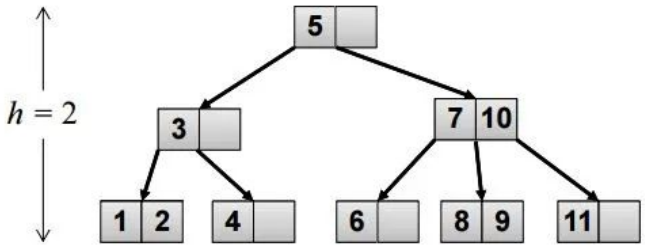


Arboles B

Ing. Max Alejandro Antonio Cerna Flores

Agenda

- Definición
- Historia
- Componentes
- Coste Computacional
- Operaciones
  - Búsqueda
  - Inserción
  - Borrado
  - dividir/reestructurar*
- Ventajas y Desventajas

Definición

En informática, un árbol B es una estructura de datos de árbol auto equilibrado, que mantiene datos ordenados y permite búsquedas, acceso secuencial, inserciones y eliminaciones en tiempo logarítmico.

Tiene un coste menor de acceso a datos que otras estructuras como los árboles AVL y árboles binarios, reduciendo la necesidad de utilizar el almacenamiento secundario.

Historia

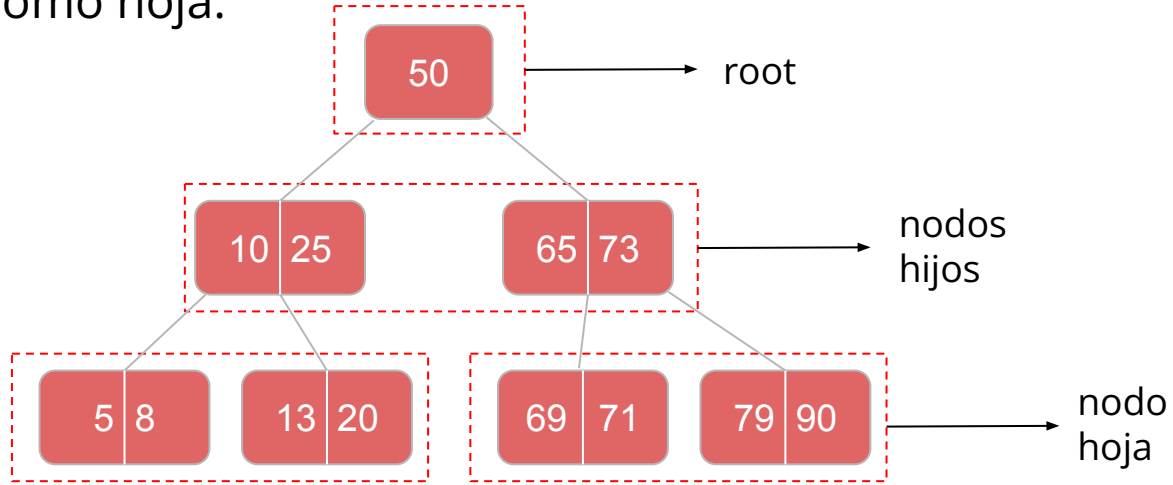
Los árboles B fueron inventados en los años 70 por Rudolf Bayer y Edward McCreight quienes trabajaban en Boeing Research Labs.

Querían diseñar una mejor versión de los árboles binarios, al abordar cargas de datos almacenados en discos.

El árbol B es ideal para sistemas de almacenamiento que leen y escriben bloques de datos relativamente grandes, como bases de datos y sistemas de archivos.

Componentes

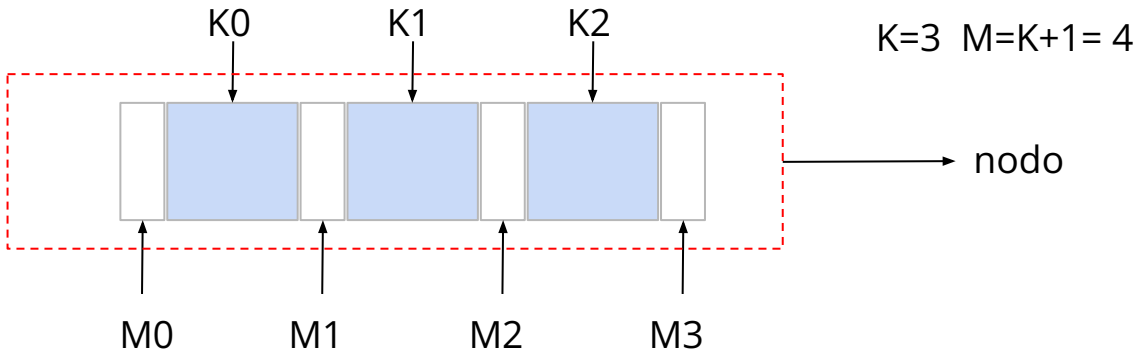
Los árboles B, están compuestos por nodos, teniendo un nodo origen conocido como raíz (root) y sus nodos hijos, cuando un nodo no tiene más hijos se conoce como hoja.



Componentes

Nodos en árboles B

- Cada nodo está compuesto de K claves (key).
- Cada nodo tendrá M hijos (M = K + 1).



Componentes

Nodos

- Un nodo de árbol-B también es llamado Pagina (Page)
- Cada nodo (excepto root) tiene como mínimo (M)/2 hijos.
- Todos los nodos hoja, están en el mismo nivel.
- El orden de las claves en cada nodo debe estar ordenado.

$$k1 > k2 > k3... > kn$$

Componentes

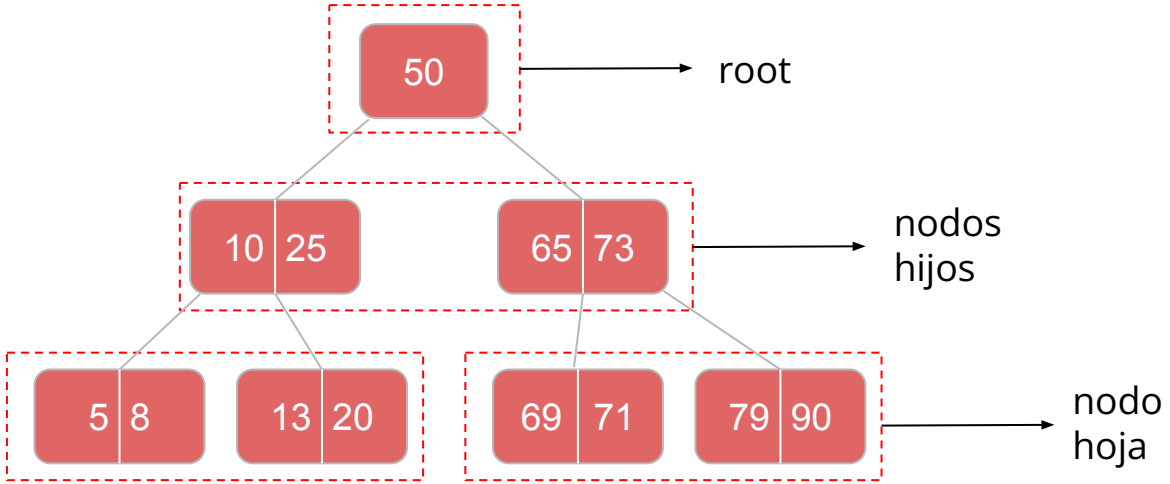
Árbol

- El árbol está ordenado.
- El grado de un árbol-B es el número máximo de hijos que pueden partir de un nodo (M).
- La profundidad es el número máximo de consultas para encontrar una clave.

Componentes

profundidad (n)

n=3
k=2
M=3



Cuanto mayor sea el grado, menor será la profundidad.

