## Abstract

Dictionaries remain the most well studied class of data structures. A dictionary supports insertions, deletions, membership queries, and usually successor, predecessor, and extract-min. In a RAM, all such operations take $O(\log N)$ time on $N$ elements. Dictionaries are often cross-referenced as follows. Consider a set of tuples $\{\langle a_i, b_i, c_i \ldots \rangle\}$. A database might include more than one dictionary on such a set, for example, one indexed on the $a$'s, another on the $b$'s, and so on. Once again, in a RAM, inserting into a set of $L$ cross-referenced dictionaries takes $O(L \log N)$ time, as does deleting. The situation is more interesting in external memory. On a Disk Access Machine (DAM), B-trees achieve $O(\log_B N)$ I/Os for insertions and deletions on a single dictionary and $K$-element range queries take optimal $O(\log_B N + K/B)$ I/Os. These bounds are also achievable by a B-tree on cross-referenced dictionaries, with a slowdown of an $L$ factor on insertion and deletions. In recent years, both the theory and practice of external-memory dictionaries has been revolutionized by ***write-optimization*** techniques. A dictionary is write optimized if it is close to a B-tree for query time while beating B-trees on insertions. The best (and optimal) dictionaries achieve a substantially improved insertion and deletion cost of $O(\frac{\log_{1+B^\epsilon} N}{B^{1-\epsilon}})$, $0 \leq \epsilon \leq 1$, amortized I/Os on a single dictionary while maintaining optimal $O(\log_{1+B^\epsilon} N + K/B)$-I/O range queries. Although write optimization still helps for insertions into cross-referenced dictionaries, its value for deletions would seem to be greatly reduced. A deletion into a cross-referenced dictionary only specifies a key $a$. It seems to be necessary to look up the associated values $b, c \ldots$ in order to delete them from the other dictionaries. This takes $\Omega(\log_B N)$ I/Os, well above the per-dictionary write-optimization budget of $O(\frac{\log_{1+B^\epsilon} N}{B^{1-\epsilon}})$ I/Os. So the total deletion cost is $O(\log_B N + L\frac{\log_{1+B^\epsilon} N}{B^{1-\epsilon}})$ I/Os. In short, for deletions, write optimization offers an advantage over B-trees in that $L$ multiplies a lower order term, but when $L = 2$, write optimization seems to offer no asymptotic advantage over B-trees. That is, no known query-optimal solution for pairs of cross-referenced dictionaries seem to beat B-trees for deletions. In this paper, we show a lower bound establishing that a pair of cross-referenced dictionaries that are optimal for range queries and that supports deletions cannot match the write optimization bound available to insert-only dictionaries. This result thus establishes a limit to the applicability of write-optimization techniques on which many new databases and file systems are based.