BUSQUEDA: Terminología

- Una tabla o archivo es un grupo de elementos.
- Cada elemento se denomina registro.
- Hay una llave asociada a cada registro, que se usa para diferenciar unos de otros. La asociación entre un registro y su llave puede ser simple o compleja.
- En la forma más simple la llave está contenida dentro del registro en un tramo a una distancia específica del principio del mismo. Esto se llama llave interna o incluida.
- En otros casos hay una tabla de llaves diferentes que incluye apuntadores a los registros. Estas se llaman llaves externas.
- Para todo archivo existe por lo menos un conjunto de llaves que es único. Dicha llave se llama llave primaria.
- Además pueden los registros estar ordenados por otro conjunto de datos, a esto se lo denomina llave secundaria.

BUSQUEDA: Terminología

- Un algoritmo de búsqueda, es un algoritmo que acepta un argumento a y trata de encontrar un registro cuya llave sea a. El algoritmo puede dar como resultado un apuntador al registro encontrado, o un apuntador a registro nulo.
- A la búsqueda exitosa se la llama recuperación.
- Muchas veces al no encontrar el registro se desea que el algoritmo inserte una nueva llave con el valor indicado, esto se llama algoritmo de búsqueda e inserción.
- Si la tabla de búsqueda esta contenida totalmente en memoria se llama búsqueda interna, si la misma está almacenada en memoria auxiliar se llama búsqueda externa.

Búsqueda secuencial

- La forma más simple de búsqueda es la secuencial, esta búsqueda se puede utilizar sobre una tabla ordenada o no, o sobre una lista ligada.
- Si k es un arreglo de n llaves, y r es un arreglo de n registros, de manera tal que k(i) es la llave del registro r(i), y si key es el argumento de búsqueda.

```
Para i que va de 0 hasta n, de a 1
        Si key == k(i) entonces FIN( i )
Fin( -1 )
```

El algoritmo revisa cada llave, si logra la coincidencia devuelve el apuntador i, sino obtiene coincidencia devuelve el valor -1.

Búsqueda secuencial

- La eficiencia del método de búsqueda, es (n +1) / 2 para el caso de una búsqueda exitosa, y de n en caso de una infructuosa. En ambos casos el orden es O(n)
- Es normal que haya un grupo de registros que son más accedidos que otros, ejemplo facultad o infractores de ley.
- También es muy difícil conocer cual va a ser el registro o grupo de registros más accedidos.
- Un método implica mover el último puntero localizado en una búsqueda exitosa al primer lugar, esto se llama moverse-al-frente.
- Otra posibilidad es al realizar una búsqueda exitosa mover el puntero localizado una posición hacia delante. Esto se llama transposición.
- Se ha demostrado que el método de transposición es más eficiente en un gran número de búsquedas en las que no cambie la distribución de probabilidad, en cambio el método de moverse al frente reacciona mejor ante los cambios de distribución de probabilidades.

Búsqueda secuencial

Si la tabla se encuentra ordenada (por ej. en orden ascendente) pueden usarse varias técnicas para mejorar la eficiencia.

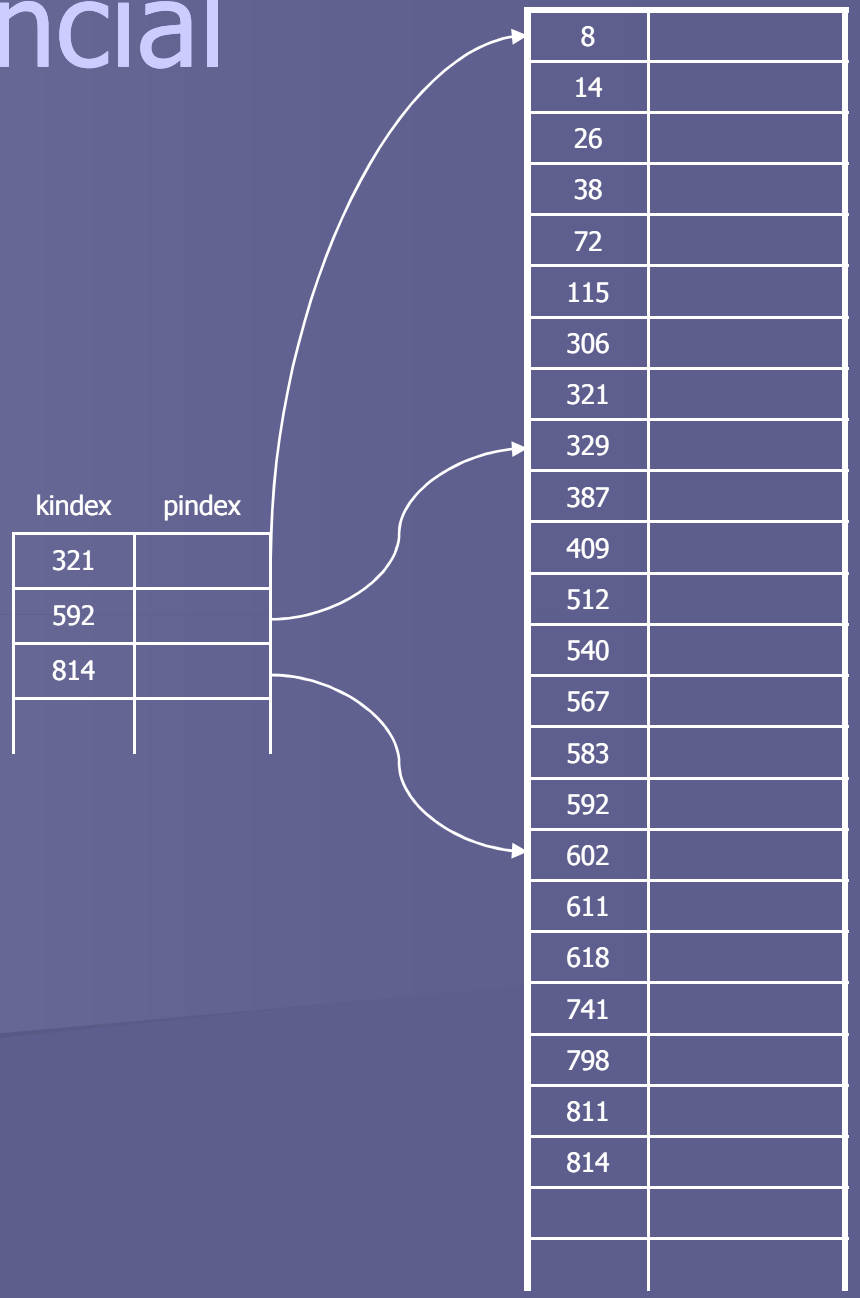
- Un caso simple es por ej. si el elemento a buscar no se encuentra en la tabla en el caso ordenado hacen falta n/2 comparaciones para detectarlo, en cambio si la tabla está desordenada se necesitan n comparaciones.
- Si juntamos un grupo de peticiones de búsqueda a la tabla ordenada, y ordenamos las peticiones, la cantidad de comparaciones a la tabla se reduce considerablemente. Si la tabla de peticiones es más grande que la tabla de búsqueda, el método secuencial de búsqueda es el más eficiente de todos.

