Level-Rebuilt B-Trees

Gerth Stølting Brodal

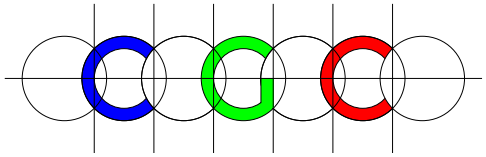
BRICS

University of Aarhus

Pankaj K. Agarwal

Lars Arge

Jeffrey S. Vitter



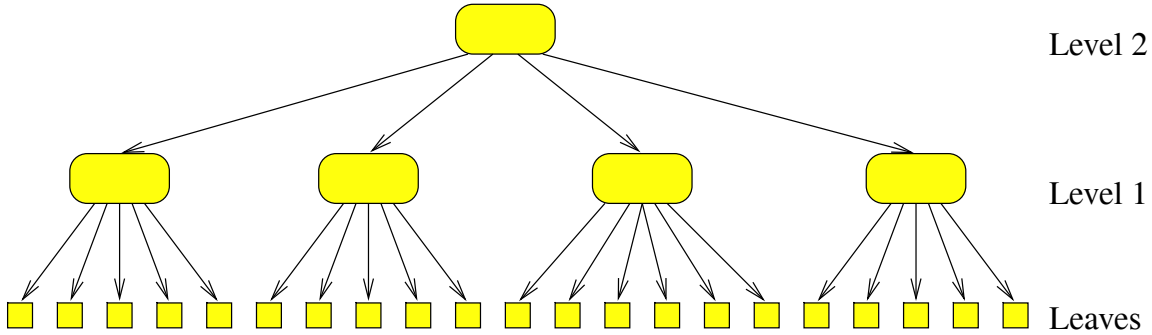
Center for Geometric Computing

Duke University

August 1998

B-Trees

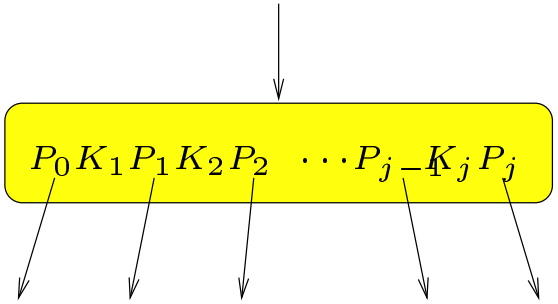
Bayer, McCreight 1972



- Each node has at most B children, i.e., each node can be stored in one block
- Each node has at least $B/2$ children, except for the root which has degree at least 2
- Height at most $\log_{B/2} N = \mathcal{O}(\log_B N)$, where $N = \#\text{Leaves}$

B-Trees – Searching

- By storing search keys in each node, a B-tree can be used as a search tree

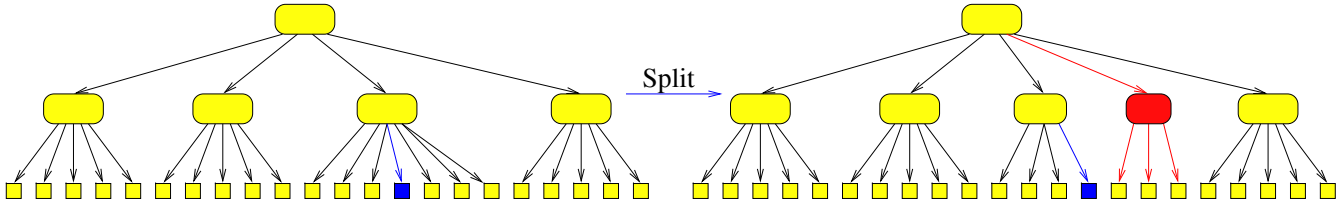


- Searching requires one block read per level of the tree, i.e., the number of I/Os for a search is $\mathcal{O}(\log_B N)$

B-Trees — Insertions

1. Find the location where to insert the new leaf
2. Insert a pointer to the new leaf
3. If the parent of the leaf has degree $= B + 1$ then
   (a) Split the parent into two nodes
   (b) Insert a pointer to the new node in the grandparent
   (c) Recursively split the grandparent if it has degree $= B + 1$

