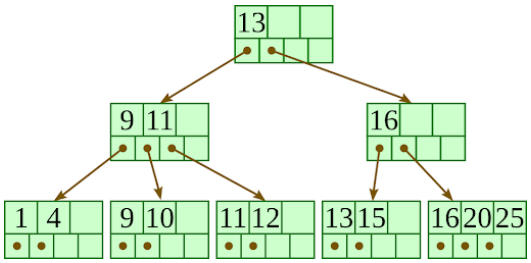
B+ Trees

Thursday, January 23, 2025 9:20 AM



- Optimized for disk-based indexing
- Minimizing disk accesses
- B Trees are for in-memory indexing alternatively

B+ Tree is an m-way tree with order M
M --> max # of keys in each node
M+1 --> max # of children

a	b	c

Pointers are vertical lines between keys

1. < a
2. >= a AND < b
3. >=b AND < c

Properties

- All nodes must be ½ full min.
- Root node DOES NOT have to be ½ full
- Insertions are always done at leaf level
- Leaves are stored as a DLL (doubly linked list)
- Internal nodes and leaf nodes
  - Internal only store keys and pointers to children
    - Act as an index
  - Leaf nodes store keys and data

m=3 tree

42		

Insert 29

29	42	

Insert 81

29	42	81

Insert 99

81 (root)		

29	42			81	99	

Insert 35

81 (root)		

29	35	42		81	99	

Insert 2 (this would make left node overflow so we split it)
Think about splitting 2, 29, 35, and 42

81 (root)		

2	29			81	99	

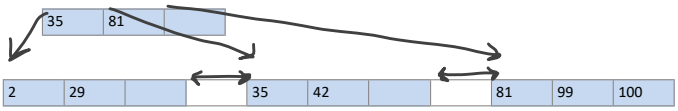
35	42	

Smallest value of new node when it's a left goes to the root (with its pointer)

35	81	

2	29			35	42			81	99	

Insert 100



Insert 30

35	81	

2	29	30		35	42			81	99	100

Insert 45

35	81	

2	29	30		35	42	45		81	99	100

Insert 82

35	81	

2	29	30		35	42	45		81	82	

99	100	

Insert 82

35	81	99

2	29	30		35	42	45		81	82			99	100	

Insert 4

35	81	99

2	4			35	42	45		81	82			99	100	

29	30	

Uh oh we need to move 29 up but can't so split the top. Since 81 was the least of this new node with 99, we move it up again (but it doesn't stay because we already have a leaf for it)

81		

29	35			99		

2	4			29	30			35	42	45		81	82			99	100