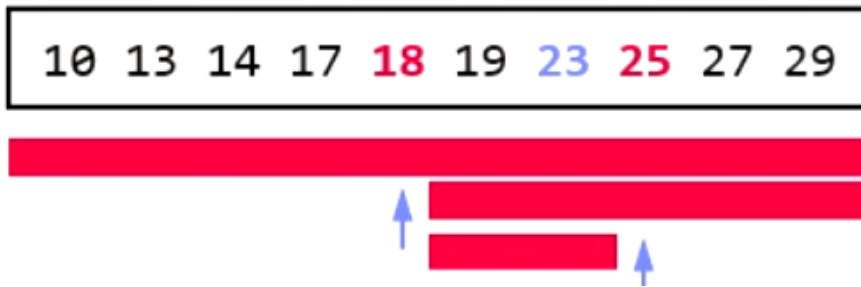


Binary Search

- `pos = binarySearch(data, key=23)`
- find key position within sorted data



-
- optimizations
 - k-ary search for SIMD data-parallelism
 - * ?
 - interpolation search: probe expected pos in key range
 - * e.g. search for “Bastian” in telephone book, don't start in the middle but rather at the beginning

BTree

- self balancing tree
- individual nodes stored as pages
 - [[Background Storage System]]
- each node contains data or reference to data
 - values sorted within node

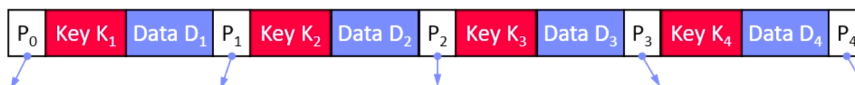
Definition B-Tree (k, h)

- All paths from root to leafs have equal length h
- All nodes (except root) have $[k, 2k]$ key entries
- All nodes (except root, leafs) have $[k+1, 2k+1]$ successors
- Data is a record or a reference to the record (RID)

$$\lceil \log_{2k+1}(n+1) \rceil \leq h \leq \left\lceil \log_{k+1} \left(\frac{n+1}{2} \right) \right\rceil + 1$$