Búsqueda secuencial

La búsqueda secuencial indexada, es una técnica para perfeccionar la eficiencia de búsqueda en un archivo ordenado.

Se requiere un adicional de memoria para almacenar una tabla auxiliar llamada INDEX.

La cantidad de comparaciones se reduce de manera considerable. Además en caso de ser muy grande la tabla, y crecer la tabla INDEX, se puede adicionar una nueva tabla INDEX Secundario, que acceda sobre la tabla INDEX y luego esta sobre la tabla secuencial.



Búsqueda binaria

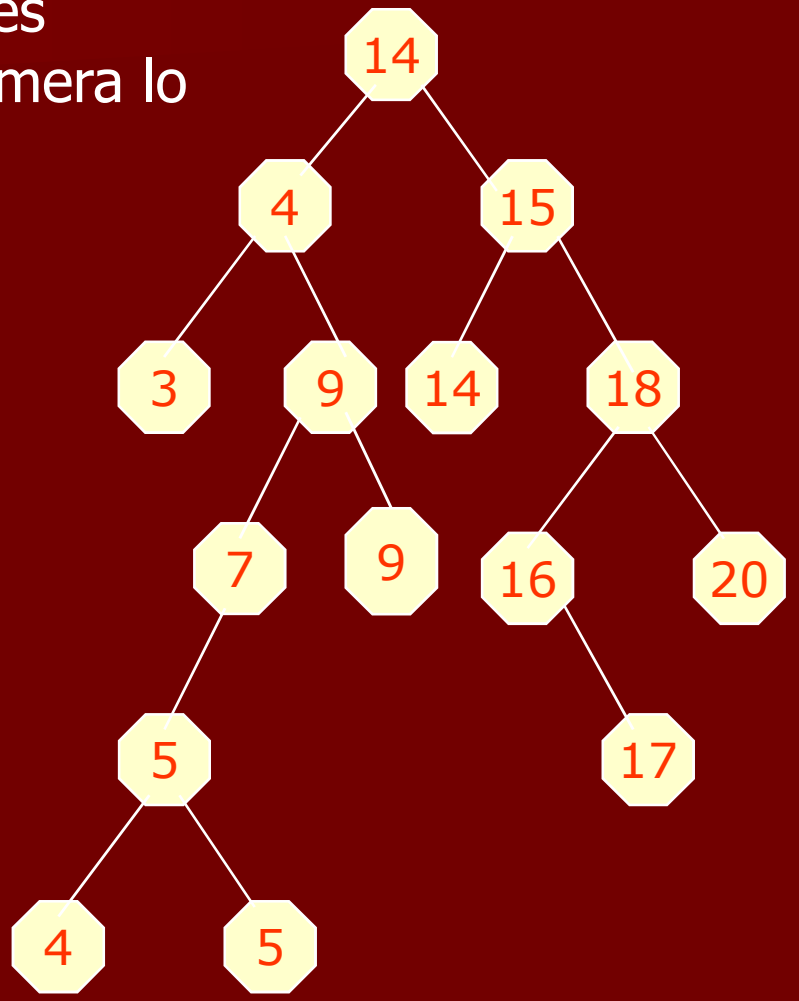
- El método más eficiente para buscar en una tabla secuencial sin usar tablas auxiliares, es el de búsqueda binaria.
- Este método reduce en un factor de 2 luego de cada comparación la cantidad de elementos a comparar.
- Este método se puede aplicar tanto para la tabla secuencial como para la tabla INDEX.
- Como dificultad podemos indicar que este método solo puede utilizarse para tablas organizadas como arreglos. Además en este tipo de tablas se hace muy dificultoso el manejo de inserciones y eliminaciones de elementos, lo cual hace muy poco común su uso.

