Outline

Last time:

- AVL Tree
- Amortized analysis

Today:

- Midterm logistics
- Finished up AVL Tree
- Multi-way Search Tree
- 2-3 tree
- B-tree
Outline

Midterm this Thu, 10/25, 6-8 pm in DOW 1013
Review session this evening Tue, 10/23, 6-8 pm in COOLG 906

Cover everything up to today’s lecture

Open book, open notes, calculator, laptop, Internet ok; cheat sheet suggested, don’t rely on $O(N)$ search during the exam

No discussion this week (Wed and Fri)

GSIs will hold office hour on Wed in the Dude instead

No class on Thu, I will hold office hour instead

I will hold office hour on Fri as usual

Sugih Jamin (jamin@eecs.umich.edu)
Multi-way Search Tree

A node on an $M$-way search tree with $M - 1$ distinct and ordered keys, \( \{k_1, k_2, k_3, \ldots, k_{M-1}\} \), has $M$ children, \( \{T_0, T_1, T_2, \ldots, T_{M-1}\} \) (see Preiss Fig. 10.9)

Every element in child $T_i$ has a value larger than $k_{i-1}$ and smaller than $k_i$

$M$ doesn’t have to be the same for every node
Searching a Multi-way Search Tree

Search on an $M$-way search tree is similar to a BST, except more than 1 CMPs are necessary at each node.

Let $n$ be the number of keys on the tree, $n = (M - 1)N$, $N$ number of internal nodes.

If all nodes have $M - 1$ keys, and you do linear search on a node, it takes $O(M \log_M N)$ to search an $M$-way search tree.

If you do binary search on the nodes, it takes $O(\log_2 M \log_M N)$ to search an $M$-way search tree.
Balanced Multi-way Search Tree

We will look at:

- 2-3
- B-tree
2-3 Tree

How do you keep an AVL tree balanced?

2-3 tree keeps tree balanced by storing more than 1 keys per node in order to maintain a perfect tree

Properties (balance condition) of 2-3 trees:

1. all leaves are at the same depth and contain 1 or 2 keys
2. an internal node either has 1 key and 2 children (a 2-node), or has 2 keys and 3 children (a 3-node); there’s no single-child internal node
3. a key in an internal node is “between” the keys in the subtrees of its adjacent children

Sugih Jamin (jamin@eecs.umich.edu)
**Search Time on 2-3 Trees**

A 2-3 tree of height $h$ has fewest number of nodes when all internal nodes are 2-nodes (a BST). Since all leaves must be at the same depth, the tree is perfect and number of nodes (and therefore keys)

$$n = 2^{h+1} - 1, \quad h = \lfloor \log n \rfloor$$

A 2-3 tree of height $h$ has most number of nodes when all internal nodes are 3-nodes. Number of nodes $N = \sum_{i=0}^{h} 3^i = (3^{h+1} - 1)/2$, number of keys $n = 3^{h+1} - 1$, and $h = \lfloor \log_3 n \rfloor$

Search time on 2-3 trees: $O(\log n)$
Insertion on a 2-3 Tree

1. Search for leaf where key belongs. Remember the search path
2. If leaf is a 2-node, add key to leaf
3. If leaf is a 3-node, split it into two 2-nodes with the 1st and 3rd keys and pass the middle key up to parent
4. If parent is a 2-node, add the child’s middle key, else split parent by Step 3 above
5. If there’s no parent, create a new root (increase tree height by 1)

Idea: whereas a BST increases height by extending a single path, a 2-3 tree increases height globally by raising the root, hence it’s always balanced

Deletion is more tedious, requiring merging with sibling nodes (see handout)
Empirical Performance Comparison

For $2^{16}$ items with integer keys

<table>
<thead>
<tr>
<th>data structure</th>
<th>search</th>
<th>insert</th>
<th>remove</th>
</tr>
</thead>
<tbody>
<tr>
<td>skip lists</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td><strong>AVL tree</strong></td>
<td>0.91</td>
<td>1.55</td>
<td>1.46</td>
</tr>
<tr>
<td><strong>2-3 tree</strong></td>
<td>1.05</td>
<td>3.2</td>
<td>3.65</td>
</tr>
<tr>
<td>splay trees</td>
<td>9.6</td>
<td>7.8</td>
<td>9.0</td>
</tr>
<tr>
<td>top-down splaying</td>
<td>3.0</td>
<td>2.5</td>
<td>3.1</td>
</tr>
</tbody>
</table>

Pugh, *Skip Lists*, CACM, June 1990
**External Search**

$M$-way search tree can be used as a kind of index for external (disk) searching of large files/databases

Characteristics of disk access:

- four or more orders of magnitude *slower* than memory access
- for efficiency, data usually transferred in blocks of 512 bytes to 8KB

Given the above characteristics, how can one speed up external search? Put as much info as possible on each disk block, for example, by making each node on an $M$-way search tree the size of a disk block
B-Tree

Invented by Bayer and McCright in 1972

A B-Tree is a balanced $M$-way search tree, with $M \geq 2$

B-Tree’s balance condition (see Preiss Fig. 10.10):

1. the root of $T$ has at least 2 children and at most $M$ children
2. all internal nodes of $T$ other than the root have
   between $\lceil M/2 \rceil$ and $M$ subtrees (think of it as an $(\lceil M/2 \rceil, M)$-tree)
3. all external nodes of $T$ are at the same level

Height of a B-Tree: $h \leq \log \lceil M/2 \rceil \frac{n+1}{2}$, $n$ number of internal nodes
Hence $h = O(\log n)$
Insertion and Removal on B-Trees

The complexity of B-Trees comes in maintaining the second requirement of the balance condition:

- after an insertion, a node may overflow and need to be split (in a manner similar to 2-3 tree, raising the root, see Preiss Figs. 10.11, 10.12)
- after a removal, a node may underflow and must be merged with its sibling or an AVL LL or RR rotation may be necessary
- see Preiss, Section 10.7

Insertion and removal each takes $O(M \log n)$ time