Comparison of Different Open Addressing Hashing Algorithms

Wenbin Luo and Gregory L. Heileman
Department of Electrical & Computer Engineering
University of New Mexico
Albuquerque, NM 87131

Abstract

Hash functions are among the oldest and most widely used data structures in computer science. Different hash functions exist. So, it is very important to compare their performance.

In this paper, we introduced our new hash function which was proposed recently in [1], and compared its performance with two different open addressing hashing algorithms: double hashing and exponential hashing. Double hashing is a widely-used hash function, which offers good performance. Exponential hashing, proposed by Smith et al [2], has been shown through experimental analysis to be superior to double hashing when the data elements are not uniformly distributed. The good performance of exponential hashing is due to its ability of spreading the table elements more randomly than double hashing. While exponential hashing has some desirable characteristics, Smith points out that it produces less than full probe length on 1/2 of the table elements. This is undesirable since it could potentially lead to insertion and search failures. Our new hash function has the ability of spreading the table elements randomly, and produces full probe length on all the table elements at the same time. From this point of view, our new hash function combines the strength of both double hashing and exponential hashing with their weakness eliminated. Experimental results are presented to support our claims, which is followed by some theoretic analysis.

1 Introduction

Hash functions have been studied over the years from a number of perspectives, but little progress in practical hashing has been made since the late 1960’s when double hashing was advanced.

The hash table is a data structure used to implement and maintain dynamic dictionaries. In a dynamic dictionary we are required to maintain a set of data elements $D$ (where each $x \in D$ is indexed by a key $k_x$ drawn from a totally ordered universe $U$) that can be accessed according to the following operations:

- $\text{Find}(k, D)$. Returns the $x \in D$ such that $k_x = k$, if such an element exists.
- $\text{Insert}(x, D)$. Adds element $x$ to $D$.
- $\text{Delete}(k, D)$. Removes the element $x \in D$ that satisfies $k_x = k$, if such an element exists.

When implementing the above operations using a hash table of size $m$, a table index is computed from the key value using a ordinary hash function $h$, that performs the mapping $h : U \rightarrow \mathbb{Z}_m$, where $\mathbb{Z}_m$ denotes the integers modulo $m$.

A collision is said to occur if given two keys $k_i$ and $k_j$, with $i \neq j$, $h(k_i) = h(k_j)$. One method of resolving collisions, termed open addressing [3], involves computing a sequence of hash slots that are successively probed until an empty hash table slot is found in the case of an Insert, or the desired item is found in the case of a Find or Delete. In open addressing, a hash function is modified so that it uses both a key, as well as a probe number when computing a hash value. Thus, hashing with open addressing uses the mapping $H : U \times \mathbb{Z} \rightarrow \mathbb{Z}_m$ and produces the probe sequence $< H(0, k), H(1, k), H(2, k), \ldots >$. When it does not lead to confusion, we will drop the dependence on $k$ (it will be accounted for in initial conditions), and refer generically to a probe sequence using $< H(0), H(1), H(2), \ldots >$. Because the hash table contains $m$ slots, there can be at most $m$ unique elements in a probe sequence. A full length probe sequence is defined to be a probe sequence that visits all $m$ entries using only $m$ probes.

The remainder of this paper is organized as follows. In section 2, an introduction to some of the most popular open address hashing algorithms is given. In section 3 we compared the performance of three different open addressing hashing algorithms: double hashing, exponential hashing, and our new hash function. Finally, the last section concludes the paper.
2 Open addressing hashing algorithms

A look at the three open addressing hashing algorithms, double hashing, exponential hashing, and our new hash function, lays the theoretical groundwork for our comparison of hash functions.

2.1 Double hashing

The family of linear probing can be expressed as
\[ H(k, i) = (h(k) + ci) \mod m, \] (1)
where \( k \) is the key, \( h(k) \) is an ordinary hash function that maps the key space to an initial location in the table, \( i = 0, 1, \ldots \) is the probe number, \( m \) is the table size, and \( c \) is a constant. This technique is known as linear hashing because the argument of the modulus operator is linearly dependent on the probe number.

