B–Trees

[Bayer & McCreight, 1972]
An Application of B–Trees

Core indexing data structure in many database management systems

TELSTRA, an Australian telecommunications company, maintains a customer database with 51,000,000,000 rows and 4.2 terabytes of data
Definition  A tree is an \((a, b)\)–tree if \(a \geq 2, \ b \geq 2a - 1\) and

- All leaves have the same depth.
- All internal nodes have degree at most \(b\).
- All internal nodes except the root have degree at least \(a\).
- The root has degree at least two.

\((a, 2a - 1)\)–trees are also denoted B–trees
Properties of \((a, b)\)–Trees

Lemma \(N\) leaves implies \(\left\lceil \frac{\log n}{\log b} \right\rceil \leq \text{height} \leq \left\lceil \frac{(\log n)-1}{\log a} \right\rceil + 1\)

Lemma Searches require \(O(\log_a n)\) I/Os if \(b = O(B)\)
Updates in \((a, b)\)-Trees

- Search for location to **insert** or **delete** a leaf
- Create/delete leaf and search key at the parent node
- Rebalance using the following transformations

\[
\begin{align*}
&\text{\underline{Split}} \\
&\begin{array}{c}
\text{b+1} \\
\text{> a} & \text{a-1}
\end{array}
\rightarrow
\begin{array}{c}
\text{b+1} \\
\frac{b+1}{2} & \frac{b+1}{2}
\end{array}
\text{\underline{Share}} \\
&\begin{array}{c}
\text{a} & \text{a-1}
\end{array}
\rightarrow
\begin{array}{c}
\text{> a} & \text{a}
\end{array}
\text{\underline{Fusion}} \\
&\begin{array}{c}
\text{a} & \text{a-1}
\end{array}
\rightarrow
\begin{array}{c}
\text{2a-1}
\end{array}
\end{align*}
\]
Example: Insert into a (2,4)–Tree

![Diagram of a (2,4)–Tree with insertion of 11]
Analysis of $(a, b)$–Trees – Insertions Only

**Theorem**

\[ n \text{ insertions imply } n/\lceil (b + 1)/2 \rceil^h \text{ splits at height } h \]

i.e. in total \( O(n/b) \) splits

**Proof**

- Nodes are created due to splits
- All nodes except the root has degree at least \( \lceil (b + 1)/2 \rceil^h \)
- The number of nodes in the lowest level dominates all other levels
Analysis of \((a, b)\)–Trees

**Theorem** \(\text{If } b \geq 2a, \text{ then } i \text{ insertions and } d \text{ deletions perform at most } O(\delta^h(i + d)) \text{ splits and fusions at height } h, \text{ where } \delta < 1 \text{ depends on } a \text{ and } b\)

**Proof (sketch)** Amortization argument, each node has a potential \(\phi\) (= measure of unbalancedness)

\[
\begin{align*}
\phi &= 1 + \delta_1 \\
\frac{1}{2} &= a - 1 \\
\delta_1 &= a - \frac{1}{2} \\
\delta_2 &= \beta - b + 1 \\
\text{degree} &= a - 1, a - \frac{1}{2}, \alpha, \beta, b + 1
\end{align*}
\]

**Theorem** \(\text{If } b \geq 2a, \text{ then the total } \# \text{ splits and } \# \text{ fusions is } O(i + d). \text{ If } b \geq (2 + \varepsilon)a, \text{ for some } \varepsilon > 0, \text{ the number of node splittings and node fusions is } O\left(\frac{1}{a}(i + d)\right)\)
Analysis of \((a, b)\)--Trees

**Theorem**

\((B/3, B)\)--trees perform \(\Theta(1/B)\) rebalancing per update

**Theorem**

\(\lceil B/2 \rceil, B\)--trees perform \(\Theta(1)\) rebalancing per update

**Theorem**

\(\lceil B/2 \rceil, B\)--trees perform \(\Theta(\log_B N)\) rebalancing per update if \(B\) odd
Lower Bound for Searching

**Theorem**  Searching for an element among $N$ elements in external memory requires $\Omega(\log_{B+1} N)$ I/Os

**Proof (sketch)**

- Adversary argument
- Algorithm knows total order of stored elements
- Initially all elements are candidates for being the query element
- If prior to an I/O there are $C$ candidate elements left, then there exists anwers leaving $\left\lceil \frac{C-B}{B+1} \right\rceil$ candidates after reading $B$ elements

**Note**  The lower bound holds even if an I/O can read $B$ arbitrary elements from memory