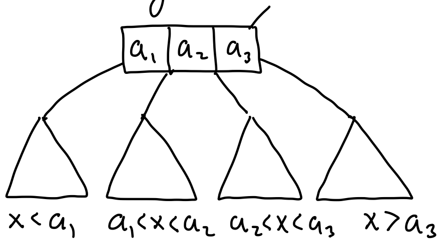
B-Trees I

Secondary Memory:

- Most large data structures reside on disk storage
- Organized in blocks - pages
- Latency: High start-up time
- Want to minimize no. of blocks accessed

Multiway Search Trees:



B-Tree:

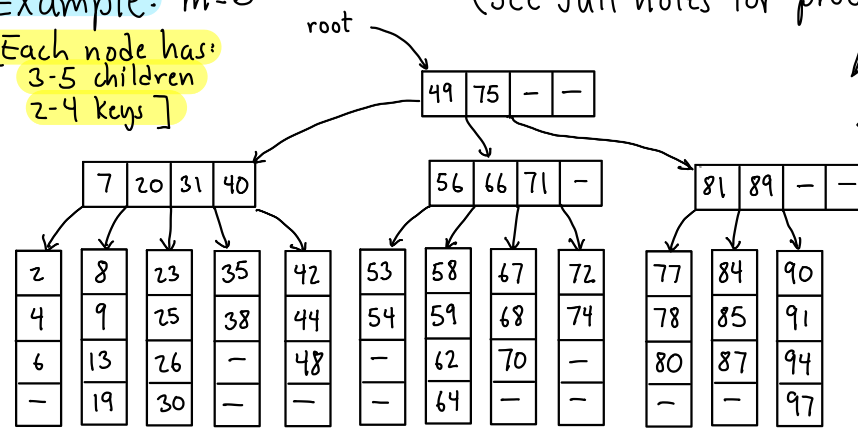
- Perhaps the most widely used search tree
- 1970 - Bayer + McCreight
- Databases
- Numerous variants

B-Tree: of order m (≥ 3)

- Root is leaf or has ≥ 2 children
- Non-root nodes have $\lceil m/2 \rceil$ to m children [null for leaves]
- k children $\Rightarrow$ k-1 key-values
- All leaves at same level

Example: m=5

[Each node has: 3-5 children 2-4 keys]



Theorem:

A B-tree of order m with n keys has height at most $(\lg n)/\gamma$, where $\gamma = \lg(m/2)$

(See full notes for proof)

Node Structure:

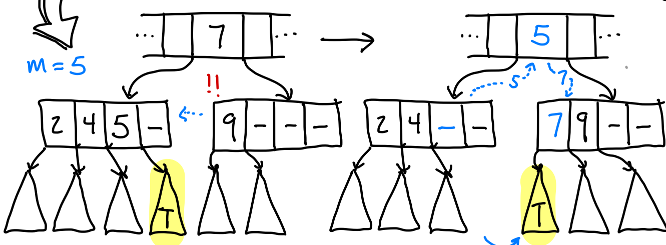
constant int M=...

```
class BTreeNode {
    int        nChild       // no. of children
    BTreeNode  child[M]     // children
    Key        key[M-1]     // keys
    Value      value[M-1]   // values
}
```

B-Trees II

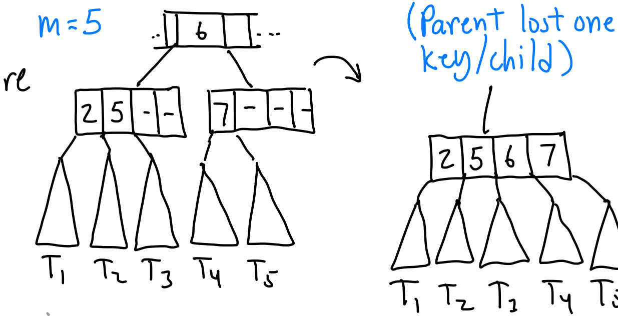
Key Rotation (Adoption)

- A node has too few children $\lceil m/2 \rceil - 1$
- Does either immediate sibling have extra? $\geq \lceil m/2 \rceil + 1$
- Adopt child from sibling & rotate keys
- When applicable - preferred



