

1 Hash Performance Test

Two hashing functions are described in Section ?? and Section ?. In order to decide which of these I would use in the algorithm, a few tests were performed in order to determine the speed of each.

To perform these tests, a short Java program was written. It randomized a given number of k -mer transformations as described in Section ?, and then counted the number of nanoseconds it took the Carter Wegman- and multiply-shift hashing schemes to hash all the transformations. The multiply-shift algorithm was set to shift only if the input was longer than 32 bits, in which case the shift was $2 \cdot k - 32$.

k	10		20		30	
H	Mult.shift	Carter	Mult.shift	Carter	Mult.shift	Carter
1 mio.	10.3 \pm 0.2	11.9 \pm 0.5	10.3 \pm 0.1	11.6 \pm 0.1	10.5 \pm 0.1	11.6 \pm 0.1
10 mio.	40.9 \pm 0.5	42.1 \pm 0.2	41.3 \pm 0.1	42.1 \pm 0.4	41.8 \pm 1.6	42.4 \pm 0.5
50 mio.	175.5 \pm 3.4	187.2 \pm 10.6	177.5 \pm 2.1	181.6 \pm 3.8	176.2 \pm 1.3	179.7 \pm 1.7
100 mio.	343.0 \pm 1.4	377.5 \pm 14.0	341.6 \pm 1.0	364.1 \pm 12.6	346.7 \pm 6.9	355.8 \pm 3.5

Table 1: Time in ms for multiply-shift- and Carter Wegman hashing schemes to 3 different which are $2 \cdot k$ bit long randomized k -mer transformations, run at four different hash function sizes H . Three tests were performed for each combination of k and H to obtain the error of the measures.

The results of the tests can be seen in Table 1. The Carter Wegman implementation is consequently slower than multiply-shift, in addition to having larger margin of error. We may note that the difference in speed is most notable at $k = 10$, diminishing at higher k . The differences in speed are however less than 50 ms, even at 100 mio. input hash values. Also, as the time increases linearly with the number of input values, so will the difference in time.

As the difference in speed is quite minimal, the method used in the algorithm will be the Carter Wegman scheme, as it is easier to implement for input of such different sizes as k -mers.