigos

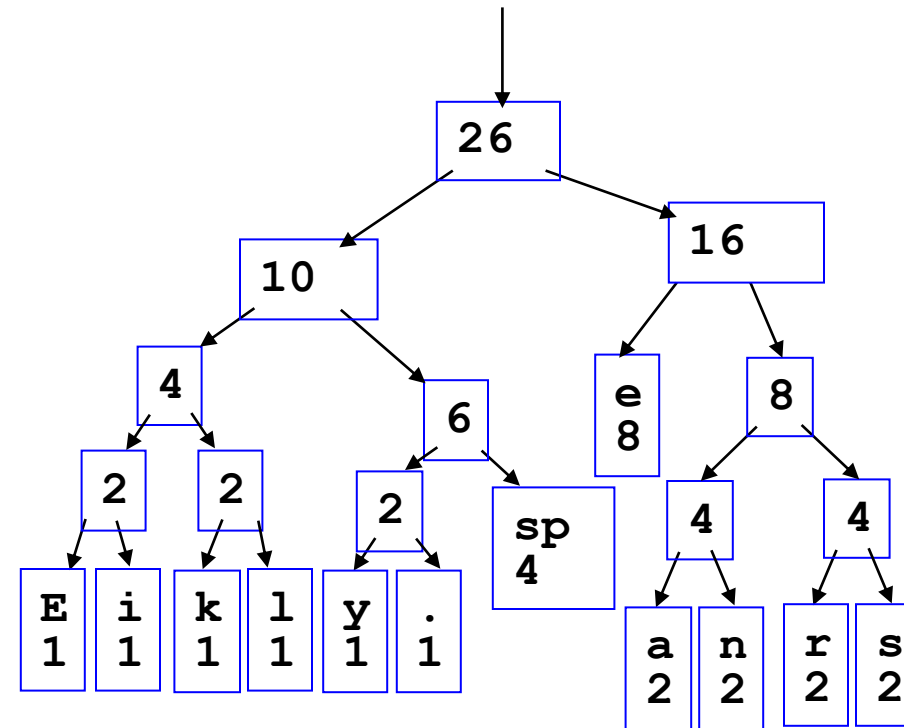
- Realice un recorrido del árbol para obtener nuevas palabras de código (secuencia de 0 y 1)
- izquierda, agregue un 0 al código
- Derecha, agregue un 1 al código
- El código sólo se completa cuando se alcanza un nodo hoja.



Codificación del Archivo

Recorrido del árbol para los códigos

Original Value	New Code
E (0100 0101)	0000
i (0110 1001)	0001
k (0110 1011)	0010
l (0110 1100)	0011
y (0111 1001)	0100
. (0010 1110)	0101
space (0010 0000)	011
e (0110 0101)	10
a (0110 0001)	1100
n (0110 1110)	1101
r (0111 0010)	1110
s (0111 0011)	1111



Códigos libres de prefijo:

El valor para un código nunca es el prefijo de otro código.

Codificando el archivo

- Vuelva a escanear el archivo original y codifique el archivo usando los nuevos códigos:

Eerie eyes seen near lake.

```
0000101111000011001110
010010111101111111010
110101111011011001110
011001111000010100101
```

Char	New Code
E	0000
i	0001
k	0010
l	0011
y	0100
.	0101
space	011
e	10
a	1100
n	1101
r	1110
s	1111

Codificación del archivo

Resultados

- ¿Hemos mejorado las cosas?
- 84 bits para codificar el archivo.
- ASCII tomaría $8 * 26 = 208$ bits

```
000010111000011001110  
01001011110111111010  
110101111011011001110  
011001111000010100101
```

- Sí, el código modificado utiliza 4 bits por carácter.
- Bits totales $4 * 26 = 104$. Los ahorros no son tan grandes.

Decodificando el archivo

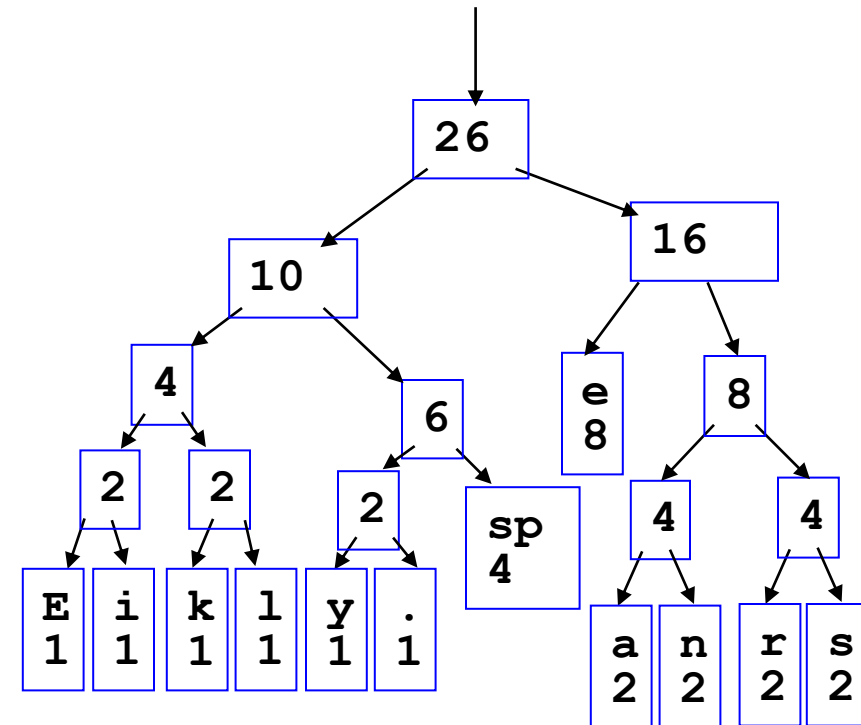
- ¿Cómo sabe el receptor cuáles son los códigos?
- Árbol construido para cada archivo.
 - Considera la frecuencia para cada archivo.
 - Gran éxito en la compresión, especialmente para archivos más pequeños
- árbol predeterminado
 - basado en análisis estadístico de archivos de texto u otros tipos de archivos

Clicker 3: decodificar el archivo

- Una vez que el receptor tiene el árbol, escanea el flujo de bits entrante
- 0 \Rightarrow ir a la izquierda
- 1 \Rightarrow ve a la derecha

1010001001111000111111
11011100001010

- A. elk nay sir
- B. eek a snake
- C. eek kin sly
- D. eek snarl nil
- E. eel a snarl



Eerie eyes seen near lake.



Universidad Nacional Mayor de San Marcos
Universidad del Perú. Decana de América

Implementar el código Huffman
