Today: Caching
   → B-trees
   → Buffer trees
   → Priority Queues
Modeling a Cache: External Memory Model

Goal: Simple model (tractable)
Sufficiently accurate (useful)

Disk

Cache

- Cache line
  - Block

Total memory: M
  \((M/B)\) lines in cache

Often assume \(M \geq B^2\)
(tall cache assumption)

Block size \(B\)
B-trees

\[(a, b)-\text{tree}\] \(a = B, \ b = 2B, \ b \geq 2a\)

**Rules:**
1) Root has \(\geq 2\) children
   Other nodes have \(\geq a\) children
2) All nodes have \(\leq b\) children
3) All leaves have same depth

\(\Rightarrow\) each leaf has size: \(a \leq |\text{leaf}| \leq b\)

(sometimes BIG leaves help)
\(\rightarrow\) all keys stored in leaves
\(\rightarrow\) internal nodes store pivots \(\leftarrow\) sorted?
\[\geq a-1\] pivots
\[\leq b-1\] pivots

Claim: height \(\leq \log_a \left(\frac{N}{a}\right) + 1\)

\(\rightarrow\) \(\leq \frac{N}{a}\) leaves

except root

\(\rightarrow\) every node has degree \(\geq a\)

\[\Rightarrow\] node at height \(\log_a \frac{N}{a}\) has
Subtree with \(\geq \frac{N}{a}\) leaves

\[\Rightarrow\] children of root are height \(\leq \log_a \frac{N}{a}\)

**Search \(K\)**

\(V = \text{root}\)

While not leaf \(V\):

if \(K \leq P_i\) then \(V \leftarrow C_i\)

else let \(l = \max j : K > P_j\)

\(V = C_{l+1}\)

return searchLeaf\(V, K\)
search tree property:
All keys $K$ in child $C_j$ satisfy:

$$P_{j-1} < K \leq P_j$$

(Pretend $P_0 = \text{min element} - 1$, and if $P_j$ undefined, $P_j = \text{max element}$.)

Claim: Search works

→ follows from search tree property

Claim: if $a = b$, $b = 2b$ then cost of search is $O\left(\log_b \frac{N}{b}\right)$

→ height of tree is $O\left(\log_b \frac{N}{b}\right)$

→ each node is $O(1)$ blocks
Insert

1) Search for insertion leaf node
2) Add key to node
3) If node has ≥ b keys:
   a) Split node; each new node has ≥ \( b/2 \geq a \) keys
   b) New nodes: X, Y
   c) Key = max element in X
   d) Node = parent (node)
   e) Go to (2) [insert key into parent]

   If node is root, Create new root
Delete

1) Search for key to delete, \( V = \text{leaf} \)

2) Delete it.

3) If \( |V| < a - 1 \):
   
   let \( u \) be sibling of \( V \)

   Case 1) \( |u| + |v| > b - 1 \)
   
   \( \Rightarrow \) divide keys between \( u \) and \( v \)
   
   each gets \( \geq \frac{b}{2} \geq a \) keys

   Case 2) \( |u| + |v| \leq b - 1 \)
   
   \( \Rightarrow \) merge \( u \) and \( v \)

   in parent, delete pivot

   separating \( u \) and \( v \) and

   recurse: \( V = \text{parent}(v) \)

   go to (3)

   if parent is root: delete parent
   
   and only one child
Claim: if \( a = b \) and \( b = 2b \) then cost of insert/delete is \( \Theta(\log \theta \frac{\Theta}{b}) \). 
\[ \Rightarrow O(1) \text{ blocks at each level of tree} \]

Claim: if \( a = b \) and \( b = 5b \) then amortized split/merges are \( O(\frac{b}{\theta}) \) per op.

Goal: After each split or merge: 
\[ \begin{align*}
&\geq 2B-1 \text{ keys} \\
&\leq 4B \text{ keys}
\end{align*} \]

\( \Rightarrow \) after split:
\[ 5B \text{ keys } \rightarrow (2.5B, 2.5B) \checkmark \]

On delete:
\( \Rightarrow \) merge if total \( \leq 4B \) \( \checkmark \)
\( (\geq B-1 + B \geq 2B-1) \)

Share if total \( > 4B \)
\( \text{each } > 2B \checkmark \)
\( \text{total } \leq 5B + B-1 \leq 6B \)
\( \text{each } \leq 3B \)
\( \Rightarrow \geq B-1 \text{ ops before next merge/split} \)
Notes:

1) Root stays in cache (almost) always
2) If $B=16$KB [Page size] and $N\leq 1000$ TB
   $\Rightarrow$ height $= 4 \Rightarrow$ 3 cache misses

3) What if you maintain parent pointers?

4) How to choose $B$?

5) How to store/search pivots?
**Buffer Tree**

- Faster inserts/deletes
- Slower searches

---

**Step 1)** **Build a (2,3)-tree**

**Step 2)** Add a buffer of size $2B$ to every non-leaf node

**Step 3)** **Guarantee:** each leaf has $\geq B$ keys

\[ \leq 5B \]

Each buffer has $\leq B$ items

\[ \leftarrow \text{Delete is similar!} \]

**Insert:**

- Add $\text{ins}[\text{key}]$ to root buffer
- Clean buffer (remove $\text{del}[\text{key}]$, duplicate $\text{ins}[\text{key}]$)
- If buffer $> B$, then empty buffer

**Search:** Walk tree as usual

- Check each buffer

\[ \leftarrow \text{Higher in tree ops have precedence!} \]
empty buffer:
1) Sort buffer
2) Move ops to proper child
3) Clean child - duplicates
   empty if needed

If non-leaf and not enough room:
   a) finish filling child
   b) empty child
   c) finish emptying buffer

If leaf:
   a) perform deletes. If underfull, postpone.
   b) perform inserts. Do splits. [split buffer too!]
   c) at end, do merges → may cause more "empty" ops
      when merge buffers