CSE 326
Lecture 17

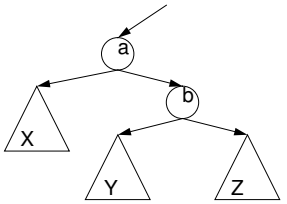
Spring Quarter 1995

Corrected Right Rotation Code

- Single_Rotate_Right(ref n) {

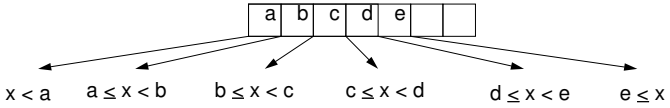
  local node pointer p;

  p := n.right;
  n.right := p.left;
  p.left := n;
  n.height := max(ht(n.left),ht(n.right)) + 1;
  p.height := max(ht(p.left),ht(p.right)) + 1;
  n := p }



B - Trees

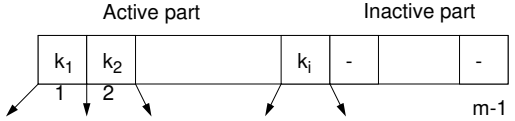
- B-trees are multiway trees which are not binary



- B-tree nodes contain search keys only, actual data is stored at the leaves.
- B-tree developed for database applications where minimizing the number of disk accesses is important. B-tree nodes are large enough to balance the disk access time with the internal search time within a B-tree node.
- B-trees are generally useful in a virtual memory environment to minimize page faults and perhaps even in a RISC environment where minimizing cache misses is important.

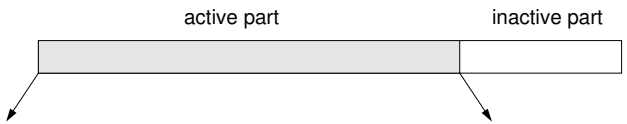
Properties of B-Trees of Order m

- The root is either a leaf or has between 2 and m children
- All non-leaf node (except the root) have between | m/2 | and m children.
- All leaves are at the same depth and contain between | m/2 | and m - 1 keys.
- A node will have keys $k_1, k_2, k_3, ..., k_i$ where | m/2 | < i < m

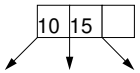


Find in a B-Tree

- If m is large then binary search within a B-tree node.



- If m is small then linear search is fine.

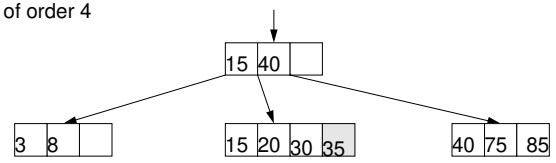


Insertion in a B-tree

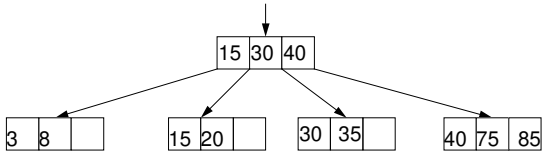
- Begin by inserting in the proper leaf.
- If the leaf ends up with m items, it has overflowed.
- An overflowed leaf node is split into two nodes with | m/ 2 | in one and | m/2 | in the other. A new child is added to the parent.
- An overflowed internal node with m+1 children is split into two nodes, one with | (m+1)/2 | children and the other with | (m+1)/2 | children. A new child is added to the parent.
- If overlow propogates to the root, then the node is split into two and a new root with two children is formed.

Insertion with Overflow - Leaf Case

B-tree of order 4

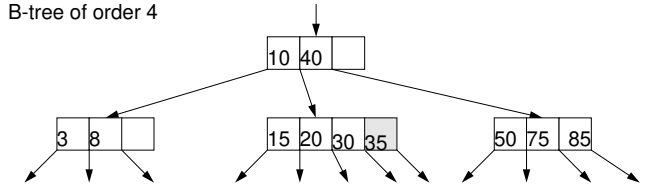


Splitting a node

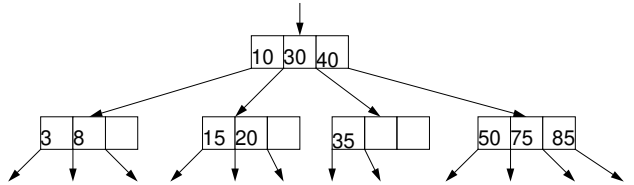


Insertion with Overflow - Internal Node Case

B-tree of order 4



Splitting a node



Example of Insertion

- Insert 1,2,3,4 5,6 7,8, ... into a B-tree of order 4

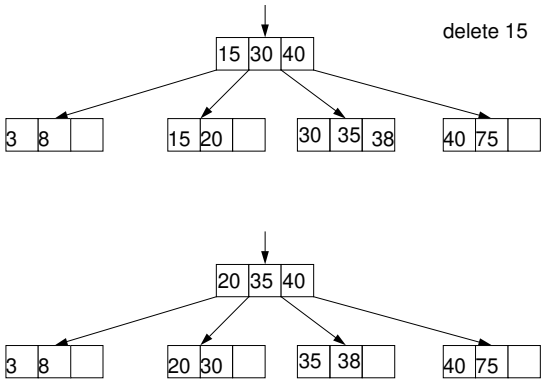
B-Tree Deletion

- In deletion a leaf may underflow, that is, have < | m/2 | items.

