Modern Hashing Made Simple

by Michael A. Bender, Martín Farach-Colton, John Kuszmaul, William Kuszmaul

Definitions

- A *hash table* is a data structure that maintains a set $\mathcal{S}$ under operations $\texttt{Insert}(\mathcal{S}, x)$, which adds x to $\mathcal{S}$, $\texttt{Delete}(\mathcal{S}, x)$, which removes x from $\mathcal{S}$, and $\texttt{Query}(\mathcal{S}, x)$ which answers whether $x \in \mathcal{S}$.
- In the case of a *fixed-capacity* hash table, we use n to denote the maximum capacity. When discussing *resizable* hash tables, we denote by $\bar{n}$ the current capacity and by n the current number of elements.
- We assume $\mathcal{S} \subseteq \mathcal{U} = [2^w]$ where $w = (1 + \Theta(1)) \log n$ is the *word size of the machine*.
- We assume access to fully random hash functions, invoked as an oracle.
- **Chernoff-Hoeffding bound.** Consider a sum of 0/1-random variables $X = \sum_{i=1}^{n} X_i$ with mean $\mu = \mathbb{E}[X] \geq \log n$. Then $X \leq \mu + \mathcal{O}(\sqrt{\mu \log n})$ with probability at least $1 - 1/\operatorname{poly}(n)$.
- **Exhaustive subtabulation.** Suppose f is a function mapping b-bit inputs to c-bit outputs. By trying all possible inputs to f, we can precompute a table of size $2^b c$ and answer $f(x)$ in constant time. E.g. if $b = \frac{1}{2} \log n$ and $c = 1$, then the table has size $\sqrt{n}$.
- **B-Trees.** A *B-tree* on n keys of w bits each is a balanced search tree where each node has up to $B + 1$ children, and the depth of the tree is $\mathcal{O}(\log_{B+1} n)$. Insertion, deletions, and queries can be implemented in time $\mathcal{O}(\log_{B+1} n)$. Note that as long as the tree consists of $2^{\mathcal{O}(w)}$ nodes (so that pointers between nodes require $\mathcal{O}(w)$ bits) then each node can be stored in one memory block including pointers to children and parent.
- **k-Tries.** For any k a power of two, a *k-trie* on w-bit keys is a k-ary search tree where the pivots are evenly spaced in the universe of possible keys. For any key $x \in [2^w]$ in the tree, the root-to-leaf path for x can be computed by:using the first (i.e. highest order) $\log k$ bits to navigate the root node, using the next $\log k$ bits to navigate depth 1 nodes, and so on, for a total of $w/\log k$ levels. The depth of a k-trie is therefore $w/\log k$.

Theorems

- **Theorem (Main result).** Let w be the machine word size, and consider hash tables storing w-bit keys, where the number n of stored keys satisfies $w = \Theta(\log n)$. There is such a hash table that uses $nw + \mathcal{O}(n \log \log n)$ bits of space, even as n changes over time, while supporting queries in $\mathcal{O}(1)$ worst-case time and insertions/deletions in $\mathcal{O}(1)$ time with probability $1 - 1/\operatorname{poly}(n)$.
- **Lemma (k-Trie Lemma).** A k-trie on n keys of w bits each takes time $\mathcal{O}(w/\log k)$ per operation and takes space $\mathcal{O}(nkw^2/\log k)$ bits. Here we assume that the arrays used to allocate nodes are already pre-allocated and initialized to zero.
- **Lemma (Space-Efficient Resizable Arrays).** Consider an array A that grows and shrinks over time. If $\bar{n}$ is the maximum size that the array is allowed to be, and n is the current size at any moment, then the array A can be implemented to use $n + \mathcal{O}(\sqrt{\bar{n}})$ machine words while supporting constant time operations.

Hash tables

- **Slow Partition Hash Table.** Split the array into *buckets* of size $\mathcal{O}(\log^3 n)$. Map each element to a bucket using a hash function and then search naively in each bucket to get time $\mathcal{O}(\log^3 n)$ per operation.
- **Indexed Partition Hash Table.** Improve buckets to support expected constant time operations by using B-trees to quickly search/insert/delete within a bucket. This needs some additional ideas to make sure that the space does not blow up past $\mathcal{O}(\log \log n)$ overhead per key.
- **Partition Hash Table.** Delegate insertions and deletions to an auxiliary data structure (two k-tries) to go from expected constant time to worst-case constant time with high probability.
- **Resizable Partition Hash Table.** Resize each bucket using space-efficient resizable arrays. When resizing the entire hash table, use two copies the same way one de-amortizes a regular array.