All nodes adhere to max constraints

k=2

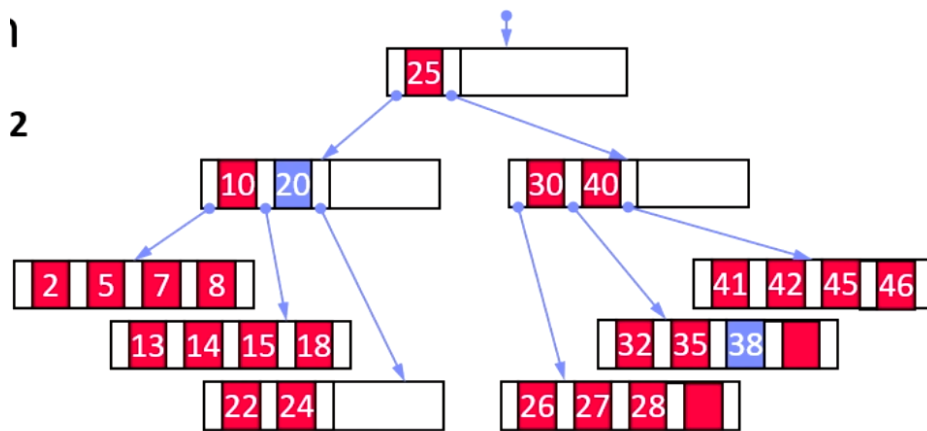


Subtree w/
keys $\leq K_1$

Subtree w/
 $K_2 < \text{keys} \leq K_3$

+ pointer left/right of value

points to leaf with smaller/bigger values



▪ **Lookup Q_K within a node**

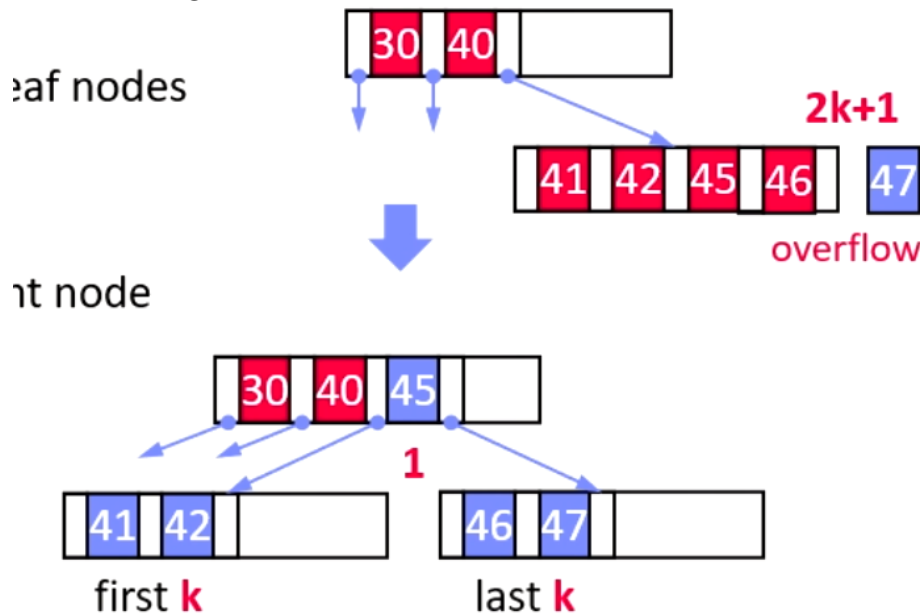
- Scan / binary search keys for Q_K , if $K_i = Q_K$, return P_i
- If node does not contain key
 - If leaf node, abort search w/ NULL (no more keys)
 - Decent into subtree P_i with $K_i < Q_K \leq K_{i+1}$

▪ **Range Scan $Q_L < K < Q_U$**

- Lookup Q_L and call next K while $K < Q_U$ (keep keys)

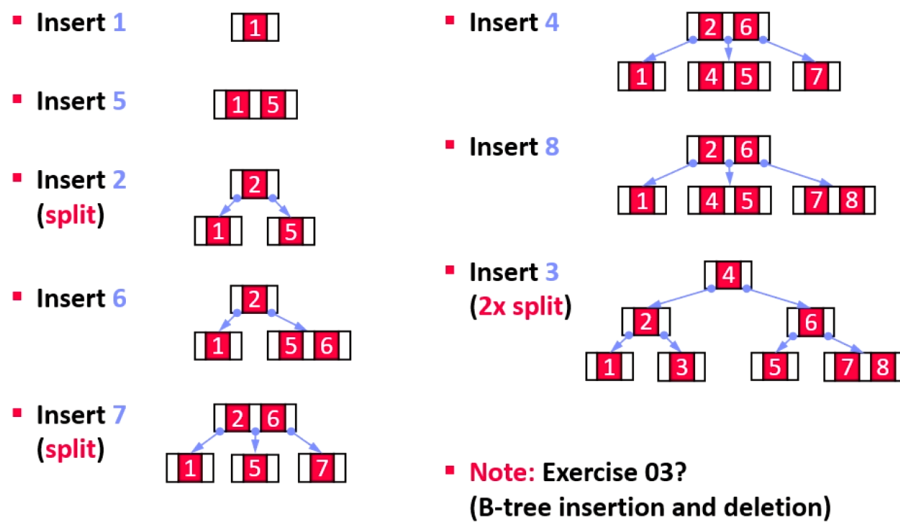
B-Tree Insert

- always insert into leaf nodes
- if node overflows (exceeds $2k$ entries) \implies node splitting
- node splitting
 - split into two leaf nodes
 - left node with first k entries
 - right node with last k entries
 - $(k+1)$ th entry inserted into parent node
 - * may cause recursive splitting
- self-balancing



