Expected Number of Hash Collisions for a given hash algorithm.

Table of Contents

1. Introduction

This exploration intends to find the expected number of collision a given hash function.

An ideal hash function has a variety of properties, one of these properties is that ideal hash functions have no hash collisions. While this is properties of hash functions, in the real world even the best cryptographic hash functions have some hash collisions.

As someone very interested in computer science, especially encryption, exploring encryption in specific hashing algorithms from a more mathematical perspective is very fascinating to me. Furthermore, I found hash collision specifically to be a counter-initiative concept which occurs in hash algorithms even though they are designed to prevent it, hence when I realised I could apply mathematics to calculate the expected number of collisions for any hash function, I was inevitably excited to proceed with the investigation.

2. Background Information

2.1 Hash Functions

A hash function is a function which can map any given input of any size to an output of a preset size. This output is called a hash and is usually represented in base16

$$MD5(test): 098F6BCD4621D373CADE4E832627B4F6$$

Shown above is an example of a hash using a hash function called MD5, where "test" is the input and "098F6BCD4621D373CADE4E832627B4F6" is the output.

Ideal hash functions have various features, but the one that is relevant to this exploration states, "it is infeasible to find two different inputs with the same hash value".

2.2 Hash Collision

A hash collision is when two input values are mapped to the same output value.

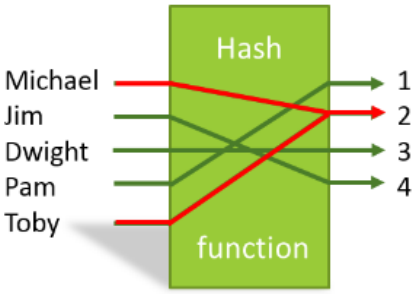

Figure 1 : Hash Collisions

The image shows a hash collision where the input of both Michael and Toby are mapped to the output value of 2.

[1]

While this contradicts the above-stated property of ideal hash functions, hash functions used in the real world all have some chance of hash collisions however small.

2.3 Other background Knowledge

2.3.1 Modulus Operator

In this exploration, the operator $x(mod\ y)$will be used frequently, this operator represents the remainder after dividing $x \div y$. When $y > x, x(mod\ y) = x$ and When $y = x\ , x(mod\ y) = 0$[2]

[1] "Qlikview Hash Functions And Collisions - The Qlik Fix!". *The Qlik Fix!*, 2019, http://www.qlikfix.com/2014/03/11/hash-functions-collisions/.

[2] "What Is Modular Arithmetic?". *Khan Academy*, 2019, https://www.khanacademy.org/computing/computer-science/cryptography/modarithmetic/a/what-is-modular-arithmetic.

2.3.2 Bits

In this exploration, the term bits and bit size will come up frequently. A bit is what a binary digit is called, it can take values of only 0 and 1. It is a representation of a number in base2. [3]

Bit size is just the representation of a number of bits used for example 0010 is of size 4 bits.

Another thing to keep in mind it that 8 bits make one byte.

[3] "Bits And Bytes". *Web.Stanford.Edu*, 2019, https://web.stanford.edu/class/cs101/bits-bytes.html.

3.Generating Hashes

3.1 Choosing Appropriate Hash Function

To make the process easy to do manually, I have chosen a simple 8-bit hashing function called Pearson hashing. Using the algorithm, I will be able to conduct a test and try to reach a general solution for the probabilities of hash collisions in hash functions.

Pearson hashing is a fast and easy algorithm which produces an 8-bit hash. Its implementation requires only a few instructions, plus a lookup table of 256 values, containing a permutation of the values 0 through 255. This lookup table is represented by the function $L(x)$below

3.2 Implementing Pearson Hashing

Below are functions and sequences which implement Pearson hashing using mathematics and will produce the required output hash value for any given input hash value.

$$Let\ I_n\ be\ a\ sequence\ which\ represents\ each\ letter\ in\ the\ user\ input$$

$$Let\ z\ be\ the\ length\ of\ sequence\ I_n$$

$$Let\ L(x)\ be\ a\ hybrid\ function\ defined\ by\ the\ below\ graph\ where\ x\ \in Z, 0 \leq x \leq 256$$

