CS 271 Project 6

Austin Burgess, Oscar Martinez

November 2019

Figure 1 shows our code for the hash function.

We used a multiplication method hash function, in which m is a large prime number (we chose 3079) and A is an irrational number (we chose Knuth's example discussed in class). We then returned $m(k(A))$ floor divided by 1 to make it an integer, then modulo 1000 to hash it into a number between 1 and 1000 (our range for the histogram) (note: k in this case is the integer value we computed of all the letters added up in whichever word we were looking at). This hash function provides the following hashing distribution showed in Figure 2 (histogram produced using pyplot).

The minimum index hashed was 2, and the maximum index hashed was 996. This is pretty promising, as both ends of the spectrum were at least accounted for in some way by our hash function. However, the standard deviation was about 288.42, which is smaller than we may have liked, because 68 percent of your data lies within one standard deviation and 95 percent lies within two. So, by that logic, 95 percent of the values we hashed using this function fell within just shy of 580 slots, leaving the remaining 420 slots to only hold 5 percent of the values we hashed. This, of course, is pretty far from uniform, and not ideal. We tried many combinations of primes and irrational numbers, but settled with this one for our analysis. So, to conclude, while our hash table does a good job of covering the full range of our slots/indices, it can definitely do better in doing so uniformly.

function code.PNG

```
def hash(k):
  knum = 0          # ensure we have a number here
  for letter in k:
    knum = knum + int.from_bytes(letter.encode(), "little")
  m = 3079                      # big prime number
  A = (math.sqrt(5) - 1) / 2    # Knuth

  return abs(( m * (knum * A) // 1)) % 1000
```

Figure 1: The code

func histogram.PNG

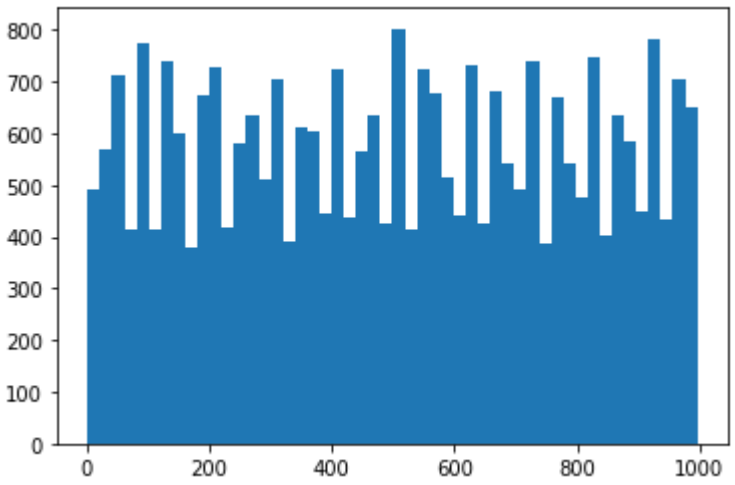

Figure 2: The histogram

As far as the IMDb stuff goes, we were unable to get the program to recompile without stretching the deadline too far. This includes an hour or three devoted only to trying to find the segmentation fault that occurs while the move data is being read in and inserted. We aren't sure what's happening, we've tried about everything.

What we anticipated would've happened is that, if our hash function was good enough, we would've shown that hash tables is much faster than BST because it searches and finds the movie/genre pairs in near constant time (except for when chaining occurs) vs. O(log n) time for BST. However, if our hash function wasn't good enough, we would find that the runtime would be similar to or perhaps even worse than that of BST, due to too much chaining and not enough variance in the indices things are being hashed to.

Also, thanks again for allowing us to submit this later tonight on the day this is due. We tried to get the seg fault issue ironed out, but it wasn't meant to be. Sorry.