CS5800: B+-Trees

Ricardo Baeza-Yates & Anurag Bhardwaj

Northeastern University 2018

Agenda

- B+-Trees
  - Introduction
  - Searching, Insertion, Deletion
  - Complexity Analysis
- B+-Tree Advantages

how memory works? Secondary, primary (OS) disk?
purpose of RAM? --> operating system load from RAM so it's fast
database to query efficiently? use B+ trees to solve disk storage problem

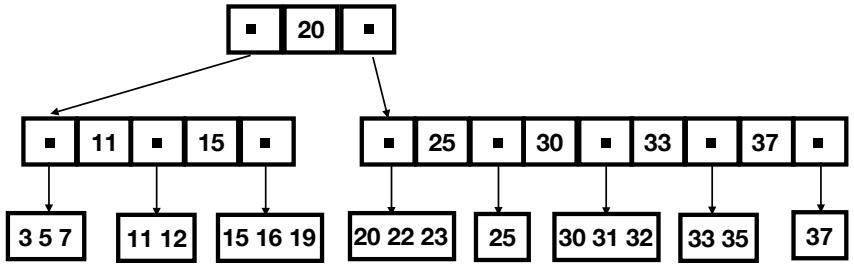
B+-Trees - Introduction

- Disk Storage Problem
  - How to store data on disk for faster access?
  - How about Binary Search trees?
  - # of reads needed?
  - # of root -> leaf traversals?
  - Locality of reference?
- B-Trees and its variants comes to rescue
- B-Tree is a variant of search tree with many children

B+-Trees - Properties

- Key Ideas
  - Reduce # of reads: store more data in each node
  - Reduce # of traversals: high branching factor (B >> 2)
  - Locality of reference: sorted keys

B+-Trees - Example

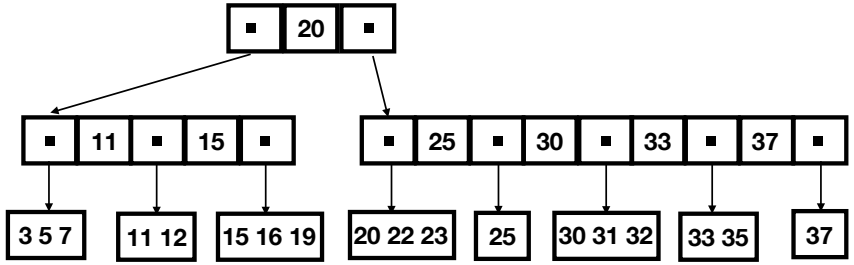


B+-Trees - Formal Definition

- For non-leaf node N:
  - (# of children in K) - (# of keys stored in K) == 1
- For every non-root node N:
  - at least (B-1) keys
- All nodes have at most 2*B children
- All leaf nodes have same depth
- Only leaf node(s) contain data, non-leaf contain routing values