Coste Computacional

Altura/Profundidad (mejor escenario/altura mínima)

$$\text{altura} = \log_m (k+1)$$

Altura/Profundidad (peor escenario/altura máxima) tomando d = M/2

$$\text{altura} = \log_d ((k+1)/2)$$

Coste Computacional

Operación	Promedio	Peor escenario
Búsqueda	O(log *n*)	O(log *n*)
Inserción	O(log *n*)	O(log *n*)
Borrado	O(log *n*)	O(log *n*)

Operaciones

Búsqueda

La búsqueda es similar a la búsqueda en un árbol de búsqueda binaria.

Comenzando en la raíz, el árbol se recorre recursivamente de arriba a abajo.

En cada nivel, la búsqueda reduce su campo de visión al subárbol.

Operaciones

Búsqueda

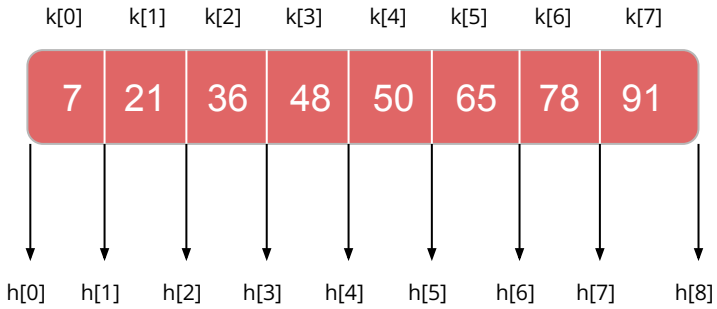
La búsqueda implica:

- Buscar entre las claves contenidas del nodo
- Sino no se encuentra, se salta al nodo hijo donde puede estar.
- Si se llega a un nodo hoja, notificar de no encontrado.

Búsqueda

Buscar: 71

Grado: 9



Búsqueda

Estructura de un Nodo (PSEUDO CÓDIGO)

```
estructura nodo {
        numClaves: Entero;
        claves: Entero[];
        numHijos: Entero;
        hijos: Nodo[];
        esHoja: booleano;
        clavesUsadas: Entero;
}
```

ArbolB (PSEUDO CÓDIGO)

```
arbol {
        grado: entero;
        raiz: nodo;
}
```

Búsqueda

PSEUDO-CODIGO

```
Busqueda(nodo, valor){
        i : Entero;
        i = 0;
        mientras(i < nodo.numClaves && nodo.clave[i] < valor)
                i = i + 1;
        if(i< nodo.numClaves && nodo.clave[i] == valor)
                return (nodo,i);
        if(nodo.hoja)
                return (nodo, null);
        else
                return busqueda(nodo, valor);
}
```

Operaciones

Inserción

Consiste en introducir elementos en el árbol B.

La aceptación de duplicados es un criterio al momento de diseñar el árbol B.

INSERTAR(nodo, valor)

Inserción

Una inserción implica:

- Salvo ciertas excepciones, los elementos serán insertados en una hoja.
  - Si el nodo no es una hoja, se debe dirigir al hijo más prometedor e intentarlo de nuevo.
  - La inserción en una hoja se hace metiendo el elemento de manera ordenada en el arreglo de claves.
- Se debe respetar el grado del árbol. Si se tienen demasiadas claves hay que ***dividir*** el nodo.

Inserción

Parámetros

Árbol B de grado M.

Máximo

- Como mucho M descendientes.
- Como mucho M-1 claves.

Mínimo

- Como mínimo M/2 descendientes.
- Como minimo (M/2)-1 claves.

Inserción

Arbol de grado 7

N = 7

Min. punteros: N/2 = 3.5 => 4.

Min. claves: (N/2) -1 = 2.5 => 3.

Max. punteros: 7

Max. claves: 6

Inserción

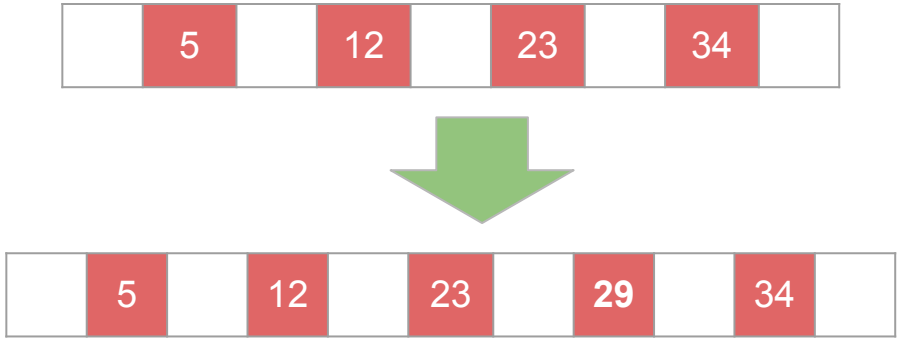
Analicemos el siguiente nodo hoja:

Elemento a insertar: 29

Inserción

Analicemos el siguiente nodo hoja:

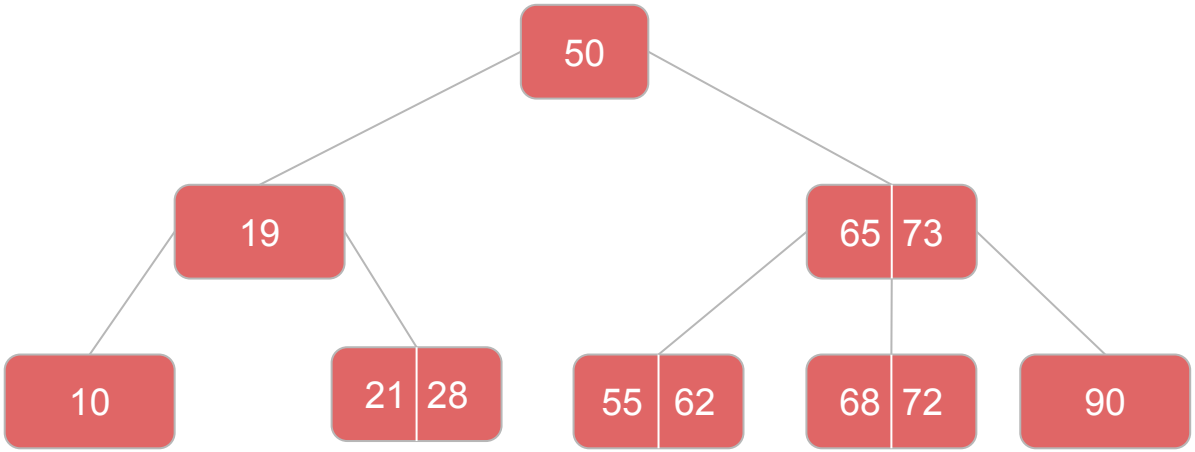
Elemento a insertar: 29



Inserción

Analicemos el siguiente árbol B:

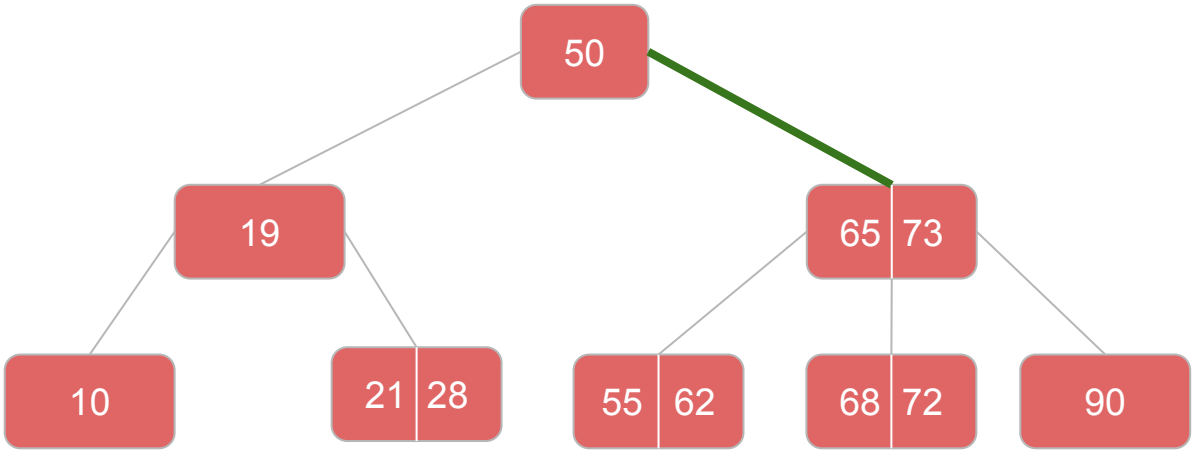
Elemento a insertar: 83



Inserción

Analicemos el siguiente árbol B:

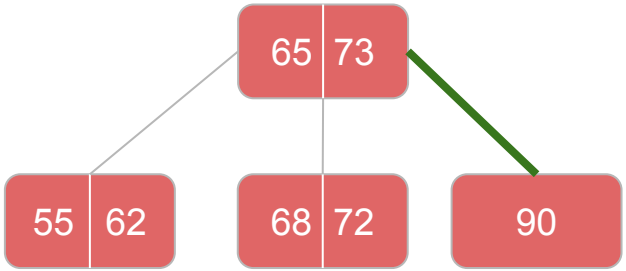
Elemento a insertar: 83



Inserción

Analicemos el siguiente árbol B:

Elemento a insertar: 83



Inserción

Analicemos el siguiente árbol B:

Elemento a insertar: 83

Inserción

Se definirá U (Upper) al número de máximo de claves entonces:

$$U = M - 1$$

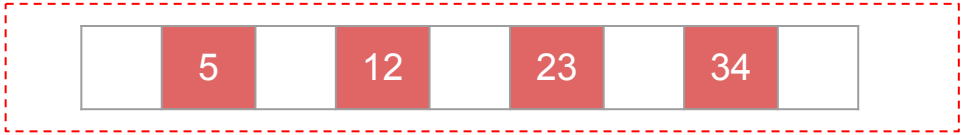
Se definirá L (Lower) al número mínimo de claves entonces:

$$L = (M/2) - 1$$

Inserción

M= 5 (L = 2, U = 4)

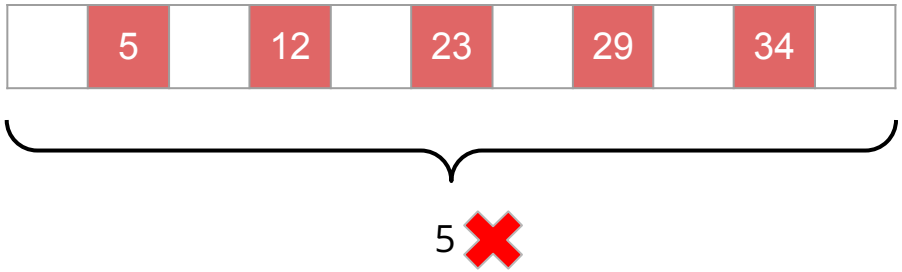
Inserción 29



nodo hoja

Inserción

M= 5 (L = 2, U = **4**)



Inserción

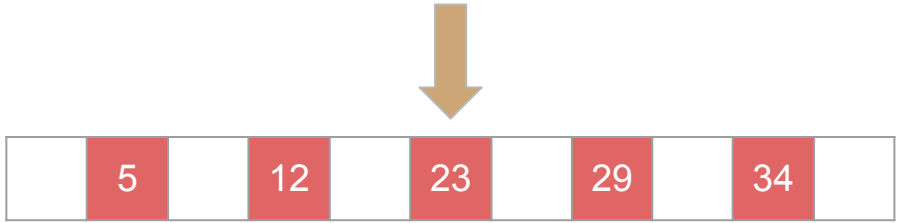
M= 5 (L = 2, U = 4)

Inserción 29

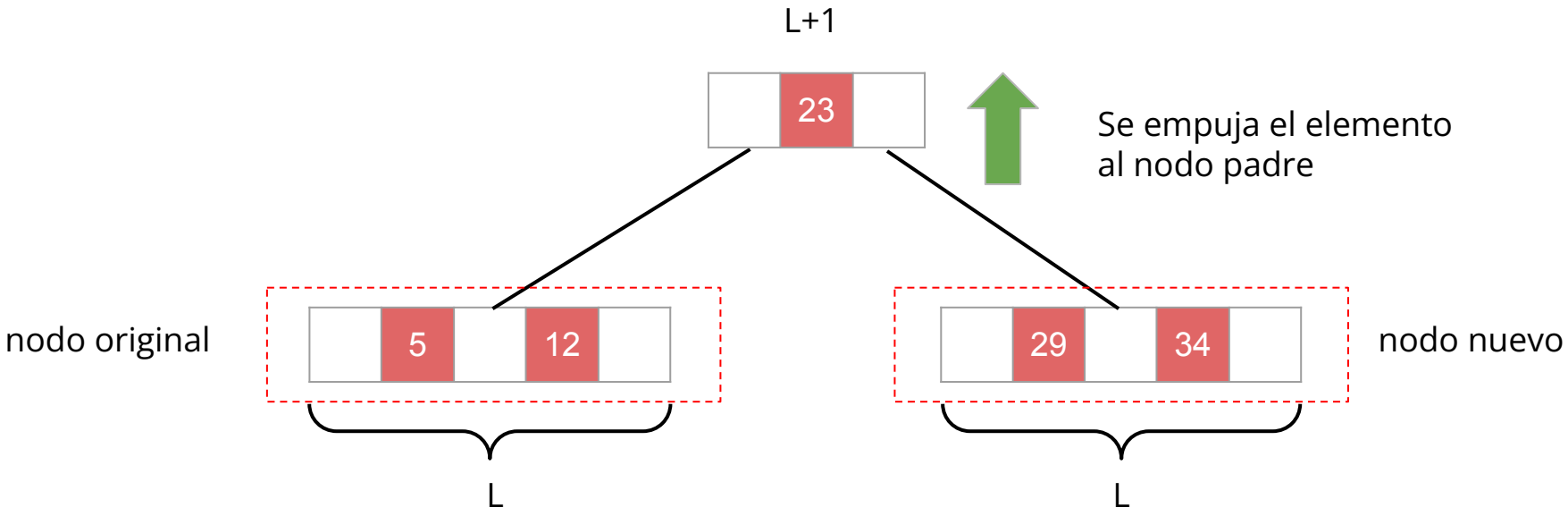
Solución: será necesario dividir.