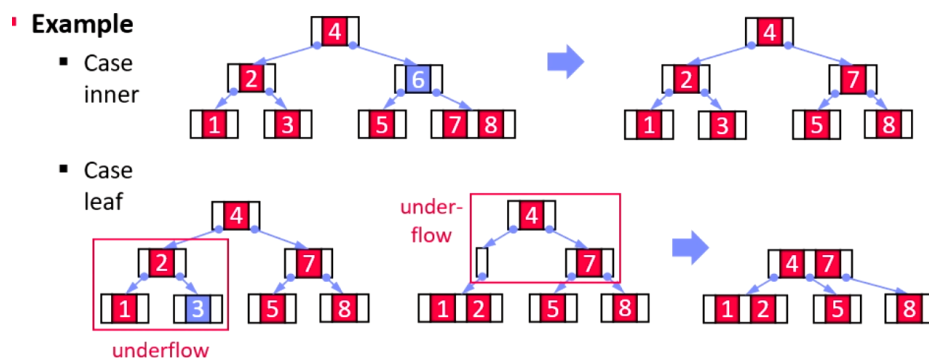
-
- Example

B-Tree Insert, cont. (Example w/ $k=1$)



B-Tree Delete

- deletion might cause underflow ($< k$ entries)
 - underflow on inner node
 - * \implies move entry from fullest successor (node below) into inner node
 - underflow on leaf node
 - * \implies merge with sibling
- example

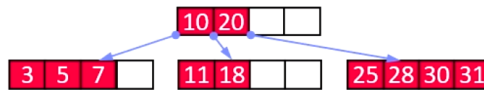


B-Tree Insert and Delete Example

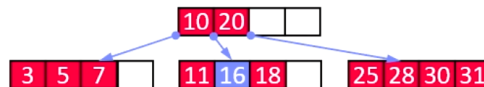
B-Tree Insert and Delete w/ $k=2$

Insert/Delete Examples

- Original



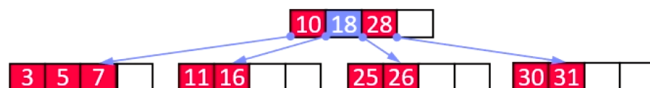
- Insert 16



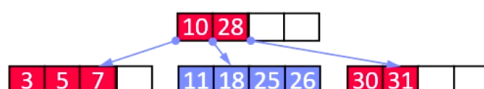
- Insert 26



- Delete 20



- Delete 16



Prefix Tree

Excursus: Prefix Trees (Radix Trees, Tries)

Generalized Prefix Tree

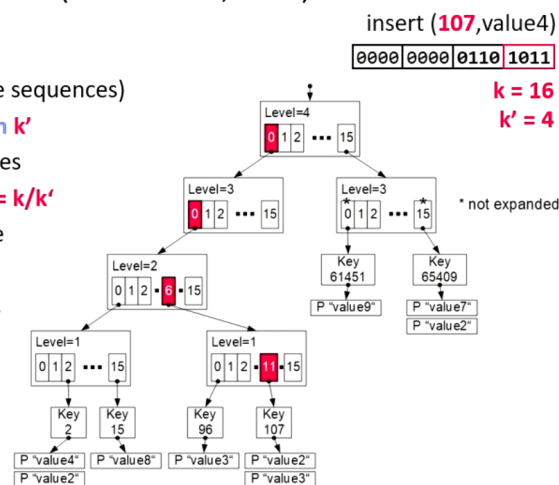
- Arbitrary data types (byte sequences)
- Configurable prefix length k'
- Node size: $s = 2^{k'}$ references
- Fixed maximum height $h = k/k'$
- Secondary index structure

Characteristics

- Partitioned data structure
- Order-preserving (for range scans)
- Update-friendly

Properties

- Deterministic paths
- Worst-case complexity $O(h)$



Excursus: Learned Index Structures

■ A Case For Learned Index Structures

- Sorted data array, predict position of key
- **Hierarchy of simple models** (stages models)
- Tries to **approximate the CDF** similar to interpolation search (uniform data)

[Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, Neoklis Polyzotis: The Case for **Learned Index Structures**. SIGMOD 2018]

