B-Trees Rules for Deletion Summary:
Source: Cormen Introduction to Algorithms 3rd edition

Rule 1: If the key *k* is part of a leaf node *x*, then just delete the key.

Rule 2A: If the key *k* belongs to an internal node *x*.

If the child *y* that precedes *k* in a node *x* has at least *t* keys, then find the predecessor *k'* of *k* in the subtree rooted at *y*. Recursively delete *k'* and replace *k* by *k'* in *x*.

Rule 2B: If the key *k* belongs to an internal node.

If *y* has fewer than *t* keys, then, symmetrically, examine the child *z* that follows *k* in node *x*. If *z* has at least *t* keys, then find the successor *k'* of *k* in the subtree rooted at *z*. Recursively delete *k'* and replace *k* by *k'* in *x*.

Rule 2C: If the key *k* belongs to an internal node *x*.

Otherwise, if both *y* and *z* have only *t – 1* keys, merge *k* and all of *z* into *y*, so that *x* loses both *k* and the pointer to *z*, and *y* now contains *2t -1* keys. Then free *z* and recursively delete *k* from *y*.

Rule 3A: If the key *k* is not part of the internal node *x*, take $x.c_i$ the root of the subtree that must contain *k* (if *k* is in the tree). If $x.c_i$ has only *t-1* keys,

If $x.c_i$ has an immediate sibling with greater than or equal to t keys, then give $x.c_i$ an extra key by:

- Moving a key from x to $x.c_i$
- Moving a key from $x.c_i$'s immediate left or right sibling up x
- Moving the appropriate child pointer from the sibling into $x.c_i$

Rule 3B: If the key *k* is not part of the internal node *x*, take $x.c_i$ the root of the subtree that must contain *k* (if *k* is in the tree).

If both $x.c_i$'s immediate sibling have *t – 1* keys, merge $x.c_i$ with one sibling, which involves moving a key from *x* down into the new merged node to become the median for that node