Level-Balanced B-Trees

Gerth Stølting Brodal

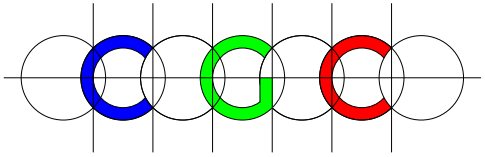
BRICS

University of Aarhus

Pankaj K. Agarwal

Lars Arge

Jeffrey S. Vitter



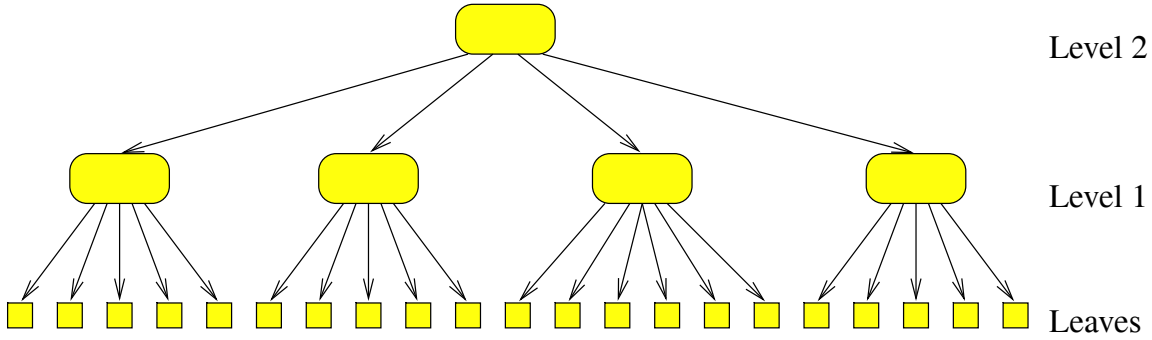
Center for Geometric Computing

Duke University

January 1999

B-Trees

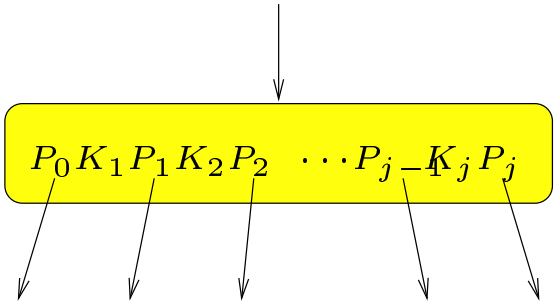
Bayer, McCreight 1972



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

B-Trees – Searching

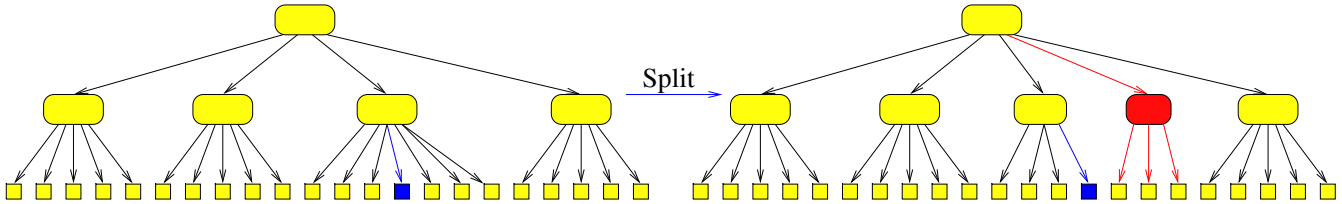
- By storing search keys in each node B-trees can be used as search trees



- Searching requires reading one disk block 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 $B + 1$