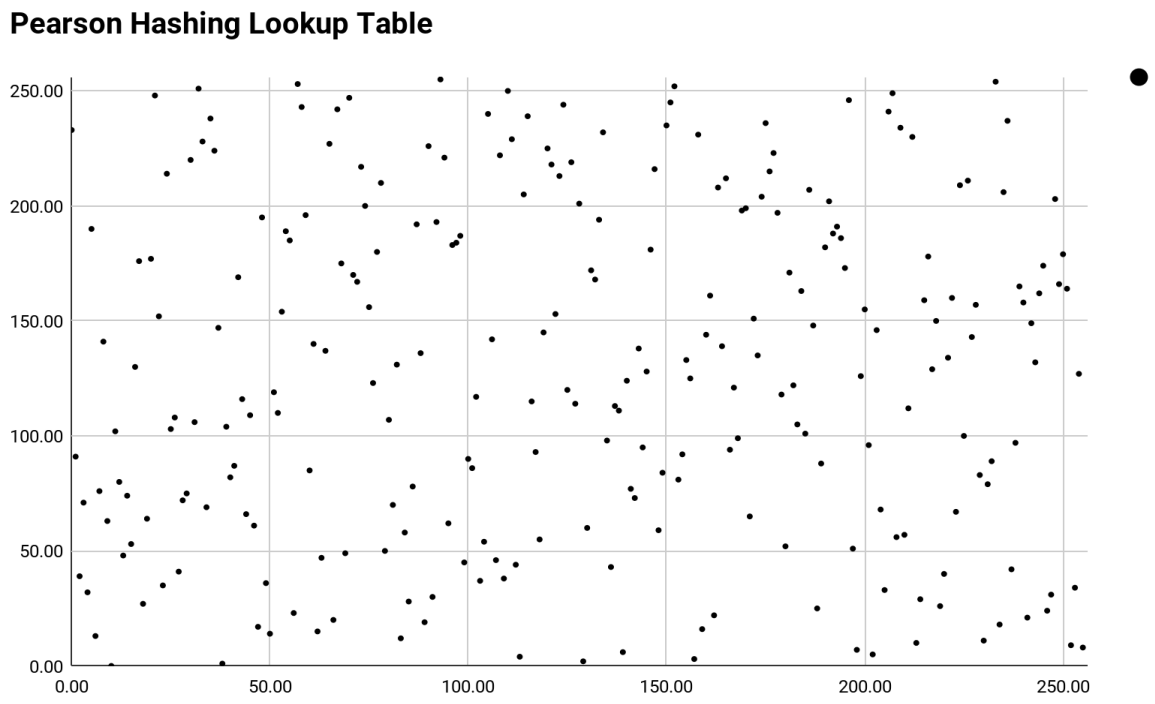

Figure 2 : Self Generated Lookup Table

Dec	Char	Dec	Char	Dec	Char	Dec	Char
0	NUL (null)	32	SPACE	64	@	96	`
1	SOH (start of heading)	33	!	65	A	97	a
2	STX (start of text)	34	"	66	B	98	b
3	ETX (end of text)	35	#	67	C	99	c
4	EOT (end of transmission)	36	$	68	D	100	d
5	ENQ (enquiry)	37	%	69	E	101	e
6	ACK (acknowledge)	38	&	70	F	102	f
7	BEL (bell)	39	'	71	G	103	g
8	BS (backspace)	40	(	72	H	104	h
9	TAB (horizontal tab)	41	)	73	I	105	i
10	LF (NL line feed, new line)	42	*	74	J	106	j
11	VT (vertical tab)	43	+	75	K	107	k
12	FF (NP form feed, new page)	44	,	76	L	108	l
13	CR (carriage return)	45	-	77	M	109	m
14	SO (shift out)	46	.	78	N	110	n
15	SI (shift in)	47	/	79	O	111	o
16	DLE (data link escape)	48	0	80	P	112	p
17	DC1 (device control 1)	49	1	81	Q	113	q
18	DC2 (device control 2)	50	2	82	R	114	r
19	DC3 (device control 3)	51	3	83	S	115	s
20	DC4 (device control 4)	52	4	84	T	116	t
21	NAK (negative acknowledge)	53	5	85	U	117	u
22	SYN (synchronous idle)	54	6	86	V	118	v
23	ETB (end of trans. block)	55	7	87	W	119	w
24	CAN (cancel)	56	8	88	X	120	x
25	EM (end of medium)	57	9	89	Y	121	y
26	SUB (substitute)	58	:	90	Z	122	z
27	ESC (escape)	59	;	91	[	123	{
28	FS (file separator)	60	<	92	\	124	|
29	GS (group separator)	61	=	93	]	125	}
30	RS (record separator)	62	>	94	^	126	~
31	US (unit separator)	63	?	95	_	127	DEL

Figure 3 : ASCII Table

$Let\ A(x)$ be a hybrid function which follows the ASCII Standard [4]represented by the table above.

$Let\ O\ be\ a\ sequence\ with\ general\ term\ O_n$

$$O_n = A(I_n\ ), where\ n: 1 \leq n \leq z, n \in Z$$

$Let\ h_n\ be\ a\ sequence$

$General\ term\ of\ sequence\ h\ is\ given\ below$

$$h_{n+1} = L((h_n + O_n)(mod\ 256)), where\ n: 1 \leq n \leq z, n \in Z$$

$h_1 = z(mod\ 256)$, This initial term of the recurrence relation shown above.

$h_{z+1} will\ be\ the\ desired\ output\ hash\ value\ represented\ in\ base\ 10$

[4] "ASCII Table". *Cs.Cmu.Edu*, 2019, https://www.cs.cmu.edu/~pattis/15-1XX/common/handouts/ascii.html.

3.3 Converting "MATH" to a hash value

Now using the method detailed above for finding a hash value, I will be finding the hash value of "MATH" using Pearson Hashing

In this case $I_n: \{'M', 'A', 'T', 'H'\}$

The sequence I_n has four terms $\Rightarrow z = 4$

$$O_n = A(I_n), where\ 1 \leq n \leq 4, as\ z = 4$$

From the ASCII Table, we know 'M' = 77, 'A' = 65, 'T' = 84, 'H' = 72

$$\Rightarrow O_n: \{77,65,84.72\}$$

$$h_1 = z(mod\ 256) \Rightarrow h_1 = 4(mod\ 256) \Rightarrow h_1 = 4$$

The final hash value for "MATH" is $h_{z+1} \Rightarrow h_5$ after the solving the below recursion relation

$$h_{n+1} = L((h_n + O_n)(mod\ 256)), where\ n: 1 \leq n \leq z, n \in Z$$

Solving for $h_5 = L((h_4 + O_4)(mod\ 256)) \Rightarrow L((h_4 + 72)(mod\ 256))$

Solving for $h_4 = L((h_3 + O_3)(mod\ 256)) \Rightarrow h_4 = L((h_3 + 84)(mod\ 256))$

Solving for $h_3 = L((h_2 + O_2)(mod\ 256)) \Rightarrow h_3 = L((h_2 + 65)(mod\ 256))$

Solving for $h_2 = L((h_1 + O_1)(mod\ 256))$

Substituting the value of h_1 in the above equation

$$\Rightarrow h_2 = L((4 + 77)(mod\ 256)) \Rightarrow h_2 = L(81) \Rightarrow h_2 = 70$$

Similarly, the required values are substituted in for the below equations

$$h_3 = L((h_2 + 65)(mod\ 256)) \Rightarrow h_3 = L((70 + 65)(mod\ 256)) \Rightarrow h_3 = L(135) \Rightarrow h_3 = 98$$

$$h_4 = L((h_3 + 84)(mod\ 256)) \Rightarrow h_4 = L((98 + 84)(mod\ 256)) \Rightarrow h_4 = L(182) \Rightarrow h_4 = 122$$

$$h_5 = L\big((h_4 + 72)(mod\ 256)\big) \Rightarrow h_5 = L\big((122 + 72)(mod\ 256)\big)$$

$$\Rightarrow h_5 = L(194) \Rightarrow h_5 = 186$$

The desired value of $h_5$which is the hash value of "MATH" is 186 represented in base10.

4. Finding Hash Collisions

For hash collisions, the two most important variables are n, k where n is the number of possible unique inputs and k is the number of possible unique outputs. Using pigeon hole principle, we know that when $n > k$ then in at least one output will co-relate to two or more inputs.

As n, k will be rather large number and as this investigation is dealing with only uppercase letters, the value of n, k can be denoted as $n = 26^l, where\ l\ is\ number\ of\ letters$ and $k = 2^b, where\ b\ is\ number\ of\ bits$

First, I will be calculating the number of Collisions for any given input represented by function $C(I)$ where I is the given input. This value will be calculated by a program I wrote in python implementing Pearson hashing. This gave me an opportunity to not only explore a field of math but to develop my programming skills and learn how to compute and solve complex mathematically problems using programming.

