Binary Search

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

10 13 14 17 **18** 19 23 **25** 27 29
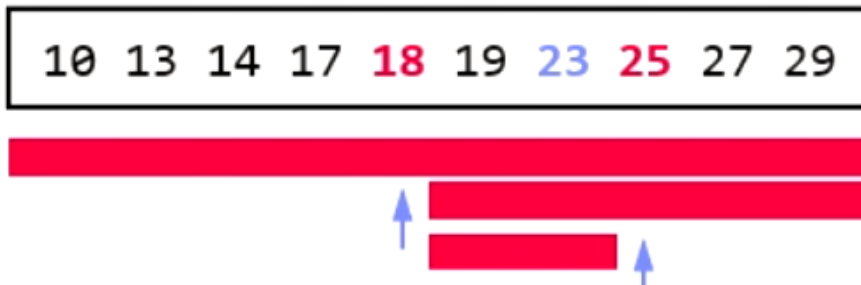
-
- optimizations
  - k-ary search for SIMD data-parallelism
    * ?
  - interpolation search: probe expexted pos in key range
    * e.g. search for "Bastian" in telephone book, dont start in the middle but rather at the beginning
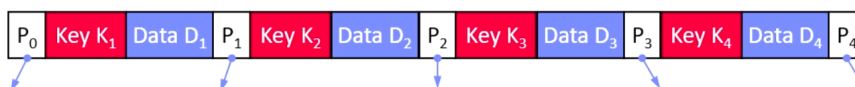
BTree

- 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)

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

All nodes adhere to max constraints

k=2

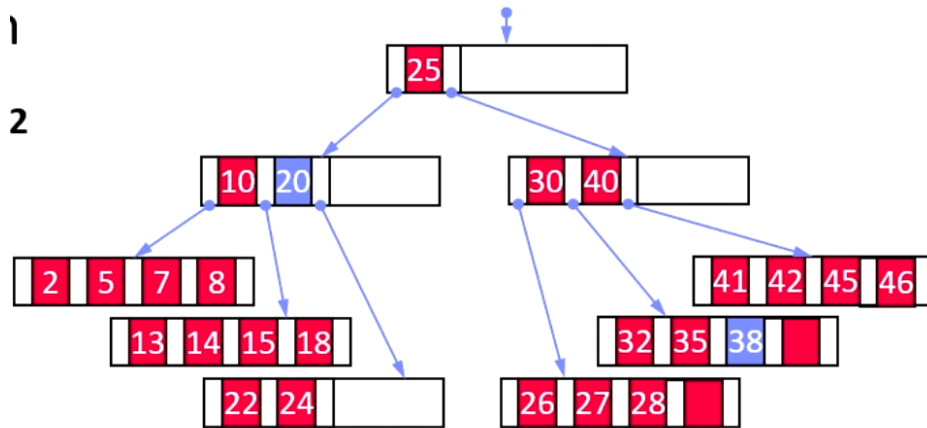| P_0 | Key K_1 | Data D_1 | P_1 | Key K_2 | Data D_2 | P_2 | Key K_3 | Data D_3 | P_3 | Key K_4 | Data D_4 | P_4 |

Subtree w/ keys ≤ K_1

Subtree w/ K_2 < keys ≤ K_3

+ pointer left/right of value points to leaf with smaller/bigger values

2

25

10 20   30 40

2 5 7 8   41 42 45 46

13 14 15 18   32 35 38

22 24   26 27 28

- **Lookup Q_K within a node**
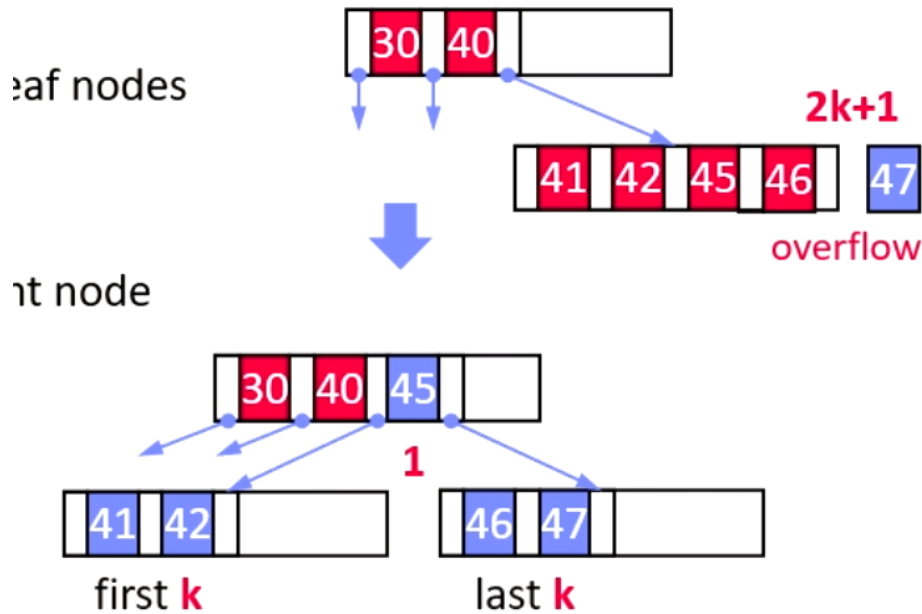 - Scan / binary search keys for Q_K, if $K_i = Q_K$, r
  - If node does not contain key
    - If leaf node, abort search w/ NULL (n
    - Decent into subtree Pi with $K_i < Q_K \leq$
- **Range Scan $Q_{L<K<U}$**
  - Lookup Q_L and call next K while $K < Q_U$ (kee

