William Huang | 8833196
CS130A Programming Assignment 1 Documentation

strToInt Explanation

The reason I implemented strToInt the way I did was because I wanted to closely follow the structure shown in the assignment documents but also reduce the chances of overflow. Essentially, I modified the equation unsigned int x = (d[0] * C^0 + d[1] * C^1 + … + d[n] * C^n) in such a way that instead of incrementing the exponents of the constant C by 1, I incremented it by 0.75. As a result, each character in the string is multiplied by 3 (a prime number) to the power of [0.75,1.5, … , 0.75n]. The reason why I utilized 0.75 was because I conducted various test trials and increments of 0.75 happened to result in the lowest probability of false positives. The reason I utilized 3 is because I tried other higher prime numbers and they all resulted in overflow at some point (became a headache halfway through).

The formula helps to reduce the chances of collision between words with the same characters because the constants raised to a specific power helps to alter the "inherent" integer value of the character at each index.

For example, even though "reed" and "deer" both have 2 'e's, 1 'd', and 1 'r' , because the 'd' and 'r' are at different indexes(1 & 4 and 4 & 1 respectively), their integer value are different due to the different constants that's being multiplied (d* C^0 and r*$C^{2.25}$ for deer and r* C^0 and d*$C^{2.25}$ for reed)

Similarly, "abc" and "aca" will also be different because of the way the constants are multiplied into the character value at each index. (a * C^0 + b * $C^{0.75}$ + c * $C^{1.5}$ vs a * C^0 + c * $C^{0.75}$ + a * $C^{1.5}$
)

Auxiliary Hash Table size

The reason I chose 97 for the size of the hash table is because I know that we will be removing 100 items each phase, and therefore I picked the closest positive prime number to 100(which is 97) as the key for the hash function.

Overall Implementation:

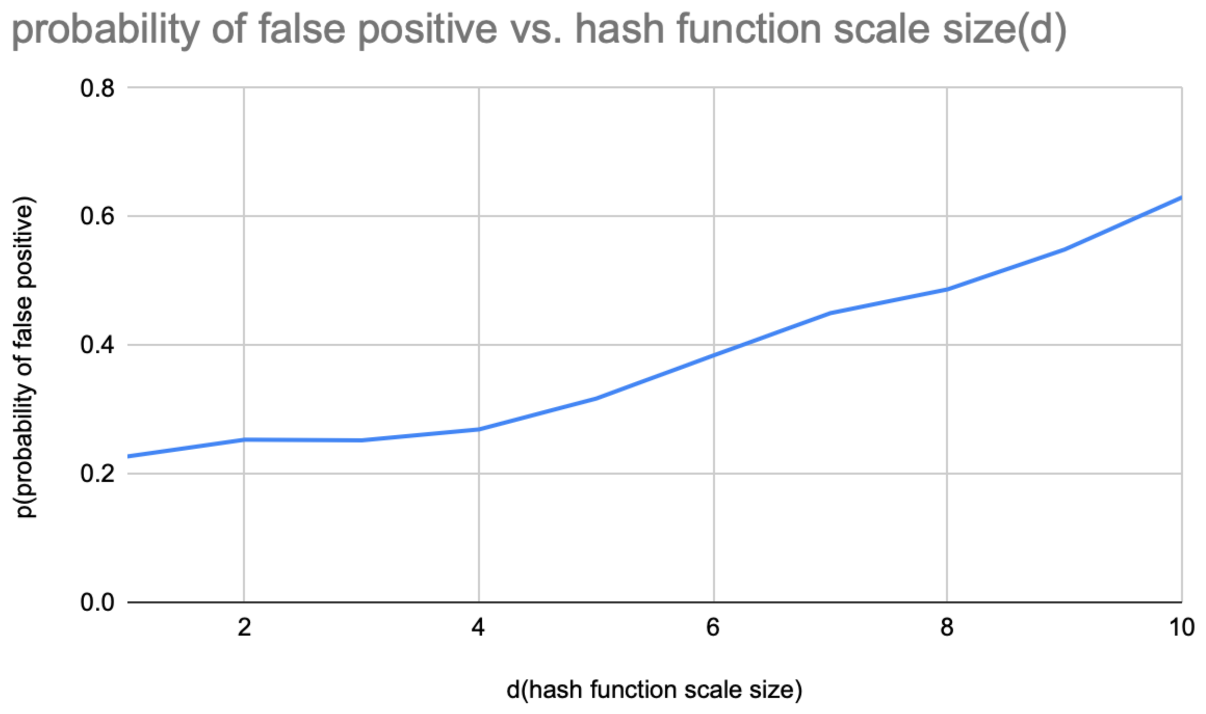
Array with linkedlist for hash table - required
Printing in main - for convenience, I conducted all the printing in main as main holds most of the required values and variables.
Boolean array on the heap for bloom filter - because I wanted to be able to initiate the array post declaration, it was necessary to declare the array on the heap.
Node structure - simple node struct that holds the value and the next node