ALICACIONES: Árboles Binarios

Muchas aplicaciones que utilizan árboles binarios proceden en 2 fases, en la primera lo construyen en la segunda lo recorren.

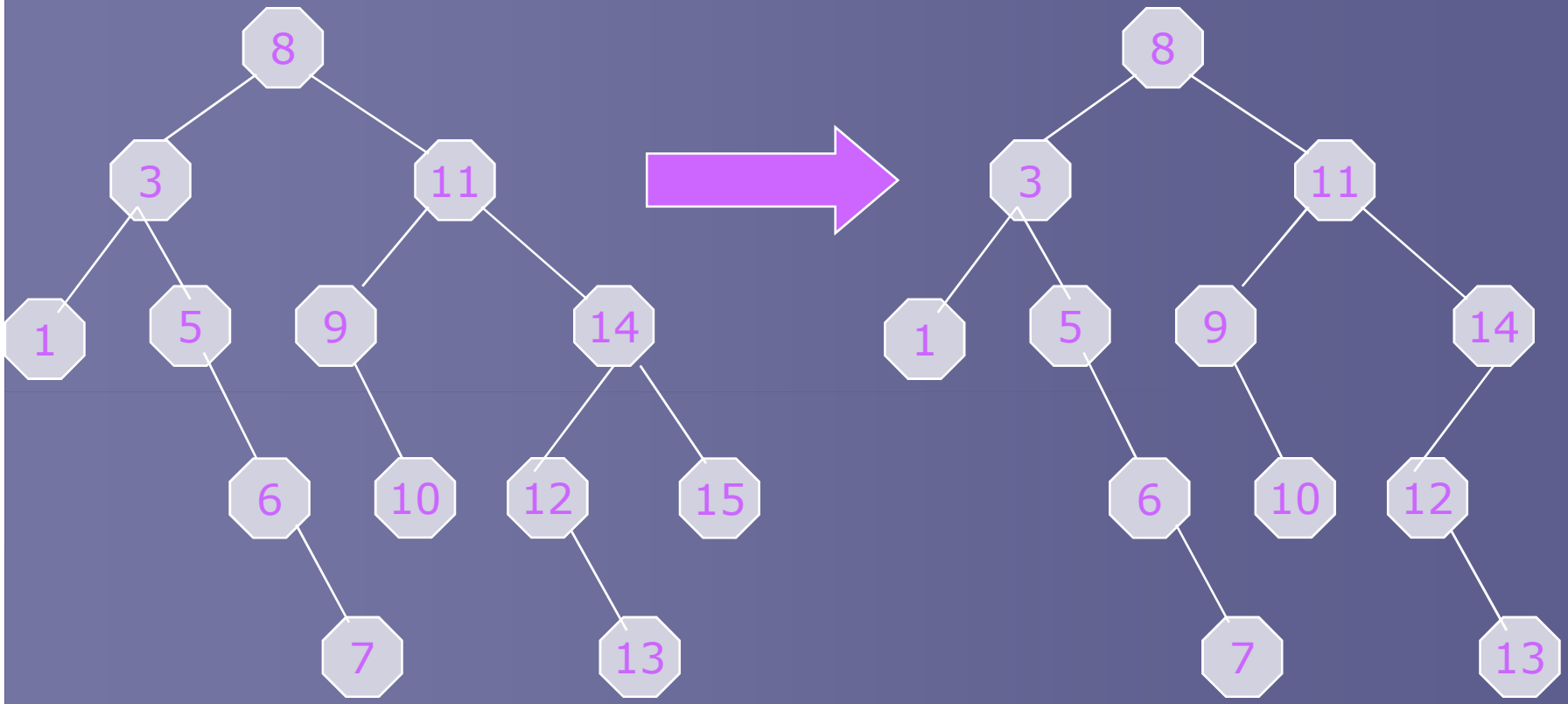
Supongamos que dada una lista de números quiero imprimirlos en orden ascendente.

Al ingresar los datos se cargan en un árbol binario, donde a la izquierda se ponen los números menores a la raíz, y a la derecha se ponen los números mayores o iguales a la raíz.



