Hash Table Length and Prime Numbers

Author: Srinidhi Viswanatha
Posted on: http://srinvis.blogspot.ca/2006/07/hash-table-lengths-and-prime-numbers.html

This has been bugging me for some time now...

The first thing you do when inserting/retreiving from hash table is to calculate the hashCode for the given key and then find the correct bucket by trimming the hashCode to the size of the hashTable by doing hashCode % table_length. Here are 2 statements that you most probably have read somewhere

- If you use a power of 2 for table_length, finding (hashCode(key) % 2^n) is as simple and quick as (hashCode(key) & (2^n -1)). But if your function to calculate hashCode for a given key isn't good, you will definitely suffer from clustering of many keys in a few hash buckets.
- But if you use prime numbers for table_length, hashCodes calculated could map into the different hash buckets even if you have a slightly stupid hashCode function.

Ok, but now can someone tell me why it is so and give me the proof for the above statements...I couldn't find much on google. Most reasons given are again just statements and I cannot accept statements (And there are some stupid proofs also on the net, so beware). And don't even try telling me that the proof is experimental.

Today i have finally been able to get rid of this thought from my head...below is the proof I came up with (Hope it doesn't fall into the stupid category, it doesn't seem atleast for now, if u think otherwise put in a comment, atleast for the sake of others). I suggest you think about the solution on your own before reading further...

If suppose your hashCode function results in the following hashCodes among others {x , 2x, 3x, 4x, 5x, 6x...}, then all these are going to be clustered in just m number of buckets, where m = table_length/GreatestCommonFactor(table_length, x). (It is trivial to verify/derive this). Now you can do one of the following to avoid clustering

1. *Make sure that you don't generate too many hashCodes that are multiples of another hashCode like in {x, 2x, 3x, 4x, 5x, 6x...}.But this may be kind of difficult if your hashTable is supposed to have millions of entries.*
2. *Or simply make m equal to the table_length by making GreatestCommonFactor(table_length, x) equal to 1, i.e by making table_length coprime with x. And if x can be just about any number then make sure that table_length is a prime number.*

If you need some more kick from hash maps take a look at Doug lea's highly performant ConcurrentHashMap http://www-128.ibm.com/developerworks/java/library/j-jtp08223/