B-Tree Insert

- always insert into leaf nodes
- if node overflows (exceeds 2k entries) ==> 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
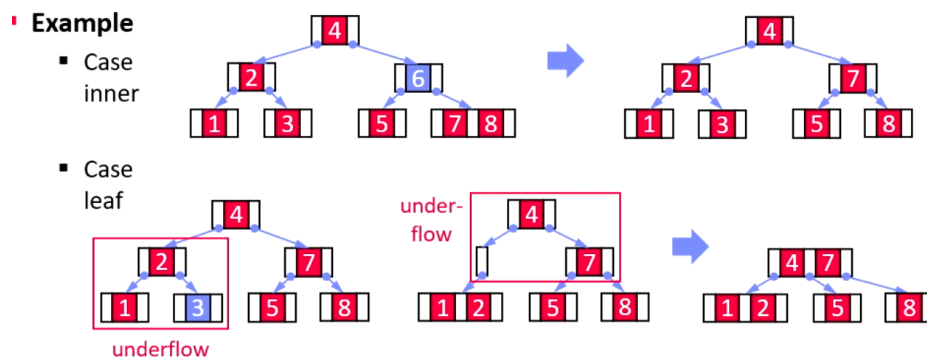   * may cause recursive splitting
- self-balancing

eaf nodes

30 40

41 42 45 46   47

2k+1

overflow

it node

30 40 45

1

41 42   46 47

first **k**   last **k**

-
- Example

2

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

- **Insert 1** `1`

- **Insert 5** `1 5`

- **Insert 2**
  (split)
  `2`
  `1`   `5`

- **Insert 6**
  `2`
  `1`   `5 6`

- **Insert 7**
  (split)
  `2 6`
  `1` `5` `7`

- **Insert 4**
  `2 6`
  `1` `4 5` `7`

- **Insert 8**
  `2 6`
  `1` `4 5` `7 8`

- **Insert 3**
  (2x split)
  `4`
  `2` `6`
  `1` `3` `5` `7 8`

- **Note: Exercise 03?**
  (B-tree insertion and deletion)
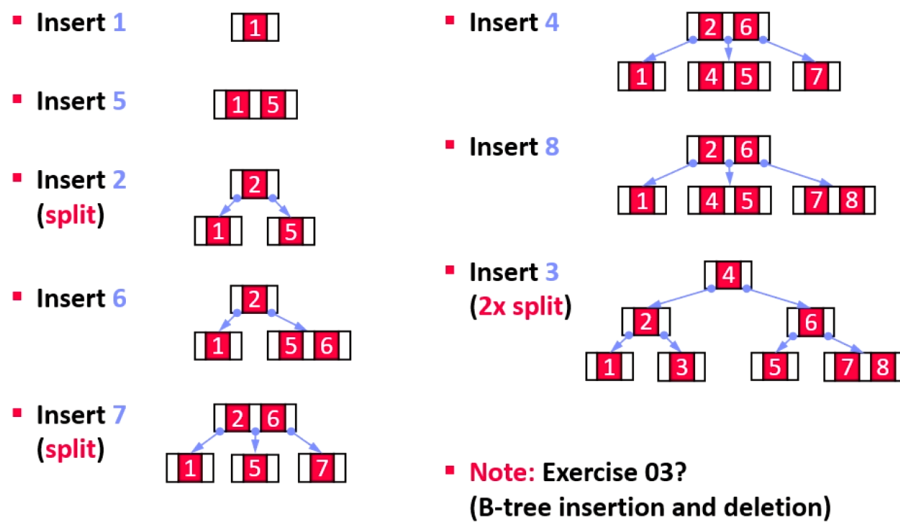
B-Tree Delete

