Arnav Shirodkar

CMSC 201 – Lab 10

[as9086@bard.edu](mailto:as9086@bard.edu)

5/5/20

Assignment: Evaluate the effectiveness of four different hash functions and find which works best for the corresponding data sets provided.

Collaboration Statement: I worked on this alone.

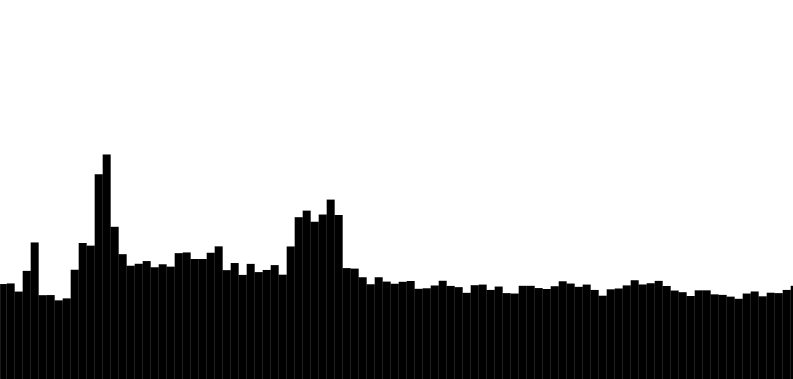
Note: I made no changes to HashEvaluation.Java and therefore have not included it in this file

**Words. Txt – case where all inputs are distinct**

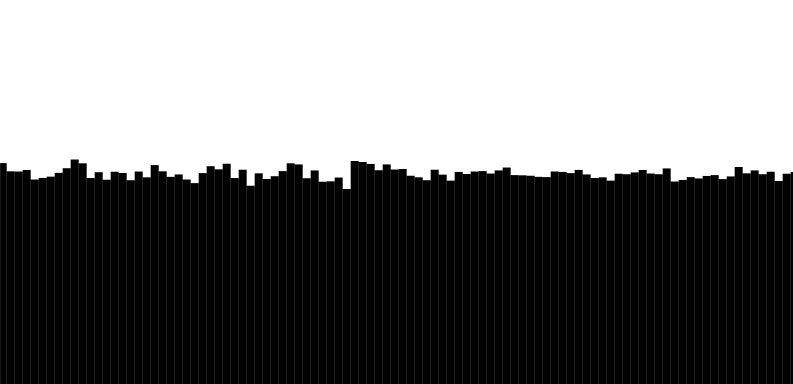
***M-Hash***

**Spread of Hash values (100 bins)**

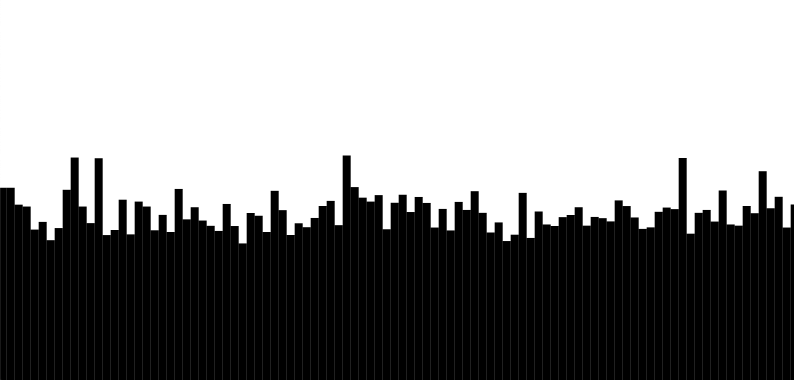
**R = 3 (small number, prime)**



**R = 31 (middle range number, prime)**



**R = 140 (large number, non prime)**



Amongst the 3 multiplier values for R, R = 31 performs the best with the most even distribution of hash values.

**“Time to collision” for 100 experiments**

**R = 3**

N min mean stdev max

----------------------------------------------------------------------

100 210.000 2764.720 1323.659 6860.000

**R = 31 (Time to collision is vastly higher)**

N min mean stdev max

----------------------------------------------------------------------------

100 10071.000 **77987.880** 37797.191 168031.000

**R = 140**

N min mean stdev max

----------------------------------------------------------------------

100 1544.000 8771.030 4632.829 21870.000

Based on the above data, M hash using a prime number within the approximate range of 20<x<80, should be optimal in delivering both a high time to collision as well.

