B-Tree Data Structure

First finish overview of deletion in red-black tree
Recall that \( y \) is node being deleted (successor if original node to delete had 2 children).

\[
\text{doubleBlack is initialized to reference child of } y \quad \text{(or Frontier/null if } y \text{ is leaf)}
\]

```c
void deleteFixUp(RBNode doubleBlack) {
    // called on a node that has an “double” black
    while (doubleBlack != root && doubleBlack.isBlack()) {
        // stop if at root or red node
        USE rotations/recoloring to move doubleBlack up towards
        root
    }
    doubleBlack.setBlack();
    // used when loop terminates with a red node as doubleBlack
    red (or root) when loop exits
}
```
**B-tree Design**

Data Structure for Ordered Collection

<table>
<thead>
<tr>
<th>Memory Type</th>
<th>approximate access time (ns)</th>
<th>cost per megabyte ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>cache</td>
<td>5-20</td>
<td>10-75</td>
</tr>
<tr>
<td>main memory</td>
<td>60-120</td>
<td>0.50-5.00</td>
</tr>
<tr>
<td>secondary storage</td>
<td>20,000,000</td>
<td>.001-0.10</td>
</tr>
</tbody>
</table>

Table 29.1  Approximate access times in nanoseconds (ns) as compared with the cost for cache, main memory, and secondary storage (disk).

Designed for situation in which \( n \) (# elements or tags in a tagged collection) is so large that B-tree + \( n \) references to elements cannot fit in main memory.

Data is moved from disk to RAM in chunk called **page**.
Page Fault

Virtual memory lets you store data in secondary storage (act as if it's in main memory).

If program accesses data not in main memory, it's a page fault.
For a binary search tree, information we must store:

- tag (comparable) date of a historical event
- location on disk
  (secondary storage) for
  the rest of information
  associated with event
- parent ref
- left child ref
- right child ref
Want to group nodes of binary search tree into a single B-tree node that fills a disk page.

Good choice because we may only look at one node. Usually look at very few.

Not a good choice.

- Make this a disk page (go as many levels as we can fit).
- $\frac{1}{16}$th of data left to consider.
Internally, a B-tree node has tags for location on disk for data and children. The maximum number of tags is $2t$, and the number of children is at most $2t$. The tags are sorted into a B-tree structure. The INORDER traversal of the tree is shown as $0, 2t^2, 2t-1$.
B-tree properties

- $t$ is a parameter (give to constructor), make as big as you can so a full node (2t children) fits on a disk page
- completely balanced, every path from root to a leaf has exactly the same # of nodes
- every node (except root) has $\geq t-1$ tags/elements
- every node has $\leq 2t-1$ tags ($\leq 2t$ children)

in node
$\#$ children is always $\#$ tags +1 (except leaf)
When $t=2$ it's called a 2-3-4 tree.

- The number of children is 2, 3, or 4.

Red-Black tree is a conversion of a 2-3-4 tree into a binary search tree.
2-3-4 tree

between \( t + 2t \) children

full node
abc  ? not balanced

d  can't do this!

Insert:

Follow path to the leaf where the new item will be inserted, split any full node encountered on the way.
abc → bcd

a c e f g → b d f

a c e g h i

d

c d f h

a c e g w