Today: the dreaded balanced tree!

1 Why Balanced Trees?

We just did skiplists specifically to avoid balanced trees. Why am I subjecting you to this?

- Sometimes, you need worst-case $O(\log n)$
- You’ll encounter these structures in practice (e.g. GNU collection types use red-black trees)
- In some situations, extra processing is worth the pain.
- We’ll discuss one such example to motivate today’s data structure.
- B-Trees!

2 Disk-bound Computations

Here’s a model of computation that is common in, e.g., the database community.

- computer consists of a CPU/memory and a disk
- CPU is arbitrarily fast, and memory is arbitrarily fast but small
- disk reads and writes are very slow, but disk is large
- disk can read or write data only in fixed-sized chunks (“blocks” or “pages”)
- algorithm’s cost measured by # of disk operations, specifically # of pages read or written

Is this a sensible model of computation? Here are some rough numbers for modern high-performance computing systems:

<table>
<thead>
<tr>
<th>storage</th>
<th>access time</th>
<th>size</th>
</tr>
</thead>
<tbody>
<tr>
<td>L2 cache</td>
<td>2 ns</td>
<td>$10^6$ bytes</td>
</tr>
<tr>
<td>L3 cache</td>
<td>4 ns</td>
<td>$10^7$ bytes</td>
</tr>
<tr>
<td>DRAM</td>
<td>12 ns</td>
<td>$10^{10}$ bytes</td>
</tr>
<tr>
<td>SSD</td>
<td>300$\mu$s</td>
<td>$10^{12}$ bytes</td>
</tr>
<tr>
<td>disk</td>
<td>4-8 ms</td>
<td>$10^{13}$ bytes</td>
</tr>
</tbody>
</table>
How much would you pay for 16 GB of DRAM today? About $90. Compare that to two TB (2000 GB) of disk for around $75.

Indexing structures for big databases are usually disk-bound.

3 Intro to Disk-based Trees

A B-tree is a type of tree adapted to be stored mostly on disk.

- Each node $x$ can hold a variable number of keys
- If $x$ has $n(x)$ keys, then it has $n(x) + 1$ children
- # of keys per node bounded by some maximum $m$
- $m + 1$ (max # of children per node) is called the branching factor

Here’s a simple example of a B-tree – more details soon!

A few comments on how this fits into our model…

- Each node fits in one disk page (choose size accordingly)
- To access a node $x$, must do disk-read($x$)
- To modify a node $x$, must do disk-write($x$)
- Assume no fancy caching (disk pages must be reread on each access), except that root node is always cached in memory.

Disk properties determine tree’s branching factor as follows.

- Branching factor determined by how many keys plus child pointers can be crammed into a disk page
- Example: suppose we can fit 999 keys into a disk page
• How many keys can we store in a tree of height 3?
• (That means root plus two levels, not the same as the book!)
• max # of nodes in tree given by
  \[ 1 + 1000 + 1000^2 = 1001001. \]
• max # of keys is 999 times max # of nodes, or almost 1 billion.
• If root is cached, any node can be reached in only two disk accesses!

4 B-tree Definition

B-trees are not just any old disk-based tree. They have special properties designed to maintain desirable ratio of size to height. **Defn**: A B-tree \( T \) is a rooted tree (with root \( \text{root}[T] \)) with the following properties:

1. Every node \( x \) in the tree has the following info:
   • \( n(x) \) – # of keys stored in \( x \)
   • stored keys \( k_1(x) \ldots k_{n(x)}(x) \) in sorted order
   • leaf\( (x) \) – true iff \( x \) is a leaf node

2. Every non-leaf node contains \( n(x) + 1 \) child pointers \( c_1(x), \ldots c_{n(x)+1}(x) \)

3. Every key in subtree rooted at \( c_i(x) \) lies between \( k_{i-1}(x) \) and \( k_i(x) \). End ranges are half-open.

4. Every leaf has same depth, which is tree height \( h \).

5. We fix a minimum degree \( t \geq 2 \) for the tree.
   • Every node may have at most \( 2t - 1 \) keys. Max branching factor for tree is therefore \( 2t \).
   • Any node with exactly \( 2t - 1 \) keys is called full (notice that a full node always has an odd # of keys.)
   • Every node except root must have at least \( t - 1 \) keys. (Root may have as little as 1 key.)

For example, if \( t = 2 \), every internal node has at least how many? [2] children and at most how many? [4] children. This special case is called a 2-3-4 tree.
5 B-tree Height

Does the funky minimum degree property guarantee a good ratio of height to size? (i.e. worst-case height $O(\log n)$?)

**Thm:** If $n \geq 1$, then any $n$-key B-tree of minimum degree $t \geq 2$ has height

$$h \leq \log_t \left( \frac{n + 1}{2} \right) + 1 = \frac{\Theta(\log(n + 1))}{\log(t)} + 1 = O(\log n).$$

**Pf:** If B-tree has height $h$, what is fewest number of nodes such a tree could have?

- In worst case, root has 1 key, all other nodes have $t - 1$ keys.
- Let’s account the # of keys like a recursion tree.

- Summing over all levels of the tree, we have

\[
\begin{align*}
    n & \geq 1 + 2(t - 1) \sum_{k=1}^{h-1} t^{k-1} \\
    & = 1 + 2(t - 1) \sum_{j=0}^{h-2} t^{j} \\
    & = 1 + 2(t - 1) \frac{t^{h-1} - 1}{t - 1} \\
    & = 2t^{h-1} - 1
\end{align*}
\]

- Conclude that

\[
\begin{align*}
    2t^{h-1} - 1 & \leq n \\
    t^{h-1} & \leq \frac{n + 1}{2} \\
    h & \leq \log_t \left( \frac{n + 1}{2} \right) + 1 \\
    & = O(\log n)
\end{align*}
\]

which is what we want. QED