FEREYDOUN Hash Table: Built-in Collision Resolution and Built-in Seamless Dynamic Resizing

Fereydoun Farrahi Moghaddam

Abstract: Fast and flexible, Elastic, built-in collision Resolution, Expandable and downsize-friendly, Yielding, built-in seamless Dynamic resizing, Open ended, Uniform, Normal distributed (FEREYDOUN) hash table provides one methodology with two functionality for collision resolution and seamless dynamic resizing without any needs for existence of a concurrent table, table partitioning, or catastrophic reshuffling.

1. Introduction

Hash tables usually need to deal with collision resolution and dynamic resizing challenges [1]. In this paper, we propose a methodology to deal with these two issues with one method at the same time. The proposed dynamic resizing of this methodology is seamless and completely elastic as the size of table can grow and shrink in-place according to the number of keys without any needs for existence of a concurrent table, table partitioning, or catastrophic reshuffling.

2. FEREYDOUN Hash Table

FEREYDOUN hash table starts with an initial size s_i and will grow as needed. It also can shrank back as the key-value pairs are deleted from the table. It can actually shrank back to its original s_i size if all the key-value pairs are removed from the hash table. In the following, we will define a few helper functions before introducing the FEREYDOUN hash table search function.

2.1. Hash function

We need a general purpose hash function $h(k)$ that returns an integer value for each key k such as md5.

2.2. Hash table virtual size

Virtual hash table size $s_v(m)$ is a function that define a virtual size for the hash table that grows with the number of misses m.

$$s_v(m) = \lceil s_i r^m \rceil \tag{1}$$

Where r represents the expansion ratio ($r > 1$). This virtual size is an abstract number and only play a rule in calculation of the position of the key-value pairs and is different from the actual size of the hash table. Basically, the size of hash table grows virtually after each miss with the factor of r.

2.3. Index function

Index function $i(k,m)$ do not uses the actual size of the hash table to calculate the location of the key-value pairs. Instead it uses the virtual size of the hash table which grows with each miss as follows:

$$i(k,m) = h(k) \bmod s_v(m) \tag{2}$$

This index can be examine by the search function to determine if a specific or any key-value pairs are stored at that location.

2.4. *Hash table expansion*

If in an INSERT operation, the index exceeds the actual size of the hash table, the hash table needs to be expanded by an expand function $e(i)$ to the location of index i. This is simply done by adding $i-s+1$ empty cells to the end of hash table where s represents the actual size of the hash table. Then, the last cell will be updated by the new key-value pair.

2.5. *Hash table shrinkage*

During a DELETE operation or during a SEARCH operation where the key-value pair are move (details in following sections) and the original index of the deleted or moved key-value pairs are equal to $s-1$, the empty cells at the end of hash table can be removed by downsize function $d()$.

2.6. *Search algorithm*

This is the main function of the FEREYDOUN hash table that provides four functionality: INSERT, UPDATE, SEARCH, and DELETE. INSERT inserts a new key-value pair that does not exists in the hash table. UPDATE updates the value of an already existing key-value pair. SEARCH returns the value of an already existing key-value pair, and DELETE deletes the an already existing key-value pair. Using INSERT with an already existing key may produce a duplicate key. Using the UPDATE, SEARCH and DELETE on non existing keys will result in maximum m_m misses before function returns Null.

Algorithm 1: INSERT algorithm
Input data: k=KEY, v=VALUE
For m= 1 to m_m **do**
.... $i(k,m) = h(k)\ mod\ s_v(m)$
.... $i_n \leftarrow$ index of first Null cell, **break**
.... $i_o \leftarrow$ First i(k, m) $\geq$ s+1, **break**
end
If i_n **then**
.... Put $k-v$ pair at i_n
else
.... e(i_o)
.... Put k-v at pair i_o
end

Algorithm 2: UPDATE algorithm
Input data: k=KEY, v=VALUE
For m= 1 to m_m **do**
.... $i(k,m) = h(k)\ mod\ s_v(m)$
.... $i_k \leftarrow$ index of k, **break**
end
If i_k **then**
.... Put v at i_k
else
.... Return key not found error
end

Algorithm 3: SEARCH algorithm
Input data: k=KEY
For m= 1 to m_m **do**
.... $i(k,m) = h(k)\ mod\ s_v(m)$
.... $i_n \leftarrow$ index of first Null cell, **break**
.... $i_k \leftarrow$ index of k, **break**

end
If i_k **then**
.... **If** i_n **then**
.... Move k-v from i_k to i_n
.... **If** i_k==s-1 **then**
.... $d()$
.... **end**
.... **end**
.... Return $v(k)$
else
.... Return key not found error
end

Algorithm 4: DELETE algorithm
Input data: k=KEY
For m= 1 to m_m **do**
.... $i(k,m) = h(k)\ mod\ s_v(m)$
.... $i_k \leftarrow$ index of k, **break**
end
If i_k **then**
.... Delete k-v at i_k
.... **If** i_k==s-1 **then**
.... $d()$
.... **end**
else
.... Return key not found error
end

3. Proof of concept

FEREYDOUN hash table is implemented as a proof of concept in [2].

References

1. "Hash table", https://en.wikipedia.org/wiki/Hash_table.
2. "Fast and flexible, Elastic, built-in collision Resolution, Expandable and downsize-friendly, Yielding, built-in seamless Dynamic resizing, Open ended, Uniform, Normal distributed (FEREYDOUN) hash table", https://github.com/frdfm/FEREYDOUN.