Simple empirical tests reveal the limitations of linear probing. For a given constant \( c \), all initial hash values \( x_0 = h(k) \) will produce the same probe sequence. This problem is known as primary clustering. Linear probing also leads to secondary clustering where for any two keys \( k_1 \) and \( k_2 \) with \( k_1 \neq k_2 \) and \( h(k_1) = h(k_2) \), the sequences \( < H(k_1, 1), H(k_1, 2), \ldots > \) and \( < H(k_2, 1), H(k_2, 2), \ldots > \) will be identical. Both types of clustering are obviously undesirable. Double hashing remedies this problem by introducing a second hash function that is used in the computation of the increment.

Given two hash functions \( g \) and \( h \), the double hash function can be written as
\[ H(k, i) = (g(k) + ih(k)) \mod m. \] (2)
where \( g(k) \) and \( h(k) \) are ordinary hash functions which return values in the range \([0, N - 1]\). For a given key, these ordinary hash functions yield constants. Notice that double hashing is the same as the linear probing except that the constant \( c \) has been replaced by the value \( h(k) \). Therefore \( h(k) \) must always be relatively prime to \( m \) in order to guarantee full length probe sequences. The easiest way to assure this is to choose \( m \) as a prime number so that any choice of \( h(k) \) in the range \([0, m - 1]\) will be relatively prime to \( m \).

The key advantage of double hashing over linear probing is that \( h(k) \) is able to vary with \( k \). A widely used member of double hashing, proposed by [4], uses hash functions \( g(k) \) and \( h(k) \) with pairs of primes \( m \) and \( m - 2 \), such that \( g(k) = k \mod m \) and \( h(k) = k \mod (m - 2) \).

One probe sequence of double hashing with \( m = 23 \) and initial key \( k = 2 \) is \( 2, 4, 6, \ldots, 22, 1, 3, 5, \ldots, 21, 0 \).

It is clear that the probe sequence is of full length, but it is an arithmetic progression, not really random.

2.2 Exponential hashing

Smith et al. [1997] proposed exponential hashing, that uses two ordinary hash functions \( g \) and \( h \) to compute a probe sequence according to
\[ H(k, i) = (g(k) + h(k)^i) \mod m. \] (3)
The exponentiation does not have to be explicitly implemented, but can be computed via successive multiplications during the probing process. For this reason, the number of mathematical operations needed to implement equation (3) is identical to the number needed to implement equation (2).

While exponential hashing has some desirable characteristics, Smith [1997] points out that it produces less than full probe length on \( 1/2 \) of the table elements. This is easily confirmed by a simple experiment with initial hash value \( h(k) = 2 \), which is followed by successive powers \( h(k)^2, h(k)^3, \ldots \) of \( h(k) \): \( 2, 4, 8, 16, 9, 18, 13, 3, 6, 12, 1, 2, 4, 8, 16, 9, 18, 13, 3, 6, 12, 1 \). As we can see, this sequence has a cycle of order 11.

2.3 Our new hash function

We proposed a new hash function recently [1], that computes a probe sequence according to
\[ H(k, i) = g(k) + a^i h(k) \mod m. \] (4)
where \( k \) is the key, \( g(k) \) and \( h(k) \) are ordinary hash functions which return values in the range \([0, m - 1]\), \( i \) is the probe index, \( m \) is the table size, and \( a \) is a constant. \( H(k, 0) = g(k) \). The probe sequence is not of length \( m \) for all values of \( a \). But as we proved in [1], if \( a \) is a primitive root of the prime \( m \), the probe sequence will be full length. This is easily confirmed by the probe sequence with \( m = 23 \), \( a = 5 \), and \( k = 2 \): \( 2, 12, 6, 22, 10, 19, 18, 13, 11, 1, 20, 0, 15, 21, 5, 17, 8, 9, 14, 16, 3, 7, 4 \). It is clear that the probe sequence is of full length, and it is more random than that of double hashing in section 2.1. Let \( \phi(m) \) denotes the Euler’s totient function whose value is the number of integers that are less than \( m \) and prime to \( m \). The symbol \( < a^i >_m \) denotes the operation of extracting the residue of \( a^i \) modulo \( m \). The primitive root of \( m \) is an element such that the period of \( < a^i >_m \), \( i = 0, 1, 2, \ldots \), is \( \phi(m) \).