Inserción - División

Buscar elemento pivote



Inserción - División

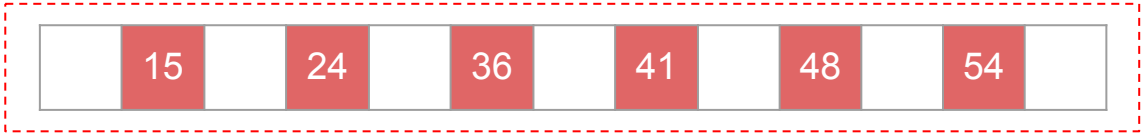


Inserción - División

Otro ejemplo

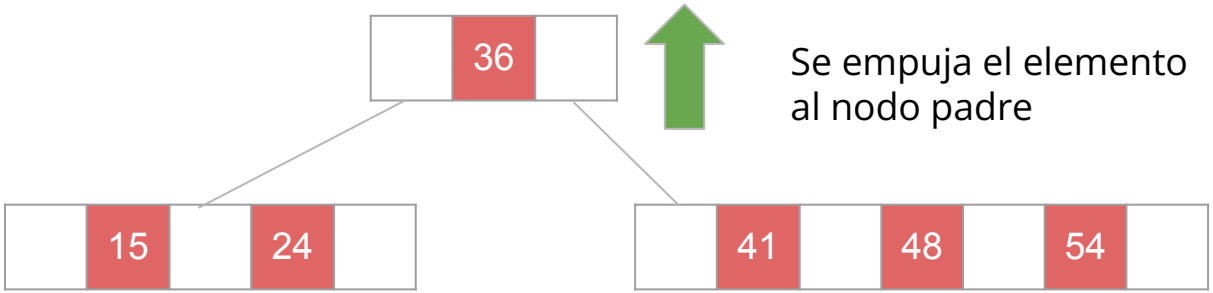
M= 6 (L = 2, U = 5)

Nodo sobrepasado



Inserción - División

M= 6 (L = 2, U = 5)

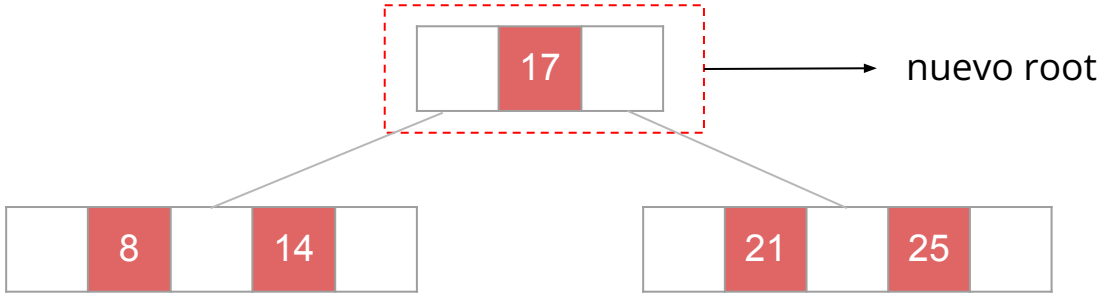


Inserción - División en Raíz

M= 5 (L = 2, U = 4)

Inserción - División en Raíz

M= 5 (L = 2, U = 4)



Resumen División

- Si luego de hacer una inserción un nodo tiene más claves que lo permitido, entonces se divide en dos subnodos el nodo.
- Se toma un elemento pivote que estará en la mediana (posición del arreglo de claves).
- Se traslada las claves e hijos mayores que L a un nuevo nodo y se retiran del nodo viejo, que será el otro subnodo.
- Se empuja el pivote hacia arriba en el árbol.
- El nodo padre que recibe el pivote puede colapsar también, pidiendo una división.
- Como en el nodo raíz no hay padre al que empujar el pivote, se redefine la raíz del árbol.

Inserción

Estructura de un Nodo (PSEUDO CÓDIGO)

```
estructura nodo {
        numClaves: Entero;
        claves: Entero[];
        numHijos: Entero;
        hijos: Nodo[];
        esHoja: booleano;
        clavesUsadas: Entero;
}
```

ArbolB (PSEUDO CÓDIGO)

```
arbol {
        t : entero;
        raiz: nodo;
}
```

NOTA: llamaremos **t** al grado mínimo (o sea ***grado/2***)

Inserción

Constructor de un Nodo (PSEUDO CÓDIGO)

```
nodo {
        constructor nodo(t) {
                clavesUsadas = 0;
                esHoja = verdadero;
                clave[(2*t -1)];
                hijo[2*t];
        }
}
```

Inserción en Nodo

```
insercionEnNodo(nodo,valor) {
        if(nodo.esHoja) {
                i: Entero;
                i = nodo.clavesUsadas;
                mientras(i>=1 && valor < nodo.clave[i-1]){
                        nodo.clave[i] = nodo.clave[i-1];
                        i --;
                }
                nodo.clave[i] = valor;
                nodo.clavesUsadas++;
        }
        else { //el nodo no es hoja
                j: Entero; j = 0;
                mientras(j < nodo.clavesUsadas && valor > nodo.clave[j]){
                        j++;
                }
                if(nodo.hijo[j].clavesUsadas == (2 * t - 1) ){// ( U - maximo de claves)
                        dividir(nodo, j, nodo.hijo[j]);
                        if(valor > nodo.clave[j]) { j++; }
                }
                insercionEnNodo(nodo.hijo[j], valor);
        }
}
```

Inserción

```
insertar( valor) {
        r : Nodo;
        r = raiz;
        if(r.clavesUsadas == (2*t -1)) { // 2*t -1 ( U - maximo de claves)
                s: Nodo(t);
                raiz = s;
                s.esHoja = falso;
                s.clavesUsadas = 0;
                s.hijo[0] = r;
                dividir(s,0,r); // es lo mismo decir dividir(s,0,s.hijo[0]);
                insercionEnNodo(s, valor);
        } else {
                insercionEnNodo(s, valor);
        }
}
```

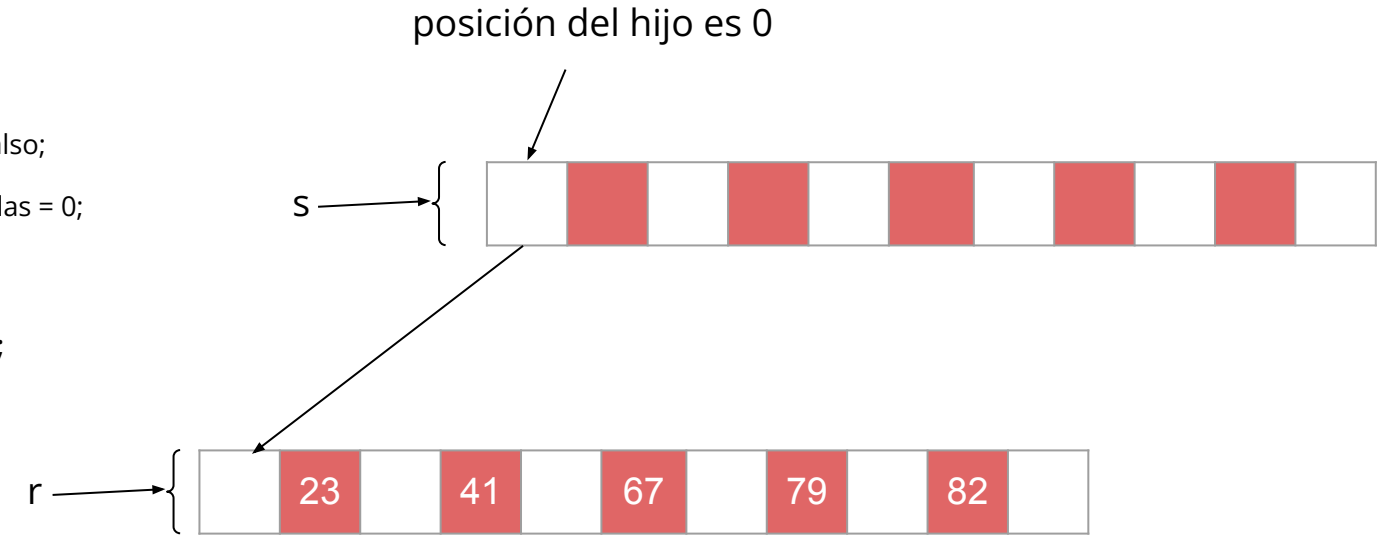
División

si t = 4

si raiz = { 23, 41, 67, 79, 82}

```
s: Nodo(t);
raiz = s;
s.esHoja = falso;
s.clavesUsadas = 0;
s.hijo[0] = r;
```

dividir(s,0,r);