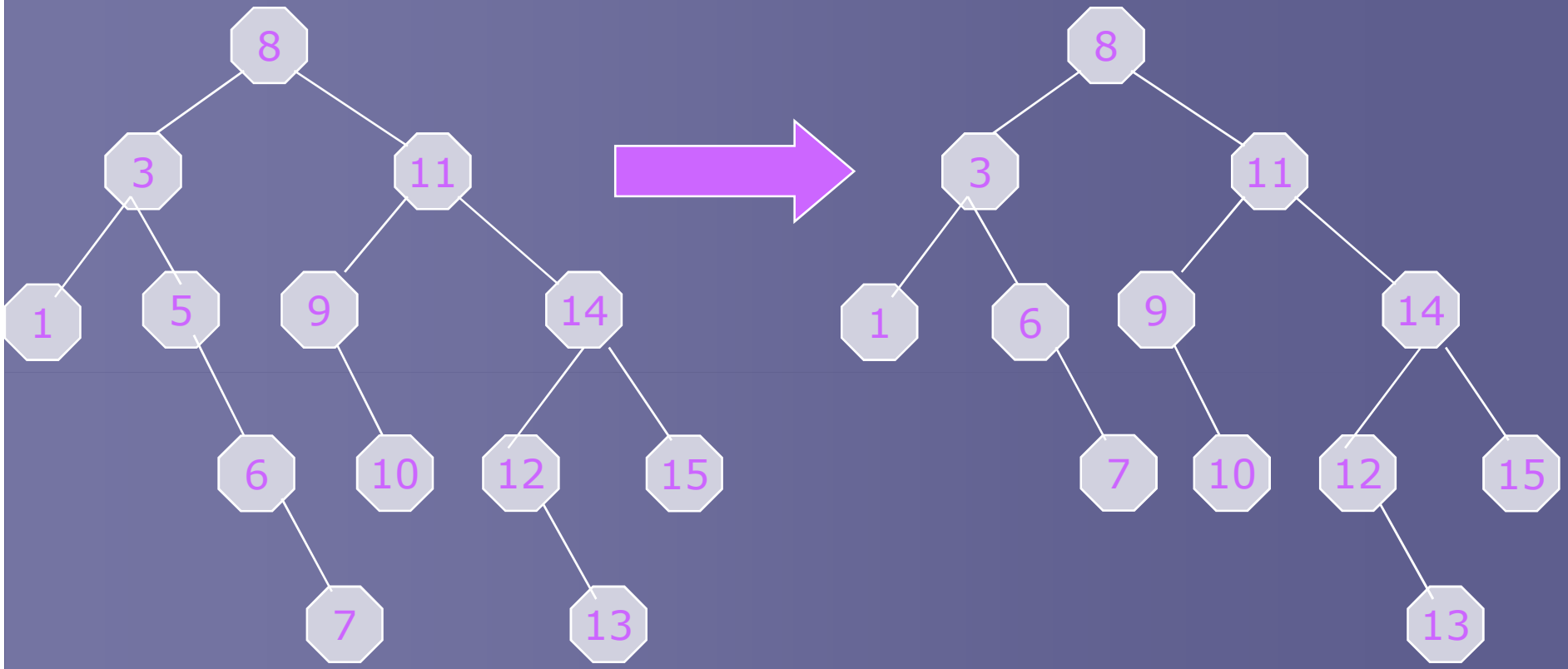
Datos Ingresados: 14 15 4 9 7 18 3 5 16 4 20 17 9 14 5

