2-3 Trees
Dictionary: an abstract data type

A container that maps keys to values

Dictionary operations

- Insert
- Search
- Delete

Several possible implementations

- Balanced search trees
- Hash tables
2-3 trees

A kind of balanced search tree
Assume keys are totally ordered \((<, >, =)\)
Assume \(n\) key/value pairs are stored in the dictionary
Time per dictionary operation is \(O(\log n)\)
Support of other useful operations as well
Basic structure: a tree

- Keys/value pairs stored only at leaves (no duplicate keys)
- All leaves at the same level, with keys sorted order
- Each internal node:
  - has either 2 or 3 children
  - has a “guide”: the maximum key in its subtree
Example
Let \( h \) := height of tree (Recall: height = length of longest path from root to leaf)

**Claim:** \( n \geq 2^h \)

- Proof by induction on \( h \)
- **Base case:** \( h = 0, n = 1 \) \( \checkmark \)
- **Induction step:** \( h > 0 \), assume claim holds for \( h - 1 \)
  - Tree has a root node, which has either 2 or 3 children
  - Each of these children is the root of a subtree, which itself is a 2-3 tree of height \( h - 1 \)
  - By induction hypothesis, if the \( i \)th subtree has \( n_i \) leaves, then \( n_i \geq 2^{h-1} \) [here, \( i = 1..2 \) (or 3)]
  - \( \therefore n = \sum_i n_i \geq \sum_i 2^{h-1} \geq 2 \cdot 2^{h-1} = 2^h \) \( \checkmark \)

**Corollary:** \( h \leq \log_2 n \)
Search(x): use guides

Search(x, p, height):  // Invoke as Search(x, root, h)
if (height > 0) then
  if (x ≤ p.child0.guide) then
    return Search(x, p.child0, height − 1)
  else if (x ≤ p.child1.guide or p.child2 = null) then
    return Search(x, p.child1, height − 1)
  else
    return Search(x, p.child2, height − 1)
else
  if x = p.guide then
    return p.value
  else
    return null (or a default value)
**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, p₁ and p₂, 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

Diagram:

```
  q
 /\  /
 u v w
```

```
  p
 /\  /
 x y
```

Diagram:

```
  q
 /\  /
 u v
```

```
  p
 /\  /
 w y
```
Harder case: give away only child

\[
\begin{align*}
q \\
\downarrow \\
\nu \quad w
\end{align*}
\]

\[
\begin{align*}
p \\
\downarrow \\
x \quad y
\end{align*}
\]

\[
\begin{align*}
q \\
\downarrow \\
\nu \quad w \quad y
\end{align*}
\]

\[
\begin{align*}
p
\end{align*}
\]
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
Augmenting 2-3 trees

Idea: augment nodes with additional information to support new types of queries

Example: 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 $\text{FlipRange}(x, y)$ flips all attribute bits of items in the range

- Assume attributes are bits

- Store an attribute bit at every node: internal nodes and leaves
  - “effective” value of the attribute is the XOR of all bits on path from root to leaf

- To perform $\text{FlipRange}(x, y)$:
  - trace paths $e, f$ to $x, y$
  - flip bits at $x, y$, and all roots of “internal” subtrees
Example:
2-3 Trees: Join and Split

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:** $h_1 = h_2$

Time: $O(1)$
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
\[ \text{Split}(T, x) \implies (T_1 \leq x, T_2 \geq x) \]
Analysis of “rejoin” step

Start with trees $T_0, T_1, \ldots, T_k$ with $h_i = \text{height}(T_i)$

$h_0 \leq h_1 \leq \cdots \leq h_k$ [Monotonicity]

There are at most two trees of any given height (except there may be three of height 0) [Diversity]

Form new trees $T_0^*, T_1^*, \ldots, T_k^*$ with $h_i^* = \text{height}(T_i^*)$:

$T_0^* = T_0, \quad T_{i+1}^* = \text{Join}(T_i^*, T_{i+1})$

Total running time: $O(S)$, where

$$S = \sum_{i=0}^{k-1} (|h_{i+1} - h_i^*| + 1)$$
Claim: For $i = 0 \ldots k - 1$: $h_i^* \in \{h_i, h_i + 1\}$

From the claim, $|h_{i+1} - h_i^*| \leq h_{i+1} - h_i + 1$, and so

$$S \leq \sum_{i=0}^{k-1} (h_{i+1} - h_i + 2) = 2k + h_k - h_0 = O(\log n)$$

Example:

* = a “fresh” tree: root has only 2 children
We prove the claim by induction, but we need to prove more: “strengthening the induction hypothesis”

For \( i = 0 \ldots k - 1 \), we have

\[
\begin{align*}
P(i) & : \quad h_i^* \in \{h_i, h_i + 1\} \\
Q(i) & : \quad h_i^* > h_{i+1} \implies T_i^* \text{ fresh}
\end{align*}
\]

Base case: \( i = 0 \) (recall \( h_0^* = h_0 \)) √

Induction step: assume \( P(i), Q(i) \) and prove

\( P(i+1), Q(i+1) \), where \( i = 0 \ldots k - 2 \)
Case I. Recall: $T_{i+1} = \text{Join}(T_i^*, T_{i+1})$

1. Assume $h_i^* \leq h_{i+1}$
2. By logic of Join and (1), either
   - $h_{i+1}^* = h_{i+1}$ or
   - $h_{i+1}^* = h_{i+1} + 1$ and $T_{i+1}^*$ is fresh
3. So $P(i + 1)$ holds
4. To prove $Q(i + 1)$:
   
   \[ h_{i+1}^* > h_{i+2} \implies h_{i+1}^* > h_{i+1} \quad \text{[By monotonicity]} \]
   
   \[ \implies T_{i+1}^* \text{ is fresh} \quad \text{[By (2)]} \]
Case II. Recall: $T_{i+1} = \text{Join}(T_i^*, T_{i+1})$

1. Assume $h_i^* > h_{i+1}$
2. Must have $i > 0$
3. By $Q(i)$ and (1), $T_i^*$ is fresh, and $\text{Join}$ logic implies $h_{i+1}^* = h_i^*$
4. We have
   \[
   h_i^* \leq h_i + 1 \quad [\text{By } P(i)]
   \]
   \[
   \leq h_{i+1} + 1 \quad [\text{By monotonicity}]
   \]
   \[
   \leq h_i^* \quad [\text{By (1)}]
   \]
   and so
   \[
   h_i^* = h_i + 1 = h_{i+1} + 1
   \]
5. By (4), we have $h_i = h_{i+1}$, and by diversity, monotonicity, and (2):
   \[
   h_{i+1} + 1 \leq h_{i+2}
   \]
6. Therefore:
   \[
   h_{i+1}^* = h_i^* = h_{i+1} + 1 \quad [\implies P(i+1)]
   \]
   \[
   \leq h_{i+2} \quad [\implies Q(i+1)]
   \]