Hash_list.h - before I realized hash table had to use an array of linkedlist, not used, but I didn't want to delete it.
testHash and testBloomFilter - both just test functions for initial data structure testing
utility
Utility.h - hold all the extra functions used both within and outside the data structures
Hash family - a and b are constants calculated using the index and various prime numbers and the actual family of hash functions consist of (a * (the element) + b) % (the size of the bloom filter)

Analysis of Hash Func Scale Size vs P(false positive)

c = constant 1

d(hash function scale size)	p(probability of false positive)
1	0.227
2	0.253
3	0.252
4	0.269
5	0.317
6	0.384
7	0.45
8	0.487
9	0.549
10	0.63

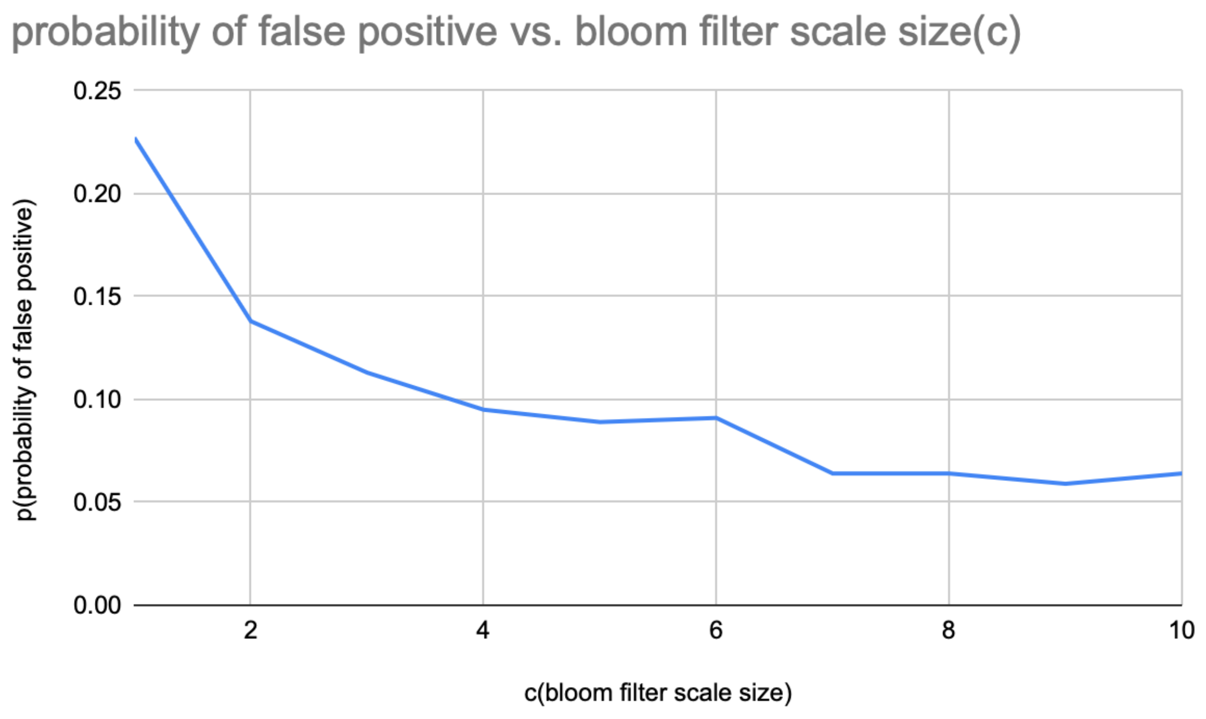


The above graph clearly demonstrates that given a hash function scale size in the range [1,10], it has a direct relationship with the probability of false positives. This makes sense because as we increase the number of hash functions(through scaling it upwards), given a constant bloom filter size, the number of buckets/spots each word/string will take up will increase. As a result, it is more likely for false positives to occur.

Analysis of Bloom Filter Scale Size vs P(false positive)

d = constant 1

c(bloom filter scale size)	p(probability of false positive)
1	0.227
2	0.138
3	0.113
4	0.095
5	0.089
6	0.091
7	0.064
8	0.064
9	0.059
10	0.064



The above graph shows that given bloom filter scale size [1,10], it has a negative correlation with respect to the probability of false positives. This makes sense logically because as we increase the size of the bloom filter(via upward scaling), we are increasing the total number of spots/buckets while keeping the number of hash function constant.