Hash

Fredrik Berzins

Fall 2023

Introduction

A Hash a kind of function that can remap keys/values to new keys/values in a consistent manner. Remapping is useful for making a LUT(look up table) of sorts by remapping a group of spread out values to a tighter grouping.

Concept Explanation and Implementation

Hash explanation

A hash function is like any other mathematical function it gives out a value dependant on input value. A easy hash function is mod with the modulus factor as new grouping size.

String/Integer explanation

The implementations with String and Integer storage doesn't have any thing to do with hash tables but they are the predecessors to hash tables(Arrays). This means the linear search takes O(n) and binary takes O($logn$).

Index explanation

Index implementation is a implementation where all values are added to the index of the value, this gives it a extremely quick lookup/find since the value used for searching is the same as the index.

Simple Hash explanation

Index implementation is a implementation where all values are added to the index of the value, this gives it a extremely quick lookup/find since the value used for searching is the same as the index.

Collisions explanation

Collisions occur when a hash function takes two different keys and return the same hashed value this means two keys would point to the same value. This can be solved with a few different methods, two methods being bucket and index shift.

Bucket explanation

The bucket method of handling collisions is to give every node a pointer that starts as null but if a secondary value is hashed to the same index the original nope on that hashed index now points to the secondary node. So in principal the bucket method is to make a linked list for every index. This can also be done with arrays as buckets but then they have to be dynamic. A bucket implementation is simple to lookup values since it hashes the key and looks thru the linked list at that index to see if the key node is in the linked list. When removing values it's as easy as doing it to a linked list.

Index Shift explanation

The shift method works by going to the hashed index in the data array and if it's occupied increment the index until theirs a empty spot the issue with this approach is that the data array can't be smaller the the data set, it also has a different situation when removing element since it needs to replace removed elements with something other then null. This is called a gravestone/dead marker some way of recognising a former node. This is needed since the lookup method is going to stop at null nodes and return null at that point.

Benchmarks

The testing was done with 5 predetermined values (111 15, 244 94, 457 32, 702 35 and 984 99).the values chosen was the lowest, highest, center and 1/4 and 3/4 value. Each time test was redone 100 times this is done for consistent values. The test each time is to find the specified value. The measured result is the minimum time (out of 100 searches/look ups) to find values with a certain method.

Result

The result in table 1 below shoes how quick it is to search for values for different Array based structures. The is the method with the most work ahead of time where the values wore sorted and converted from strings to integers.

Zip Code	Linear S	Binary S	Linear I	Binary I
111 15	490	5200	370	1500
244 94	600000	1100	200000	590
457 32	1500000	780	510000	390
702 35	2200000	1200	840000	620
984 99	2800000	6700	1100000	2400

Table 1: Time to search for a value 100 times in different arrays in ns (S/I stands for String/Integer depending on what the array stores.)

The table below shows the time difference to lookup elements when using the key as the index in a large array compared to hashing the key and using a bucket implementation to minimize the main data array. The result show that h the time difference is minimal but he memory usage is significantly lower.

Zip Code	Lookup Index	Lookup Keys
111 15	307	325
244 94	370	375
457 32	376	388
702 35	378	412
984 99	319	347

Table 2: Time to lookup a 100 values in different arrays in ns. (modulo was 8123 for this test.)

The results in the table below shows a how much better it is to use a prime value as a modulus value for the hash function since it wont consistently pile up similar values in the same spot like 111 15 and 411 15 would be written to the same bucket hash if the mod values are something like 1000, 10000. In the table below its clear that 8123 is a great choice for mod values since its small and not many collisions. if less collisions are wanted and memory can be spared a prime value around 18000 wold probably be good. The reason to not use the primes around the nice values like 10000 or 20000 is to sufficiently shift the hash compared to previous keys.

Mod	No	1	2	3	4	5	6	7	8
8123	6006	2782	759	115	12	0	0	0	0
10000	4465	2414	1285	740	406	203	104	48	9
10007	6157	2500	788	189	36	4	0	0	0
19997	7100	2032	466	68	8	0	0	0	0
20000	6404	2223	753	244	50	0	0	0	0
40000	8040	1521	113	0	0	0	0	0	0
40009	8890	760	24	0	0	0	0	0	0
79999	9468	206	0	0	0	0	0	0	0
80000	9459	215	0	0	0	0	0	0	0

Table 3: Amount of keys hashed to the same value dependent on size of mod value.

Conclusion

A Hash table makes the task of storing and then retrieving items take slightly more space but be a lot quicker. Hash tables are probably best used for larger data sets since the time to set it up for smaller data set is probably not worth the cost if the values are not stored for a long time and often retrieved. The best method for small data set or almost continuous(integer) keys is index Array where every index is the key. it's the fastest data structure to lookup values in and dose not have the downside of more complicated function for Hash tables.