División

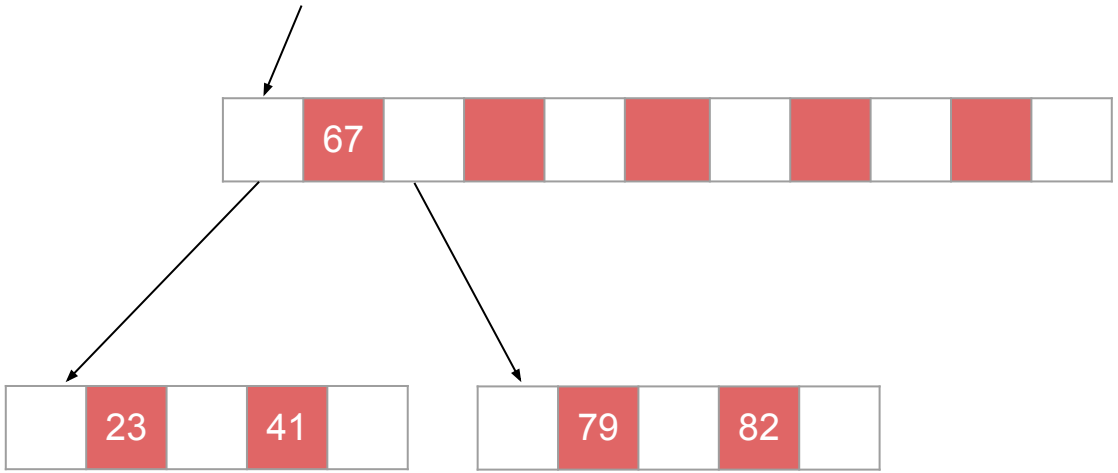
si t = 3

si raiz = { 23, 41, 67, 79, 82}

```
s: Nodo(t);
raiz = s;
s.esHoja = falso;
s.clavesUsadas = 0;
s.hijo[0] = r;
```

dividir(s,0,r);

posición del hijo es 0



División

```
dividir(npadre, posicion, nhijo) {
        newNodo : Nodo;
        newNodo.esHoja = nhijo.esHoja;
        newNodo.clavesUsadas = (t -1); // L  - minimo de claves
        para(int j = 0; j < (t -1); j++) { // copia ultimas claves del nodo hijo al newNodo
                newNodo.clave[j] = nhijo.clave[(j+t)];
        }
        if(nhijo.esHoja == falso){
                para(int k=0; k < t; k++) {
                        newNodo.hijo[k] = nhijo.hijo[(k+t)];
                }
        }
        nhijo.clavesUsadas = t -1;
        para(int j = npadre.clavesUsadas; j < posicion ; j++) {
                npadre.hijo[(j+1)] = npadre.hijo[(j)];
        }
        npadre.hijo[(posicion+1)] = newNodo;
        para(int j = npadre.clavesUsadas; j < posicion ; j--) {
                npadre.clave[(j+1)] = npadre.clave[j];
        }
        npadre.clave[posicion] = nhijo.clave[(t-1)];
        npadre.clavesUsadas++;
}
```

Borrado

Proceso similar a una inserción, el borrado implica:

- En lugar de dividir , se realiza una unión o redistribución.
- El borrado puede ocurrir en cualquier lugar del árbol, contrario a la inserción que ocurría únicamente en las hojas.

Borrado

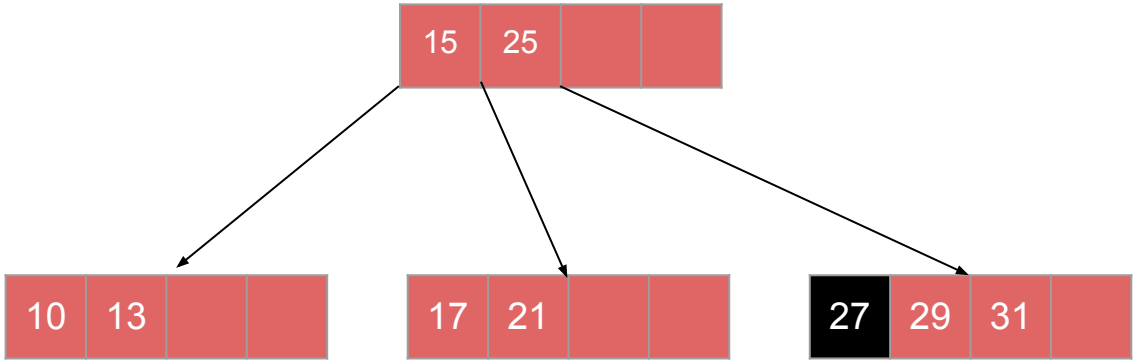
- Si la clave a eliminar se encuentra en una hoja, se elimina directamente.
  - Si al realizar la eliminación, el nodo mantiene el mínimo número de claves (M/2 -1), finaliza.
  - De lo contrario se debe hacer una redistribución.
- Si se encuentra dentro de un nodo interno, no se puede suprimir de forma directa
  - Se debe “subir” la clave que se encuentra más a la derecha en el subárbol izquierdo o más a la izquierda en el subárbol derecho.
- Si al subir la clave, en el nodo respectivo no se cumple el mínimo número de claves (M/2 -1), se debe realizar una redistribución.

Borrado - Redistribución

- **NOTA:** d = ((M/2) -1)
- Si un nodo vecino tiene suficiente número de claves (**>*d***), realiza un ***préstamo.***
  - La clave que se encuentra más a la izquierda “sube”.
  - La clave del nodo padre “baja”.
- Si un nodo vecino no tiene suficiente número de claves (**<=*d***) , se realiza una unión entre el nodo donde se realiza el borrado (**nodo de supresión),** su vecino y su padre.

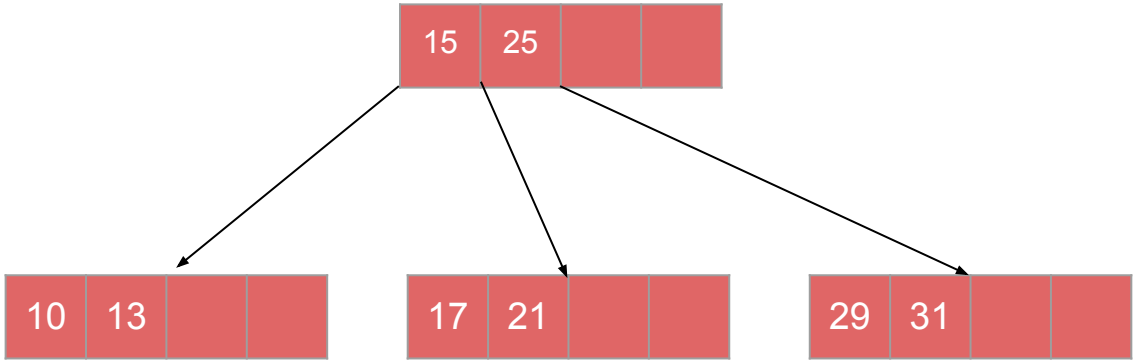
Ejemplo 1- Borrado

Eliminar valor 27 (M=5, U=4, L=2)