4.1 N Constant

For the first case, I will keep n constant by restricting the character set to only uppercase letters and choosing a four-letter word, for this test I will be using the word "MATH" for which I had previously demonstrated the Pearson Hash Value of 186.

$$[26] \times [26] \times [26] \times [26]$$

The total number of possible combinations of upper case four letters is $26^4 = 456976$, this can be derived using the multiplicative principle of combinatorics. Four positions are using only four characters, using only upper-case characters there are 26 possible letters in each position hence $n = 26^4$

The above described Pearson hash function was an 8-bit function with $b = 8\ and\ k = 256$

Varying the bit size Pearson hash function to generate outputs for different values of k

Input Word	l	$n = 26^l$	b	$k = 2^b$	Collisions- $C(I)$
MATH	4	456976	8	256	1849
MATH	4	456976	9	512	931
MATH	4	456976	10	1024	425
MATH	4	456976	11	2048	220
MATH	4	456976	12	4096	84
MATH	4	456976	13	8192	69
MATH	4	456976	14	16384	22
MATH	4	456976	15	32768	20

The above values of Collisions are for the specific word “MATH” which I choose to run this test, but it is very likely that with a different four-letter uppercase word the individual value of Collisions might change. This can have an effect on the accuracy of the conclusions drawn from this data, but this shouldn’t impede this investigation severely as the key focus is on the relationship between $k\ and\ C(I)$ not necessarily the exact values of $C(I)$.

To further this investigation a clear relationship between $k\ and\ C(I)$ must be established. From the looking at a scatterplot of the data above, the data seems to falls a form of quadratic relationships in the form $C(I)\ \cong \mathrm{x}k^y$.

By taking two random data points from the above eight data points, the above relationship can be solved as a simultaneous equation with two variables.

$$C(I) = xk^y$$

$$\rightarrow 220 = x2048^y \text{ and } 22 = x16384^y$$

For the above, the above two equations the values of x, y are $102115, -1.11$.

Similarly, this process is repeated for the other six data points with the x, y as the following

$C(I)$	k	$C'(I)$	k'	x	y
220	2048	22	16384	102115	-1.11
1849	256	69	8192	356351	-0.948801
20	32768	425	1024	191914	-0.881878
931	512	84	4096	126754	-1.15677

From the above values average x, y values can be calculated as $194283.5, -1.022$ respectively.

A relationship $C(I) \cong 194283.5k^{-1.022}$ can be established, however, to ensure that this relationship is an accurate representation of the data the residuals for each point must be calculated.

The residual at a given point is nothing but the difference between the actual $C(I)$ value and the $C(I)$ predicted by the above relationships.

Taking the point $A(84,4096)$ where $C(I) = 4096$ and $k = 84$, hence the residual $r(I)$ is

$$R(I) = 4096 - (194283.5(84)^{-1.022}) \rightarrow 44.49$$

Using the same method to calculate the residual $r(I)$ for all the data points

k	$C(I)$	$r(I)$
256	1849	1177.24
512	931	600.20
1024	425	262.10
2048	220	139.78
4096	84	44.49
8192	69	49.54
16384	22	12.42
32768	20	15.28

Using the residuals for each point, the R^2 score can be calculated which shows how accurate the relationship is to the actual data set.

$$R^2 = 1 - \frac{\sum(r(I))^2}{\sum(C(I) - \overline{C(I)}}$$ [5]

By substituting the required values then simplifying the R^2 value is 0.3634.

An R^2 value of 0.3634 implies that the relationship is accurate only 36% of the time which is not a very good relationship. This is likely because the relationship was found by taking random data points and solving simultaneous equations. This method is constrained in its ability to accurately describe the relationship. Hence to derive a significant relationship between $k\ and\ C(I)$ the assistance of technology is needed.

[5] "1.5 - The Coefficient Of Determination, R-Squared | STAT 501". Newonlinecourses.Science.Psu.Edu, 2019, https://newonlinecourses.science.psu.edu/stat501/node/255/.

By using regression software on a scatterplot of the data a more accurate relationship can be established

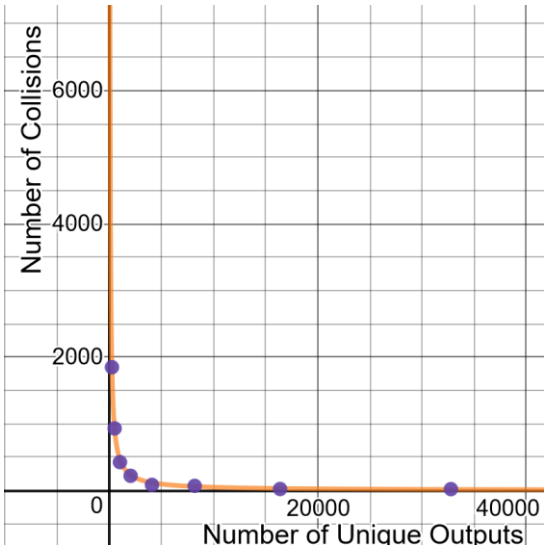

Figure 4 : Plotted Relationship between C(I) and bit size

$$C(I) \cong \mathrm{x}k^{y}, where\ x = 456976 \text{ and}$$

$$y = -0.99478\ with\ R^2 = 0.99$$

Looking at the above relationship, one can conclude that $b = n$ and the relationship can be rewritten as

$$C(I) \cong \frac{n}{k^{0.99478}}.$$

This relationship has a far superior R^2 of 0.99 when compared to the manually modelled relationship.

4.2 K Constant

With a relationship established for $C(I)\ and\ k$, I intend to establish a relationship for $C(I)\ and\ n$ by keeping k constant and varying n. For this investigation, the bit size for Pearson hash function is $b = 10 => k = 1024$

As shown above, each slot has 26 possible inputs. Therefore the total number of inputs is $n = 26^l$

Input Word	l	$n = 26^l$	b	$k = 2^b$	Collisions- $C(I)$
BAT	3	17576	10	1024	40
MATH	4	456976	10	1024	1491
MATHS	5	11881376	10	1024	34745
CHANCE	6	308915776	10	1024	1960937

As shown above in the previous test, manually modelling the above data would result in poor correlation which is not a statistically significant relationship. Hence, I used a computer programmer to plot a regression line through the data and calculate the relationship.

Plotting the above table where n is the x-axis and $C(I)$ is the y-axis.