***P-Hash***

**Spread of Hash values (100 bins)**



P-hash has an uneven distribution of hash values even where every input string is unique, making it unsuitable to be used. There is a pattern observed of certain equally spaced ranges where hash values occur more frequently.

**“Time to collision” for 100 experiments**

N min mean stdev max

----------------------------------------------------------------------

100 2.000 21.300 9.485 45.000

Compare to M hash, the time to collision is far smaller, implying that collisions appear very frequently even when the input is distinct. As a result, this version of P hash is not viable in this situation

***Q-Hash (P-Hash Variant)***

**Spread of Hash values (100 bins)**

****

For Q-Hash, the distribution is far more uniform than its P-Hash counterpart, comparable to that of M-hash with an optimal choosing of the multiplier R.

**“Time to collision” for 100 experiments**

N min mean stdev max

---------------------------------------------------------------------------

100 8756.000 94533.560 45049.104 201674.000

Furthermore, the time to collision for this version of P-Hash is better than the earlier data from M-Hash.

***J-Hash***

**Spread of Hash values (100 bins)**

J-Hash also appears to have a fairly uniform distribution of hash values comparable to that of M-Hash and Q-Hash.

**“Time to collision” for 100 experiments**

N min mean stdev max

----------------------------------------------------------------------

100 2604.000 17594.270 9455.062 46689.000

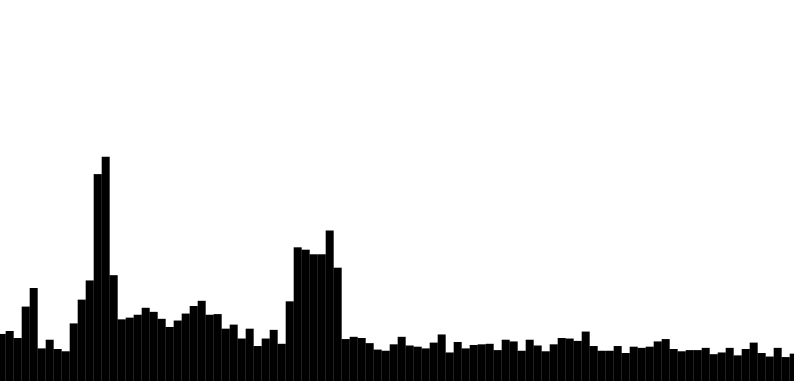
Despite its visible uniformity, J-Hash still falls short compared to Q-Hash with respect to its time to collision Statistic. **As a result, Q-hash seems to be the best option for use in words.txt due to an equal distribution of hash values as well as the highest time to collision in a set of fully distinct inputs.**

**Tale.txt – case for distinct inputs but also some frequently repeated inputs**

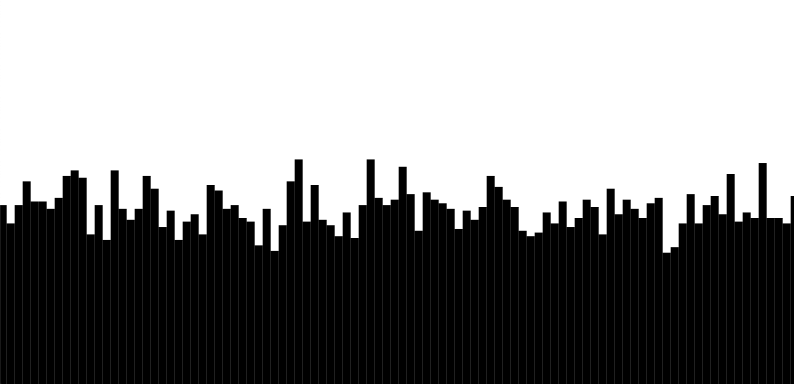
***M-Hash***

**Spread of Hash values (100 bins)**

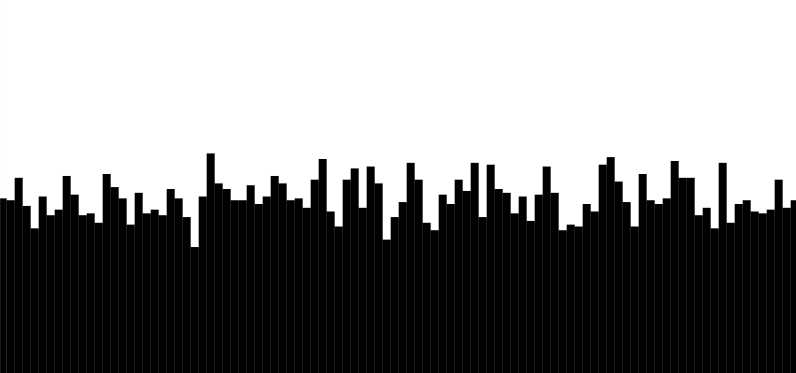
**R = 3**



**R = 31**

****

**R = 140**

****

*Note: Apart from the images above, I also checked R values of 7,19,67,91. All led to similar distributed hash values as the R = 31 and R = 140 case.*