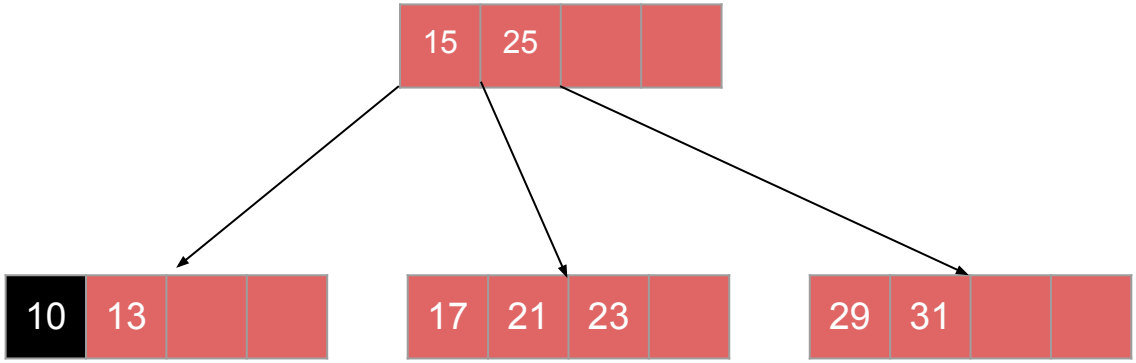
Ejemplo 1- Borrado

Eliminar valor 27 (M=5, U=4, L=2)



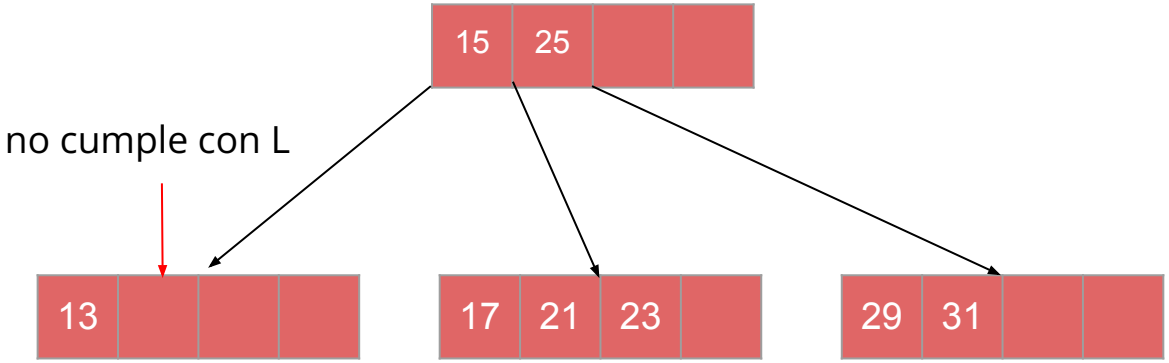
Ejemplo 2- Borrado

Eliminar valor 10 (M=5, U=4, L=2)



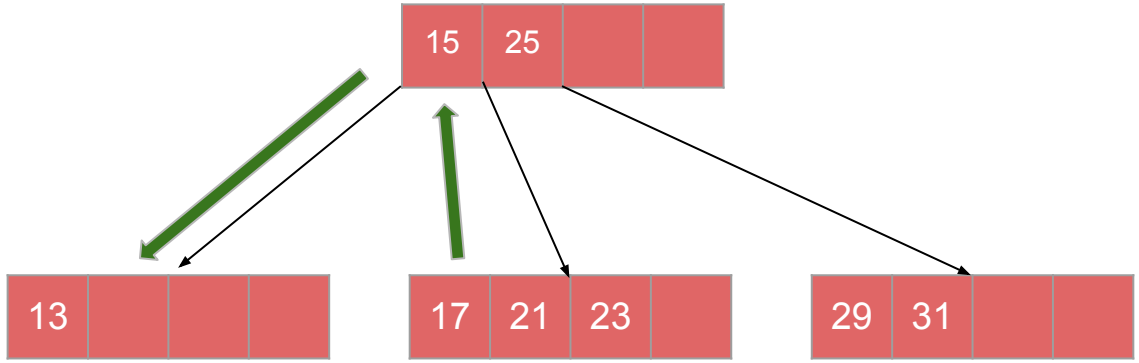
Ejemplo 2- Borrado

Eliminar valor 10 (M=5, U=4, L=2)



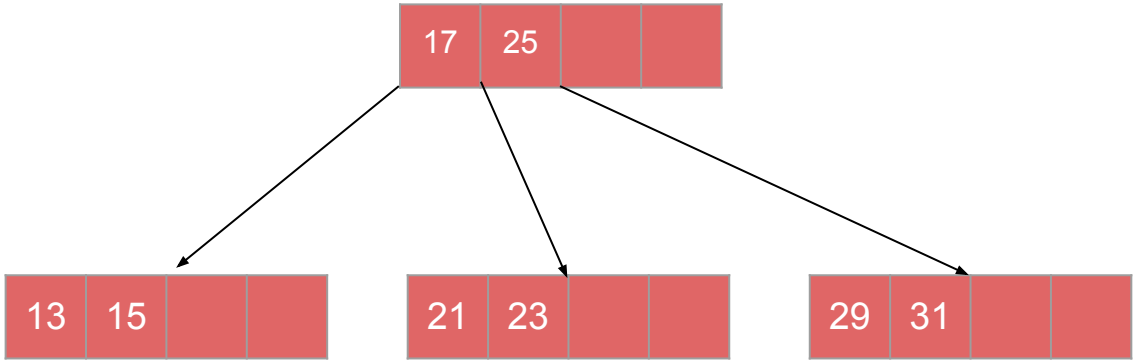
Ejemplo 2- Borrado

Eliminar valor 10 (M=5, U=4, L=2)



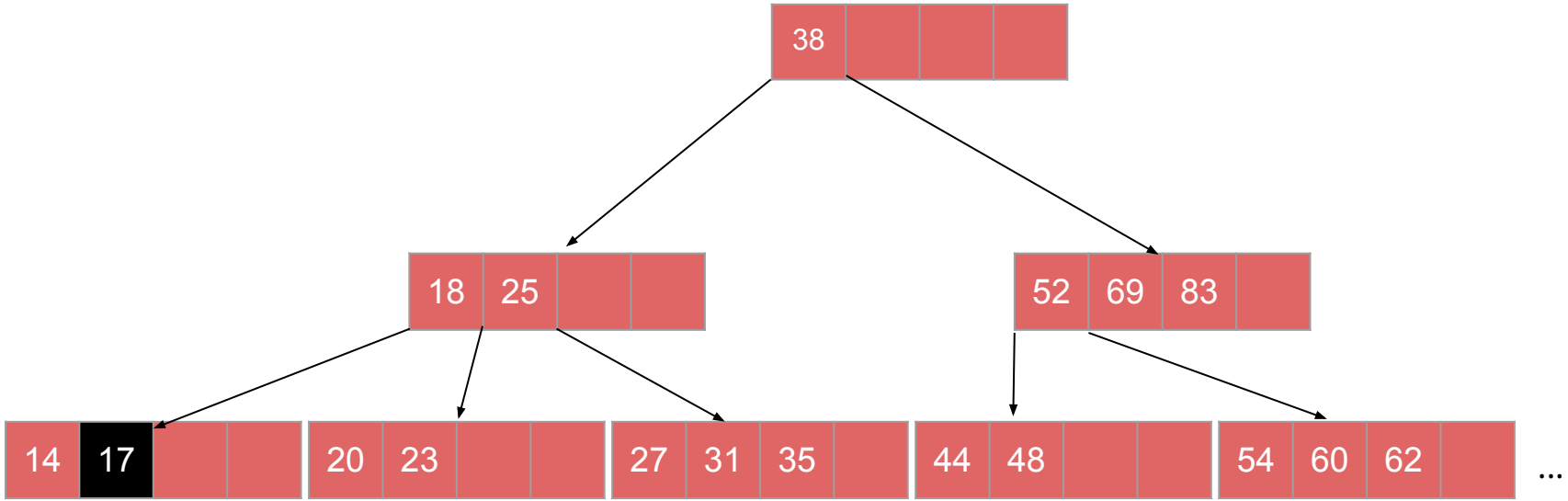
Ejemplo 2- Borrado

Eliminar valor 10 (M=5, U=4, L=2)



