What is a B+tree?

A B+-tree index takes the form of a balanced tree in which every path from the root of the tree to a leaf of the tree is of the same length. Each branch node in the tree has between \( \text{ceiling}(n/2) \) and \( n \) children, where \( n \) is fixed for a particular tree. All key values are duplicated & stored with the leaves; also, each leaf node points to the next.
Why do we use B+trees?

- The B+-tree structure can do insertion & deletion in logarithmic time.
- Hash based indices can do insertion in “constant” time.
- However, B+ trees do range queries very well.
- Compared to ordered indices, the overhead is acceptable even for frequently modified files, since the cost of file reorganization is avoided.
Root, Internal, and Leaf Nodes

- **Root Node**: Can hold fewer than \( n/2 \) pointers; however, it must hold at least two pointers, unless the tree consists of only one node.

- **Internal Node**: Form a multilevel (sparse) index on the leaf nodes. The structure of internal nodes is the same as that for leaf nodes, except that all pointers are pointers to tree nodes. An internal node may hold up to \( n \) pointers, and must hold at least \( n/2 \) pointers. The number of pointers in a node is called the *fanout* of the node.

- **Leaf Node**: Points to actual record in the database
Root, Internal, and Leaf Nodes
Insert, Update, Delete

These are the most important functions for the B+tree. You can read more about them in the textbook pages 485 - 500.

**B+tree demo**

https://www.cs.usfca.edu/~galles/visualization/BPlusTree.html

*B tree demo for comparison*

https://www.cs.usfca.edu/~galles/visualization/BTREE.html
### Example

<table>
<thead>
<tr>
<th>ID</th>
<th>Student</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>30405</td>
<td>Harsha Yeddanapudy</td>
<td>2.3</td>
</tr>
<tr>
<td>10324</td>
<td>Jake Small</td>
<td>4.0</td>
</tr>
<tr>
<td>40591</td>
<td>Krishna Aluru</td>
<td>3.7</td>
</tr>
<tr>
<td>78691</td>
<td>Steven Kim</td>
<td>2.0</td>
</tr>
</tbody>
</table>

Total # of records: 960  
Disk Block Size: 1024 bytes  
Disk performs sequential read at 2ms and a random read at 5ms  
ID - 8 bytes, Name - 16 bytes, GPA - 8 bytes
Questions

How many tuples can be stored in a disk block?

\[
\frac{\text{size of disk block}}{\text{size of tuple}}
\]

How many disk blocks do we need to store all student data?

\[
\frac{\text{total tuples}}{\text{tuples per block}}
\]

How long would it take to read two blocks? random read at the first block of each disk, sequential reads for the rest of the blocks after.

\[
2 \times \left( \left( \text{random read} \times 1 \right) + \left( \text{sequential read} \times (\text{tuples per block} - 1) \right) \right)
\]
Questions cont.

New Data:
B+ tree created on secondary index.
Each tree node has a size of 35 bytes
Tree fanout of 3 and is 60% loaded
pointer & key size are each 5 bytes
162 tuples total

Questions on next slide
Questions Cont.

How many records per leaf node?

\[
\text{floor}\left(\frac{(\text{size of node} - \text{size of pointer}) \times \text{percent full}}{\text{size of key} + \text{size of pointer}}\right)
\]

**Explanation:** we subtract the size of the pointer from the initial size of the tree node, because we know a B+tree guarantees a pointer at the end of each leaf node connecting it to the next leaf node, & branch nodes have one more pointer than keys.

The minimum height of the tree?

\[
\text{ceiling}\left(\log_{\text{fanout}}(\text{number of leaf nodes})\right)
\]
Open Question Time