Motivation for looking at linear mappings as hash functions

by Martin

January 30, 1997

Abstract

A may-be paragraph in the STOC paper on hashing. Please comment, improve, condemn ...

There is no doubt that the method of implementing a dictionary by hashing with chaining, recommended in textbooks [CLR90, GBY90] especially for situations with many update operations, is a practically important scheme.\textsuperscript{1} Starting with the seminal paper of Carter and Wegman [CW79], it has been pointed out many times that this scheme should be used in combination with a function chosen at random from a universal class, since in this way the effects of particular data sets interfering badly with a fixed hash function can be excluded, and since no penalty as to efficiency is involved. (Techniques described in [CW79] make it possible to implement hash families consisting of linear mappings over $\mathbb{Z}_2$ very efficiently.) In situations in which a good bound on the cost of single operations is important, e.g., in real-time applications, the expected maximal bucket size as formed by all keys ever present in the dictionary during a time interval plays a crucial role. Our results show that, at least as long as the size of the hash table can be determined right at the start,\textsuperscript{2} using a hash family of linear functions over $\mathbb{Z}_2$ will perform very well in this respect. For other simple hash classes such bounds on the worst case bucket size are not available or are even wrong (see Theorem ?\textsuperscript{3}); other, more sophisticated hash families [S89, DM90, DGMP92] that do guarantee small maximal bucket sizes consist of functions with higher evaluation time.\textsuperscript{4} Of course, if worst case constant time for certain operations is absolutely necessary, the known two-level hashing schemes\textsuperscript{5} can be used, e.g., the FKS scheme [FKS84] for static dictionaries; dynamic perfect hashing [DKMHRT94] for

\textsuperscript{1}Dear co-authors: if you feel unable to sign this, I am afraid there is no point in saying anything is practically relevant at all.

\textsuperscript{2}Presumably we don’t have to argue that this will be the case more often than not.

\textsuperscript{3}Theorem 3

\textsuperscript{4}All these functions involve evaluating polynomials of degree at least 3 over a finite field that contains the universe of keys. That this is more costly than Carter and Wegman’s way of implementing matrix multiplication can be and has been demonstrated by simple experiments carried out, e.g., at our institute.

\textsuperscript{5}the term “double hashing” starts to be used for this; I find “dynamic perfect hashing” or “two-level scheme” more appropriate, especially since “double hashing” is the name of a well-known and important open addressing hashing scheme, see [GBY90].
the dynamic case with constant time lookups and expected time $O(n)$ for $n$ update operations; and the “real-time dictionaries” from [DM90] that perform each operation in constant time, with high probability. It should be noted, however, that a price is to be paid for the guaranteed constant lookup time in the dynamic schemes: the (average) cost of insertions is significantly higher than in simple schemes like chained hashing; the overall storage requirements are higher as well.  

References


*A series of experimental studies carried out by students in Saarbrücken, Paderborn, Dortmund, Copenhagen, and observations by St. Nähr in connection with dictionary implementations in the LEDA project consistently gave this result. We don’t have to write this, I think. A recent reference with real figures is [KL96]. In discussions, one could also mention the following: Implementing the FKS scheme is o.k. Implementing dynamic perfect hashing requires quite some effort and much more extensive code than chained hashing, say. I know of one master’s thesis that studied details of implementing the [DM90] scheme; it’s not simple at all, and I wouldn’t recommend it for practical use.