Ideally, \( g(k) \neq h(k) \) for all \( k \). This will assure that no two keys have the same probe sequence, avoiding
the primary and secondary clustering problem. Each probe location can be calculated with a single multiplication, addition, and modulo division, which compares favorably with double hashing which requires two additions and a modulo division.

3 Experimental results

In order to compare the performance of double hashing, exponential hashing, and our new hash function, we implemented all three hashing algorithms. For simplicity, we used table size $m = 3023$ for all experiments.

3.1 Uniform data distribution

The first experiment is to test the performance of different hash functions under uniform initial data distribution. Data elements (keys) chosen at random from uniform distribution were successively added to the table. Assume that there are $k$ elements inserted into the hash table. The capacity or table load is defined as $k/m$. Hash tables of equal size were created and filled to 95% capacity by different hash functions. The test was repeated 100 times using a different random number seed for each run to determine if any statistical difference in total number of probes to fill the table could be detected. The measure of merit was the average number of probes required per key added. For example, if $k$ elements take a total of $n$ probes, the average probes per key is simply $n/k$. A summary of the results are presented in table 1 in which D. H. means double hashing, E. H. means exponential hashing, and New H. means our new hash function. From table 1, we see that no statistically significant difference in performance could be detected. The difference in average number of probes between different hash functions is small. Our new hash function performs slightly better than both double hashing and exponential hashing.

3.2 Clustered data distribution

The second experiment is to test the performance of different hash functions under clustered initial data distribution. The clustered data represents many real world data sets where data is far from evenly distributed. The data cluster size is set to 10% of the table size, and all of the data was chosen at random from this cluster. Samples of the average number of probes per data element were taken for every 5% of table size, up to a total table load of 95%. The results are summarized in Fig. 1.

From Fig. 1, we see that our new hash function performs consistently better than both double hashing and exponential hashing. Assume that the probes needed to fill the hash tables to a specific table load are $P_{exp}$ and $P_{new}$ by using exponential hashing and our new hash function, respectively. We define the probes saved by our new hash function over exponential hashing as $P_{saved} = P_{exp} - P_{new}$. To take a clearer look at how our new hash function outperforms exponential hashing, we plot the probes saved $P_{saved}$ versus table load factor in Fig. 2.

From Fig. 2, we see that higher the table load factor is, more probes our new hash function saves over exponential hashing. The same is true when it is compared with double hashing.

3.3 Variation of cluster size

This experiment was similar to the previous one, but in this case the cluster size was varied from 2% to 20% of the overall table size. Hash tables were created using different hash functions. Data were taken at random from a cluster of size varying from 2% to 20% of the table size, and the total number of probes was sampled for each table to reach 95% capacity.

The results, summarized in Fig. 3 show that our new hash function and exponential hashing uses far
fewer probes than double hashing.

The relative advantage seems to be larger for more tightly clustered data. To take a clearer look at how our new hash function outperforms exponential hashing, we also plot the probes saved $P_{\text{saved}}$ versus cluster size in Fig. 4.

From Fig. 4, we see that our new hash function performs consistently better than exponential hashing. The probes saved is very large for tightly clustered data.

### 4 Conclusions

In this paper, we introduced our new hash function which combines the strength of both double hashing and exponential hashing, i.e., full length probe sequence property of double hashing and random probe sequence property of exponential hashing, and compared the performance of three open addressing hashing algorithms. Results indicate that our new hash function outperforms both double hashing and exponential hashing for nonuniform data distributions. Our new hash function has the advantage of consistent behavior over all data distributions, while double hashing doesn’t perform well for some clustered distributions. Generally speaking, the relative advantage of our new hash function over double hashing and exponential hashing is larger for more tightly clustered data, higher table load, or larger table size.

### References


