2-3 Trees
2-3 trees: a dictionary for general data

Assume data items are totally ordered ($<$, $>$, $=$)

Assume $n$ items in the dictionary

Structure: a tree

- Data stored only at leaves (no duplicates)
- All leaves at the same level, in sorted order
- Each internal node:
  - has either 2 or 3 children
  - has a “guide”: the maximum data item in its subtree

Height of tree is $O(\log n)$
Example
Search($x$): use guides

Insert($x$): Search for $x$, and if it should belong under $p$:

add $x$ as a child of $p$ (if not already present)

if $p$ now has 4 children:

• split $p$ into two two nodes, $p_1$ and $p_2$, each with two children

• process $p$’s parent in the same way

• Special case: no parent — create new root, increasing height of tree by 1

Also need to update “guides” — easy

Time = $O$(height) = $O$(log $n$)
Case when $p$ ends up with 4 children

$p$

$w \ x \ y \ z$

$p$

$w \ y \ z$

$p_1$

$w \ x$

$p_2$

$y \ z$
Delete(x): Search for x, and if found under p:
remove x
if p now only has one child:
  • if p is the root: delete p (height decreases by 1)
  • if one of p’s siblings has 3 children: borrow one
  • if none of p’s siblings has 3 children:
      ◦ one sibling q must have 2 children
      ◦ give p’s only child to q
      ◦ delete p
      ◦ process p’s parent
Easy case: borrow from sibling
Harder case: give away only child
2-3 trees: summary

Assume $n$ items in dictionary

Running time for lookup, insert, delete:
  $O(\log n)$ comparisons, plus $O(\log n)$ overhead

Space: $O(n)$ pointers
Dictionaries for strings: a comparison

Hash tables or balanced trees (e.g., 2-3 trees)?

Assume $n$ strings of length $t$ over an $m$ letter alphabet

Time per lookup:

- balanced trees: $O(t \log n) - O(\log n)$ comparisons, each takes time $O(t)$
- hash tables: $O(t)$ (expected)

Support for other operations:

- balanced trees support fast in-order traversal (and other things)
- hash tables: nothing
2-3 Trees: Join and Split

$\text{Join}(T_1, T_2)$ joins two 2-3 trees in time $O(\log n)$

Assume $\max(T_1) < \min(T_2)$

Assume $T_i$ has height $h_i$ for $i = 1, 2$

Case 1: $h_1 = h_2$
Case 2: $h_1 < h_2$

- Attach $\nu$ as the left-most child of $p$
- If $p$ now has 4 children, we split $p$, and proceed up the tree as in Insert
- Time: $O(h_2 - h_1) = O(\log n)$

Case 3: $h_1 > h_2$ — similar
$\text{Split}(T, x) \Rightarrow (T_1 [\leq x], T_2 [> x])$
Observations:

- Initially: at most 2 trees of any given height — except there may be 3 of height 0
- Let $T_1, T_2$ have heights $h_1, h_2$, where $h_1 \geq h_2$
  
  Then $\text{Join}(T_1, T_2)$ takes time $O(h_1 - h_2 + 1)$, and produces a tree of height $h_1$ or $h_1 + 1$
- Let $T_1, T_2, T_3$ have heights $h_1, h_2, h_3$, where $h_1 = h_2 \geq h_3$
  
  Then $\text{Join}(T_1, \text{Join}(T_2, T_3))$ takes time $O(h_2 - h_3 + 1)$, and produces a tree of height $h_1$ or $h_1 + 1$
If the distinct heights of the trees to merge are 
\[ h_1 > h_2 > \cdots > h_k, \]
then the total cost is \( O(t) \), where
\[
t \leq (h_1 - h_2 + 1) + (h_2 - h_3 + 1) + \cdots + (h_{k-1} - h_k + 1)
\]
\[ = h_1 - h_k + k - 1 \]
\[ \leq 2h, \]
where \( h \) is the height of the original tree

Conclusion: total time for Split is \( O(\log n) \)
Augmenting 2-3 trees

Examples

Store # of items in subtree at each internal node

Queries:

- What is the $k$th smallest item?
- How many items are $\leq x$?
Items may be marked with an attribute, say, “active”/“inactive”

Store a count of active items in subtree at each internal node

Queries:

- What is the $k$th smallest active item?
- How many active items are $\leq x$?
• Attribute flipping . . .
• Operation $Flip(x, y)$ flips all attribute bits of items in the range
• Assume attributes are bits
• Store an XOR-bit at each internal node
  ◦ “effective” value of the attribute is the XOR of all bits on path from root to leaf
• To perform $Flip(x, y)$:
  ◦ trace paths $e, f$ to $x, y$
  ◦ flip bits at $x, y,$ and all roots of “internal” subtrees
Example: