Henry Chang Lab 9

In this lab, we compare four different kinds of hash functions: modular hashing, 8-bit Pearson hashing, 32-bit Pearson hashing, and Java 1.1 String hashing on three different datasets: a dictionary, a novel, and DNA strings. By doing so, we hope to find the best hash function for each dataset.

Before comparing different hash functions, I find the best parameter **r** for modular hashing by keeping track of **fh,** the number of hashes that happened before the first collision, and continue to use the **r** value for later comparison with other hash functions. We average out the results of 50 trials to reduce error, and the results shown that r=37 has the best performance, since it has the largest fh, meaning it would less likely result in a collision and would distribute keys more uniformly into hash buckets.

r	0	1	2	3	4	10	13	31	37	39	64
hf	5	35	268	722	1027	4184	7671	75301	87683	80191	66

This result is easy to understand since 37 is the largest prime number tested, so it creates many buckets for keys while avoiding frequent collisions for datasets with many keys that shares common factors. On the other hand, r = 0 is the worst parameter as it doesn't really do the job of modular function during the hashing process.

I perform the same tests in part I on the four hash functions, and the results are:

	modular (r = 37)	8-bit Pearson	32-bit Pearson	Java 1.1 String
dictionary hf	92259	18	68681	16496
a novel hf	7366	18	15888	6727
DNA strings hf	No Collision	22	No Collision	1126

We see that 8-bit Pearson is the worst for all files. For the dictionary file, modular hash works the best, but 32-bit Person is not far behind. For the novel file, 32-bit Pearson wins it significantly. For the DNA files, both modular and 32-bit Person are PERFECT HASH functions!

***

For the coding this week in HashEvaluation.java, I only modified one line that checks whether the hash function is perfect or not to: if (hashes.size() >= keys.size()), and everything else remains as given.