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 \( \lceil \frac{m}{2} \rceil \) to m children [null for leaves].
- k children \( \Rightarrow \) k-1 key-values.
- All leaves at same level.

Secondary Memory:
- Node Structure: constant int M...

Node Structure:

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

Example:

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

Example of m = 5

Theorem: An B-tree of order m with n keys has height at most \( \frac{\log n}{\log \frac{m}{2}} \), where \( \gamma = \log(\frac{m}{2}) \). (See full notes for proof.)
Key Rotation (Adoption)
- A node has too few children \([m/2] - 1\)
- Does either immediate sibling have extra? \(\geq [m/2] + 1\)
- Adopt child from sibling & rotate keys
- When applicable - preferred

B-Tree restructuring:
- Generalizes 2-3 restructuring
- 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 \(m' = [m/2]\) and \(m'' = m+1-[m/2]\)

Lemma: For all \(m \geq 2\),
\([m/2] \leq m+1-[m/2] \leq m\)
\(\Rightarrow m' + m''\) are valid node sizes

Node Merging:
- A node has too few children \([m/2] - 1\)
- Neither sibling has extra (both \([m/2]\))
- Merge with either sibling to produce node with \(([m/2] - 1) + [m/2]\) children

\(m = 5\)
\([m/2] = 2\)
\([m/2] - 1 = 1\)
\(\Rightarrow\) 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

**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:**
- For m = 5
- Insertion example:
  - Insert(29)
  - Split
- Deletion example:
  - Delete(30)
  - Merge
  - Key Rotation

**B-Trees III**
**History:**
1989: Seidel & Aragon

Explosion of randomized algorithms

Later discovered this was already known: Priority Search Trees from different context (geometry)

McCreight 1980

**Intuition:**
- Random insertion into BSTs $\Rightarrow O(\log n)$ expected height
- Worst case can be very bad $O(n)$ height
- Treap: A tree that behaves as if keys are inserted in random order

**Example:** Insert: k, e, b, s, f, h, w
(Std. BST)

Along any path - Insertion times increase

**Randomized Data Structures**
- Use a random number generator
- Running in expectation over all random choices
- Often simpler than deterministic

**Monte Carlo** - possibly incorrect

**Las Vegas** - always correct

**Geometric Interpretation:**

**Treaps I**

**Obs:** In a standard BST, keys are by insertion times are in heap order (parent < child)

**Tree + Heap**

**Example:**

<table>
<thead>
<tr>
<th>Key</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>37</td>
</tr>
<tr>
<td>e</td>
<td>13</td>
</tr>
<tr>
<td>f</td>
<td>51</td>
</tr>
<tr>
<td>h</td>
<td>1</td>
</tr>
<tr>
<td>k</td>
<td>3</td>
</tr>
<tr>
<td>m</td>
<td>78</td>
</tr>
<tr>
<td>n</td>
<td>45</td>
</tr>
<tr>
<td>o</td>
<td>45</td>
</tr>
<tr>
<td>w</td>
<td>67</td>
</tr>
</tbody>
</table>

**McCreight Priority Search Tree**

**Treap:** Each node stores a key + a random priority

Keys are in inorder
Priorities are in heap order

? Is it always possible to do both?

Yes: Just consider the corresponding BST
**Insertion:** As usual, find the leaf and create a new leaf node.
- Assign random priority
- On backing out - check heap order + rotate to fix.

**Theorem:** A treap containing $n$ entries has height $O(\log n)$ in expectation (averaged over all assignments of random priorities).

**Proof:** Follows directly from BST analysis.

**Example:**

**Implementation:** (See pdf notes)

**Node:** Stores priority + usual...

**Helpers:**
- Assists with random priority assignment
- On backing out - check of random priorities

**Restructure:**
- Performs rotation (if needed) to put lowest priority node at $p$.

**Deletion:** (Cute solution) Find node to delete. Set its priority to $+\infty$. Rotate it down to leaf level + unlink.
Announcements 4/27

- Prog Assign 3 posted
  - Due, Wed, May 9
- Skeleton code/test data
  - Coming