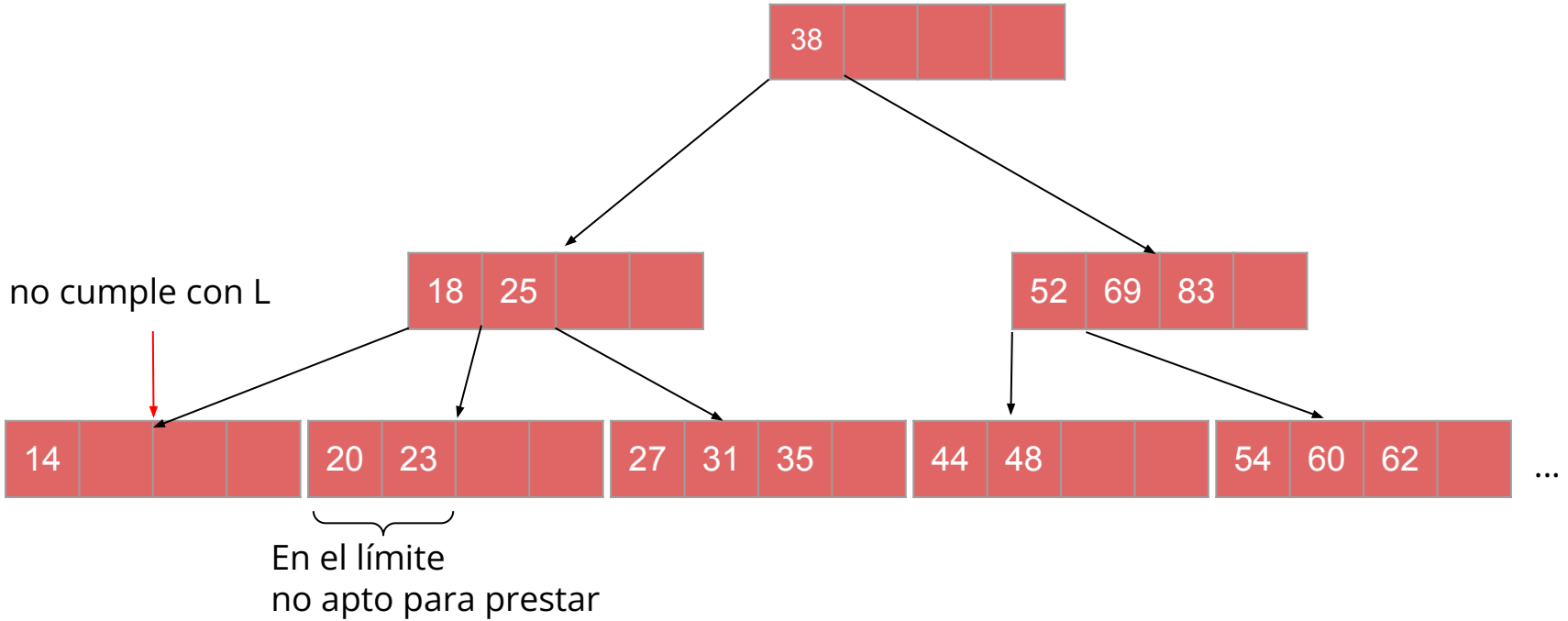
Ejemplo 3- Borrado

Eliminar valor 17 (M=5, U=4, L=2)



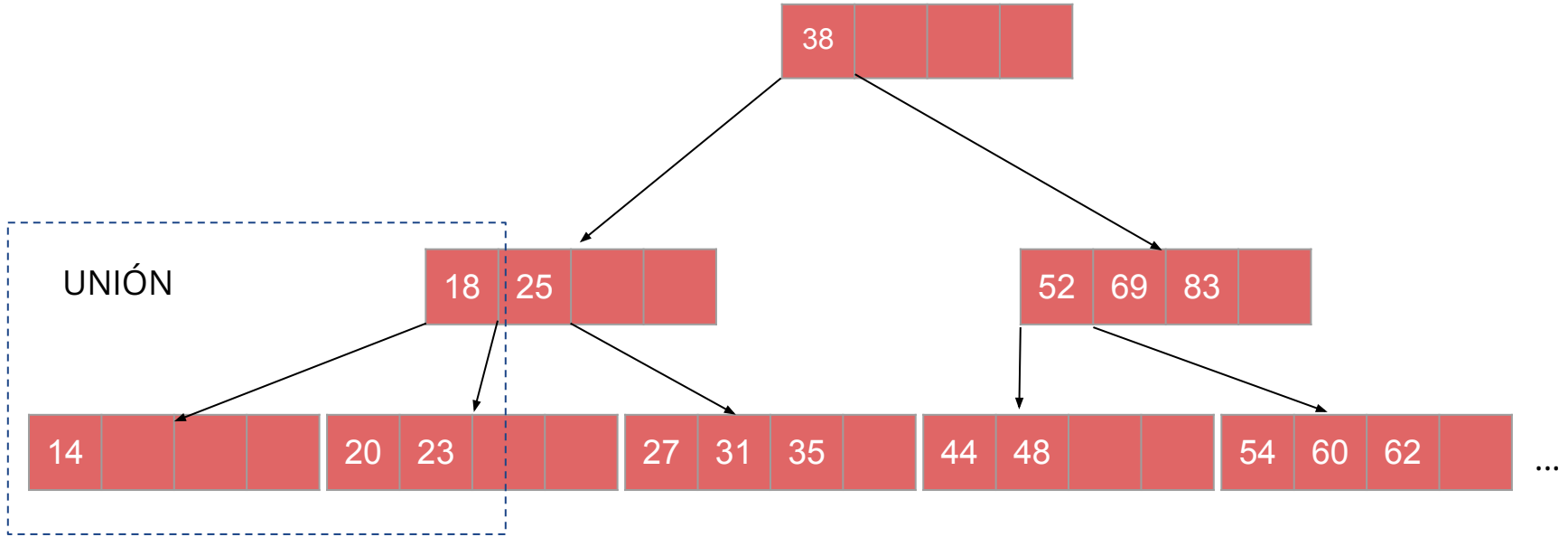
Ejemplo 3- Borrado

Eliminar valor 17 (M=5, U=4, L=2)



