Today: 2-3-4 trees
       Search, Insert, Delete
       Connection to red-black trees
       B-trees

<table>
<thead>
<tr>
<th>Dynamic set data structures</th>
<th>Insert/Delete</th>
<th>Exact Search</th>
<th>Predecessor/Successor</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>linked list</td>
<td>O(1)</td>
<td>O(n)</td>
<td>O(n)</td>
<td>&quot;=' only</td>
</tr>
<tr>
<td>sorted array</td>
<td>O(n)</td>
<td>O(lg n)</td>
<td>O(lg n)</td>
<td>comparison</td>
</tr>
<tr>
<td>heap</td>
<td>O(lg n)</td>
<td>O(1) exp.</td>
<td>O(n)</td>
<td>comparison</td>
</tr>
<tr>
<td>hash table</td>
<td>O(lg n)</td>
<td>O(1) exp.</td>
<td>O(n)</td>
<td>hashable</td>
</tr>
<tr>
<td>heap + hash table</td>
<td>O(lg n) + O(1)exp.</td>
<td>O(1) exp.</td>
<td>O(n)</td>
<td>comparison + hashable</td>
</tr>
<tr>
<td>BST</td>
<td>O(lg n)</td>
<td>if uniform Insert's</td>
<td>O(lg n)</td>
<td></td>
</tr>
<tr>
<td>balanced search tree</td>
<td>O(lg n)</td>
<td>O(lg n)</td>
<td>O(lg n)</td>
<td>comparison</td>
</tr>
</tbody>
</table>

Today’s Goal
2-3-4 trees
- force leaves to be all at the same depth & all nonleaves to branch
  ⇒ height = O(\lg n)
- achieve by relaxing binary constraint
- allow nonleaf \( x \) to have \( c \in \{2,3,4\} \) children:
  \[ \text{child}_{1}[x], \text{child}_{2}[x], \ldots, \text{child}_{c}[x] \]
- also store \( c-1 \in \{1,2,3\} \) keys:
  \[ \text{key}_{1}[x], \ldots, \text{key}_{c-1}[x] \]
- leaves store 1 to 3 keys as well

Example:

```
    20
   /\  \
  10 12
 / \ / \  \
1 4 5 11 15 17
  / \       / \       / \
  21 24 27 33
```

Search tree property:
- keys in node are in sorted order
- keys in subtree left of key k are \(\leq k\)
- keys in subtree right of key k are \(\geq k\)

![Tree Diagram](image)

keys in \(\text{child}_i[x]\) \(\leq \text{key}_i[x]\) \(\leq \) keys in \(\text{child}_{i+1}[x]\)

2-3-4 Search \((x, k)\):

1. search for key \(k\) in \(\text{keys}[x]\) \(\rightarrow O(1)\)
2. if \(\text{key}_i[x] = k\) then return \(\text{key}_i[x]\)
3. else suppose \(\text{key}_i[x] < k < \text{key}_{i+1}[x]\)
   return 2-3-4 Search \((\text{child}_{i+1}[x], k)\)

Time: \(O(\log n)\)

Example:

![Example Tree](image)

Predecessor, Successor, Inorder Traversal similar
2-3-4 Insert \((T, k)\):

1. 2-3-4 Search \((\text{root}[T], k)\) for leaf \(x\) where \(k\) belongs
2. Insert key \(k\) into \(\text{keys}[x]\), keeping \(\text{keys}[x]\) in sorted order
3. While \(x\) is overflowing (has 4 keys):
   1. Split node into left half, median, & right half

\[
\begin{array}{c}
\text{k} \quad \text{l} \quad \text{m} \quad \text{p} \\
\text{A} \quad \text{B} \quad \text{C} \quad \text{D} \quad \text{E} \\
\end{array} \quad \Rightarrow \quad \begin{array}{c}
\text{k} \quad \text{l} \\
\text{A} \quad \text{B} \quad \text{C} \\
\text{D} \quad \text{E} \\
\end{array}
\]

4. If \(\text{parent}[x] \neq \text{nil}\) (\(x\) isn't root), then promote median up to \(\text{parent}[x]\), inserting into \(\text{keys}[\text{parent}[x]]\),
   else create new \(\text{root}[T]\) with one key, the median & two children, left & right halves

5. \(x \leftarrow \text{parent}[x]\)

\text{Time: } O(\log n)
Example:

Insert(16)

Insert(18)

Insert(2)

viola!
2-3-4 Delete ($T, k$):

1. 2-3-4 Search ($\text{root}[T], k$) $\rightarrow$ key$_i[x]$

2. if $x$ is not a leaf
   then 2-3-4 Successor ($x, k$) $\rightarrow$ key$_j[y]$  $\min_{\text{child}_{i+1}[x]}$ key$_i[x] \leftarrow$ key$_j[y]$ replace with successor
   i.e. $x \leftarrow y$  & delete successor $\Rightarrow$ in leaf

3. delete key$_i[x]$, keeping keys$_i[x]$ sorted

4. while $x$ is underflowing (has no keys):
   i) try to steal from siblings:
      if an adjacent sibling $y$ of $x$ has $\geq 2$ keys
      then shift a key through parent$_[x]$ & steal a child from $y$

--- continued on next page ---
2-3-4 Delete \((T, k)\):

...continued from previous page...

(ii) if necessary, steal from parent:
if adjacent sibling(s) have only 1 key
then merge with one of them
& intervening parent key

\[ \Rightarrow \]

(iii) \( x \leftarrow \text{parent}[x] \)

**Time:** \(O(lg \ n)\)
Representing 2-3-4 trees as BSTs:
- use two kinds of nodes: red & black

- root always black
- never two adjacent red nodes
- "black depth" (# black ancestors) of all leaves are the same

→ RED-BLACK TREES [Bayer 1972] → Recitation '7

B-trees [Bayer & McCreight 1972]
- parameter \( t \geq 2 \) (\( t=2 \Rightarrow 2\cdot3\cdot4 \) trees)
- nonleaf node has \( c \) children, \( t \leq c \leq 2t \)
  except root where \( 2 \leq c \leq 2t \)
- also store \( c-1 \) keys (\( \leq 2t-1 \))
- height \( = O(\log_t n) \)
- B-Tree Search: \( O(\log_t n \cdot \lg t) = O(\lg n) \)
  \( \frac{\text{binary search}}{\text{using balanced search tree for node}} \)
- B-Tree Insert/Delete: \( O(\log_t n \cdot t) \)
  or \( O(\lg n) \)

Motivation:
- caches read whole lines of data
- want entire line to be useful
  \( \Rightarrow \) set \( t = \text{line size} \)
  \( \Rightarrow O(\log_t n) \) line reads/writes

Typical cache size:
- 512B for RAM \( \frac{\lg(t/4)}{7x} \)
- 1MB for disk \( \frac{\lg(t/4)}{18x} \)

Used by most databases & most file systems

\[ \text{Sleepycat/Berkeley DB} \quad \downarrow \quad \text{MacOS HFS/HFS+} \]
\[ \quad \text{ReiserFS} \quad \text{Linux ext3, xfs, shmfs} \]
\[ \text{Windows NTFS} \]