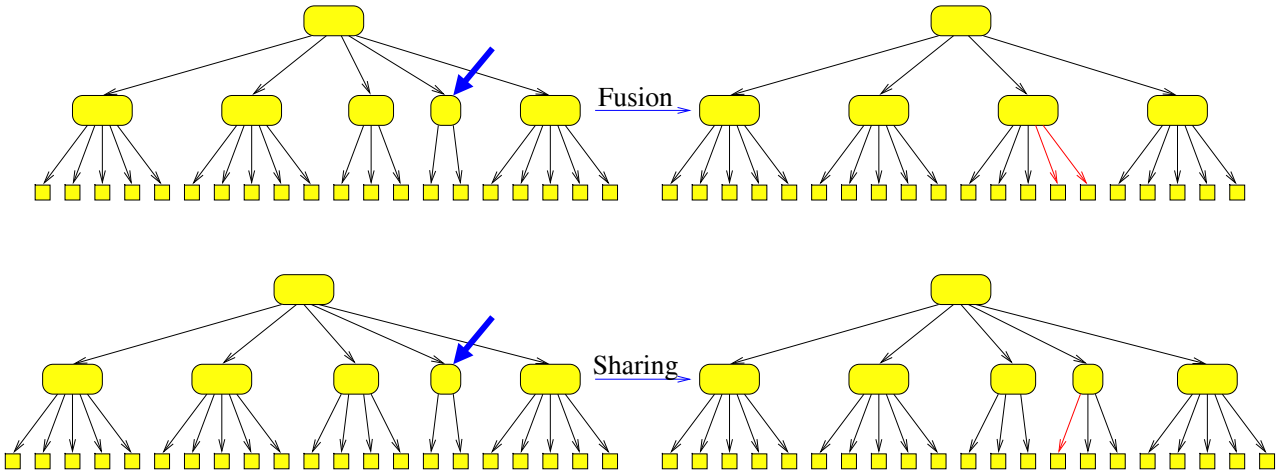
Total # I/Os $= \mathcal{O}(\log_B N)$



$B = 6$

B-Trees — Deletions

1. Delete the leaf
2. If the parent has degree $B/2 - 1$ then either
   (a) Fusion the parent with one of its siblings and recurse on grandparent if degree $= B/2 - 1$
   (b) Move children of a sibling of the parent to the parent (sharing)



$B = 6$

Total # I/Os $= \mathcal{O}(\log_B N)$

B-Trees — Amortized Restructuring Cost

- The number of splitting/fusion/sharing steps per insertion and deletion is amortized $\mathcal{O}(1)$
- By decreasing the lower bound on the degrees to $B/4$
  - The height remains $\mathcal{O}(\log_B N)$
  - Splitting/fusion/sharing steps can be done such that the resulting nodes have degree between $B/3$ and $2B/3$
  - Insertions and deletions require $\mathcal{O}(1)$ I/Os to create/delete a leaf pointer and amortized $\mathcal{O}(1/B)$ splitting/fusion/sharing steps to restructure the tree

B-Trees — Parent Pointers

Why parent pointers ?

Example: Applications where the left-to-right traversal of the tree defines the order of the leaves, and queries are of the form: Is leaf x to the left of leaf y?

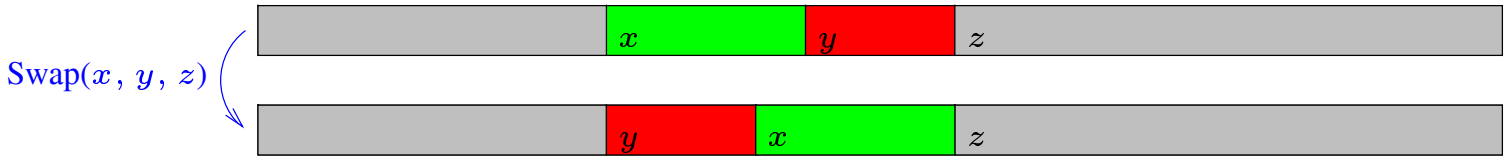
Algorithm: Queries can be done by traversing the two leaf-to-root paths starting at x and y until the nearest common ancestor of x and y is found.

