Red-Black Trees

(You will need to make notes on this lecture ...)
4-node sorted order

3-node

2-node

N keys = N children - 1
**Insertion**

**Special rules**

2 → 3

3 → 4

\[ ab + c \]

\[ abc + d \]

\[ \text{median promoted} \]

\[ \text{level} 3 \]
Example
Always balanced!
Why 2-3-4 Trees Remain Balanced

- **Binary Tree**
  - Additions add nodes to the bottom
  - This can unbalance the tree

- **2-3-4 Tree**
  - Additions only push nodes up
  - *So the base of the tree is effectively pushed down as a single unit, remaining balanced*
Difficulties with 2-3-4 Trees

- The main problem is that those different node sizes have different storage requirements
  - Converting a node from one type to another is going to be costly in a computer
  - You could just implement 4-nodes and not use the spare links when you want a 2- or 3-node.
    - Can be very wasteful
Binary tree implementation

- Let's keep our beloved binary trees and try to 'hack' 2-3-4 capabilities into it.
- Firstly we have be able to represent the different node types:
Insertion Rules

Insert as per BST
Colour new link red
if (two reds in a row) {
    if (4-node equivalent) promote to fix
else rotate to fix
}
The 'Normal' View

- Normally the links aren't represented as objects with properties in our code, just the nodes
  - So we 'colour' the nodes according to the incoming (parent) link
  - This is the usual way to view a "red-black tree"
The Red-Black Properties

- Every node is red or black

- The root node is black
  - Because red nodes are linked to the ends of the red links we had. The root has no parent so cannot be on the end of any link

- If a node links to a NULL node, the NULL 'node' is black
  - Otherwise we'd have an incomplete 2-3-4 node
The Red-Black Properties

- Every red node has black children
  - None of the binary representations of 2-3-4 nodes requires two consecutive red links so black must follow red

- Every path from the root to a leaf must visit the same number of black nodes
  - There is one black node for each 2-3-4 node and we know 2-3-4 trees are balanced so the black nodes represent the path through the balanced 2-3-4 tree.
Red-Black Analysis

- We know that whichever route we take from the root to a leaf, we meet the same number of black nodes
  - Since the 2-3-4 tree nodes equate to black nodes in the RB tree
  - Call this the **black height** of the tree, BH

\[
\begin{align*}
\text{BH} &= 3 \\
\text{BH} &= 2
\end{align*}
\]
How many nodes?

- If the tree was purely black, we would have:
  
  Always a full tree if purely black
  
  \[ n = 2^{B^+} - 1 \]

- Adding in red nodes doesn't change the black height so we know

  \[ \text{Total # of nodes} = n \geq 2^{B^+} - 1 \]
How many nodes?

- In the worst case, there is one red node for every black node

\[ BH \geq \frac{h}{2} \]
So what is $h$?

\[ B^{ht} \geq \frac{n}{2} \quad n \geq 2^{B^{ht}} - 1 \]

\[ n \geq 2^{B^{ht}/2} - 1 \]

\[ n^h \geq 2^{n/2} \]

\[ \log_2(n^h) \geq \frac{n}{2} \]

\[ h \leq 2\log_2(n^h) \]

$h$ is $o(\log n)$

All BST operations were $\sim O(h)$

\[ \therefore \Rightarrow O(\log n) \quad \text{for Red-Black tree} \]
<table>
<thead>
<tr>
<th></th>
<th>Worst Case</th>
<th>Average</th>
<th>Space</th>
<th>Insert</th>
<th>Delete</th>
<th>Search</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>O(n)</strong></td>
<td>O((\log n))</td>
<td>O((\log n))</td>
<td>O((\log n))</td>
<td>O((\log n))</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>