Búsqueda en árboles: eliminación de una llave



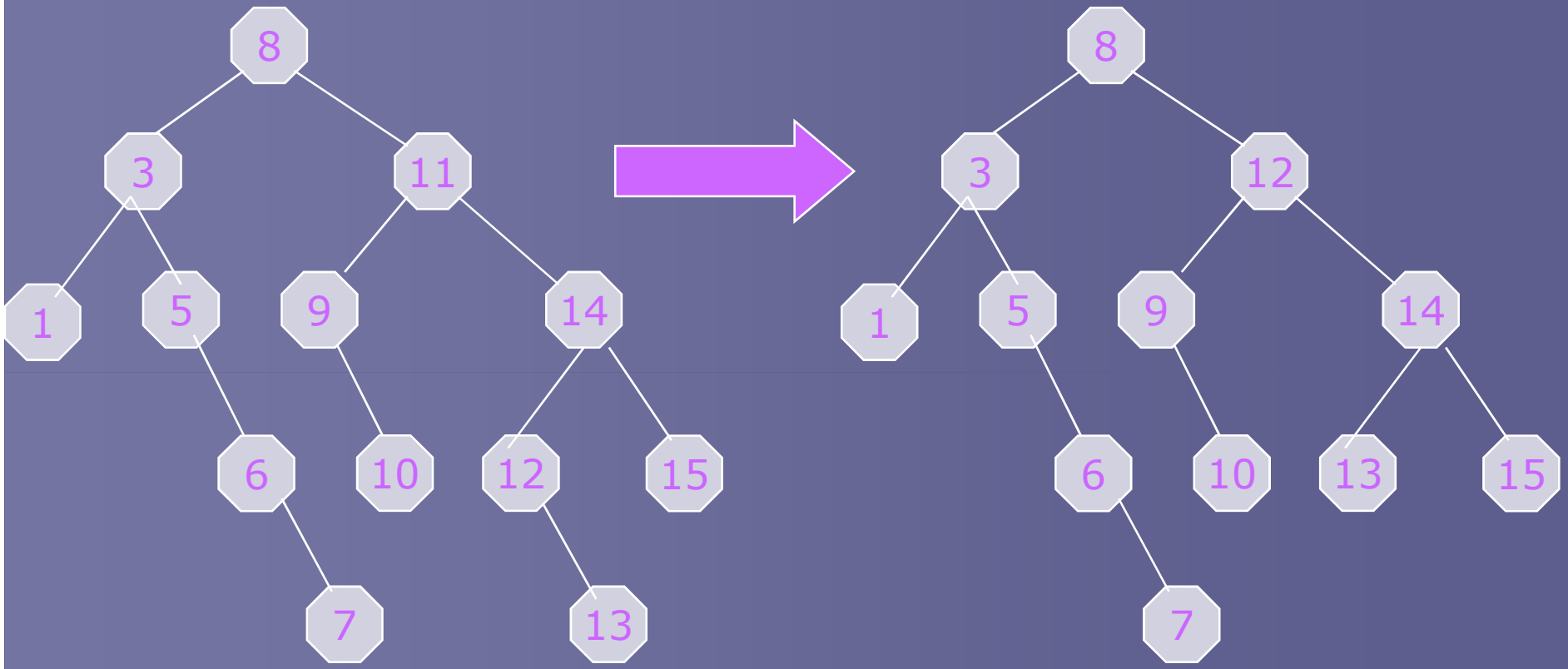
Eliminación del nodo con llave 15

Búsqueda en árboles: eliminación de una llave



Eliminación del nodo con llave 5

Búsqueda en árboles: eliminación de una llave



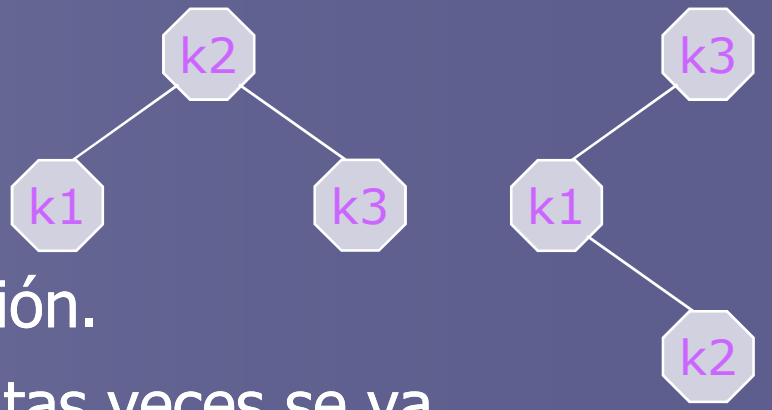
Eliminación del nodo con llave 11

Búsqueda en árboles

- Como vimos anteriormente, el ingresar los datos ordenadamente en un árbol es una operación del orden n log n.
- Aquí debemos considerar dependencias como el orden en que se ingresan los datos. (si los datos están ordenados, el ingreso es más lento y la búsqueda es de tipo secuencial)
- Si los datos se ingresan en forma aleatoria, lo más probable que el árbol se encuentre balanceado.

Eficiencia en Búsqueda en árboles

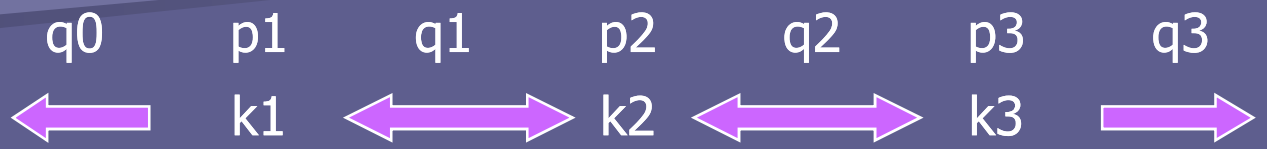
Supongamos 2 árboles:



En el primer caso recuperar k3, implica 2 comparaciones, en el segundo sólo implica una comparación.

Lo importante a considerar, es cuantas veces se va a buscar k1, k2, o k3, y cuantas veces la búsqueda será infructuosa.

Supongamos p1, p2 y p3 las probabilidades de buscar a k1, k2, k3, y q0, q1, q2, q3 las probabilidades de buscar valores diferentes de k1, k2, k3 en los rangos correspondientes.



Eficiencia en Búsqueda en árboles

Entonces p1 + p2 + p3 + q0 + q1 + q2 + q3 = 1

El número esperado de comparaciones está dado por la probabilidad de acontecimiento multiplicada por las comparaciones a realizar para encontrar la llave.

Entonces para árbol1: 2p1 + p2 + 2p3 + 2q0 + 2q1 + 2q2 + 2q3

en cambio para árbol2: 2p1 + 3p2 + p3 + 2q0 + 3q1 + 3q2 + q3

Este número esperado de comparaciones puede usarse como una medida de cuán "bueno" es un árbol para un conjunto dado de llaves y para un conjunto de probabilidades.

Así para las probabilidades de la izq. es mejor el árbol1, y para las de la derecha es mejor el árbol2.

p1 = 0,1	q0 = 0,1	p1 = 0,1	q0 = 0,1	p1 = 0,3	q0 = 0,025
p2 = 0,3	q1 = 0,2	p2 = 0,1	q1 = 0,1	p2 = 0,3	q1 = 0,025
p3 = 0,1	q2 = 0,1	p3 = 0,3	q2 = 0,1	p3 = 0,3	q2 = 0,025
	q3 = 0,1		q3 = 0,2		q3 = 0,025
Valor esperado arb1 : 1,7		Valor esperado arb1 : 1,9		Valor esperado arb1 : 1,7	
Valor esperado arb2 : 2,4		Valor esperado arb2 : 1,8		Valor esperado arb2 : 2,03	

Árboles de búsqueda óptimos

Un árbol de búsqueda binaria que minimice el número esperado de comparaciones para un conjunto dado de llaves y probabilidades se llama óptimo.

El algoritmo más rápido conocido para producir un árbol de búsqueda binaria óptimo es $O(n^2)$ en el caso general.

Esto es muy costoso a menos que el árbol se mantenga sin cambiar durante un largo número de búsquedas.

Sin embargo, aunque no existe un algoritmo eficiente para construir un árbol óptimo en el caso general, hay varios métodos de construcción de árboles cercanos a óptimos en tiempo de O(n)

Veamos algunos métodos:

Árboles de búsqueda óptimos: método de balanceo

SI p(i) = a la probabilidad de buscar la llave k(i)

SI q(i) = a la probabilidad de búsqueda infructuosa entre k(i-1) y k(i)

Defino s(i, j) = q(i) + p(i+1) +.....+ q(j)

Este método intenta encontrar i que minimice el valor absoluto de s(0, i-1) – s(i, n) y entonces establece k(i) como raiz del árbol, donde k(1) a k(i-1) sean el subárbol izquierdo y k(i+1) a k(n) sean el subárbol derecho.
Este proceso se aplica de manera recursiva para construir los subárboles izquierdo y derecho.

