CS 370 - Programming Project 2 Writeup

Doug Lloyd

Oregon State University

CS 370

16 November 2021

1) I chose to use the mmh3 library (murmur) hash, seeded differently on each run. Since murmur is a seeded hash, altering the seed every loop results in a different hash each time. Murmur is a non-cryptographic function. I used a non-cryptographic function because cryptographic functions are much more resource intensive. Since the dictionary contained over 600,000 words, using a cryptographic function would be extremely resource intensive, especially since we hashed multiple times when populating the bit array. The range of murmur is 2^32 (4,294,967,296) because it's a 32-bit hashing function. I chose to use a bloom filter size of 5,000,000 for both the 3-hash and 5-hash implementations, because I was hoping to see some noticeable difference in the % false positive.
2) Checking 1 word in the case of the 3-hash function takes about 28,000 nanoseconds. Checking 1 word in the case of the 5-hash function takes about 45,000 nanoseconds. This is likely because the majority of the time spent computing is spent doing the hashing. This is still an extraordinarily fast operation.
3) The probability of a false positive in the 3-hash function is about 2.9% with a bit array size of 5,000,000 and ~615,000 items in the bloom filter (the dictionary). The probability of a false positive in the 5-hash function is about 2.0%, with the same bit array size and amount of items in the bloom filter. Getting a false negative in a bloom filter is impossible – if any of the bits from an entry to be checked are not flagged, then that entry cannot possibly be in the bloom filter. Bloom filters are a probabilistic data structure that are only prone to false positives, not false negatives.
4) You can reduce the rate of false positives by increasing the number of bits in the bit array, reducing the number of items in the filter to begin with (more items in the filter means more set bits, up until the point where the chances of a positive are 100% for any entry), or by finding the sweet spot of # of cryptographic hashes. Increasing the bits is the most reliable way, but increasing the bits in the array is computationally expensive.