Reviewing the 3 cases above, M-hash seems to work best when R = 31 as the hash values are the most uniform of the three. However, with the introduction of the file “tale.txt” which has regular sentences compared to a dictionary of distinct words, the values are no longer as well distributed as before.

**“Time to collision” for 100 experiments**

**R = 3**

N min mean stdev max

----------------------------------------------------------------------

100 24.000 111.480 56.987 298.000

**R = 31**

N min mean stdev max

----------------------------------------------------------------------

100 93.000 2549.440 1550.258 8312.000

**R = 140**

N min mean stdev max

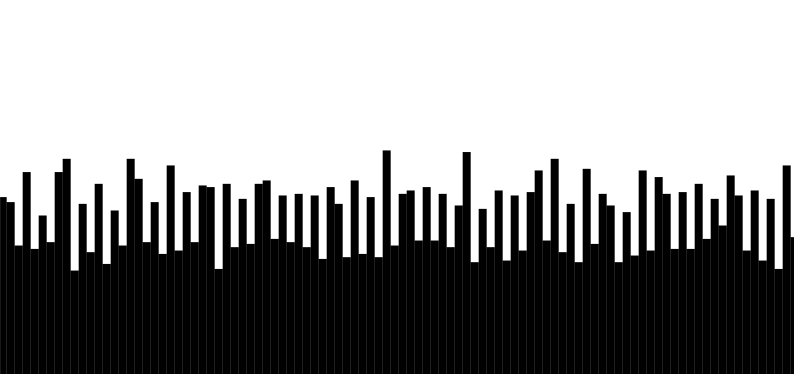
----------------------------------------------------------------------

100 3833.000 12719.180 3645.146 19191.000

In this scenario, our arbitrary case of R = 140 has the best time till collision. It is possible that in this case, words that are used in regular sentences might be smaller and simpler than those in the dictionary and hash to similar values . Thus using a large value of R may spread them out.

***P-Hash***

**Spread of Hash values (100 bins)**

****

Again, this version of P hash has an uneven distribution of hash values. With a pattern observed of certain equally spaced ranges of hash values occurring more frequently.

**“Time to collision” for 100 experiments**

N

min mean stdev max

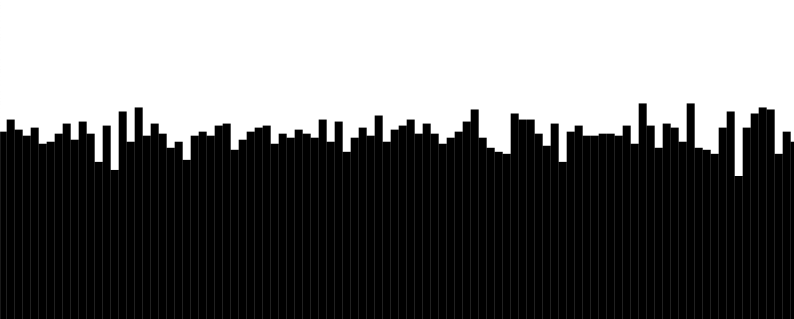
----------------------------------------------------------------------

100 2.000 16.780 7.618 39.000

The time to collision for this hash function is still very small even though there are still a significant number of distinct inputs.

***Q-Hash***

**Spread of Hash values (100 bins)**

****

For Q-hash, the hash values seem relatively well distributed, more so than any of the M-hash histograms.