Splitting/fusion/sharing steps require parent pointers in $\mathcal{O}(B)$ nodes to be updated

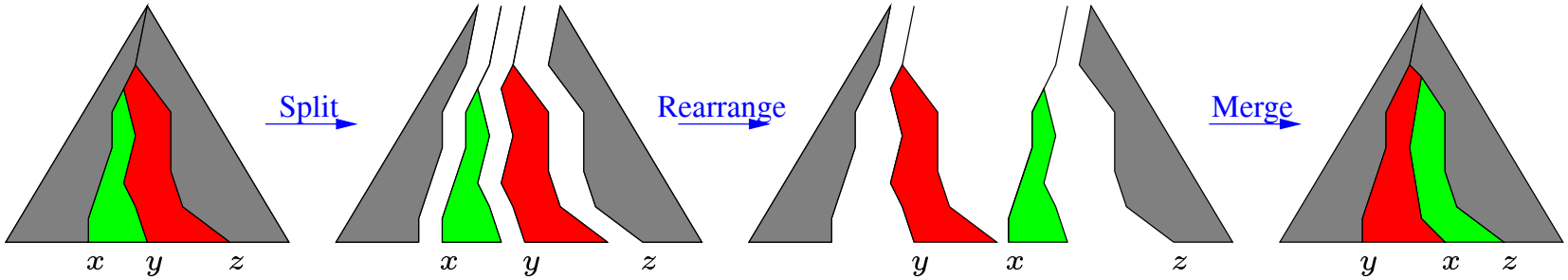
Theorem Insertions and deletions in a $(B/4, B)$–tree requires amortized $\mathcal{O}(1)$ I/Os to restructure the tree

The Swap Operation

Swap(x, y, z) swaps two consecutive subsequences in a list



If the list is represented by a B-tree, the Swap operation can be implemented by 3 split operations on the tree and 3 merge operations on the tree



The Swap operation requires $\mathcal{O}(1)$ node splittings/fusions/sharings at each level of the tree, i.e., without parent pointers Swap requires $\mathcal{O}(\log_B N)$ I/Os

B-Trees + Parent Pointers + The Swap Operation

With parent pointers the Swap operation requires $\mathcal{O}(B \cdot \log_B N)$ I/Os, because each splitting/fusion/sharing step must update $\mathcal{O}(B)$ parent pointers

Why a factor of B ?

- Splitting a node moves children from the node to a new node (in a new block)
- Fusion/sharing moves some children from one node to another node (the two nodes being in two different blocks)

Level Rebuild B-trees – The Ideas

Splits: When splitting a node keep the resulting nodes in the same block

$\Rightarrow$ parent pointers do not need to be updated

$\Rightarrow$ several nodes from a level in each block (random order)

Adding child pointers:

1) If a node gets too many children split the node (internally in block)

2) If not sufficient space for pointers in the block, split the nodes contained in the block evenly between two blocks and update all necessary parent pointers using $\mathcal{O}(B)$ I/Os

Fusions/sharings: Do nothing!

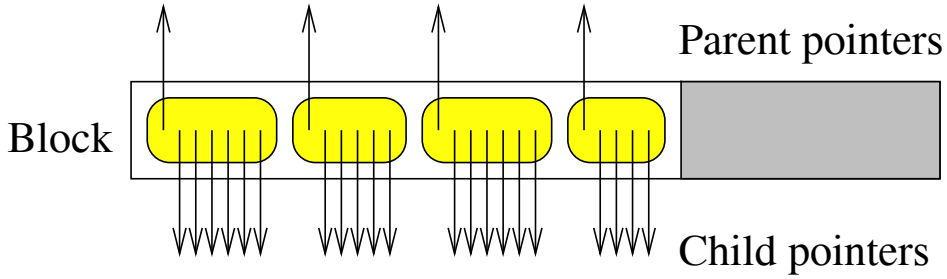
Bounding the height:

Require that level i of the tree contains at most $\frac{N}{B^{\mathcal{O}(i)}}$ nodes

$\Rightarrow$ rebuild levels with too many nodes

Level Rebuild B-Trees

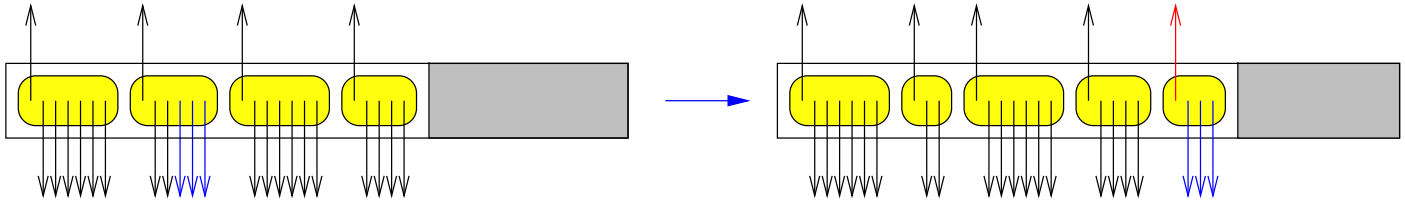
- Each node has degree at least one and at most $2b$ ($b \approx \sqrt{B}$)
- Each block stores several nodes from the same level such that each block contains at least $\Theta(B)$ pointers