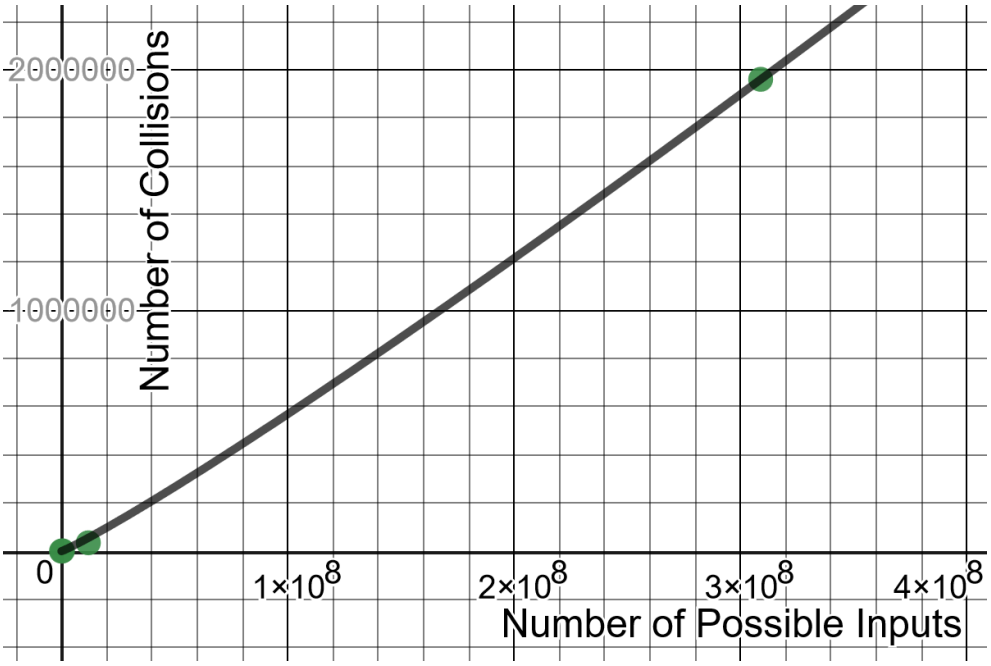

Figure 5: Plotted Relationship between C(I) and the Number of Unique Inputs

Using regression, a relationship between $n\ and\ C(I)$ can be established:

$$C(I) \cong \frac{n^a}{b}, where\ a = 1.0957\ and\ b = 1024\ with\ R^2 = 0.9999$$

Similar to the relationship between $k\ and\ C(I)$ looking at the above relationship, one could conclude $b = k$. Therefore, the relationship can be rewritten as $C(I) \cong \frac{n^{1.0957}}{k}$

4.3 Collisions for given Input

Both the derived relationships for $k, C(I)\ and\ n. C(I)$ are rather similar in nature and considering they are only approximations with limited data values, it is possible to derive an approximate value for $C(I)$ using both relationships

$$C(I) \cong \frac{n^{1.0957}}{k} \text{ and } C(I) \cong \frac{n}{k^{0.99478}}$$

As the position of $n\ and\ k\ relative\ to\ C(I)$ is the same in both relationships is clear that $C(I) \propto \frac{n}{k}$. Apart from this both numbers in the respective powers of $n\ and\ k$ are relativity close

to 1 and can be approximated as 1. Therefore, $C(I)$ a function for the number of Collisions for any given input I is approximately the following:

$$C(I) \cong \frac{n}{k}$$

This above relationship can also be proven theoretically, given that there are n inputs and k hash locations/outputs. Let X_i be a random variable that checks if an input hashed to location 1 in trial i, so X_i value can be either $0\ or\ 1$. The probability of that any one item hashes to location 1 is $1/k$ as in hash functions, the probability of hashing to any locations is equally likely. Therefore, the expected value of $X_i = 1/k$. Now let X be random variable $X_1 + X_2 + X_3 + \cdots + X_n$.

$$E(X) = E(X_1 + X_2 + \cdots + X_n)$$

$E(x + y) = E(x) + E(y)$ for any given random variables x, y[6]

Using the above formula and solving for $E(X)$

$$E(X) = E(X_1) + E(X_2) + \cdots E(X_n)$$

$$=> E(X) = \frac{1}{k} + \frac{1}{k} + \cdots + \frac{1}{k} = \frac{n}{k}$$

Hence expected value of $X = n/k$ as X is the sum of n terms of value $1/k$. As this same expected value applies to any locations the above approximation is proven.

In fact, the above-shown approximation is, in fact, one of many theorems about hash functions, the theorem states: *In hashing n items into a hash table of size k, the expected number of items that hash to any one location is n/k.*[7]

[6] "Random Variable Combinations". *Stattrek.Com*, 2019, https://stattrek.com/random-variable/combination.aspx.

[7] *PROBABILITY CALCULATIONS IN HASHING*. Darthmouth, 2019, pp. 245-247, https://math.dartmouth.edu/archive/m19w03/public_html/Section6-5.pdf. Accessed 4 Mar 2019.

4.4 Expected Number of Collisions

With the value for Collisions $C(I)$ for any given input I, now we can find $E(Collisions)$ which is the expected number of collisions. $E(Collisions)$ will be the total number of inputs n minus the number of occupied locations as each occupied location will contain one input that would not have collided in the process of hashing.

$$E(Collisions) = n - E(occupied\ locations)$$

$E(occupied\ locations)$ is total number of locations or total number of unique outputs k minus the $E(empty\ locations)$. Hence, the equation for $E(Collisions)$ can be written as

$$E(Collisions) = n - k + E(empty\ locations)$$

Using the similar logic used to find the number of collisions or any given input, we can find the expected number of empty locations. The probability that any position i will be empty after 1 item is hashed is $1 - \frac{1}{k}$. Considering the trial process independent with two outcomes, input hashes to slot i or it doesn't, it is clear the probability of no input hashing to slot i from n inputs is $(1 - \frac{1}{k})^n$.

Let X_i be a random variable equal to 1 if location i is empty and 0 if it is not. The expect number of empty locations will be equal to $X_1 + X_2 + X_3 + \cdots + X_k$, thus an equation for the expected number of empty locations can be written in terms of n, k

$$E(Empty\ Locations) = k(1 - \frac{1}{k})^n$$

Substituting the above equation in the equation for $E(Collisions)$, we get

$$E(Collisions) = n - k + k(1 - \frac{1}{k})^n$$

Similar to the equation derived for Collisions for a given input both the above equations are in fact theorems about hashing, which we were able to derive using data.