Puede demostrarse que el orden de realizar este trabajo es del orden n log n

Árboles de búsqueda óptimos: método exhaustivo

En lugar de construir el árbol de arriba hacia abajo como en el método anterior, este método construye el árbol de abajo hacia arriba.

Este método utiliza una lista doblemente ligada, con 4 punteros (izqy der dobles), un valor llave y 3 valores de probabilidad (izq der y de la llave)

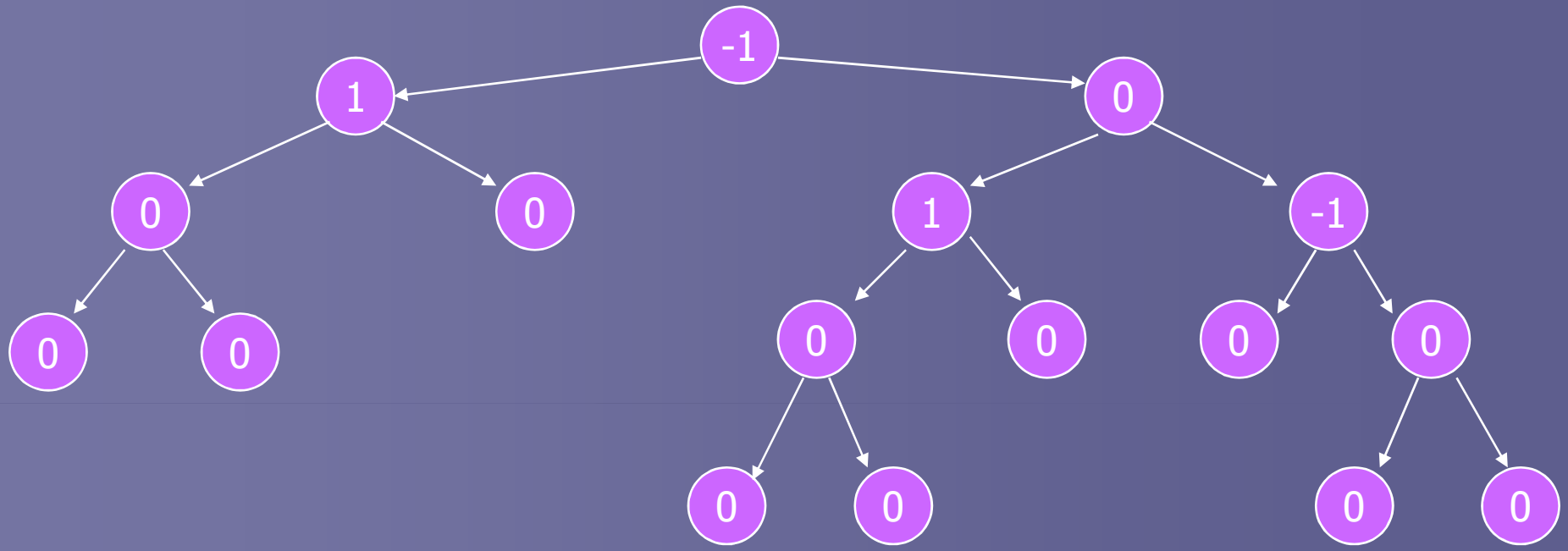
Árboles de búsqueda óptimos: método de división

Este método contiene 2 llaves en cada nodo, una con el valor de éxito (si coincide la búsqueda culmina) y otra con el valor de división, que utiliza la mediana como llave del nodo más frecuente para subdividir los árboles. Este requiere un tiempo de construcción del orden de n log n