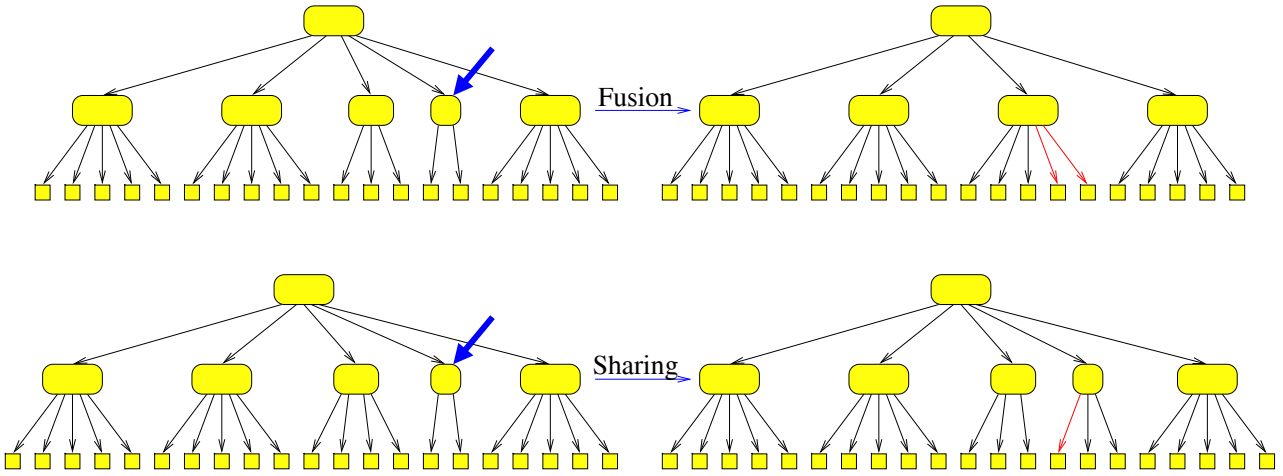


$B = 6$

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

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?

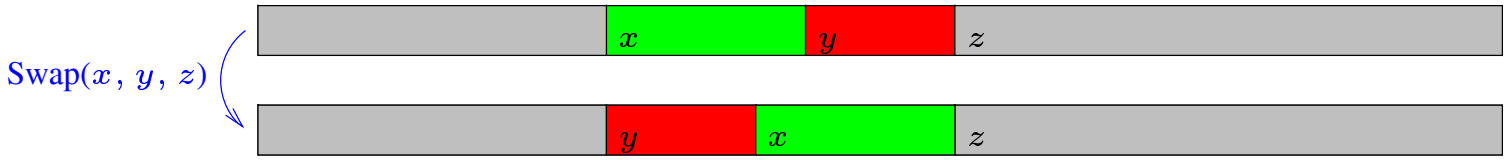
Algorithm: Queries can be answered by traversing the two leaf-to-root paths starting at x and y until the nearest common ancestor of x and y is found.

Splitting/fusion/sharing steps require parent pointers in $\Theta(B)$ nodes to be updated

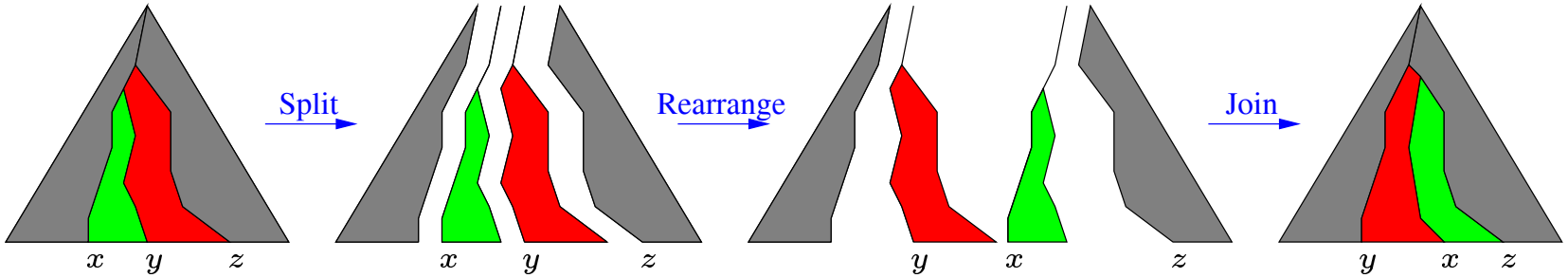
Theorem Insertions and deletions in a $(B/4, B)$–tree with parent pointers requires amortized $\mathcal{O}(1)$ I/Os

A 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 join 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 + Split and Join

— The Problem —

With parent pointers the split operation requires $\Theta(B \cdot \log_B N)$ I/Os, because each node along the splitting path must be split and requires $\Theta(B)$ parent pointers to be updated

Level-Balanced 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 disk block in 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:

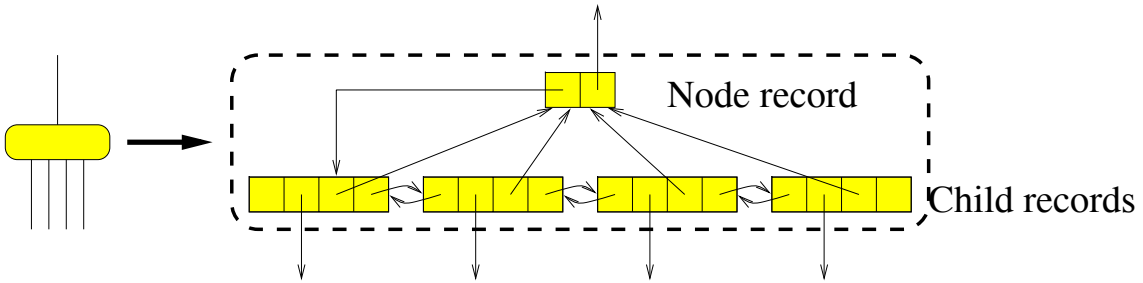
Require that level h of the tree contains at most $\frac{N}{B^{\mathcal{O}(h)}}$ nodes

