B-Trees, Part II

Review of Properties

Let $t$ be the order of the B-tree (parameter given to constructor)

**BALANCED** - all leaf nodes are at same height. So an internal node with $x$ elements has $x+1$ non-empty children

**NODE UTILIZATION** - With the exception of root all nodes have $\geq t$ children (+ so $\geq t-1$ elements). The root has $\geq 2$ children. All nodes have $\leq 2t$ children (+ so $\leq 2t-1$ elements)
Split (+ Merge)

**Insertion**
- **Split**: When a node is full, it splits into two nodes.
- **Merge**: After a split, the parent node merges with one of the child nodes.

**Delete**
- When an element is deleted, the node may become empty or underfull.

**2t-1 elements**
- Each node can have up to 2t-1 elements.

**2t children**
- Each node has 2t children.

**Full + we want to split**
- If a node is full and we want to insert, we split it.

**Roam for growth**
- If a node becomes empty, we roam for growth to find a parent node that can accommodate the empty node.
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
\[ \text{EHQTEU} \rightarrow \text{EHJIKTU} \]

\[ \rightarrow \text{FIQ} \rightarrow \text{EHJKTV} \]

\[ \boxed{t=3} \]

5 elements is a full node

Insert Z

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.
Relation between red-black tree insertion & 2-3-4 bottom-up insertion

Insert L

Red node is part of its parents DHU node

AC E KNP W

Insert L bottom-up

Last step make H black
Analysis of height

Suppose the B-tree has height \( h \)

What is the min # of elements it might hold?

\[ n \geq f(h) \quad \text{solve for } h \quad h \leq f(n) \]
The diagram illustrates the relationship between the order of the tree and the number of elements and nodes at different levels.

- **Level 0:**
  - Elements: $1$
  - Nodes: $2(t-1)$

- **Level $h$:**
  - Elements: $\frac{2t}{h-1} \cdot (t-1)$
  - Nodes: $2t^2(t-1)$

- **Level $h-1$:**
  - Elements: $2t^{h-1}(t-1)$
  - Nodes: $2t^h(t-1)$
\[ n \geq 1 + 2(t-1) \left(1 + t + t^2 + \cdots + t^{h-1}\right) \]

\[ n \geq 1 + 2(t-1) \left(\frac{t^h - 1}{t-1}\right) \]

\[ n \geq 1 + 2(t^h - 1) \]

\[ 2t^h \leq n + 1 \]
$2t^h \leq n+1$

$t^h \leq \frac{n+1}{2}$

$h \leq \log_t (\frac{n+1}{2})$

$\log_t \left( \frac{n+1}{2} \right) = \log_t (n+1) - \log_t 2$

$\sim \log_t n$

Maximum height of B-tree with $n$ elements
Cost for insertion

$\Theta \left( \log_t n \cdot \log_2 (2t+1) \right) = O(\log_2 n)$

- # nodes on path down
- time per node (sorted array)
- Max # of page faults
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 \( \text{cmd} \times \) not minimum sized

\[ \frac{t-3}{2-5} \]

Elements

Diagram:
- min size
- merge
- \( dx \times yz \)
- \( \text{abc} \)
max sizing

empty

ShiftRight
ShiftLeft

delete

Something here
(not minimum-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
B+ tree

$t = 2$ For letters “abstraction”

Internal purely for navigation

Keep a copy in leaf

All elements in leaves & they are linked together in a doubly-linked chain

Insert p

```
no
pr
```