The theorems states:

1. *In hashing n items into a hash table with k locations, the expected number of empty locations is* $k(1-\frac{1}{k})^n$.[8]
2. *In hashing n items into a hash table with k locations, the expected number of collisions is* $n-k+k(1-\frac{1}{k})^n$.[8]

The above equation can also be converted to use bit size b instead of k by substituting $k = 2^b$

$$E(Collisions) = n - 2^b + 2^b\left(1-\frac{1}{2^b}\right)^n, where\ n.b\ \in Z^+$$

4.5 E(Collisions) For Common Hashing Algorithms

First, I will calculate using the above formula the expected number of collisions for the Pearson Hashing algorithm which has been used in this investigation, after this I will be calculating the expected number of collisions for other popular hashing algorithms.

The initial Pearson hashing algorithm we used for calculating the hash of the word "MATH" had a bit size of 8 and we used only four upper case characters, for this case the expected number of collisions:

$$E(Collisions) = 456976 - 2^8 + 2^8\left(1-\frac{1}{2^8}\right)^{456976}$$

$$=> 456720$$

The expected number of collisions is so high as the $n \gg 2^b$

[8] *PROBABILITY CALCULATIONS IN HASHING*. Darthmouth, 2019, pp. 245-247, https://math.dartmouth.edu/archive/m19w03/public_html/Section6-5.pdf. Accessed 4 Mar 2019.

In the real-world hashing algorithms with so many collisions would practically be useless, but most modern hashing algorithms have enormous bit sizes and do not collide unless n is very large. For practical applications ,n cannot be restricted to only four upper case characters, for showcasing expected number of collisions of modern hashing algorithms I will be using the entire English alphabet, both upper and lower case, which is 52 characters. Additionally, all 10 digits will be included and two symbols for good measure, this brings the number of characters to 64. The number of slots used will be 10 which would result in a total input size of

$$64^{10} = 1.15292 \times 10^{18}$$

Hashing Algorithm	$b(Bit\ Size)$	$n(Input\ Size)$	$k = 2^b$	$E(Collisions)$
Pearson	8	1.152×10^{18}	256	1.153×10^{18}
MD-5	128	1.152×10^{18}	3.403×10^{38}	0
SHA-1	160	1.152×10^{18}	1.462×10^{48}	0
SHA-2	224	1.152×10^{18}	2.696×10^{67}	0
SHA-2	256	1.152×10^{18}	1.158×10^{77}	0
SHA-2	384	1.152×10^{18}	3.940×10^{115}	0
SHA-2	512	1.152×10^{18}	1.340×10^{154}	0

None of the modern hashing algorithms collides $n = 1.15292 \times 10^{18}$ as $n \ll k$. Such large bit sizes enable modern hashing algorithms to minimize collisions.

Out of all the above show algorithms, MD5 has the small bit size after Pearson with a bit size of $b = 128$, below shown is the graph of a discontinuous function which showcases the first value of n for which MD5 collides.

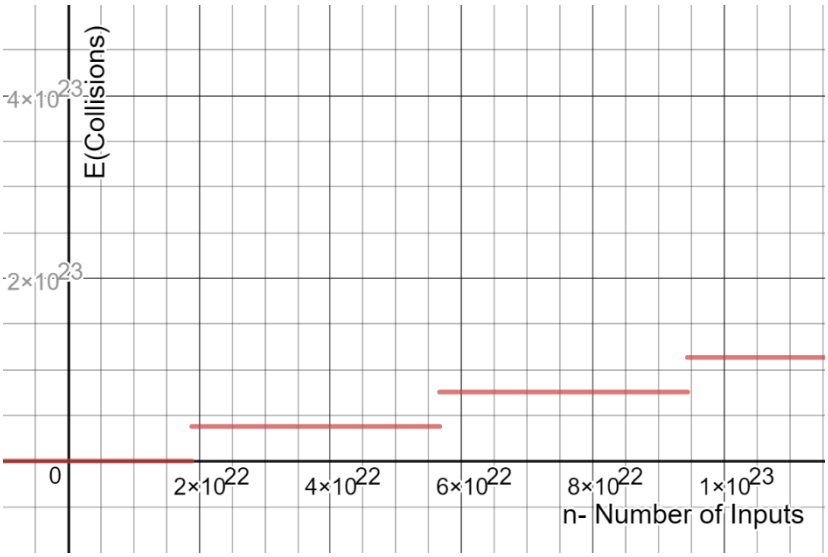

Figure 6 : Relationship between E(Collisions) and Number of Unique Inputs

For $n > 1.89 \times 10^{22}$, the function of $E(Collisions) > 0$

4.6 Optimizing E(Collisions) for N and K

The function for $E(Collisions)$ would be more useful to help create secure hashing algorithms if it could be optimized, finding a n, k values for a minimum $E(Collisions)$

$$E(Collisions) = n - 2^b + 2^b\left(1 - \frac{1}{2^b}\right)^n, where\ n.b\ \in Z^+$$

While the function $E(Collisions)$ is a discrete function and the properties of differentiation are not supposed to be used to find a minimum, there are times that this method does work to solve for a minimum. Hence, I am assuming the function $E(Collisions)$ to be a continuous function to attempt to find a minimum using partial derivatives. The function $E(Collisions)$ is rewritten with x, y instead of $n, k\ where\ x = n\ and\ y = k$.

$$F(x, y): x - 2^y + 2^y\left(1 - \frac{1}{2^y}\right)^x, where\ x, y\ \in \mathbb{R}$$

As the above function is a multivariable function in terms of n, k, to optimize the function I will have to use partial derivates and multivariable calculus. This was another great learning opportunity for me, I was able to learn a new part of calculus about which I knew little to nothing before.

The first condition for a minimum at point $A(x_0, y_0)$ for the function $F(x, y)$ is that

$$\nabla F(x_0, y_0) = 0^9$$

Another notation for $\nabla F(x_0, y_0)$ is below

$$\begin{bmatrix} F'_x(x_0, y_0) \\ F'_y(x_0, y_0) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \mathbf{0}$$

First, I will evaluate the derivate of the function with respect to x which is $\frac{\partial}{\partial x} F(x, y)$

While evaluating $\frac{\partial}{\partial x} F(x, y)$ the variable y can be considered a constant, hence the function can be re-written as

$$\frac{\partial}{\partial x}\left(2^y \left(1 - \frac{1}{2^y}\right)^x - 2^y + x\right) \rightarrow a^x - b + x$$

$$where\ a = \left(1 - \frac{1}{2^y}\right)\ and\ b = 2^y$$

$$\frac{\partial}{\partial x}(b \times a^x - b + x) = b \times a^x ln(a) + 1$$

Substituting a, b back in the above equation

$$\frac{\partial}{\partial x}\left(2^y \left(1 - \frac{1}{2^y}\right)^x - 2^y + x\right) = 2^y \left(1 - \frac{1}{2^y}\right)^x \ln\left(2^y \left(1 - \frac{1}{2^y}\right)\right) + 1$$

[9] "Introduction To Partial Derivatives". *Khan Academy*, 2019, https://www.khanacademy.org/math/multivariable-calculus/multivariable-derivatives/partial-derivative-and-gradient-articles/a/introduction-to-partial-derivatives.

