Hashing and bloom filter

# Experiment with different load factors

The variations of hashing algorithms implemented were done using 1,000,000 random integers ranging from 0 to 10,000,000. Multiple hashing algorithms and variations of them were tested using a 1.0 load factor for benchmarking.

Multiple hashing algorithms were tested with multiple variations of numbers within the hashing algorithms. All performed about the same with around 36% of the hash table empty and the longest linked list ranging from 5 to 8 elements. A little bit of memory is wasted but lookups and insertions are basically constant time with O(1) time complexity.

**Important note**: When selecting a load factor or recommended hash table size, it is not guaranteed that you will get *exactly* this size. For performance benefits the requested table size or load factor will be rounded up to the nearest prime number larger than the requested number (unless the requested number is a prime number itself).

The experiments were also run with varying sizes of *n*, or input numbers. Ranging from 1,000 – 1,000,000. The hash table percent empty results were proportionally the same each time remaining at almost exactly 36%. The longest list however remained almost constant at 7 – 8 elements.

After settling down on one particular hash function I began testing different load factors, while varying the size of n.

**Information on running my program**

* My program accepts command line arguments for specifying the size of the hash table as well as how many numbers to process from a given file.
* There are two ways to specify the size of the hash table.
  + Specify the size of the hash table as a number
    - Use the ‘-s’ flag for using a specific size
  + Specify the size of the hash table as a load factor
    - Use the ‘-l’ flag for specifying a load factor
* Syntax for executing program:
  + Hashing <file> <# of elements to process> [<-s> <size of hash table> | <-l> <load factor>]
* Example for reading 1,000 numbers from the file num.txt with a hash table size of 500
  + Hashing nums.txt 1000 -s 500
* Example for reading 1,000 numbers from the file num.txt with a load factor of .75
  + Hashing nums.txt 1000 -l .75

### Analysis of results

As one can see from the results the build times are almost exactly identical for a particular size of n, regardless of the load factor. This is the exact definition of O(n) or linear time to build a particular hash table. Retrieving an element from a table, regardless of load factor will be done in constant time, or O(1).

With a million elements the largest list is 26, which takes no time at all to scan through for a computer. Since the results of the operations on a hash table are the same regardless of load factor, the best load factor really comes down to memory management.

I think a load factor of 3 to 10 is ideal for best performance. Therefore the hash table will be anywhere from 4% to 0% empty – most spots in the hash table full – will the longest list ranging from 14 to 26. So this will save memory for the table allocation and still allow operations in constant time of O(1) – really O(longest list), however in a particular instance this is constant and therefore becomes O(1).

# Bloom filter