- Child pointer = a pair <block id, node number in block>
- Parent pointer = block id (no node number!)
- Invariant: Level i contains at most $2\frac{N}{b^i}$ nodes

Splittings

Node splitting



- Adds one child pointer and one parent pointer to the structure, which can trigger recursive node and block splittings
- Requires $\mathcal{O}(1)$ I/Os

Block splitting

- Requires $\mathcal{O}(B)$ I/Os, but each pointer added to the structure at most costs amortized $\mathcal{O}(1/B)$ block splittings, i.e., amortized $\mathcal{O}(1)$ I/Os for each pointer added to the structure.
- Does not change the structure of the tree, i.e., does not trigger any recursive splittings

Level Rebuilding – I

After any operation the following level rebuilding is applied to the lowest level i with more than $2N/b^i$ nodes

1. Generate a list of all pointers to the children at level $i-1$ by a left-to-right traversal of the top of the tree $\mathcal{O}(\frac{N}{b^i})$ I/Os
2. Construct levels $j \geq i$ such that level j contains exactly N/b^j nodes, and setup all parent and child pointers for level j $\mathcal{O}(\frac{N}{Bb^{i-1}})$ I/Os
3. Setup $\mathcal{O}(\frac{N}{b^{i-1}})$ new parent pointers for level $i-1$
   a. Construct from the child list at level i a permuted list of all triples <parent block, child block, child number in block>
   b. Sort the list of triples w.r.t. the child block number $\mathcal{O}(\frac{N/b^{i-1}}{B} \log_{M/B} \frac{N/b^{i-1}}{B})$ I/Os
   c. Update the parent pointers at level $i-1$ by a simultaneous scan of the blocks storing level $i-1$ and the sorted list constructed above $\mathcal{O}(\frac{N}{Bb^{i-2}})$ I/Os

Level Rebuilding – II

The total # I/Os to rebuild level i is

$$\mathcal{O}(\frac{N}{Bb^i} + \frac{N}{Bb^{i-1}} + \frac{N}{Bb^{i-1}} \log_{M/B} \frac{N}{Bb^{i-1}} + \frac{N}{Bb^{i-2}})$$

$$= \mathcal{O}(\frac{N}{b^i} \cdot (\frac{b^2}{B} + \frac{b}{B} \log_{M/B} N))$$

$$= \mathcal{O}(\frac{N}{b^i} \cdot (1 + \frac{\log_{M/B} N}{\sqrt{B}}))$$

Swap(x, y, z)

- Split the nodes on the paths to x, y, z, except for the root
- Rearrange the children at the root to satisfy the new ordering
- If necessary apply level rebuilding to the lowest level i with more than $2N/b^i$ nodes

Because each Swap operation at most introduces $\mathcal{O}(1)$ new nodes at each level of the tree, level i will only be rebuild once for every $\mathcal{O}(N/b^i)$ Swap operation, implying that the amortized restructuring cost for level i of a Swap operation is $(1 + \frac{\log_{M/B} N}{\sqrt{B}})$ I/Os

Theorem Level rebuild B-trees support Swap operations in $\mathcal{O}((1 + \frac{\log_{M/B} N}{\sqrt{B}}) \log_B N)$ I/Os (amortized) and order queries in $\mathcal{O}(\log_B N)$ I/Os (worst-case)

Applications of Level Rebuild B-Trees

Reachability in planar st-graphs

Updates in $\mathcal{O}((\frac{\log_{M/B} N}{\sqrt{B}} + 1) \log_B N)$ I/Os (amortized)

Reachability queries $\mathcal{O}(\log_B N)$ I/Os (worst case)

Point location in planar monotone subdivisions

Updates (vertex/edge insertion/deletion) in $\mathcal{O}(\log_B^2 N)$ I/Os (amortized)

Queries in $\mathcal{O}(\log_B^2 N)$ I/Os (worst-case)

Conclusion

B-trees can be extended with parent pointers and efficiently support Swap operations by applying the following ideas

- Store several nodes in each block
- Replace the lower bound on the degrees by an upper bound on level sizes
- Apply level rebuilding to satisfy the level size constraint

Open Problem

- Improve I/O bound for updates: $(\frac{\log_{M/B} N}{\sqrt{B}} + 1) \log_B N \rightarrow \log_B N$