Similarly, differentiation the function with respect to y, the below equation can be written with x as a constant

$$\frac{\partial}{\partial y}\left(2^y\left(1-\frac{1}{2^y}\right)^x-2^y+x\right)\rightarrow\frac{\partial}{\partial y}\left(2^y\left(1-\frac{1}{2^y}\right)^a-2^y+a\right), where\ a=x$$

Applying the Sum/Difference Rule: $(f\pm g)'=f'+g'$

$$\frac{\partial}{\partial y}(a)=0$$

$$\frac{\partial}{\partial y}(2^y)=\ln(2)\times 2^y$$

$$\frac{\partial}{\partial y}\left((2^y(1-\frac{1}{2^y})^a\right)$$

Apply the product rule on the previous derivative: $(f\times g)'=f'\times g+g'\times f$

$$f=2^y, g=(1-\frac{1}{2^y})^a$$

$$\rightarrow\frac{\partial}{\partial y}(2^y)\left(1-\frac{1}{2^y}\right)^a+\frac{\partial}{\partial y}\left(1-\frac{1}{2^y}\right)^a(2^y)$$

$$\frac{\partial}{\partial y}(2^y)=\ln(2)\times 2^y$$

$$\frac{\partial}{\partial y}\left(1-\frac{1}{2^y}\right)^a=a(1-\frac{1}{2^y})^{a-1}\times\ln(2)\times\frac{1}{2^y}\times 2^y$$

$$\rightarrow\left(1-\frac{1}{2^y}\right)^a\times\ln(2)\times 2^y+(2^y)\times(a\left(1-\frac{1}{2^y}\right)^{a-1}\times\ln(2))$$

$$\rightarrow\left(1-\frac{1}{2^y}\right)^a\times\ln(2)\times 2^y+(2^y)\times\left(a\left(1-\frac{1}{2^y}\right)^{a-1}\times\ln(2)\right)-\ln(2)\times 2^y+0$$

Simplifying the above equation and substituting $a=x$ back in

$$\rightarrow\ln(2)(x\left(1-\frac{1}{2^y}\right)^{x-1}+2^y(\left(1-\frac{1}{2^y}\right)^x-1)$$

$$\text{Therefore } \nabla F(x,y) = \begin{bmatrix} F'_x(x,y) \\ F'_y(x,y) \end{bmatrix} = \begin{bmatrix} 2^y \left(1-\frac{1}{2^y}\right)^x \ln\left(2^y\left(1-\frac{1}{2^y}\right)\right)+1 \\ \ln(2)(x\left(1-\frac{1}{2^y}\right)^{x-1} + 2^y(\left(1-\frac{1}{2^y}\right)^x - 1) \end{bmatrix}$$

To proceed with optimising the function, I should be able to find a solution for the above set of simultaneous equations, however, after extensive attempts to manually solve the above equation and further attempts to use software like wolfram alpha or even custom written program, I was unable to find a solution for above equations.

Within the scope and limitation of my investigation I found it impossible to find a solution for the above system of equations and hence cannot proceed in optimising the function. The techniques I attempted couldn't solve the system of equations and techniques that could possibly solve this system of equation are far beyond the scope of both the high school and even undergraduate mathematics.

For the above-stated reasons, I will assume that the system of equations has no solution and hence the function has no minima and cannot be optimised.

5. Conclusions

The goal of the investigations was to explore the mathematics behind hash functions and to determine a function for the expected number of collisions for any given hash functions.

The investigation was able to achieve both of these goals, first by exploring how simple hash functions uses mathematics to convert a given input of any length to a fix length output which was demonstrated using the Pearson Hash Function.

Furthermore, the investigation was able to derive theorems valid for any given hash function by a custom program to run collision tests on the Pearson Hash Function.

Finally, the investigation was able to derive a function for the expected number of collisions,$E(C)$ dependent on the size of input and bits size of hash functions.

$$E(C) = n - 2^b + 2^b\left(1 - \frac{1}{2^b}\right)^n, where\ n.b\ \in Z^+$$

The investigation was even able to explore the applications of this function with real world hashing algorithms like MD5 and SHA. While the investigation was limited by its scope and was unable to optimise the function, finding an exact minimum the investigation found that

$$E(C) \to 0\ as\ n \ll k$$

For me, this investigation gave me an opportunity to learn a lot of new skills and develop my existing skills in both mathematics and computer science, I had a chance to learn multivariable calculus which is something I might not have discovered otherwise, furthermore having a practical application of my knowledge in python to assist this investigation helped me become more comfortable with the language and with using programming to solve mathematical problems. This investigation gave me new insights in the field of encryption and hashing algorithms which is a field that has fascinated me for the past few years, it gave me a deeper and

more mathematical understanding of a concept which I once thought was primarily a computer science concept.

The two primarily limitations of this investigation were Time and Computational Power, with limited time to both conduct and write this investigation, I was unable to achieve somethings which might have been possible given longer time. Similarly, the data collected by the custom program was limited in the scope of hashing algorithms and bit sizes it could test due to a limitation in computational power and the time the program could run for. This limited the data to only one hashing algorithm and one smaller bit sizes. Of course, the investigation is inevitable limited by some human error on my part and my ability and knowledge in mathematics.

To carry this investigation forward I would like to collect data on collisions for more hashing algorithms with a larger variance in bit size, I would also like to derive a function for Probability of Collision for any given input for any given hash function.

6.References

1. "Bits And Bytes". *Web.Stanford.Edu*, 2019, https://web.stanford.edu/class/cs101/bits-bytes.html.

2. "Introduction To Partial Derivatives". *Khan Academy*, 2019, https://www.khanacademy.org/math/multivariable-calculus/multivariable-derivatives/partial-derivative-and-gradient-articles/a/introduction-to-partial-derivatives.

3. "Qlikview Hash Functions And Collisions - The Qlik Fix!". *The Qlik Fix!*, 2019, http://www.qlikfix.com/2014/03/11/hash-functions-collisions/.