$\Rightarrow$ rebalance levels with too many nodes

Level-Balanced B-Trees

A forest of level-balanced B-Trees with a total of at most N leaves is stored as follows:

- Each node has degree at least 1 and at most b, $2b \leq B$
- Level h contains $\leq 2N_h$ non-root nodes, $N_h = N/(\frac{b}{3})^h$
- Each node is stored as a node record and a double-linked list of child records



- Each disk block stores either only node records or child records from some level h; all child records of a node are stored in the same block; each block stores $\Theta(B)$ records

Splittings

Node splitting

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

Block splitting

- Split a block with B records into two blocks each containing at most $\frac{3}{4}B$ records
- Requires $\mathcal{O}(B)$ I/Os, but each record added to the structure requires amortized $\mathcal{O}(1/B)$ block splittings, i.e., amortized $\mathcal{O}(1)$ I/Os for each record added to the structure.
- Does not change the structure of the tree, i.e., does not trigger any recursive splittings

Level Balancing – I

After any operation the following level balancing is applied to the lowest level h with $> 2N_h$ non-root nodes

1. Generate a list of all pointers to the children at level $h-1$ by a left-to-right traversal of the top of the tree(s) $\mathcal{O}(N_h)$ I/Os
2. Reconstruct levels $i \geq h$ such that nodes at level i have degree between $b/2$ and b, and setup all parent and child pointers for level i $\mathcal{O}(N_h)$ I/Os
3. Setup $\leq 2N_{h-1}$ parent pointers for level $h-1$
   a. Construct from the list of child records at level h pairs $< p, c >$, where p is a pointer to a child record at level h and c the corresponding pointer to the node record at level $h-1$
   b. Sort the list of pairs w.r.t. c $\mathcal{O}(\frac{N_{h-1}}{B} \log_{M/B} \frac{N_{h-1}}{B})$ I/Os
   c. Update the parent pointers at level $h-1$ by a simultaneous scan of the blocks storing level $h-1$ and the sorted list constructed above $\mathcal{O}(\frac{N_{h-1}}{B})$ I/Os

Level Balancing – II

Using $N_{h-1} = \frac{b}{3} N_h$, the total # I/Os to rebalance level h becomes

$$\mathcal{O}(N_h + N_h + \frac{N_{h-1}}{B} \log_{M/B} \frac{N_{h-1}}{B} + \frac{N_{h-1}}{B})$$

$$= \mathcal{O}(N_h + \frac{bN_h}{B} \log_{M/B} \frac{N}{B} + \frac{bN_h}{B})$$

$$= \mathcal{O}(N_h(1 + \frac{b}{B} \log_{M/B} \frac{N}{B}))$$

Split and Join

Split operation:

- Split the nodes along the splitting path
- If necessary rebalance the lowest level h with $> 2N_h$ non-root nodes

Join operation:

- Link trees by adding one or two edges, and split nodes of degree $b+1$
- If necessary rebalance the lowest level h with $> 2N_h$ non-root nodes

Because each insert, delete, split and join operation at most introduces $\mathcal{O}(1)$ new nodes at each level of the forest, a rebalancing is only triggered at level h once every $\Omega(N_h)$ operation, implying that the amortized restructuring cost for level h of an operation is $\mathcal{O}(1 + \frac{b}{B}\log_{M/B}\frac{N}{B})$ I/Os

Theorem Level-balanced B-trees support insert, delete, split and join operations in $\mathcal{O}((1 + \frac{b}{B}\log_{M/B}\frac{N}{B})\log_b N)$ I/Os (amortized) and searches and order queries in $\mathcal{O}(\log_b N)$ I/Os (worst-case)

Applications of Level-Balanced B-Trees

Reachability in planar st-graphs

Updates : $\mathcal{O}((1 + \frac{b}{B} \log_{M/B} \frac{N}{B}) \log_b N)$ I/Os (amortized)

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

— I/O bounds are resp. $\mathcal{O}(\log_B^2 N)$ and $\mathcal{O}(\log_B N)$ for $b = B/\log B$

Point location in planar monotone subdivisions

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

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

Conclusion

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

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

Open Problem

- Improve I/O bound $\rightarrow \log_B N$?