**“Time to collision” for 100 experiments**

N min mean stdev max

----------------------------------------------------------------------

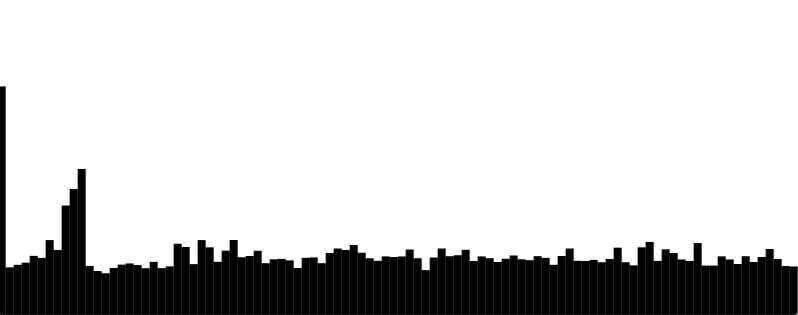
100 19696.000 19696.000 0.000 19696.000

The results for time to collision in the case of Q-hash for “tale.txt” suggests that there are no collisions whatsoever.

Using the Zipf and frequency counter classes from Lab 8&9, I verified that the file tale.txt has 19695 distinct words. Q-hash spreads these values over the 2^31 range for any positive integer value (primitive type). Therefore it is definitely viable that Q-hash in this case prevents the case of collisions completely.

***J-Hash***

**Spread of Hash values (100 bins)**

****

For J-hash, the hash values seem to be very well distributed amongst the larger hash values values, but poorly distributed for smaller hash values.

**“Time to collision” for 100 experiments**

N min mean stdev max

----------------------------------------------------------------------

100 1045.000 6495.850 3381.027 15475.000

Compared to the case of R = 140, J hash a similar standard deviation, but it’s mean time to collision value is almost half as small, suggesting that both M-hash and Q-hash are superior.

**As a result, Q-hash seems to be the best option as it not only distributes the hashes regularly but also ensures no collisions whatsoever in a case where the number of distinct entries is much smaller than 2^31.**

**Dna3. Txt – case of very large strings composed of repeating characters**

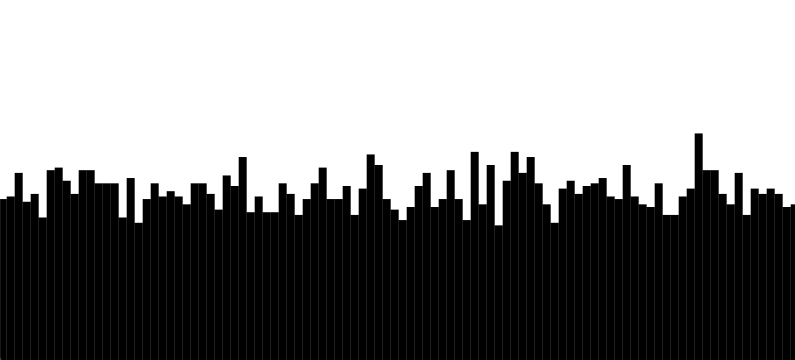
***M-Hash***

**Spread of Hash values (100 bins)**

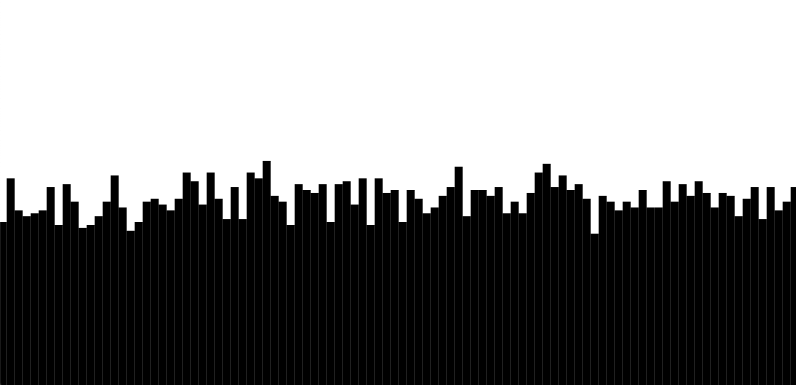
**R = 3**

****

**R = 31**

****

**R = 140**

****

Reviewing the 3 cases above, M-hash seems to work well for R = 31 and R = 140 but very poorly for R = 3, with a large number of majority of hash values within a fairly small range.