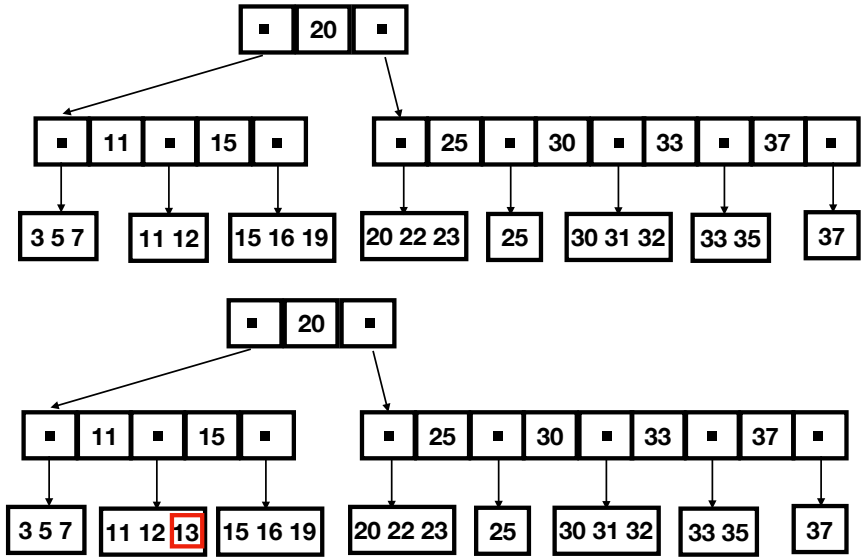
store sorted array in each node

B+-Trees - Search

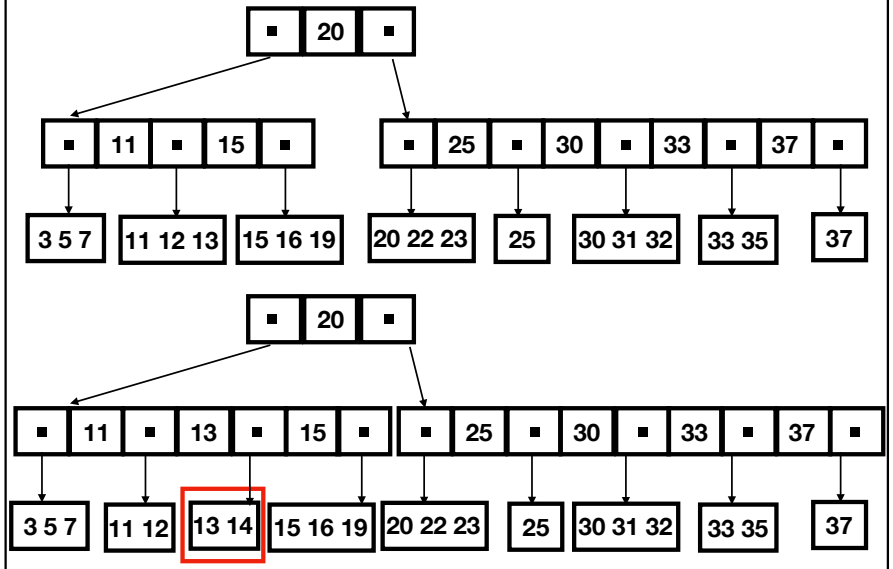


- Linear or binary search in each node - given max. # of keys in node
- If not found, follow the appropriate child pointer
- Complexity: $\log_B(N)$

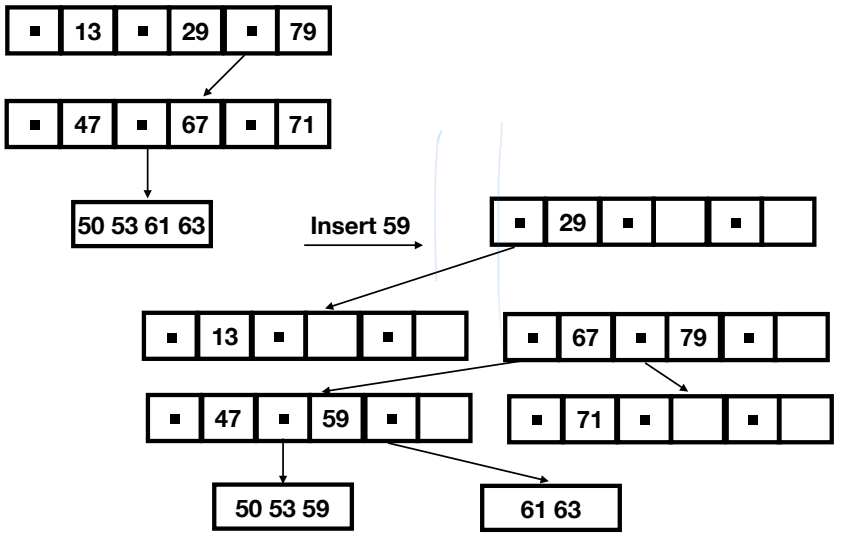
B+-Trees - Insertion (1/3)



B+-Trees - Insertion (2/3)



B+-Trees - Insertion (3/3)



B+-Trees - Deletion

- An element is removed from the leaf
  - if leaf becomes empty, remove key from parent
  - Keep adjusting parent and its siblings to satisfy invariants
  - May ripple to root and reduce height of tree by 1
  - If the node has < 50% occupancy, merge it with left/right node (standard B+-Trees have a minimal occupancy of 50%)

Node has maximum b keys
Each node has maximum b-2 keys
Otherwise have to merge

- More examples:
  - https://www.cs.usfca.edu/~galles/visualization/BTree.html

B+-Trees - Complexity

- Time Complexity

B is the max number of children can have

  - Search: $O(B * \log_B(n))$
  - Insertion: $O(B * \log_B(n))$
  - Deletion: $O(B * \log_B(n))$

- Space Complexity
  - O(n)

B+-Trees - Advantages

- Multi-level Indexing
- Bulk Insertion
- Aggregation queries
- Value of B can be set to match the size of a disk sector or its multiples

give an example of an application when B+ tree is more useful than binary search tree

tell me if this is a valid B+ tree or not

number of keys / children have for B+ tree?