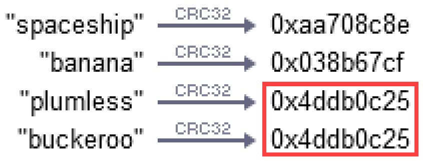
May 04, 2011

# Hash Collision Probabilities

A hash function takes an item of a given type and generates an integer hash value within a given range. The input items can be anything: strings, compiled shader programs, files, even directories. The same input always generates the same hash value, and a good hash function tends to generate different hash values when given different inputs.

A hash function has no awareness of "other" items in the set of inputs. It just performs some arithmetic and/or bit-magic operations on the input item passed to it. Therefore, there's always a chance that two different inputs will generate the same hash value.

Take the well-known hash function CRC32, for example. If you feed this function the two strings "plumless" and "buckeroo", it generates the same value. This is known as a hash collision.


"spaceship" CRC32 0xaa708c8e
"banana" CRC32 0x038b67cf
"plumless" CRC32 0x4ddb0c25
"buckeroo" CRC32 0x4ddb0c25

What is the probability of a hash collision? This question is just a general form of the birthday problem from mathematics. The answer is not always intuitive, so it's difficult to guess correctly. Let's derive the math and try to get a better feel for those probabilities.

## Calculating the Probability of a Hash Collision

There are many choices of hash function, and the creation of a good hash function is still an active area of research. Some hash functions are fast; others are slow. Some distribute hash values evenly across the available range; others don't. If you're interested in the real-world performance of a few known hash functions, Charles Bloom and strchr.com offer some comparisons. For our purposes, let's assume the hash function is pretty good — it distributes hash values evenly across the available range.

In this case, generating hash values for a collection of inputs is a lot like generating a collection of random numbers. Our question, then, translates into the following:

> Given $k$ randomly generated values, where each value is a non-negative integer less than $N$, what is the probability that at least two of them are equal?

It turns out it's actually a bit simpler to start with the reverse question: What is the probability that they are all unique? Whatever the answer to the reverse question, we can just subtract it from one, and we'll have the answer to our original question.

Given a space of $N$ possible hash values, suppose you've already picked a single value. After that, there are $N-1$ remaining values (out of a possible $N$) that are unique from the first. Therefore, the probability of randomly generating two integers that are unique from each other is $\frac{N-1}{N}$.

After that, there are $N-2$ remaining values (out of a possible $N$) that are unique from the first two, which means that the probability of randomly generating three integers that are all unique is $\frac{N-1}{N} \times \frac{N-2}{N}$. (We can multiply the probabilities together because each random number generation is an independent event.)

In general, the probability of randomly generating $k$ integers that are all unique is:

$$\frac{N-1}{N} \times \frac{N-2}{N} \times \cdots \times \frac{N-(k-2)}{N} \times \frac{N-(k-1)}{N}$$

On a computer, this can be quite slow to evaluate for large k. Luckily, the above expression is approximately equal to:

$$e^{\frac{-k(k-1)}{2N}}$$

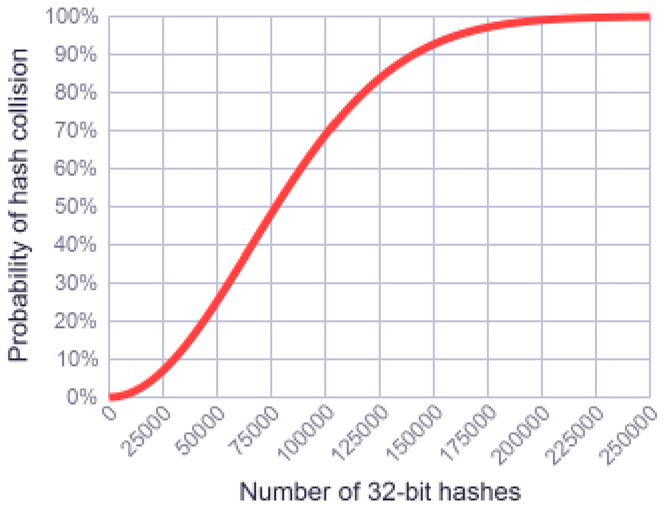
which is a lot faster to compute. How do we know this is a good approximation? Well, it can be shown analytically, using the Taylor expansion of $e^x$ and an epsilon-delta proof, that the approximation error tends to zero as $N$ increases. Or, you can just compute both values and compare them. Run the following Python script with different $N$, and you'll get a feeling for just how accurate the approximation is:

```
import math
N = 1000000
probUnique = 1.0
for k in xrange(1, 2000):
    probUnique = probUnique * (N - (k - 1)) / N
    print k, 1 - probUnique, 1 - math.exp(-0.5 * k * (k - 1) / N)
```

Great, so this magic expression serves as our probability that all values are unique. Subtract it from one, and you have the probability of a hash collision:

$$1 - e^{\frac{-k(k-1)}{2N}}$$

Here is a graph for $N = 2^{32}$. This illustrates the probability of collision when using 32-bit hash values. It's worth noting that a 50% chance of collision occurs when the number of hashes is 77163. Also note that the graph takes the same S-curved shape for any value of $N$.



## Simplified Approximations

It’s interesting that our approximation takes the form $1 - e^{-X}$, because it just so happens that for any $X$ that is very small, say $\frac{1}{10}$ or less:

$$1 - e^{-X} \approx X$$

In other words, the exponent makes a pretty good approximation all by itself! In fact, the smaller the $X$, the more accurate it gets. So for small collision probabilities, we can use the simplified expression:

$$\frac{k(k-1)}{2N}$$

This is actually a handy representation, because it avoids some numerical precision problems in the original expression. Floating point numbers are not very good at representing values extremely close to 1.

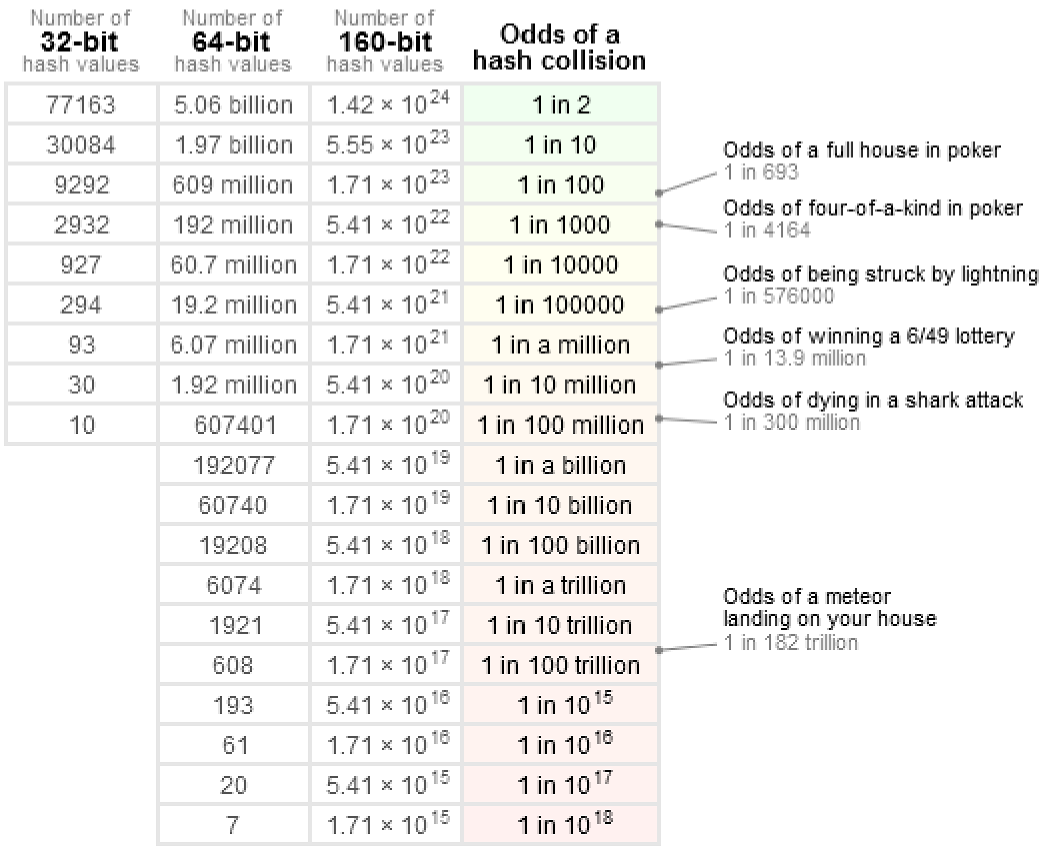
Furthermore, if you’re talking about more than a handful of $k$, there isn’t a very big difference between $k(k-1)$ and $k^2$. So the absolute simplest approximation is just:

$$\frac{k^2}{2N}$$

## Small Collision Probabilities

In certain applications — such as when using hash values as IDs — it can be very important to avoid collisions. That’s why the most interesting probabilities are the small ones.

Assuming your hash values are 32-bit, 64-bit or 160-bit, the following table contains a range of small probabilities. If you know the number of hash values, simply find the nearest matching row. To help put the numbers in perspective, I’ve included a few real-world probabilities scraped from the web, like the odds of winning the lottery.



| Number of **32-bit** hash values | Number of **64-bit** hash values | Number of **160-bit** hash values | **Odds of a hash collision** |
|---|---|---|---|
| 77163 | 5.06 billion | $1.42 \times 10^{24}$ | 1 in 2 |
| 30084 | 1.97 billion | $5.55 \times 10^{23}$ | 1 in 10 |
| 9292 | 609 million | $1.71 \times 10^{23}$ | 1 in 100 |
| 2932 | 192 million | $5.41 \times 10^{22}$ | 1 in 1000 |
| 927 | 60.7 million | $1.71 \times 10^{22}$ | 1 in 10000 |
| 294 | 19.2 million | $5.41 \times 10^{21}$ | 1 in 100000 |
| 93 | 6.07 million | $1.71 \times 10^{21}$ | 1 in a million |
| 30 | 1.92 million | $5.41 \times 10^{20}$ | 1 in 10 million |
| 10 | 607401 | $1.71 \times 10^{20}$ | 1 in 100 million |
| | 192077 | $5.41 \times 10^{19}$ | 1 in a billion |
| | 60740 | $1.71 \times 10^{19}$ | 1 in 10 billion |
| | 19208 | $5.41 \times 10^{18}$ | 1 in 100 billion |
| | 6074 | $1.71 \times 10^{18}$ | 1 in a trillion |
| | 1921 | $5.41 \times 10^{17}$ | 1 in 10 trillion |
| | 608 | $1.71 \times 10^{17}$ | 1 in 100 trillion |
| | 193 | $5.41 \times 10^{16}$ | 1 in $10^{15}$ |
| | 61 | $1.71 \times 10^{16}$ | 1 in $10^{16}$ |
| | 20 | $5.41 \times 10^{15}$ | 1 in $10^{17}$ |
| | 7 | $1.71 \times 10^{15}$ | 1 in $10^{18}$ |

Odds of a full house in poker
1 in 693

Odds of four-of-a-kind in poker
1 in 4164

Odds of being struck by lightning
1 in 576000

Odds of winning a 6/49 lottery
1 in 13.9 million

Odds of dying in a shark attack
1 in 300 million

Odds of a meteor landing on your house
1 in 182 trillion

[Hash Table Performance Tests »]

## Comments (66)

**Commenting Disabled**
Further commenting on this page has been disabled by the blog admin.

RCL · *544 weeks ago*

He is Charles, not Chris (Bloom) :) Thanks for great article, anyway!

Reply **1 reply** · *active 463 weeks ago*

Jeff Preshing · *544 weeks ago*

Man, that's twice I did that! Good thing so many people out there know his name.

Reply

João Salada · *532 weeks ago*

Great Article! do you know any good paper that contains what you've just posted?

Reply **2 replies** · *active 463 weeks ago*

Jeff Preshing · *532 weeks ago*

The math is also derived in the Wikipedia article I linked. You'll find additional references there.