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

  - **Example**
    - Case inner

      `4`
      `2` `6`
      `1` `3` `5` `7 8`
      ==>
      `4`
      `2` `7`
      `1` `3` `5` `8`

    - Case leaf

      `4`
      `2` `7`
      `1` `3` `5` `8`
      underflow

      under-flow
      `4`
      ` ` `7`
      `1 2` `5` `8`
      ==>
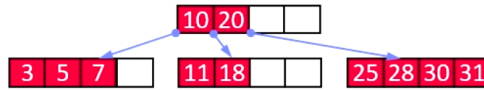   `4 7`
      `1 2` `5` `8`

3

B-Tree Insert and Delete Example

B-Tree Insert and Delete w/ k=2

- **Insert/Delete Examples**
  - Original

    | 10 | 20 | | |

    | 3 | 5 | 7 | | | 11 | 18 | | | | 25 | 28 | 30 | 31 |

  - Insert 16

    | 10 | 20 | | |

    | 3 | 5 | 7 | | | 11 | 16 | 18 | | | 25 | 28 | 30 | 31 |

  - Insert 26

    | 10 | 20 | 28 | |

    | 3 | 5 | 7 | | | 11 | 16 | 18 | | | 25 | 26 | | | | 30 | 31 | | |

  - Delete 20

    | 10 | 18 | 28 | |

    | 3 | 5 | 7 | | | 11 | 16 | | | | 25 | 26 | | | | 30 | 31 | | |

  - Delete 16

    | 10 | 28 | | |

    | 3 | 5 | 7 | | | 11 | 18 | 25 | 26 | | 30 | 31 | | |
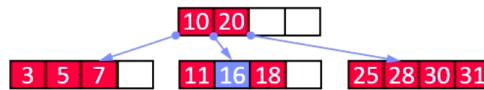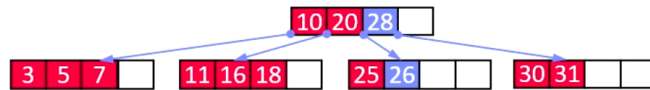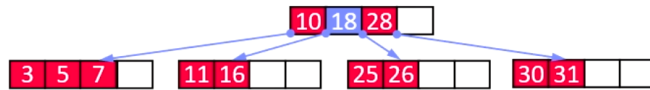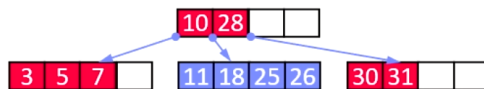
Prefix Tree

Excursus: Prefix Trees (Radix Trees, Tries)

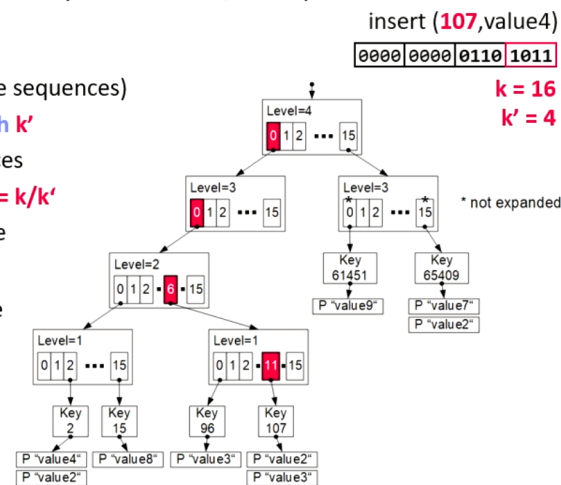- **Generalized Prefix Tree**
  - **Arbitrary data types** (byte sequences)
  - **Configurable prefix length k'**
  - Node size: $s = 2^{k'}$ references
  - Fixed maximum height $h = k/k'$
  - Secondary index structure
- **Characteristics**
  - Partitioned data structure
  - **Order-preserving**
    (for range scans)
  - **Update-friendly**
- **Properties**
  - **Deterministic paths**
  - Worst-case complexity O(h)

insert (**107**,value4)

| 0000 | 0000 | 0110 | 1011 |

$k = 16$
$k' = 4$

Level=4
| 0 | 1 | 2 | ••• | 15 |

Level=3
| 0 | 1 | 2 | ••• | 15 |

Level=3
| 0 | 1 | 2 | ••• | 15 | * not expanded

Level=2
| 0 | 1 | 2 | 6 | 15 |

Key 61451   Key 65409
P "value9"   P "value7"
               P "value2"

Level=1
| 0 | 1 | 2 | ••• | 15 |

Level=1
| 0 | 1 | 2 | 11 | 15 |

Key 2   Key 15   Key 96   Key 107

P "value4"   P "value8"   P "value3"   P "value2"
P "value2"   P "value3"

4

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**]