AVL Trees
AVL Trees

- Another, more simple way to auto-balance a BST
  - (Named after inventors)
  - It is essentially our manual approach of applying rotations, but using a nice metric to decide when and what to rotate

The rule:
The heights of sibling nodes must differ by no more than one
Insertion

- Insert as usual, then look for violations

Rotate to fix

"Rebalance the tree"
Deletion

Leaf node ⇒ Delete, rebalance
Non-leaf node ⇒ Decide left/right based on height ⇒ Find predecessor or successor

Delete 78

Delete 62
AVL vs Red-Black

- AVL trees are more rigidly balanced than RB trees
  - i.e. on average they are shorter than their RB equivalents
  - Marginally faster to search
- But the extra work needed to get that shorter tree means insertions and deletions are slower
- In most cases, not much to choose between them
B-Trees
Databases

- A database is just a collection of records
  - Each record has a series of data fields
  - We want to be able to find a given record quickly
  - But there are typically lots of records
Databases

- We end up putting all the data on to hard drives due to size problems
  - We choose a field to sort by and write the records to the disc in sorted order

- A binary search finds a given record in \( \log n \) comparisons
But...

- We have a new problem
  - Every time we follow a branch on the tree it has to load in a record from disc in order to perform a comparison
  - And disc reads are sloooow

- Easy solution: pick out the index fields and keep them in memory
  - A lookup table for records
Apply Red-Black Trees?

- Sadly, the tree gets so big (lots of keys) we need to split the tree between memory and disc too!
- So we have to go to the disk almost every time we follow a branch
  - Discs are too slow for this
2-3-4 Trees Again

- We disliked 2-3-4 trees because the nodes had a variable size and we'd be wasting lots of memory if not all nodes were 4-nodes.
- But now we're trying to minimise the number of levels and they certainly did that compared to BSTs...

- Generalise to much wider trees: B-trees
B-Tree Definition and Terminology

1. "B-tree of order $x" \Rightarrow \left\lceil \frac{m}{2} = x \right\rceil \begin{cases} \text{No more than } x \text{ children} \\ \text{at least } x \text{ children} \end{cases}

[Confused]
Most operations \( \Rightarrow O(h) \) ~ what is \( h \)??

Consider smallest tree of height \( h \), min degree \( t \)

Nodes keys

\[
N_{\text{keys}} = 1 + 2(t-1) + 2t(t-1) + \ldots + 2t^{h-2}(t-1) + 2t^{h-1}
\]

\[
= 1 + 2(t-1) \sum_{i=0}^{h-1} t^i
\]

Geometric progression

\[
= 1 + 2(t-1) \frac{1 - t^h}{1 - t}
\]

\[
= 1 + 2(t-1) \frac{1 - t^{h-1}}{1 - t}
\]

\[
= 1 + 2(t-1) \frac{1 - t^{h-1}}{1}
\]

\[
= 1 + 2(t-1) \frac{1}{1 - t}
\]

Factor log better than \( \log_t \) tree

\[
h \leq \log_t \left( \binom{n-1}{t-1} \right), \quad n \geq t^{h+1} - 1
\]

\[
\frac{n+1}{2} \leq \log_t \left( \binom{n-1}{t-1} \right) \Rightarrow h+1
\]

\[
\log_t \left( \frac{2}{n+1} \right) \geq h
\]

\[
B\text{-Tree Height}
\]
Insertion

- Same idea as 2-3-4 except we split nodes into two based on the median being pushed up

```
<table>
<thead>
<tr>
<th>Blobfishrules</th>
</tr>
</thead>
<tbody>
<tr>
<td>BB</td>
</tr>
<tr>
<td>BB</td>
</tr>
<tr>
<td>BB</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>FLR</th>
</tr>
</thead>
<tbody>
<tr>
<td>BBE</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>FL</th>
</tr>
</thead>
<tbody>
<tr>
<td>BB</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
```

n=13
Order 5
min degree 5=3
Deletion

1. \( t = 3 \) \( m = 5 \) 
   - \( H \) is a leaf node
   - Delete

2. Delete R

More than half full sibling that can refill the parent if we steal from it.
Deletion

3. Delete w

4. Delete y

Steal from parent but siblings cannot refill

Merge

Height reduced!
Limitations

- B-trees only make sense when we want to minimise the tree height i.e. when traversing branches is costly.
  - B-trees mess up your lectures.

- Consider traversing the records in order
  - Just like with the BST, we often need to explore branches to find successors
  - Which means we have to load in lots of records
  - Slow

- Unfortunately, we often want both random and sequential access to records...

  $\Rightarrow B^+ \text{ Tree}$
**Idea**

- Store *all* the records at the leaves of the B-tree
  - Replace 'keys' with 'indexes'

![Diagram of B-tree]

- Now we always traverse \((h-1)\) links on search
- BUT we can sequentially link our records
- Called a “B+ tree”