**“Time to collision” for 100 experiments**

**R = 3**

N min mean stdev max

----------------------------------------------------------------------

100 310.000 3056.370 1563.117 8349.000

**R = 31 (No Collision)**

N min mean stdev max

----------------------------------------------------------------------

100 13086.000 13086.000 0.000 13086.000

**R = 140**

N min mean stdev max

----------------------------------------------------------------------

100 244.000 2440.960 1274.995 6951.000

Reviewing the 3 cases above, M-hash seems to work best for R = 31 as there seems to be a case of no collisions across 100 experiments. Upon verifying with the frequency counter class from Lab 8 & 9 indeed has 13085 distinct DNA sequences present in the text file. Furthermore, R = 3 and R = 140 have similar time to collision values. Given the huge lengths of each DNA sequence, it is possible that the hash values of with R = 140 exceed the range of 2^31 and as a result of the “& 0x7FFFFFFF” operation, we might have skewed results in this situation.

***P-Hash***

**Spread of Hash values (100 bins)**

****

Again, this version of P hash has an uneven distribution of hash values. With a pattern observed of certain equally spaced ranges of hash values occurring more frequently.

**“Time to collision” for 100 experiments**

N min mean stdev max

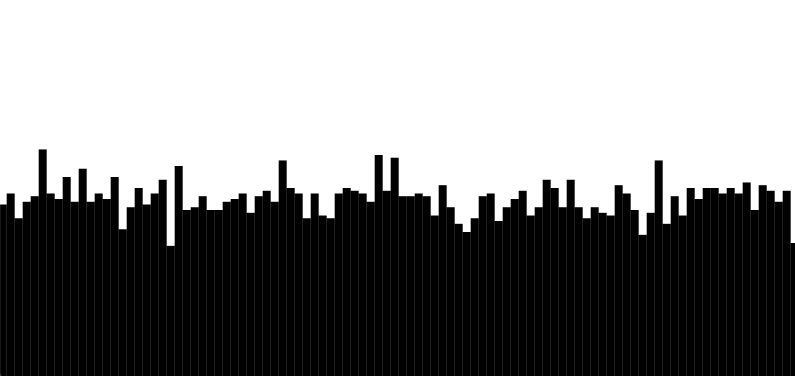
----------------------------------------------------------------------

100 3.000 19.130 9.056 46.000

The time to collision for this hash function is still very small compared to those from M-hash, making it ineffective.

***Q-Hash***

**Spread of Hash values (100 bins)**

****

Like the case for R = 31 in M-Hash, Q-hash has a well distributed graph. However, on closer visual inspection it seems (to me) like the M-Hash function still has a slightly more even distribution.

**“Time to collision” for 100 experiments**

N min mean stdev max

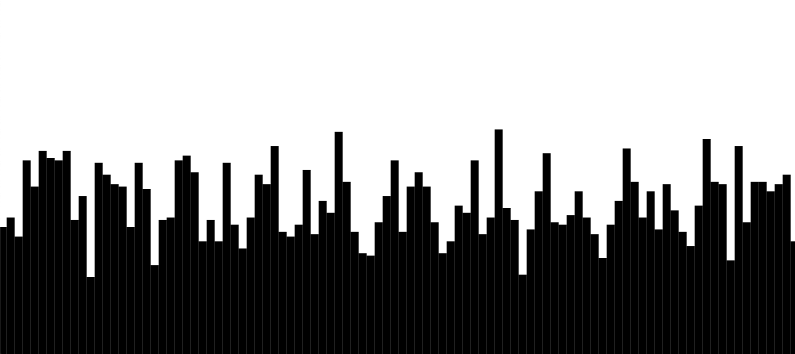
----------------------------------------------------------------------

100 13086.000 13086.000 0.000 13086.000

Similar to the R=31 case, Q-Hash exhibits no collisions.

***J-Hash***

**Spread of Hash values (100 bins)**

****

Compared to M-Hash and Q-Hash, the distribution of hash values for J-hash is far more uneven, making it less effective than the other two hashes.

**“Time to collision” for 100 experiments**

N min mean stdev max

----------------------------------------------------------------------

100 125.000 1082.840 591.200 2729.000

J hash has a much smaller value for time to collision as well, even less so than the M-hash case for R = 3 which had a very uneven distribution in the centre. This suggests it is less effective for this case.

**Reviewing all the cases above, both Q-hash and M-Hash for R = 31 seem to work very well, with a good distribution of Hash values as well as no collisions. However, the M-hash seems to have a slightly more even distribution based on a visual comparison and is also less computationally intensive than Q-Hash, performing noticeably faster upon multiple attempts. Therefore for this data set, I would recommend using M-hash with R = 31.**