B-trees

- Operations
  - insert($k, v$)
  - delete($k$)
  - $v = \text{search}(k)$
  - $[v_1, v_2, \ldots] = \text{range-query}(k_1, k_2)$

Elements in leaves in sorted order: between $B/2$ and $B$ in each leaf

Node:

- $\leq 10$ to the left
- $> 10$ and $\leq 65$ to the right
- $> 65$ and $\leq 83$ further right
- $> 83$ further right

Branching factor of root: $[2, B]$

Branching factor of leaves: $(B/2, B)$
B-trees

- A node/leaf can be stored in $O(1)$ blocks.
- Search uses $O(\log_B N)$ I/Os.

Elements in leaves in sorted order: between $B/2$ and $B$ in each leaf

Branching factor of root: $[2, B]$
B-trees

Elements in leaves in sorted order: between B/2 and B in each leaf

- Range query $[q_1, q_2]$:
  - search down $T$ for $q_1$ and $q_2$ (or successor and predecessor).
  - report the elements in the leaves between the leaves containing (successor of) $q_1$ and (predecessor of) $q_2$.
- $\#I/Os = O(\log_B N + \text{occ}/B)$. 

branching factor of root: $[2,B]$
Insertion in B-tree

- **Insert**\( (k, v) \)
  - search for relevant leaf \( u \) and insert \((k,v)\) in \( u \).
  - If \( u \) now contains \( B+1 \) elements:
    - split it into two leaves \( u' \) and \( u'' \).
    - update parent(\( u \))
  - If parent(\( u \)) now has degree \( B+1 \) recursively split it.
  - If root split: add a new root node with degree 2 (height of tree grows)

- **Example.** \( B=5 \). Insert(24, v)
Insertion in B-tree

- **Insert(k, v)**
  - search for relevant leaf u and insert (k,v) in u.
  - If u now contains B+1 elements:
    - split it into two leaves u’ and u’’.
    - update parent(u)
    - If parent(u) now has degree B+1 recursively split it.
  - If root split: add a new root node with degree 2 (height of tree grows)

- **Example.** B= 6. Insert(18, v)
Insertion in B-tree

- Insert(k, v)
  - search for relevant leaf u and insert (k,v) in u.
  - If u now contains B+1 elements:
    - split it into two leaves u’ and u’’.
    - update parent(u)
  - If parent(u) now has degree B+1 recursively split it.
  - If root split: add a new root node with degree 2 (height of tree grows)

- #I/Os = O(logₐ N)
Deletion in B-tree

- **Delete(k)**
  - search for relevant leaf $u$ and delete element with key $k$ in $u$.
  - If $u$ now contains $B/2 - 1$ elements:
    - merge $u$ with its sibling $u'$. If this results in $u$ containing more than $B$ elements split it into two leaves.
    - update parent($u$)
    - If parent($u$) now has degree $B/2 - 1$ recursively merge it.
  - If root has degree 1: delete root (height decreases)

- **Example.** $B=6$. Delete(24)
Deletion in B-tree

- **Delete(k)**
  - search for relevant leaf $u$ and delete element with key $k$ in $u$.
  - If $u$ now contains $B/2 - 1$ elements:
    - merge $u$ with its sibling $u'$.
    - If this results in $u$ containing more than $B$ elements split it into two leaves.
    - update parent($u$)
  - If parent($u$) now has degree $B/2 - 1$ recursively merge it.
  - If root has degree 1: delete root (height decreases)

- **Example.** $B = 6$. Delete(18)
Deletion in B-tree

- **Delete(k)**
  - search for relevant leaf $u$ and delete element with key $k$ in $u$.
  - If $u$ now contains $B/2 - 1$ elements:
    - merge $u$ with its sibling $u'$. If this results in $u$ containing more than $B$ elements split it into two leaves.
    - update parent($u$)
      - If parent($u$) now has degree $B/2 - 1$ recursively merge it.
  - If root has degree 1: delete root (height decreases)

- **Example.** $B = 6$. Delete(18)
Deletion in B-tree

- **Delete(k)**
  - search for relevant leaf $u$ and delete element with key $k$ in $u$.
  - If $u$ now contains $B/2 - 1$ elements:
    - merge $u$ with its sibling $u'$. If this results in $u$ containing more than $B$ elements split it into two leaves.
    - update parent($u$)
      - If parent($u$) now has degree $B/2 - 1$ recursively merge it.
  - If root has degree 1: delete root (height decreases)

- $\#I/Os = O(\log_B N)$
(a,b)-trees

Height $\log_a N$

Branching factor of root: $[2,b]$

Branching factor: $[a,b]$

Elements in leaves in sorted order: between $a$ and $b$ in each leaf

- Operations
  - insert($k$, $v$)
  - delete($k$)
  - search($k$)
  - $v = \text{range-query}(k)$
  - $[v_1, v_2, \ldots] = \text{range-query}(k_1, k_2)$

Node:

- $\leq 10$
- $> 10 \text{ and } \leq 65$
- $> 65 \text{ and } \leq 83$
- $> 83$

A B-tree is an (a,b)-tree with $a,b=\Theta(B)$
Amortized updates in (a,b)-trees

- If $b \geq 2a$ then the number of rebalancing operations caused by an update $O(1/a)$ amortized
B$^\varepsilon$ trees

For $0 \leq \varepsilon \leq 1$:
- Updates: $O\left(\frac{\log_{1+B^\varepsilon} N}{B^{1-\varepsilon}}\right)$
- Point query: $O(\log_{1+B^\varepsilon} N)$
- Range query:
  - $O\left(\log_{1+B^\varepsilon} N + \frac{\text{occ}}{B}\right)$

$\varepsilon = 1/2$:
- Updates: $O\left(\frac{\log B N}{\sqrt{B}}\right)$
- Point query: $O(\log B N)$
- Range query:
  - $O\left(\log B N + \frac{\text{occ}}{B}\right)$