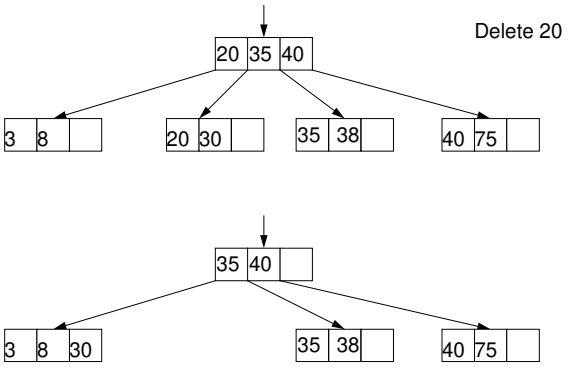
  1. Borrow from a sibling.

  2. If borrowing is impossible then give all to siblings evenly and physically delete the node. Deletion may propogate to the parent.

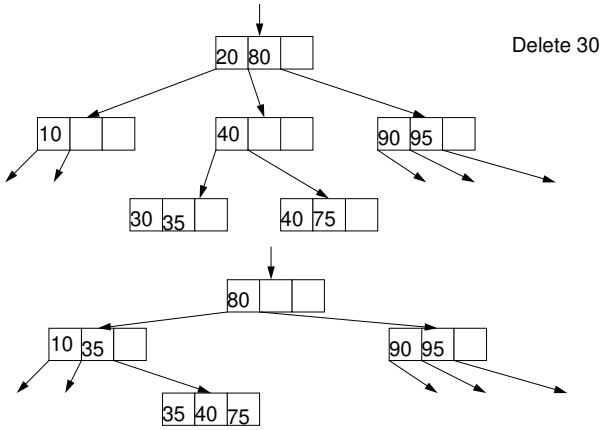
Borrowing



Elimination without Propogation



Deletion with Propogation



Summary of B-Trees

- B-trees of order 3 are often called 2-3 trees and B-trees of order 4 are often called 2-3-4 trees.
- B-trees are used when accessing a new node can be costly, like in data bases. The root and next level are often kept in main memory.
- B-tree insertion causes splitting which may propogate.
- B-tree deletion causes borrowing (good) or physical deletion which may propogate. Propogation is quite rare if the order of the tree is large.
- If order of the tree is large like 64 or 128, then only a few levels are need to store very large data sets. For example, if m = 128 then a B-tree of height 4 will store at least 30 million items.

Summary of Search Trees

- Plain vanilla binary search trees

  Good on average - simple to understand and use.
- AVL trees - height balanced trees

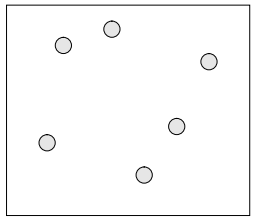
  Good in worst case - more complicated.
- Splay trees

  Good amortized cost and good locality properties - slightly more complicated than AVL
- B-trees

  Good in worst case when node access is expensive - most complicated of the algorithms.

Multidimensional Search Trees

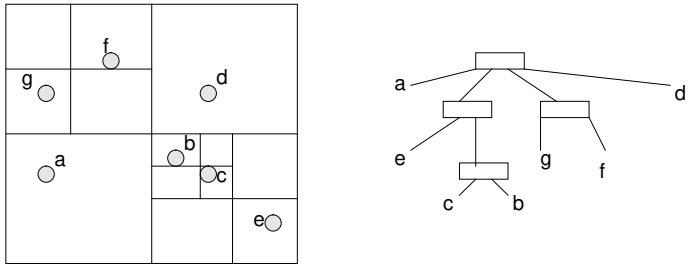
- Let us examine just two dimensions. We want to store a set of ordered pairs (x_1,y_1), (x_2,y_2), (x_3,y_3), ... , (x_n,y_n). There is no specific order to the points as there is with say integers or character strings.



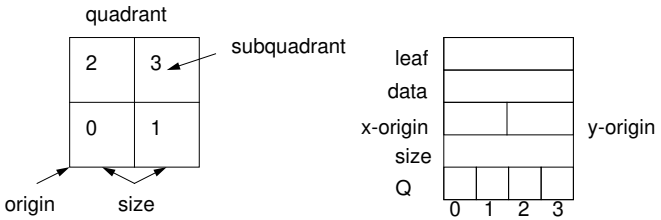
Application of Multidimensional Search

- Astronomy - simulation of galaxies, 3 dimensions.
- Protein folding in molecular biology - 3 dimensions.
- Lossy data compression - 4 to 16 dimensions.
- Image processing - 2 dimensions
- Graphics - 2 or 3 dimensions
- Geographical Databases - 2 or 3 dimensions

Quad Trees



Quad Tree Node

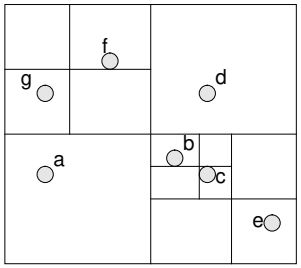
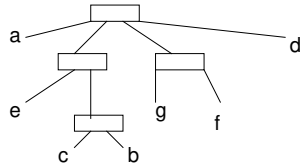


- leaf is boolean. If leaf is true then data holds data or pointer to data.
- origin is the bottom left corner of the quadrant.
- size is the width and height of the the quadrant.
- Q is an array which contains pointers to other active subquadrants.

Find in Quad Trees

- Find(x,y,n) finds the node which has the pair (x,y) in it and returns null if there is no such node.
- Find(x,y,n) {

  if n.leaf = true then
  if n.data = (x,y) then return(n) else return null
  else
  return(find(x,y,Q[quadrant(x,y,n)]) }
- quadrant(x,y,n) {

  return((x < n.x_origin + size/2) + 2(y < n.y_origin + size/2)) }

Example