B-Tree restructuring:

- Generalizes 2-3 restructure
- Key rotation (Adoption)
- Splitting (insertion)
- Merging (deletion)



Lemma: For all $m \geq 2$,

$$\lceil m/2 \rceil \leq 2\lceil m/2 \rceil - 1 \leq m$$

$\Rightarrow$ Resulting node is valid

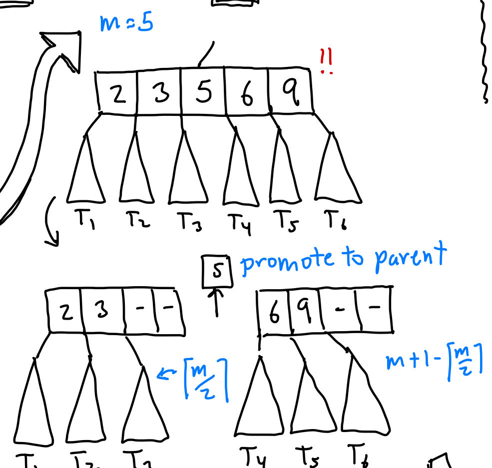
Node Splitting:

- After insertion, a node has too many children... $m+1$
- We split into two nodes of sizes $m' = \lceil m/2 \rceil$ and $m'' = m+1-\lceil m/2 \rceil$

Lemma: For all $m \geq 2$,

$$\lceil m/2 \rceil \leq m+1-\lceil m/2 \rceil \leq m$$

$\Rightarrow$ m' + m'' are valid node sizes



Node Merging:

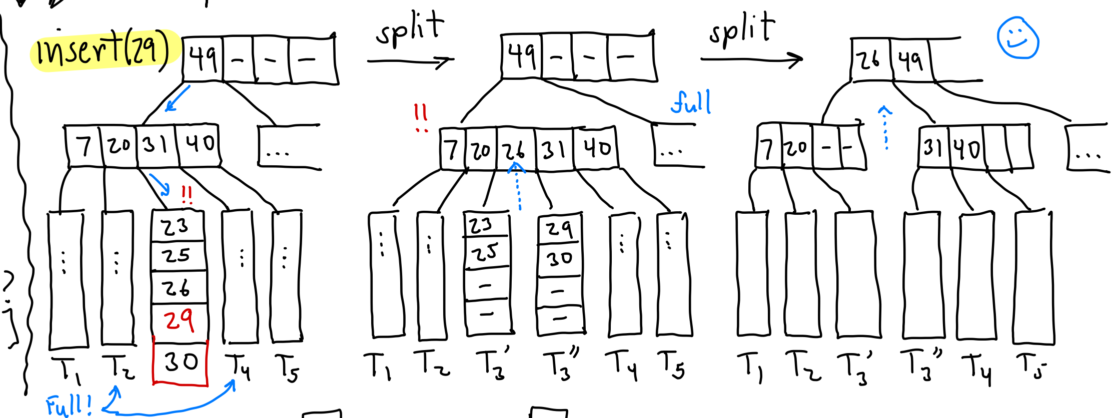
- A node has too few children $\lceil m/2 \rceil - 1$
- Neither sibling has extra (both $\lceil m/2 \rceil$)
- Merge with either sibling to produce node with $(\lceil m/2 \rceil - 1) + \lceil m/2 \rceil$ child

B-Trees III

Insertion:

- Find insertion point (leaf level)
- Add key/value here
- If node overfull (m keys, m+1 children)
  - → Can either sibling take a child (< m)?
  - ⇒ Key rotation [done]
  - → Else, split
    - → Promotes key
    - → If root splits add new root

Example: $m=5$



Deletion:

- Find key to delete
- Find replacement / copy
- If underfull ($\lceil m/2 \rceil - 1$) child
  - → If sibling can give child
    - → Key rotation
  - → Else (sibling has $\lceil m/2 \rceil$)
    - → Merge with sibling
- → Propagates → If root has 1 child → collapse root

Example: $m=5$