B+ Trees

Diferencias del B+ con el B-Tree:

- los nodos internos solo tienen los keys que actuan como routers para encontrar las hojas, dado que el las hojas se encuentran los datos correspondientes a cada key. por esta razón, hay keys duplicados en el B+
- los nodos hoja están linkeados con un double linked list para que el acceso secuencial sea rápido
- el insert y el delete son “más fáciles” que el B
- como los nodos internos solo tienen los keys, significa que cada nodo puede guardar más datos y por lo tanto el árbol termina teniendo menos altura que un B, por lo tanto operaciones más eficientes
- hay más…

Insert

Nodo a la izquierda: <

Nodo a la derecha: ≥

Se busca el nodo con el algoritmo típico del Btree hasta llegar a la hoja donde se debería insertar el nodo, porque la inserción siempre ocurre en las hojas.

Se ubica la ubicación correcta del key dentro del nodo hoja, si el nodo tiene más que el número máximo de $keys$, entonces hacer un split

Split en un leaf

Se crean dos nodos:

1. Los que están desde el índice 0 hasta $\lfloor \frac{count}{2} \rfloor$ - 1
2. Los que están desde $\lfloor \frac{count}{2} \rfloor$ hasta $count$

El nodo en la posición $\lfloor \frac{count}{2} \rfloor$ sube al padre en la posición correcta y se reajustan los punteros a los hijos

Mientras la recursión se va desenrollando, revisar si el nodo interno tiene más que el número máximo de $keys$, si es así se hace un **split de nodo interno**

Split en un nodo interno

Se crean dos nodos:

1. Los que están desde el índice 0 hasta $\lfloor \frac{count}{2} \rfloor - 1$
2. Los que están desde $\lfloor \frac{count}{2} \rfloor + 1$ hasta $count$

El nodo en la posición $\lfloor \frac{count}{2} \rfloor$ sube al padre en la posición correcta y se reajustan los punteros a los hijos

> ❗ Notar que el **split de nodo interno** difiere del **split de un leaf**, porque en el **split del nodo interno**, el nodo del medio **sube** y no se queda abajo

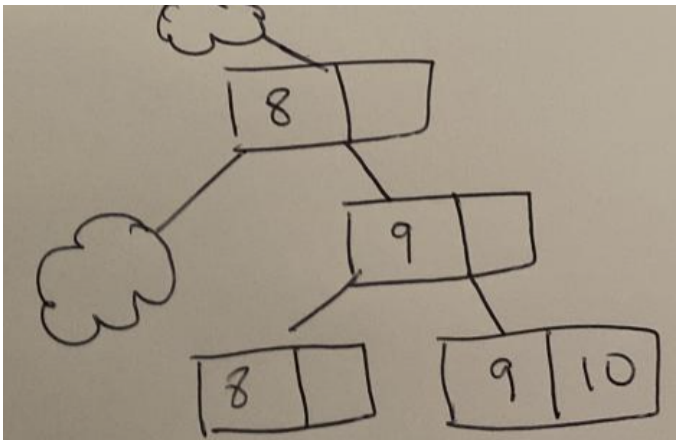
Delete

Ubicar el nodo hoja donde está el key que se quiere borrar y mantener un puntero al nodo interno donde está ese key si aplica el caso

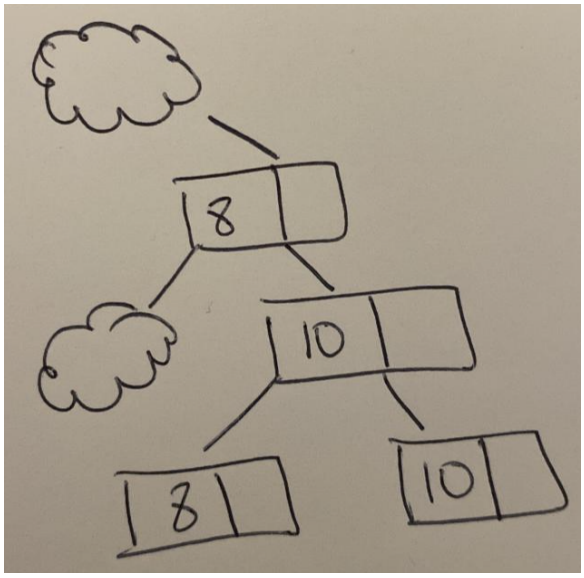
Borrar el key del nodo hoja

Si el nodo tiene más $keys$ que el número mínimo de $keys$

> Si el puntero al nodo interno es distinto a nullptr, entonces copiar el key al lado (a la derecha, osea el **sucesor**) del key que acabamos de borrar y ponerlo en el lugar del key del puntero al nodo interno:
>
> Por ejemplo, borremos el 9



Antes de borrar el 9.



Luego de borrar el 9. Notar que “el nodo de la derecha”, osea el sucesor de 9 es 10.

