# ANALYSIS OF COMPRESSED LITERATURE 2

## By Jake McKenzie

For my probing strategy I set out to find a hashing function that was meant to data of small byte length, as I found the average byte length 2.510645470812479 bytes. I found two hash function commonly used for 8-byte length words and one for 4-byte length words. They all performed better than the built-in hash function in java for both linear probing and quadratic probing. All of the quadratic hash functions outperformed the linear probing methods in terms of space allocation, but many of the hash functions had far improved runtimes over the built-in hash function in java. The max linear probe length of Javas built in hash function was 335 where the average probe length was 1.727. The max quadratic probe length of this has function was 21 where the average probe length was 0.834. The runtime of the built in hash function was quite slow at 5.588 seconds for the linear probing method and 1.386 seconds for the quadratic probing method.

The first hash function I implemented was the Bernstein hash which is meant for small character keys and can outperform algorithms with more random distributions. The max linear probe length of this hash function was 263 where the average probe length was 1.406. The max quadratic probe length of this has function was 20 where the average probe length was 0.811. This outperformed hash function outperformed the runtime of java's built in hash at 1.576 seconds for the linear probe and 1.296 seconds for the quadratic probe.

The second hash function I implemented was Jenkin's 'one at a time' hash which is meant for small character keys around 3-bytes up to 8-bytes. This hash is widely used on character strings and is the standard implementation for the Perl programming language. The max linear probe length of this hash function was 59 where the average probe length was 1.1594. The max quadratic probe length of this has function was 21 where the average probe length was 0.812. This outperformed hash function outperformed the runtime of java's built in hash at 1.486 seconds for the linear probe and 1.319 seconds for the quadratic probe.

The third hash function I implemented was FNV hash which is meant for speed while maintaining a slow collision rate. This hash was given freely to the public as an IEEE standard meant for use on URLs, hostnames, text, IP addresses, etc. The max linear probe length of this hash function was 46 where the average probe length was 1.1382. The max quadratic probe length of this has function was 19 where the average probe length was 0.7991. This outperformed hash function outperformed the runtime of java's built in hash at 1.302 seconds for the linear probe and 1.283 seconds for the quadratic probe. This was the best hash function I implemented. This was because I purposely chose "FNV primes" for the average byte length in the file by following the algorithm specifications on Wikipedia.

In the analysis of these hash functions and it became clear that when constructing a good hash table it is best to have your data in mind. Hash functions meant for different types of data sets can produced high clustered and weakly dispersed hashes, so can java's built in hash function. Hash functions should be tailored for your data set and your needs. Quadratic probing is quite better than linear probing for these hashing functions. Bernstein hash perform the best when paired with quadratic probing and FNV hash outperformed all the other hash functions in linear probing. These hashing functions all produce clusters

of integers due to the homogeneity of the input data and inability to create perfectly unique hashes. Quadratic probing seems to be attempting to make smaller clusters by growing in powers of two instead of two instead of one. From the histograms of all the hashes, all produce clusters, but the hash functions and probing techniques that perform best are the ones that produce smaller clusters. They don't need to converge to a bucket immediately but to perform well they need to reach a bucket quickly.