Árboles balanceados

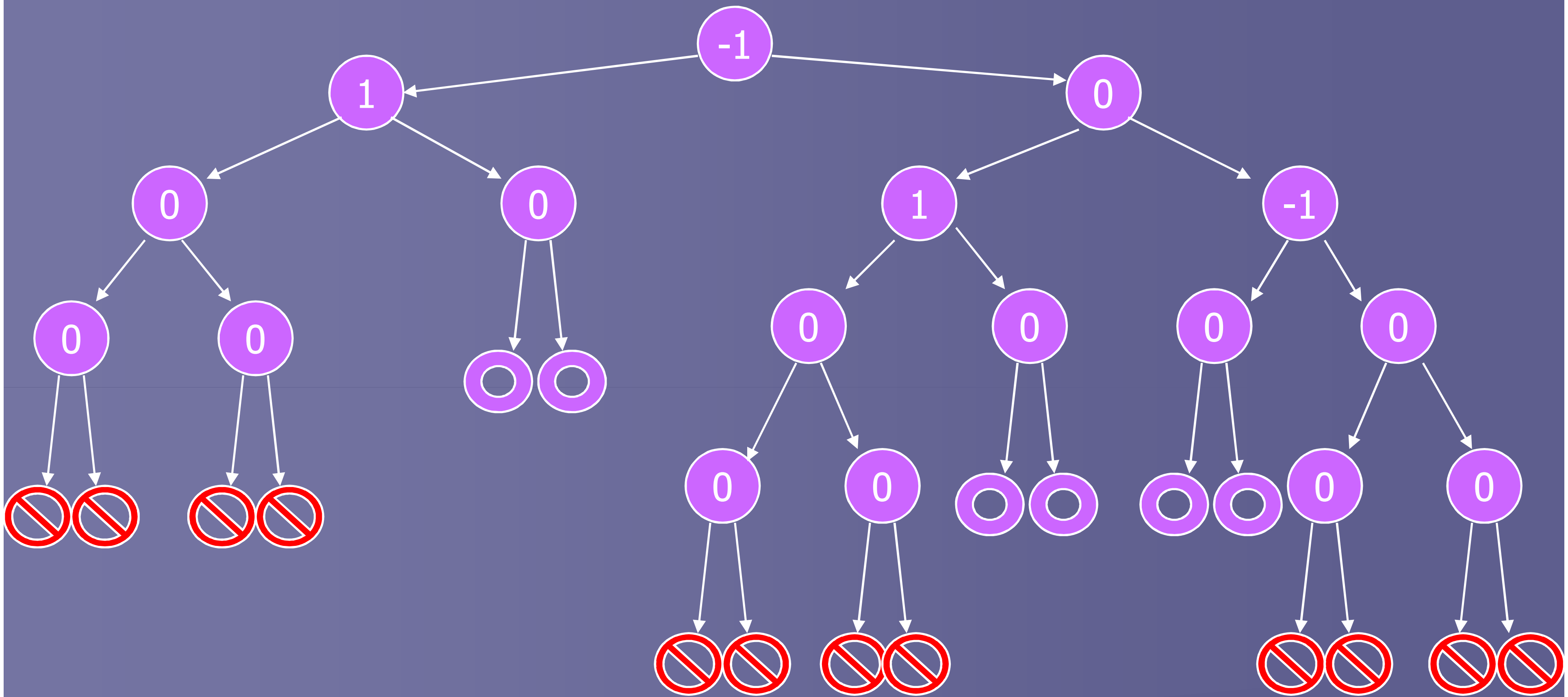
- Si la probabilidad de buscar una clave en una tabla es la misma para todas las llaves, la búsqueda más eficiente se efectúa en un árbol binario balanceado.
- La altura o profundidad de un árbol está dada por el nivel máximo de sus hojas.
- La altura de un árbol nulo se define como -1
- Un árbol binario balanceado es un árbol en el cual las alturas de los dos subárboles de todo nodo difieren a lo sumo en 1.
- El balance de un nodo se define como la altura del subárbol izquierdo menos la altura del subárbol derecho.
- Entonces, cada nodo de un árbol balanceado tiene balance igual a 1, 0 o -1, dependiendo de las alturas de los subárbol izquierdo o derecho.

Árboles balanceados



Si ha este árbol balanceado agregamos un nuevo nodo (o elemento), puede que el árbol permanezca balanceado o que se desbalancee.

Árboles balanceados

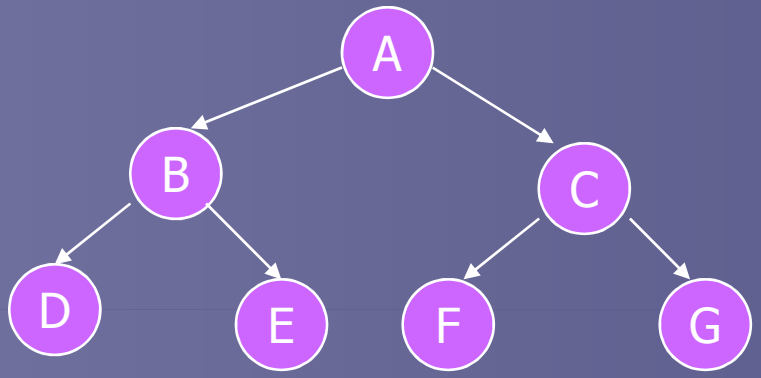


Los nodos indicados en rojo producen que el árbol se desbalancee, los indicados en rosa mantienen el árbol balanceado.

Árboles balanceados

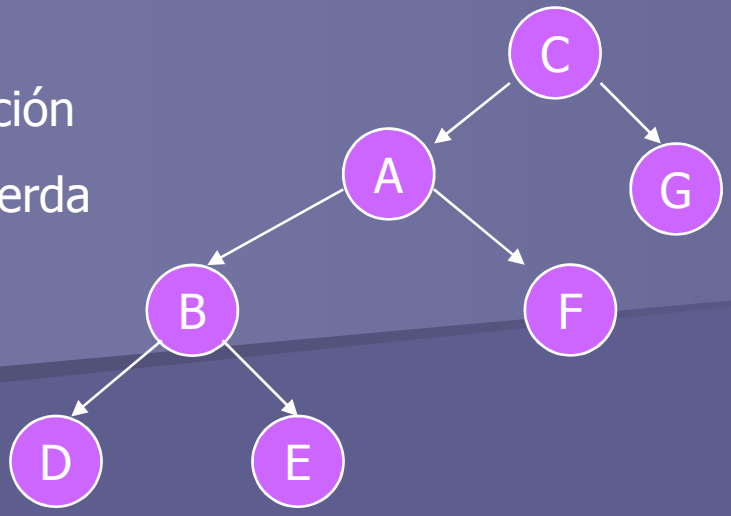
¿ Como vamos a mantener los árboles balanceados al agregar nuevos nodos ?