Sino

Pedirle prestado al hermano inmediato izquierdo o derecho si tienen más del número mínimo de $keys$

Prestar a través del padre:

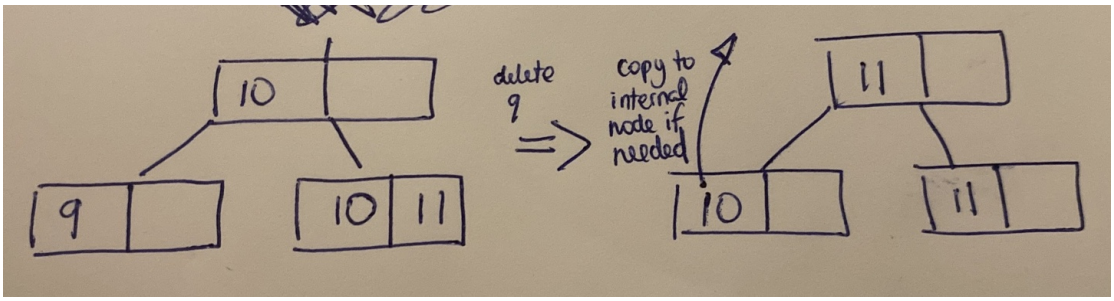
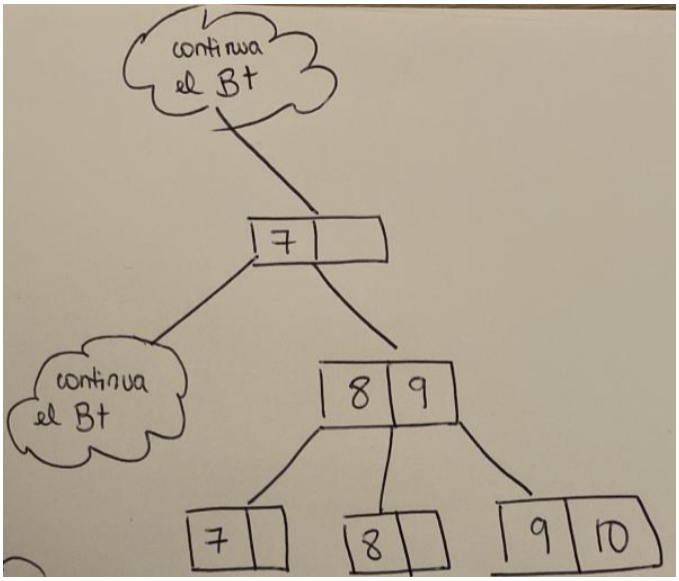


Ilustración del préstamo del hermano inmediato derecho **a través del padre.** Notar que el *key* prestado tiene que chancar el *key* si está tmb presente en un nodo interno.

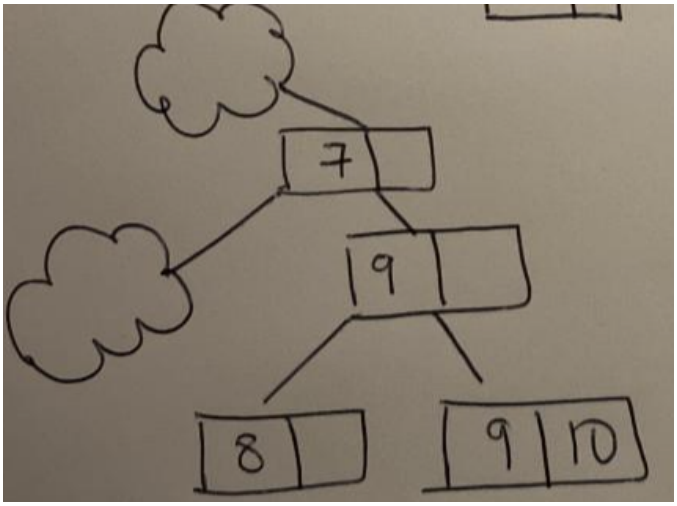
Sino (ambos hermanos inmediatos tienen el mínimo número de $keys$)

El key padre baja al nodo del key a borrar y luego se hace merge con el hermano inmediato izquierdo o derecho y luego se borra el key y cualquier key que esté repetido (solo habrá como máximo 1 repetición). Copiar el key del padre y ponerlo en el lugar del key del puntero al nodo interno:

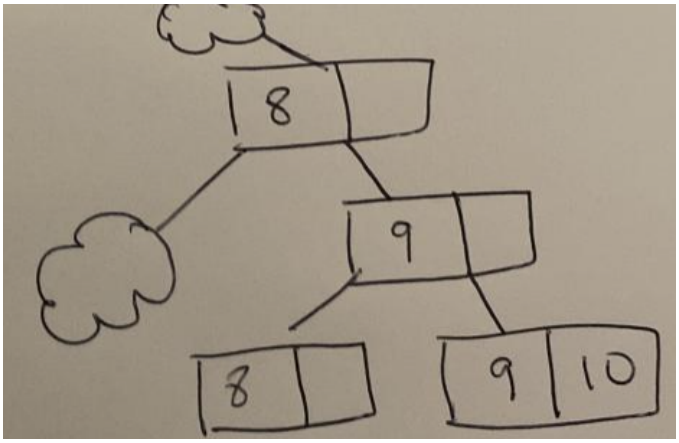
Por ejemplo, del siguiente B+ borremos el 7



Antes de borrar.



El padre **8** bajó y por lo tanto el nodo se convirtió en 7 8**,** luego se hizo merge con el nodo derecho que dio nacimiento del nodo 7 8 8, borramos el 7 y el repetido quedando el nodo 8. Además, como bajó el padre **8**, se reordenaron los punteros en ese nodo padre.



Ya que el 7 estaba también presente en nodos internos, se copió el valor del 8 al lugar del 7.

Luego **mientras la recursión se desenrolla**, revisar si el padre (en el ejemplo de arriba el nodo 9) tiene menos del mínimo número de $keys$ y si es así, replicar el proceso de pedir prestado al heramano inmediato izquiero o derecho, sino hacer merge