4. "Random Variable Combinations". *Stattrek.Com*, 2019, https://stattrek.com/random-variable/combination.aspx.

5. "What Is Modular Arithmetic?". *Khan Academy*, 2019, https://www.khanacademy.org/computing/computer-science/cryptography/modarithmetic/a/what-is-modular-arithmetic.

6. "1.5 - The Coefficient Of Determination, R-Squared | STAT 501". *Newonlinecourses.Science.Psu.Edu*, 2019, https://newonlinecourses.science.psu.edu/stat501/node/255/.

7. *PROBABILITY CALCULATIONS IN HASHING*. Darthmouth, 2019, pp. 245-247, https://math.dartmouth.edu/archive/m19w03/public_html/Section6-5.pdf. Accessed 4 Mar 2019.

8. "ASCII Table". *Cs.Cmu.Edu*, 2019, https://www.cs.cmu.edu/~pattis/15-1XX/common/handouts/ascii.html.

9. Tarak, Rahul. "Cryogenicplanet/Mathia". *Github*, 2019, https://github.com/CryogenicPlanet/mathIA.

7. Appendix

All the programs write for this investigation can be found below and at the repository https://github.com/CryogenicPlanet/mathIA with all the lookup tables also

K Constant Test Code

```
      import pickle
ten = []
with open ('ten.txt', 'rb') as fp:
    ten = pickle.load(fp)
correctHash = 186
trueCases  = 0
def hash8(message, table,mod):
    global trueCases
    hashed = len(message) % mod
    #print("Intial:")
    #print(hashed)
    for i in message:
        #print("Ord:")
        #print(ord(i))
        #print("postion :")
        #print( (hashed+ord(i))%256)
        #print("In loop hashed:")
        hashed = table[(hashed+ord(i)) % mod]
        if hashed == correctHash:
            trueCases = trueCases +1
        #print(hashed)
    return hashed
from itertools import product
from string import ascii_uppercase
#userInput = input("Enter 4 Letter Word : ")
keySet = []
for i in range(3,6):
    keywords = [''.join(i) for i in product(ascii_uppercase,
repeat = i)]
    keySet.append(keywords)
```

```
#print(len(keywords))
testWords = ["BAT","MATH","MATHS","CHANCE"]
number = 0
for i in range(4):
    print("Word : ")
    word = testWords[i]
    print(word)
    correctHash = hash8(word,ten,len(ten))
    if i != 3:
        keywords = keySet[i]
        trueCases = 0
        for word in keywords:
            output = hash8(word,ten,1024)
    else :
        with open("outputfile.txt") as infile:
            for line in infile:
                trueCases = 0
                line.split("\n")
                output = hash8(line,ten,1024)

        #byte = bytes(word,encoding='utf-8')
        #temp = hashlib.sha256(byte).hexdigest()
        #if temp == correctHashSha:
            #trueForSha = trueForSha + 1
    #print(trueCases)
    #print(len(keywords))
    #print(trueForSha)
    print(trueCases)
    print(" ------- ")
```

