

B-Trees, 2-3 Trees

Recitation 7: B-Trees, 2-3 Trees

1

Outline

- Trade-off when using secondary storage
- B-tree definition
- Height of B-tree
- Operations
 - Search
 - Insert
 - Delete

Recitation 7: B-Trees, 2-3 Trees

2

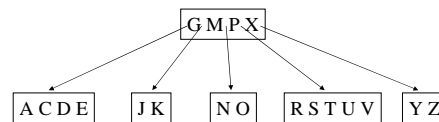
Secondary Storage

- Magnetic disks is approx 100 time cheaper, but 10,000 times slower than main memory
- For applications such as database systems, main memory not large enough - secondary storage needed
- Total running time = CPU time + disk access time
- Search trees that operate on secondary memory need to minimize disk accesses - minimize height of the search tree.

Recitation 7: B-Trees, 2-3 Trees

3

B-Tree



B-tree with minimum degree $t = 3$

Recitation 7: B-Trees, 2-3 Trees

4

- Every node x has the following fields:
 - $n[x]$, the number of keys currently stored in x
 - the $n[x]$ keys, stored in nondecreasing order
 - $key_1[x] \leq key_2[x] \leq \dots \leq key_{n[x]}[x]$
 - $leaf[x]$, a boolean value that is true iff x is a leaf
- Each **internal** node x also contains $n[x]+1$ pointers $c_1[x], c_2[x], \dots, c_{n[x]+1}[x]$ to its children.
- The keys $key_i[x]$ separate the ranges of keys stored in each subtree: if k_i is any key stored in the subtree with root $c_i[x]$, then
 - $k_1 \leq key_1[x] \leq k_2 \leq key_2[x] \leq \dots \leq key_{n[x]}[x] \leq k_{n[x]+1}$
- All leaves have the same depth, which is the tree's height h

Recitation 7: B-Trees, 2-3 Trees

5

- The minimum degree of the B-tree $t \geq 2$ is an integer such that
 - Every node other than the root must have at least $t - 1$ keys. Every internal node other than the root must have at least t children. The root must have at least one key
 - Every node can contain at most $2t-1$ keys. An internal node can have at most $2t$ children. The node is full if it contains exactly $2t-1$ keys.

Recitation 7: B-Trees, 2-3 Trees

6

Height of B-tree

- **Theorem:**
If $n \geq 1$, then for any n -key B-tree T of height h and minimum degree $t \geq 2$,
 $h \leq \log_t (n+1)/2$
- **Proof:**
 - The root contains at least one key
 - All other nodes contain at least $t-1$ keys.
 - There are at least 2 nodes at depth 1, at least $2t$ nodes at depth 2, at least 2^{i-1} nodes at depth i and $2^{t^{i-1}}$ nodes at depth h

Recitation 7: B-Trees, 2-3 Trees

7

$$\begin{aligned}
 n &\geq 1 + (t-1) \sum_{i=1}^h 2t^{i-1} \\
 &= 1 + 2(t-1) \left(\frac{t^h - 1}{t-1} \right) \\
 &= 2t^h - 1
 \end{aligned}$$

- So $t^h \leq (n+1)/2$ as required.
- B-trees save a factor of about $\lg t$ over red-black trees in the number of nodes examined (disk accesses).

Recitation 7: B-Trees, 2-3 Trees

8

Search

- **Search(x,k)**
 - while $i \leq n[x]$ and $k > \text{key}_i[x]$
 - increment i
 - if $i \leq n[x]$ and $k = \text{key}_i[x]$
 - then return (x,i)
 - if leaf[x]
 - then return nil
 - else return Search($c_i[x],k$)
- Run time in each node is $O(t)$
- Height of tree is $O(\log_t n)$
- Total run time is $O(t \log_t n)$

Recitation 7: B-Trees, 2-3 Trees

9

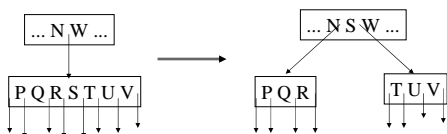
Insert

- Cannot just create a new leaf node and insert it
 - resulting tree is not B-tree
- Insert new key into an existing leaf node
- If leaf node is full
 - Split full node y (with $2t-1$) keys around its median key, y into two nodes each having $t-1$ keys
 - Move the median key into y 's parent.
 - If parent is full, recursively split, all the way to the root node if necessary.
 - If root is full, split root - height of tree increase by one.