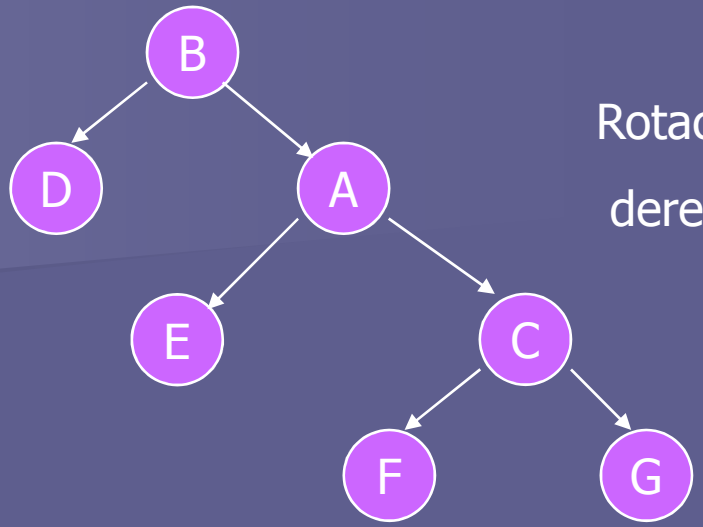
La primer técnica a manejar es la de rotación derecha e izquierda



Árbol Original

Rotación Izquierda





Rotación derecha

Árboles balanceados

- Esta técnica permite que un árbol que se encuentra balanceado y que al ingresar un nuevo nodo se va a desbalancear, el realizar una rotación izquierda o una derecha o ambas, pueda mantenerse balanceado.

- En la práctica, los árboles de búsqueda binaria balanceados se comportan mejor, produciendo tiempos de búsqueda de orden $\log_2 n$ para valores grandes de n. Además se requiere una rotación en el 46,5% de las inserciones.

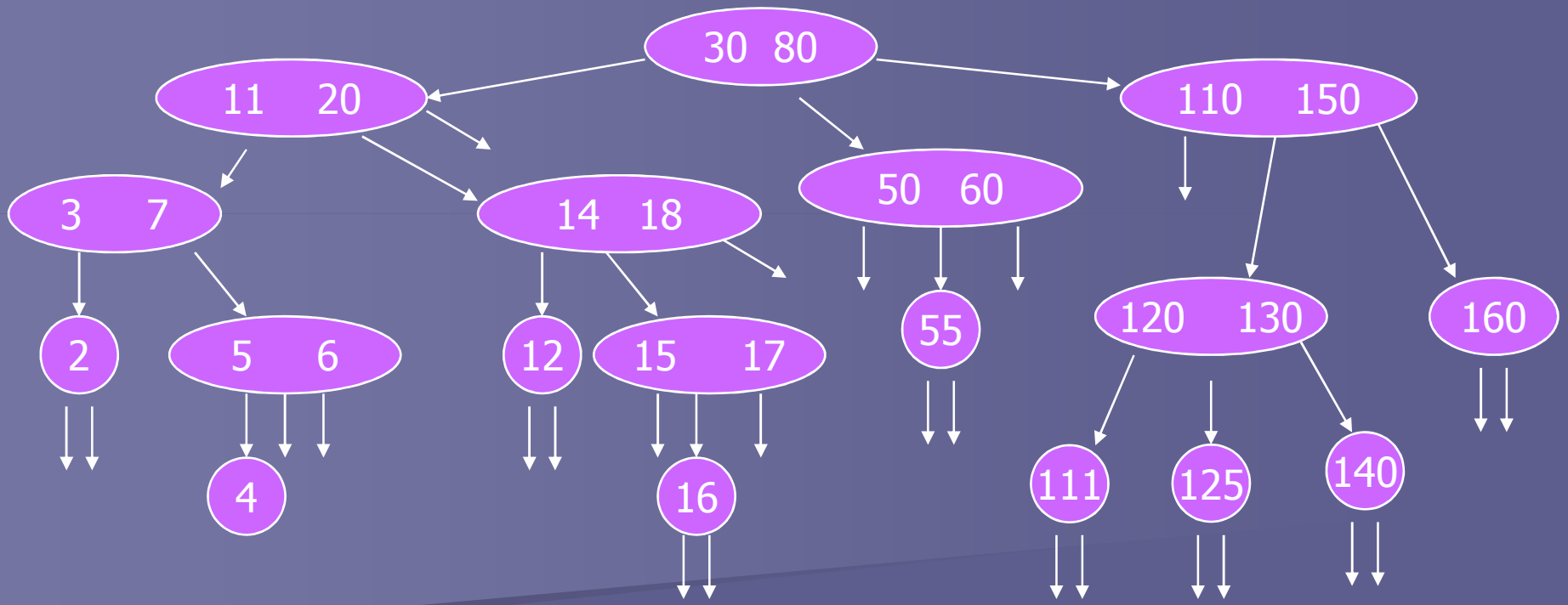
- El algoritmo para eliminar un nodo de un árbol balanceado es más complejo, y puede requerir una rotación doble o simple en cada nivel del árbol.

Árboles de búsqueda multivias

- Un árbol de búsqueda de accesos múltiples de orden N, es un árbol general en el cual cada nodo tiene n o menos subárboles y contiene una llave menos que la cantidad de subárboles.

- Uno o más subárboles de un nodo pueden estar vacíos.

Árboles multivías



Árboles B

- En los árboles B la técnica de inserción es más compleja, aunque esto se compensa con el hecho de crear árboles balanceados, de manera de reducir el número de nodos accesados para encontrar la llave es menor.

- Un árbol de búsquedas multivías balanceado de orden n en el cual cada nodo raíz contiene al menos n/2 llaves se llama árbol B de orden n

- Para mantener el árbol balanceado se intenta que los nodos de cada subárbol mantengan un número similar de llaves para cada subnivel, de manera que cuando el árbol tiende a desbalancearse se deben subdividir los nodos, y realizar rotaciones izquierdas y derechas de los elementos para mantener el árbol balanceado.