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
\[ \text{Split}(T, x) \implies (T_1 [\leq x], T_2 [> x]) \]
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 $Join(T_1, T_2)$ 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 $h_1 + 1$
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 $\text{Flip}(x, y)$ 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:
Priority Queues

Priority Queue operations:

- Insert
- Delete Min

Recall basic “heap” data structure

Structure: “nearly” perfect binary tree

Heap condition: $val(v) \geq val(parent(v))$
Insert:

```
  1
 /  \
 5   3
 |   |
 9   8
 |   |
 6   6
```

``float up''

Delete Min:

```
  1
 /  \
 5   3
 |   |
 9   6
 |   |
 9   9
```

``sink down''
Insert and Delete Min: time $O(\log n)$
Array layout (an optimization)

If array is indexed from 1:

- $LeftChild(i) = 2i$
- $RightChild(i) = 2i + 1$
- $parent(i) = \lfloor i/2 \rfloor$
Building a heap from scratch in time $O(n)$

- Put all items in the array
- Let $h$ be the height of the (implicit) tree
- Process nodes at levels $h-1, h-2, \ldots, 0$:
  - let the value at node $v$ “sink” to its correct position in the subtree rooted at $v$ (as in Delete Min)
- After processing level $j$, each node at level $j$ is the root of a heap
- Cost for level $j$: $O((h - j)2^j)$
  - $2^j$ nodes at level $j$, each costs time $O(h - j)$ to process
• Total Cost is $O(t)$, where

$$t \leq 1 \cdot 2^{h-1} + 2 \cdot 2^{h-2} + 3 \cdot 2^{h-3} + \cdots + h \cdot 2^0$$

$$= 2^h \sum_{i=1}^{h} \frac{i}{2^i}$$

Now, $h = \lceil \log_2 n \rceil$, so $2^h \leq n$

Also, $\sum_{i=1}^{\infty} i/2^i = 2$:

\[
\begin{array}{cccc}
1/2 & 1/4 & 1/4 \\
1/8 & 1/8 & 1/8 \\
\vdots & \vdots & \vdots \\
1 & 1/2 & 1/4 \\
\end{array}
\]

$\therefore t \leq 2n$
Mergeable Priority Queues

Operations:
- Insert
- Delete Min
- Merge two queues

Using heaps:
- need to re-build — time $O(n)$

Using 2-3 trees:
- Can support all 3 operations in time $O(\log n)$
Mergeable Priority Queues using 2-3 trees

- Same tree structure as ordinary 2-3 trees
- Items stored at leaves, but
  - duplicates allowed
  - values not in any particular order
- Internal nodes contain “min values” as guides
- Insert: just make a new leaf (anywhere), and update guides
- Delete Min: follow guides to find min, delete, and update guides
- Merge: use Join procedure, and update guides
This data structure supports Insert, Delete Min, and Merge, in time $O(\log n)$

It does not directly support Search and Delete

Implementation notes for both heaps and 2-3 trees:

- data structure stores pointers to objects
- objects may contain a “hook” into the data structure
Using “hooks,” we can also implement operation Adjust Value

- heaps: change value and “float up” or “sink down”
  
  Time: $O(\log n)$

- 2-3 trees: change value and update guides
  
  Time: $O(\log n)$