The Reason As To Why The Given Hash Function Fails

The given hash function is okay at best as it fails to account for the simple fact of position vs value. When dealing with the hash function, you will immediately recognize that the array positions don’t hold any weight. All positions of the strings are equally weighed, creating the issue where a string such as “ABC”, for example, would yield the same key as “BCA”. And while the hash function does utilize folding technique with division remainder, the lack of position weight from a string leaves for much to be desired, as collisions are all to common with this hash function.

# The Reason As To Why The Custom Hash Function Works

The custom hash Function relies on the same folding and remainder division technique to accomplish hash keys but at the start, all bytes of a given letter are modified by the bytes of the next letter. First, each byte of each letter is added by the next byte of the next letter, next the resulting byte is XOR’d, again by the next letters byte. This results in a unique byte for each letter of the word, unique to its position based on the next letter. I also extend the range of folding, so each word is more unique when it comes to generating its hash address. The results are instantly better than the original hash function, all the while retaining mostly the same methodology as the original hash function.

# Linear Probe At 50 Percent Fill Using Custom Hash Function

**E = ( 1 - α / 2 ) / ( 1 - α ) where α equals number of keys in the table divided by table size. At 50 percent fill, number of keys in the table are 64 and the size of the table is 128, this would make α equal 0.5 and E would equal 1.5 probes on average. The first 30 keys are instantly better using the custom hash function as the average probe drops from 7 to an average of 1. The last 30 keys do fall short a bit from the theoretical average probe but still out perform the original hash function as the average drops from 36 probes to 19 probes. Although it is worth mentioning that the minimum probes do drop from 16 to 0 going from the original hash function to the custom hash function.**

**Original Hash Function:**

First Thirty Probes :: Min=0 Max=26 Avg=7

Last Thirty Probes :: Min=16 Max=58 Avg=36

**Custom Hash Function:**

First Thirty Probes :: Min=0 Max=10 Avg=1

Last Thirty Probes :: Min=0 Max=61 Avg=19

# Linear Probe At 90 Percent Fill Using Custom Hash Function

**E = ( 1 - α / 2 ) / ( 1 - α ) where α equals number of keys in the table divided by table size. At 90 percent fill, number of keys in the table are 116 and the size of the table is 128, this would make α equal 0.9 and E would equal 5.5 probes on average. The first 30 keys are instantly better using the custom hash function as the average probe drops from 7 to an average of 1. The last 30 keys do fall short a bit from the theoretical average probe but still outperform the original hash function as the average drops from 54 probes to 43 probes. Although it is worth mentioning that the minimum probes do drop from 33 to 4 going from the original hash function to the custom hash function. One thing that I was unable to explain for the last 30 probes was why the max probe in the custom hash function always came out higher on the linear probe vs the original hash function but the min values and avg values showed significant improvements.**

**Original Hash Function:**

First Thirty Probes :: Min=0 Max=26 Avg=7

Last Thirty Probes :: Min=33 Max=74 Avg=54

**Custom Hash Function:**

First Thirty Probes :: Min=0 Max=10 Avg=1

Last Thirty Probes :: Min=4 Max=76 Avg=43

# Random Probe At 50 Percent Fill Using Custom Hash Function

**E = - ( 1 / α ) ln ( 1 - α ) ) where α equals number of keys in the table divided by table size. At 50 percent fill, number of keys in the table are 64 and the size of the table is 128, this would make α equal 0.5 and E would equal 1.39 probes on average. The custom hash function excels with the use of random probe. With the first 30 keys having 0 probes on average vs 1 probe on average with the original hash function. The last 30, while not below the theoretical value, still come close with an average of 2 probes, max out at 6 probes and having a minimum of 0 probes.**

**Original Hash Function:**

First Thirty Probes :: Min=0 Max=5 Avg=1

Last Thirty Probes :: Min=0 Max=13 Avg=4

**Custom Hash Function:**

First Thirty Probes :: Min=0 Max=4 Avg=0

Last Thirty Probes :: Min=0 Max=6 Avg=2

# Random Probe At 90 Percent Fill Using Custom Hash Function

**E = - (1 / α ) ln ( 1 - α ) ) where α equals number of keys in the table divided by table size. At 90 percent fill, number of keys in the table are 116 and the size of the table is 128, this would make α equal 0.9 and E would equal 2.56 probes on average. At 90 percent fill, the custom hash function only manages to beat the theoretical expected probes, once more only on the first 30 keys. Last 30 keys fail to reach the milestone and barely manages to get an edge on the on the original hash function. This can be attributed to the inherent properties of the random probe and how its efficiency decreases as the hash table approaches capacity, meaning fewer empty spaces for the random probe to randomly run into, increasing probe count.**

**Original Hash Function:**

First Thirty Probes :: Min=0 Max=5 Avg=1

Last Thirty Probes :: Min=0 Max=30 Avg=8

**Custom Hash Function:**

First Thirty Probes :: Min=0 Max=4 Avg=0

Last Thirty Probes :: Min=0 Max=23 Avg=7