N Constant Code

```
import pickle
thirteen = []
with open ('thirteen.txt', 'rb') as fp:
    thirteen= pickle.load(fp)
fourteen =[]
with open ('fourteen.txt', 'rb') as fp:
    fourteen= pickle.load(fp)
fifthteen = []
with open ('fifthteen.txt', 'rb') as fp:
```

```
    fifthteen= pickle.load(fp)
twevle = []
with open ('twevle.txt', 'rb') as fp:
    twevle= pickle.load(fp)
eleven = []
with open ('eleven.txt', 'rb') as fp:
    eleven= pickle.load(fp)
ten = []
with open ('ten.txt', 'rb') as fp:
    ten = pickle.load(fp)
nine = [12, 438, 80, 6, 401, 403, 458, 78, 408, 127, 100, 138, 183, 182, 348,
61, 308, 230, 103, 159, 500, 414, 128, 55, 339, 445, 451, 441, 452, 425, 480,
333, 469, 291, 434, 156, 487, 86, 397, 144, 5, 363, 502, 508, 269, 303, 9,
19, 71, 58, 137, 420, 51, 320, 486, 482, 353, 505, 92, 398, 26, 326, 476,
135, 345, 352, 391, 313, 49, 171, 373, 481, 489, 57, 467, 1, 77, 443, 292,
180, 140, 245, 223, 427, 433, 293, 54, 325, 321, 3, 283, 217, 328, 56, 185,
279, 455, 202, 220, 113, 278, 132, 461, 196, 114, 106, 41, 209, 79, 385, 446,
134, 59, 497, 179, 270, 16, 453, 285, 232, 426, 60, 372, 44, 139, 336, 15,
43, 164, 493, 120, 307, 437, 161, 383, 470, 4, 305, 238, 411, 241, 47, 52, 2,
216, 369, 316, 395, 173, 151, 150, 471, 299, 286, 311, 46, 253, 175, 110,
211, 82, 302, 181, 459, 210, 346, 130, 394, 341, 473, 8, 96, 143, 7, 510,
509, 197, 25, 412, 334, 474, 50, 317, 155, 239, 91, 39, 439, 323, 374, 10,
33, 131, 236, 335, 506, 281, 200, 424, 384, 165, 264, 263, 312, 199, 295,
468, 98, 122, 243, 499, 405, 492, 306, 503, 337, 409, 258, 501, 342, 99, 62,
73, 24, 17, 366, 193, 157, 252, 400, 227, 329, 375, 152, 483, 436, 386, 108,
13, 475, 142, 115, 125, 111, 121, 166, 224, 169, 255, 332, 126, 465, 63, 407,
231, 294, 276, 380, 389, 53, 207, 359, 287, 149, 261, 498, 90, 349, 29, 116,
421, 107, 371, 67, 64, 192, 404, 347, 212, 318, 38, 485, 68, 472, 249, 396,
399, 233, 229, 42, 177, 491, 496, 240, 248, 274, 288, 356, 72, 189, 415, 145,
419, 430, 153, 74, 235, 406, 284, 504, 257, 97, 431, 271, 119, 160, 260, 324,
208, 327, 40, 266, 31, 136, 477, 65, 234, 32, 117, 388, 310, 101, 435, 190,
322, 95, 174, 123, 195, 194, 423, 221, 351, 112, 330, 14, 66, 511, 273, 85,
94, 104, 361, 392, 488, 387, 146, 83, 275, 393, 418, 410, 254, 226, 133, 69,
89, 314, 23, 11, 358, 228, 129, 319, 494, 268, 417, 205, 454, 118, 162, 201,
355, 449, 36, 298, 167, 331, 338, 390, 242, 203, 413, 381, 93, 444, 218, 204,
365, 370, 460, 262, 168, 18, 315, 344, 402, 186, 237, 378, 163, 21, 301, 362,
272, 70, 429, 191, 87, 20, 484, 364, 296, 81, 368, 265, 456, 422, 416, 22,
340, 267, 495, 463, 105, 300, 154, 250, 219, 350, 462, 246, 148, 290, 76,
277, 466, 343, 507, 176, 34, 289, 30, 376, 448, 0, 147, 172, 428, 382, 247,
442, 280, 141, 84, 222, 282, 184, 188, 187, 377, 297, 27, 479, 124, 198, 457,
490, 37, 178, 440, 109, 309, 379, 102, 88, 225, 367, 447, 432, 48, 259, 158,
206, 28, 213, 244, 478, 35, 357, 256, 45, 75, 251, 304, 464, 354, 214, 170,
450, 215, 360]
eight = [233, 91, 39, 71, 32, 190, 13, 76, 141, 63, 0, 102, 80, 48, 74, 53,
130, 176, 27, 64, 177, 248, 152, 35, 214, 103, 108, 41, 72, 75, 220, 106,
251, 228, 69, 238, 224, 147, 1, 104, 82, 87, 169, 116, 66, 109, 61, 17, 195,
36, 14, 119, 110, 154, 189, 185, 23, 253, 243, 196, 85, 140, 15, 47, 137,
227, 20, 242, 175, 49, 247, 170, 167, 217, 200, 156, 123, 180, 210, 50, 107,
70, 131, 12, 58, 28, 78, 192, 136, 19, 226, 30, 193, 255, 221, 62, 183, 184,
```

```
187, 45, 90, 86, 117, 37, 54, 240, 142, 46, 222, 38, 250, 229, 44, 4, 205,
239, 115, 93, 55, 145, 225, 218, 153, 213, 244, 120, 219, 114, 201, 2, 60,
172, 168, 194, 232, 98, 43, 113, 111, 6, 124, 77, 73, 138, 95, 128, 181, 216,
59, 84, 235, 245, 252, 81, 92, 133, 125, 3, 231, 16, 144, 161, 22, 208, 139,
212, 94, 121, 99, 198, 199, 65, 151, 135, 204, 236, 215, 223, 197, 118, 52,
171, 122, 105, 163, 101, 207, 148, 25, 88, 182, 202, 188, 191, 186, 173, 246,
51, 7, 126, 155, 96, 5, 146, 68, 33, 241, 249, 56, 234, 57, 112, 230, 10, 29,
159, 178, 129, 150, 26, 40, 134, 160, 67, 209, 100, 211, 143, 157, 83, 11,
79, 89, 254, 18, 206, 237, 42, 97, 165, 158, 21, 149, 132, 162, 174, 24, 31,
203, 166, 179, 164, 9, 34, 127, 8]
seven = [48, 29, 17, 123, 45, 49, 7, 53, 120, 25, 20, 92, 51, 77, 16, 43, 97,
107, 65, 82, 67, 22, 79, 116, 68, 113, 32, 66, 124, 69, 84, 102, 37, 4, 31,
54, 34, 85, 44, 122, 52, 72, 111, 110, 63, 83, 28, 39, 59, 23, 40, 75, 70,
11, 6, 60, 61, 71, 38, 91, 73, 27, 1, 125, 10, 12, 21, 19, 0, 78, 112, 104,
118, 13, 18, 98, 88, 100, 76, 56, 117, 108, 101, 33, 2, 115, 109, 81, 57,
127, 5, 35, 46, 80, 106, 96, 58, 62, 47, 86, 114, 99, 30, 94, 42, 93, 15, 74,
126, 55, 24, 36, 87, 119, 121, 14, 90, 103, 89, 95, 9, 8, 3, 50, 41, 105, 64,
26]
tables = [eight,nine,ten,eleven,twevle,thirteen,fourteen,fifthteen]
correctHash = 186
trueCases  = 0
def hash8(message, table,mod):
    global trueCases
    hashed = len(message) % mod
    #print("Intial:")
    #print(hashed)
    for i in message:
        #print("Ord:")
        #print(ord(i))
        #print("postion :")
        #print( (hashed+ord(i))%256)
        #print("In loop hashed:")
        hashed = table[(hashed+ord(i)) % mod]
        #print(hashed)
    return hashed
from itertools import product
from string import ascii_uppercase
#userInput = input("Enter 4 Letter Word : ")

keywords = [''.join(i) for i in product(ascii_uppercase, repeat = 4)]
#print(len(keywords))
for table in tables:
    print("Size : ")
    print(len(table))
    correctHash = hash8("MATH",table,len(table))
    print("Value of Math: ")
    print(correctHash)
    print("Collisions :")
    trueCases = 0
```

```
    for word in keywords:
        output = hash8(word,table,len(table))
        if output == correctHash:
            trueCases = trueCases +1
        #byte = bytes(word,encoding='utf-8')
        #temp = hashlib.sha256(byte).hexdigest()
        #if temp == correctHashSha:
            #trueForSha = trueForSha + 1
    #print(trueCases)
    #print(len(keywords))
    #print(trueForSha)
    print(trueCases)
    print(" ------- ")
```

Creating Lookup Tables

```
from random import shuffle
import hashlib
from statistics import mean
import pickle
from itertools import product
from string import ascii_uppercase

#example_table = list(range(0,32768))
#shuffle(example_table)
#print(example_table)
#print(len(example_table))
#keywords = [''.join(i) for i in product(ascii_uppercase, repeat = 6)]
with open('outputfile.txt', 'w') as outfile:
    for combo in product(ascii_uppercase, repeat=6):
        outfile.write(''.join(combo) + '\n')
#keywords = []
#for item in product(ascii_uppercase, repeat=6):
    #keywords.append(item)
#with open('outfile.txt', 'wb') as fp:
    #pickle.dump(keywords, fp)
```

Pearson Hash Function

```
def hash8(message, table,mod):
    #global trueCases
    hashed = len(message) % mod
    #print("Intial:")
    #print(hashed)
```

```
    for i in message:
        #print("Ord:")
        #print(ord(i))
        #print("postion :")
        #print( (hashed+ord(i))%256)
        #print("In loop hashed:")
        hashed = table[(hashed+ord(i)) % mod]
        #print(hashed)
    return hashed
```