Recitation 7: B-Trees, 2-3 Trees

10

Split

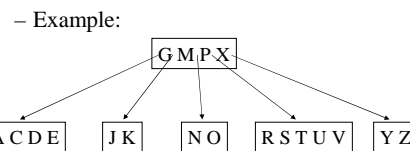


- Splitting a node with $t=4$

Recitation 7: B-Trees, 2-3 Trees

11

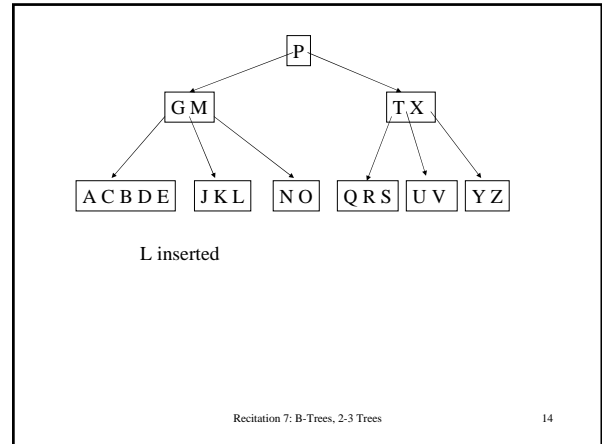
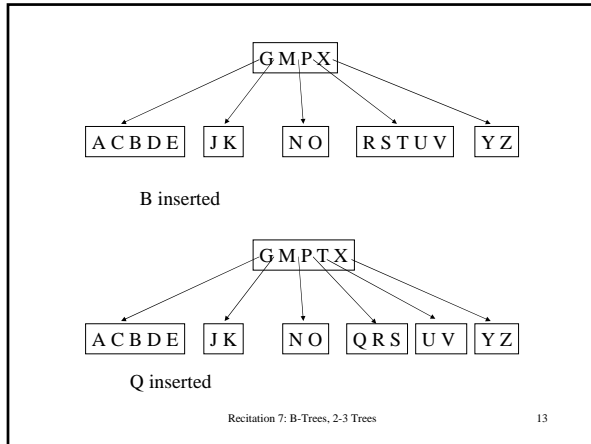
- Can also insert in a single pass down the tree, instead of going down during search then recursively splitting on the way up
 - Split full nodes encountered on the way down during search.



Initial tree, $t=3$

Recitation 7: B-Trees, 2-3 Trees

12



Delete

- If k is in an internal node, swap k with its inorder successor (in a leaf node) then delete k from the leaf node.
- Deleting k from a leaf x may cause $n[x] < t-1$.
 - if the left sibling has more than $t-1$ elements, we can transfer an element from there to retain the property $n[x] \geq t-1$. To retain the order of the elements, this is done by moving the largest element in the left sibling to the parent and moving the parent to the left most position in x

Recitation 7: B-Trees, 2-3 Trees 15

- else, if right sibling has more than $t-1$ element, transfer from right sibling through the parent.
- else, merge x with left sibling. One pointer from the parent needs to be removed in this case. This is done by moving the parent element into the new merged node. If the parent now has fewer than $t-1$ element, recurse on the parent.

- Height of the tree may be reduced by 1 if root contains no element after delete.
- Can also do delete in one pass down, similar to insert (see textbook).

Recitation 7: B-Trees, 2-3 Trees 16

Summary

- Minimize the number of disk access by increasing the branching factor.
- Height $O(\log_t n)$, where t is min degree
- Dynamic set operations search, successor, predecessor, min, max, insert, delete run using $O(\log_t n)$ disk accesses.

Recitation 7: B-Trees, 2-3 Trees 17