2-3 Trees
Dictionary data type

Dictionary operations

- Insert
- Search
- Delete

2-3 trees:

- A kind of balanced search tree
- Assume data items are totally ordered (<, >, =)
- Assume $n$ items in the dictionary
- Time per dictionary operation is $O(\log n)$
- Support of other useful operations as well
Basic 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

Let $h :=$ height of tree

$n \geq 2^h$ — why? because every internal node has at least two children

$\therefore h \leq \log_2 n$
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 = $O(\text{height}) = O(\log n)$
Case when $p$ ends up with 4 children
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

\[ q \]
\[ v \quad w \]

\[ p \]
\[ x \quad y \]

\[ q \]
\[ v \quad w \quad y \]

\[ p \]
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
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 $v$ 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
Split($T, x$) $\iff$ ($T_1 \leq x$, $T_2 \geq x$)

join from inside out
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 $Join(T_1, T_2)$ 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 $Join(T_1, Join(T_2, T_3))$ takes time $O(h_2 - h_3 + 1)$, and produces a tree of height $h_1$ or $h_1 + 1$
Suppose the distinct heights of the trees to merge are
\[ h_1 > h_2 > \cdots > h_k \]

**Invariants:**

- After merging trees of height \( h_i, \ldots, h_k \), we obtain a tree of height \( h_i \) or \( h_i + 1 \)
- Time to merge this new tree with the original trees of height \( h_{i-1} \): \( O(h_{i-1} - h_i) \)

Total cost of all merge steps is \( O(t) \), where
\[
t \leq (h_1 - h_2) + (h_2 - h_3) + \cdots + (h_{k-1} - h_k)
= h_1 - h_k
\leq h,
\]
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 kth smallest active item?
- How many active items are ≤ x?
- Attribute flipping ...
- Operation $\text{Flip}(x, y)$ 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 $\text{Flip}(x, y)$:
  - trace paths $e, f$ to $x, y$
  - flip bits at $x, y$, and all roots of “internal” subtrees
Example: