What are the best *k* hashes and *m* bits values to store one million *n* keys (E.g. e52f43cd2c23bb2e6296153748382764) suppose we use the same MD5 hash key from pickle_hash.py and explain why?

I think it up to your acceptance of the probability of false positivity and the space efficiency.
In ideal assumption, P=0 will never create false positivity, but it needs infinity k hashes and infinity m bits values. (The more k hashes, the less m bits values, and vice versa)
Based on the formula from https://www.geeksforgeeks.org/bloom-filters-introduction-and-python-implementation/,

> **Probability of False positivity:** Let **m** be the size of bit array, k be the number of hash functions and **n** be the number of expected elements to be inserted in the filter, then the probability of false positive **p** can be calculated as:

$$P = \left(1 - \left[1 - \frac{1}{m}\right]^{kn}\right)^{k}$$

we could set P lim to 0, n as 1000000, then see whether there are constant numbers for k and m. (k, m > 0)

wolframalpha.com/input/?i=0%3D%281-%5B1-%281%2Fm%29%5E%28k*1000000%29%5D%29%5Ek

YEEZY SUPPLY | kdlasbang | 光影笔墨的个人空... | Instacart - Instacar... | Inventory issues -... | 订单详情-用户中心... | Ma

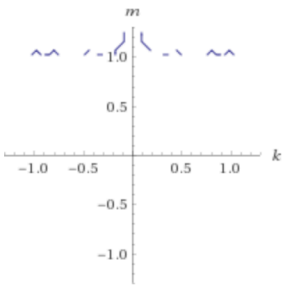
Input:

$$0 = \left(1 - \left(1 - \left(\frac{1}{m}\right)^{k \times 1000\,000}\right)\right)^{k}$$

Result:

$$0 = \left(\left(\frac{1}{m}\right)^{1000\,000\,k}\right)^{k}$$

Implicit plot:



Alternate form assuming k and m are positive:

$$0 = m^{-1000\,000\,k^2}$$

Solutions:

(no solutions exist)

k is in inverse ratio to/with m.

If we set k = 1, 2, 3…

0=(1-[1-(1/m)^(1*1000000)])^1

Extended Keyboard | Upload | Examples | Random

Assuming "m" is a variable | Use as a unit instead

Input:

$$0 = \left(1 - \left(1 - \left(\frac{1}{m}\right)^{1 \times 1000\,000}\right)\right)^{1}$$

Result:

$$0 = \frac{1}{m^{1000\,000}}$$

Alternate form:

False

Solutions:

Enlarge | Customize | Plain Text

(no solutions exist)

Download Page

POWERED BY THE WOLFRAM LANGUAGE

0=(1-[1-(1/m)^(2*1000000)])^2

Extended Keyboard | Upload | Examples | Random

Assuming "m" is a variable | Use as a unit instead

Input:

$$0 = \left(1 - \left(1 - \left(\frac{1}{m}\right)^{2\times 1000\,000}\right)\right)^2$$

Result:

$$0 = \frac{1}{m^{4000\,000}}$$

Alternate form:

False

Solutions: Enlarge | Customize | Plain Text

(no solutions exist)

Download Page

POWERED BY THE **WOLFRAM LANGUAGE**

0=(1-[1-(1/m)^(3*1000000)])^3

Extended Keyboard | Upload | Examples | Random

Assuming "m" is a variable | Use as a unit instead

Input:

$$0 = \left(1 - \left(1 - \left(\frac{1}{m}\right)^{3\times 1000\,000}\right)\right)^3$$

Result:

$$0 = \frac{1}{m^{9000\,000}}$$

Alternate form: Enlarge | Customize | Plain Text

False

Solutions: Step-by-step solution

(no solutions exist)

Download Page

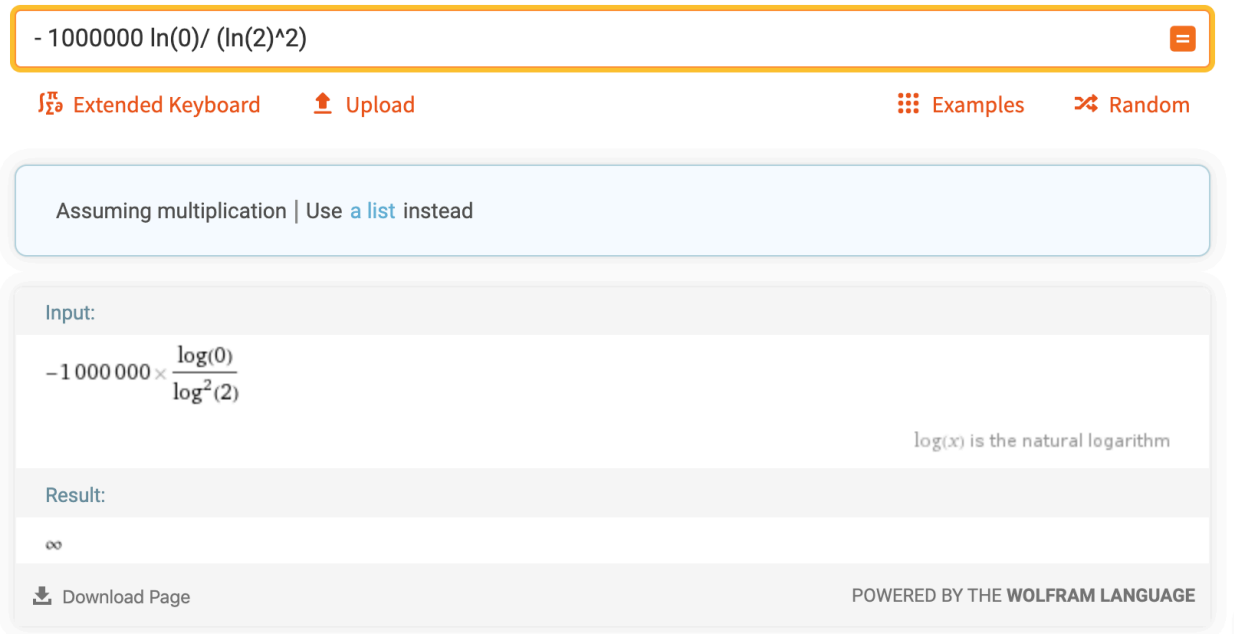
POWERED BY THE **WOLFRAM LANGUAGE**

We can find that the more k hashes we have, the less m bits values we will need to achieve (P=0).

Then, let choose the second formula, and set P lim to 0, n = 1000000, to find what will be m bit value is.

> **Size of Bit Array:** If expected number of elements **n** is known and desired false positive probability is **p** then the size of bit array **m** can be calculated as :

$$m = -\frac{n \ln P}{(ln2)^2}$$

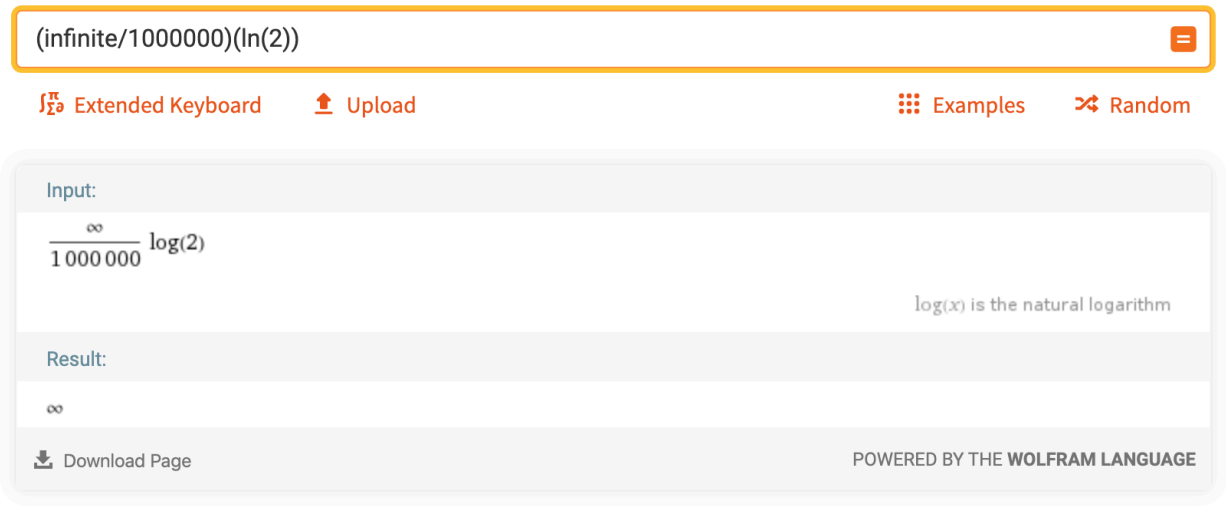


Then we get m -> infinity

Moreover, if we input m= infinity, n=1000000 into third formula to get k, then we have:

> **Optimum number of hash functions:** The number of hash functions **k** must be a positive integer. If **m** is size of bit array and **n** is number of elements to be inserted, then k can be calculated as :

$$k = \frac{m}{n} ln2$$



So in ideal situation, if we have infinity k hashes and infinity m bit values, then we won't have false positivity.

In practical assumption, -> P as small as possible(1%,0.1%,0.01%…) and consider our space/memory practically.
Below are some discussions I picked on stack overflow:

"Case 2 : I use a bloom filter. The entire list of 1 million URLs are run through the bloom filter using multiple hash functions and the respective positions are marked as 1, in a huge array of 0s. Let's say we want a false positive rate of 1%, using a bloom filter calculator (http://hur.st/bloomfilter?n=1000000&p=0.01) , we get the size of the bloom filter required as only 1.13 MB. This small size is expected as, even though the size of the array is huge, we are only storing 1s or 0s and not the URLs as in case of the hash table.This array can be treated as a bit array. That is, since we have only two values 1 and 0, we can set individual bits instead of bytes. This would reduce the space taken by 8 times. This 1.13 MB bloom filter, due to its small size, can be stored in the web browser itself !! Thus when a user comes along and enters a URL, we simply apply the required hash functions (in the browser itself), and check all the positions in the bloom filter (which is stored in the browser). A value of 0 in any of the positions tells us that this URL is DEFINITELY NOT in the list of malicious URLs and the user can proceed freely. Thus we did not make a call to the server and hence saved time. A value of 1 tells us that the URL MIGHT be in the list of malicious URLs. In these cases we make a call to the remote server and over there we can use some other hash function with some hash table as in the first case to retrieve and check if the URL is actually present. Since most of the times, a URL is not likely to be a malicious one, the small bloom filter in the browser figures that out and hence saves time by avoiding calls to the remote server. Only in some cases, if the bloom filter tells us that the URL MIGHT be malicious, only in those cases we make a call to the server. That 'MIGHT' is 99% right.

So by using a small bloom filter in the browser, we have saved a lot of time as we do not need to make server calls for every URL entered."

"But wait a minute: we already know a data structure that can answer all of this without vague 'possible' and also without all of the limitations (can't remove, can't show all). And it is called a set. And here comes a main advantage of a bloom filter: it is **space efficient and space constant**.
This means that it does not matter how many elements do we store there, the space will be the same. Yes a bloom filter with 10^6 elements (useless bloom filter) will take the same amount of space as a bloom filter with 10^20 elements and the same space as bloom filter with 0 elements. So how much space will it take? It is up to you to decide (but there is a trade of: the more elements you have the more uncertain you are with you possible in the set answer.

Another cool thing is that it is space constant. When you save the data to a set, you have to actually save this data. So if you store this long string in the set you have to at least use 27 bytes of space. But for a 1% error and an optimal value of k **, you will need ~ 9.6 bits (< 2 bytes) per any element (whether it is a short int or a huge wall of text).

Another property is that all the operations are taking constant time, which is absolutely not the same as amortized constant time in case of sets (remember that if the set has collisions, it can deteriorate in O(n) time)."

More Discussion Regarding to this question & The Reference:

https://www.geeksforgeeks.org/bloom-filters-introduction-and-python-implementation/
https://stackoverflow.com/questions/4282375/what-is-the-advantage-to-using-bloom-filters