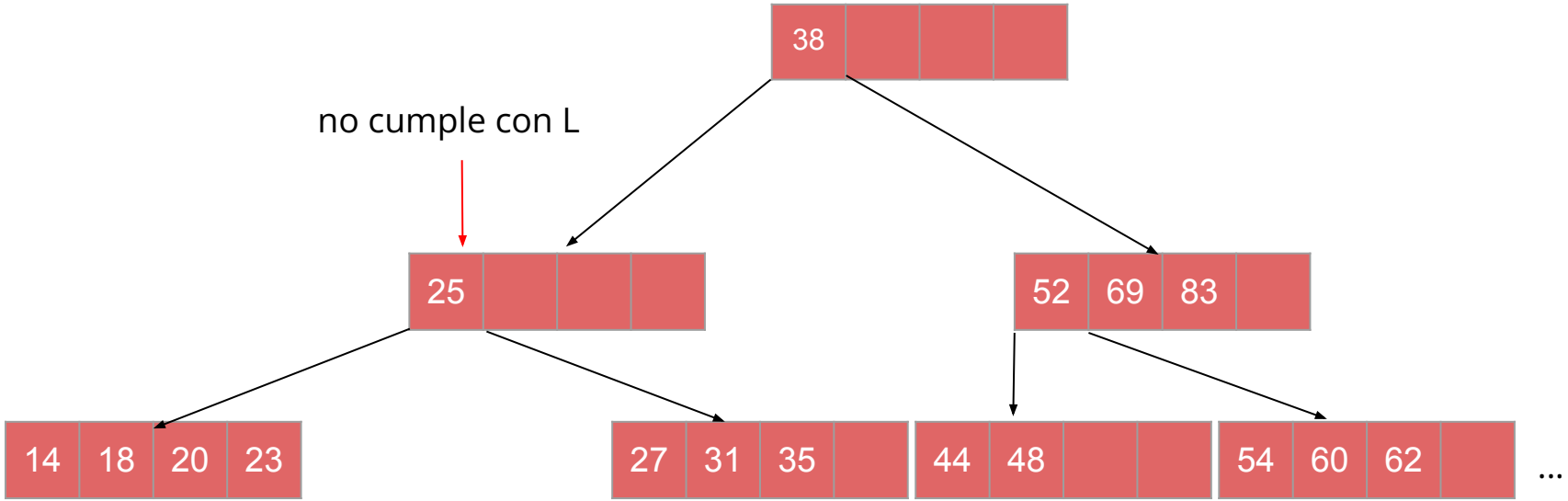
Ejemplo 3- Borrado

Eliminar valor 17 (M=5, U=4, L=2)



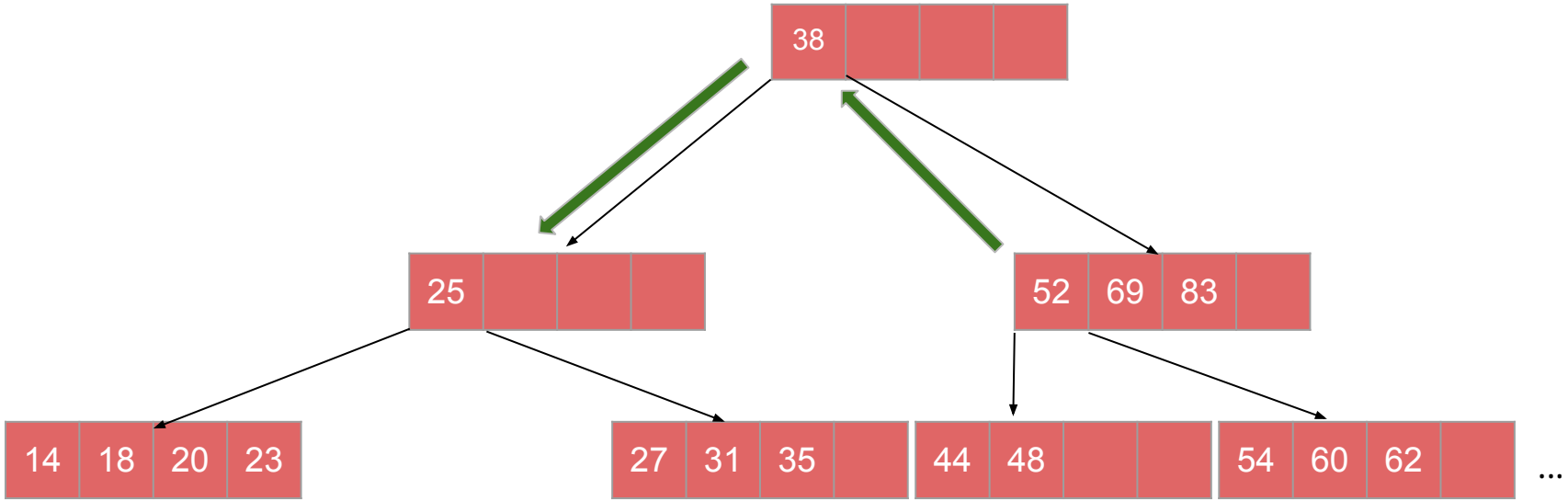
Ejemplo 3- Borrado

Eliminar valor 17 (M=5, U=4, L=2)



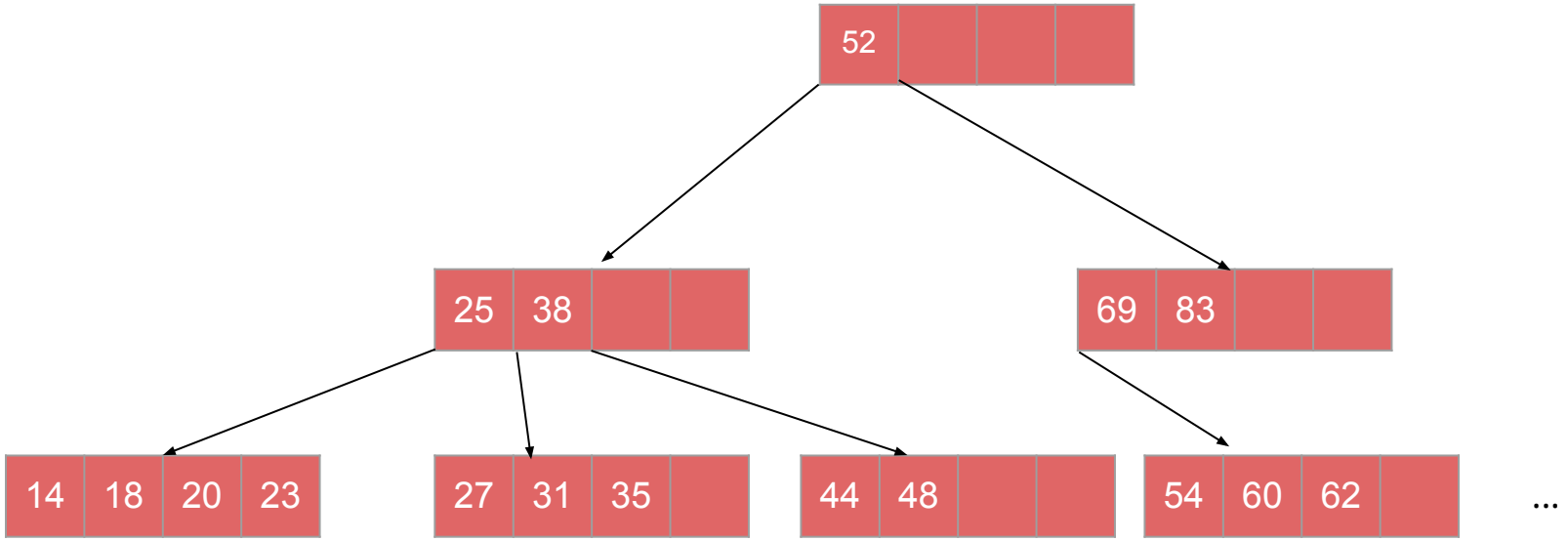
Ejemplo 3- Borrado

Eliminar valor 17 (M=5, U=4, L=2)



Ejemplo 3- Borrado

Eliminar valor 17 (M=5, U=4, L=2)



Ventajas

- Todos los nodos hoja están al mismo nivel, la recuperación de cualquier registro toma el mismo tiempo.
- Permanecen balanceados automáticamente.
- Proveen un buen desempeño para consultas exactas y por rangos.
- Inserciones, modificaciones y eliminaciones son eficientes, y se mantiene el orden de las claves para una recuperación rápida.