# DAT470/DIT065 Assignment 5

Mirco Ghadri
mircog@chalmers.se

May 13, 2024

In this assignment, we will implement tabulation hashing, the HyperLogLog algorithm[1], and have a look at some of their properties.

## Problem 1

*(a) Implement tabulation hashing to hash unsigned 64-bit integer keys into 32-bit unsigned hash values. Utilize a 16-bit alphabet and adhere to the interface provided in the file `tabulation_hash.py`. Ensure consistent behavior of your implementation by using a fixed random seed, guaranteeing the same results upon construction. Utilize bitwise operations exclusively; refrain from converting integers into strings or lists.* (6 pts)
**Answer:** See `assignment5_problem1.py`.

*(b) Utilize the implemented tabulation hashing to hash integers from 1 to 1,000,000. Plot a histogram depicting the distribution of hash values.* (2 pts)
**Answer:** We created a plot in python using pyplot hist() function.

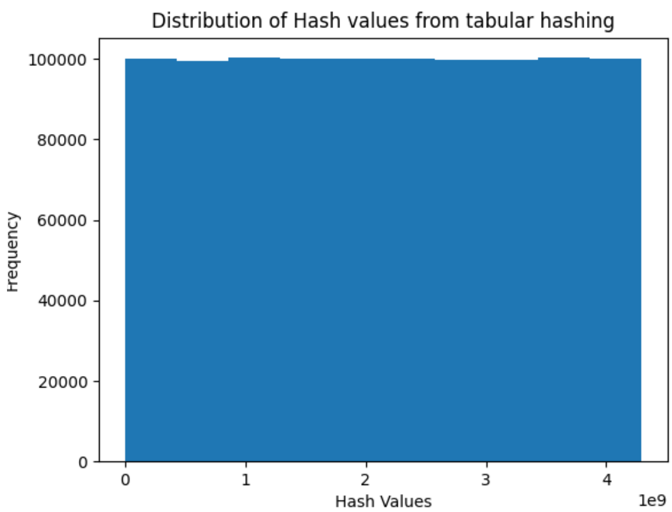


Figure 1: Distribution of 1 million 32 bit unsigned Hash Values

From the figure we can see that the hash values range from 0 to $4 * 10^9$. This makes sense we use 32 bit unsigned integers as hash values, which can take on a value at most $2^{32} \approx 4.3 * 10^9$. We can see that the hash values are evenly distributed. This means that is is equally likely to find a hash value that is within any given range within 0 to $4 * 10^9$. In other words, our hash function is uniformly random.

# Problem 2

**(a)** Implement the function $\rho$ using bitwise operations. (2 pts)
**Answer:** See `assignment5_problem2.py`.

**(b)** Implement the HyperLogLog algorithm. Utilize the interface provided in the file `hyperloglog.py`. Ensure that the memory usage for registers does not exceed the allocation specified in the interface file (i.e., for $m$ registers, use at most $m$ bytes). Employ the tabulation hash implemented in Problem 1 as the hash function $h$, passing the seed value to its constructor. For the hash function $f$, use a multiply-shift function with the constant parameter $a = \text{0xc863b1e7d63f37a3}_{16}$. (8 pts)
**Answer:** See `assignment5_problem2.py`

**(c)** Construct a sketch with $m = 1024$ registers, and input the integers $1, 2, \ldots, 10^6$ into it. Report the estimate obtained. (2 pts)
**Answer:** To construct a sketch with 1024 registers and feed it the values and calculate the cardinality estimate for the number of values in the sketch, we used the code:

```
hyper_log_log_sketch = HyperLogLog(m, seed)
for i in range(1,1000001):
    hyper_log_log_sketch(i)
print(hyper_log_log_sketch.estimate())
```

When we printed the cardinality estimate, we got the value: -

**(d)** Choose two other values of $m$ besides $m = 1024$, and repeat the process 1000 times with different seeds for generating the data: construct ten thousand ($10^4$) random integers, i.e., estimate the cardinality of $n = 10^4$. Produce a table where each row lists the present value of $m$, the average cardinality estimate, and the fraction of runs where the estimate was within $n(1 \pm \sigma)$ and within $n(1 \pm 2\sigma)$, where $\sigma = \frac{1.04}{\sqrt{m}}$. Thus, the table should have three rows (one for each value of $m$) and four columns. (4 pts)
**Answer:** We used the following values for m (all values were a power of 2):

| $m$ |
|---|
| 1024 |
| 2048 |
| 4096 |

We got the following table of results when running our experiments 1000 times(each time with a different seed) for each value of m.

Table 1: Cardinality Estimation Results

| $m$ | Average Estimate | $n(1 \pm \sigma)$ | $n(1 \pm 2\sigma)$ |
|---|---|---|---|
| 1024 | - | - | - |
| 2048 | - | - | - |
| 4096 | - | - | - |

# References


[1] Philippe Flajolet et al. "HyperLogLog: The analysis of a near-optimal cardinality estimation algorithm". In: *Proceedings of the 2007 Conference on Analysis of Algorithms (AofA 07)*. 2007.