Multiway Search Trees:

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

B-Tree:
- Perhaps the most widely used search tree.
- Databases.
- Numerous variants.

B-Tree: of order m (≥ 3)
- Root is leaf or has ≥ 2 children.
- Non-root nodes have \([m/2]\) to m children [null for leaves].
- k children \(\Rightarrow k-1\) key-values.
- All leaves at same level.

B-Trees (I)

Example: \(m=5\)

Each node has
- 3-5 children
- 2-4 keys.

Node Structure:

```java
class BTree.Node {
    int nChild; // no. of children
    BTreeNode[] children;
    Key[] keys; // keys
    Value[] values; // values
}
```

Theorem:
A B-tree of order \(m\) with \(n\) keys has height at most \((\log n) / \gamma\), where \(\gamma = \log(m/2)\).

(See full notes for proof.)
Key Rotation (Adoption)
- A node has too few children \( \left\lceil \frac{m}{2} \right\rceil - 1 \)
- Does either immediate sibling have extra? \( \geq \left\lceil \frac{m}{2} \right\rceil + 1 \)
- Adopt child from sibling & rotate keys
- When applicable - preferred

B-Tree restructuring:
- Generalizes 2-3 restructure
- Key rotation (Adoption)
- Splitting (insertion)
- Merging (deletion)

Node Splitting:
- After insertion, a node has too many children ... \( m+1 \)
- We split into two nodes of sizes \( m' = \left\lceil \frac{m}{2} \right\rceil \) and \( m'' = m+1 - \left\lceil \frac{m}{2} \right\rceil \)

Lemma: For all \( m \geq 2 \),
\[ \left\lceil \frac{m}{2} \right\rceil \leq m+1 - \left\lceil \frac{m}{2} \right\rceil \leq m \]
\[ \Rightarrow m' + m'' \text{ are valid node sizes} \]

Node Merging:
- A node has too few children \( \left\lceil \frac{m}{2} \right\rceil - 1 \)
- Neither sibling has extra (both \( \left\lceil \frac{m}{2} \right\rceil \))
- Merge with either sibling to produce node with \((\left\lceil \frac{m}{2} \right\rceil - 1) + \left\lceil \frac{m}{2} \right\rceil \text{ child} \)

B-Trees II

Lemma: For all \( m \geq 2 \),
\[ \left\lceil \frac{m}{2} \right\rceil \leq m+1 - \left\lceil \frac{m}{2} \right\rceil \leq m \]
\[ \Rightarrow \text{Resulting node is valid} \]
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 ([m/2]-1) child
  → If sibling can give child
     → Key rotation
  → Else (sibling has [m/2])
     → Merge with sibling
  → Propagates → If root has 1 child → collapse root

Example: m=5