\[ a \rightarrow ab \rightarrow abc \]

\[ \text{Split} \]

\[ \ldots w \ y \ldots \]

\[ \text{Full node} \]

\[ 2t-1 \text{ tags} \]

\[ 2t \text{ children} \]

\[ t = 2 \]

\[ 1, 2 \text{ or } 3 \text{ tags} \]

\[ 2, 3, \text{or } 4 \text{ children} \]

\[ \sum \min \text{ size allowed!} \]
abc \[d\] ? not balanced
can't do this!

Insert:
Follow path to the leaf where the new item will be inserted, split any full node encountered on the way.
Split (+ Merge)

Insertion

Split

Merge

Delete

2t-1 elements

2t children

Full + we want to split

Roam for growth
Top-Down Insertion

Follow path to leaf where you'd insert (with natural extension of binary search tree insertion)

Whenever a full node (2t children) encountered Split it & then Continue

Goal: minimize possibility of a page fault occurring twice on same page
EHQTU → EHIJK TU

→ FQ → EH JK TV

Insert 2

t=3

5 elements is a full node

c1 c2 c3 c4 c5 [TU2]

top down

bottom-up
Bottom-up Insertion

Do standard insertion in leaf if there is room.

Otherwise split leaf (which could propagate to the root). Stop as soon as parent is not full.

Reduces unnecessary splits.
Overview of B-Tree Deletion

Like binary search tree, for removing element in an internal node, then replace $x$ by its successor & remove the successor from marked node take leftmost child until reach leaf & succ. leftmost element

*hold this in memory until successor is found to replace it*
Focus on removing an element in a leaf.

Leaf $cdmx$ is not minimum size.

Diagram showing the process of removing an element and merging nodes.
not minimum-sized

Something here (min sized)
Basic Flow for B-tree deletion

Top-down

On search to leaf (for element to delete or its successor)

If a minimum-sized node is encountered "fix that" by

1. merging or 2. shift left/right

Ensures that the leaf holding element to remove is not min sized
Pictorial Summary of B-Tree Deletion

Key idea is to ensure as you move down the tree that for the nodes visited (i.e. those on the path from the root to the node with the key to delete), the number of keys is always at least one more than the minimum allowed (i.e. at least $1 + (t-1) = t$) with the exception of the root.

Repeat until Case 1 is reached:

Let $x$ be the current node when searching for the node with key $k$ to delete.

Case 1: $x$ is a leaf (which must then contain $k$). Then just delete $k$.

Case 2: $k$ is in $x$ where $x$ is not a leaf

Case 2a: at least $t$ keys in $y$

Case 2b: at least $t$ keys in $z$

Case 2c: $t-1$ keys in $y$ and $z$

Combine $y$ and $z$ into one node

Case 3: Node we want to go to next has only $t-1$ keys

Case 3a: at least $t$ keys in $y$

Case 3b: $t-1$ keys in $y$

Combine $y$ and $z$ into one node

Case 3c: $t-1$ keys in $y$ and $z$

Combine $y$ and $z$ into one node

Now delete $k$ starting at node $x$ (If $k$ was the root of size 1 then the height shrinks).

In both cases you now continue the search for the desired key continuing from $x$.