# Using Hashing for Large Sets of Large Objects with Efficient Membership

Ernie Davis

## Technique 1

Suppose that you have a large set of large objects; let's say a million high-quality images, each 3 MBytes. You want to be able to check efficiently whether a new image is already in your collection, and you don't have 3 TBytes of RAM. You can do this as follows:

Hash each image to a 64 bit key. Store the 1,000,000 keys in a hash table of size 1,000,000. This hash table is the representation of $S$, the set of all images. The total space required is 8 MBytes,

Note that there are two hash functions here: $h_1(X)$ maps a 3MByte image $X$ to a 64 bit key (i.e. an integer between 0 and $2^{64} - 1$) and $h_2(K)$ maps a 64 bit key $K$ to a hash-table index between 0 and 999,999.

To check whether a new image $Y$ is already in your set, compute $K = h_1(Y)$, and then compute $h_2(K)$ to look up $K$ in your hash table. If $Y$ is in $S$, then $K = h_1(Y)$ is in the table, so the algorithm returns TRUE. If $Y$ is not in $S$, then since there are only 1,000,000 $= 2^{20}$ different keys in the table, and there are $2^{64}$ different possible 64-bit keys, the probability that a random object outside $S$ happens to hash to one of the keys is $2^{20}/2^{64} = 2^{-44}$, much smaller than the probability that your computer will crash while you are doing the computation.

Most of the running time is spent computing $h_1(Y)$, but that is just linear in $|Y|$; you're not going to do better than that.

To recapitulate and generalize. If there are $N$ items, each of size $Q$ and you used $K$-bit keys where $K \gg \lg(N)$:

- The space required is $N \cdot K$ bits.
- The time required to create the data structure is $\Theta(N \cdot Q)$. The time required to check an object is $\Theta(Q)$.
- The probability of a false negative (rejecting a true element of the set) is 0. The probability of a false positive (validating a non-element) is $N/2^K$.

Note that the size of the objects only enters into the running time and not into the space or the probability.

## Technique 2: Bloom Filters

If you can live with a (much) higher false positive rate, then you can get a (substantially) higher degree of compression. For instance, suppose that you can only afford 2 Bytes per item rather than 8. Modifying Technique 1 to use 16-bit keys rather than 64-bit keys won't work. $2^{16} = 65,536 \ll 1,000,000$, so you would end up with all keys being taken 15 times, and nothing ever being rejected.

Instead you proceed as follows: Let $A$ be a bit vector of size 16,000,000; thus 2 MBytes. Construct 8 different independent hash functions $h_1(X) \ldots h_8(X)$. Initialize the bit vector as follows:

$A \leftarrow 0$;
for $(X \in S)$
    for $(i \leftarrow 1 \ldots 8)$
        $A[h_i(X)] = 1$;

The result is the representation of the set $S$. Note that the assignment statement in the loop is executed 8,000,000 times; hence at most half the values in $A$ are 0.

Now to check whether an item $Y$ is in the set, just evaluate:

$A[h_1(Y)]$ AND $A[h_2(Y)]$ AND $\ldots$ AND $A[h_8(Y)]$

If $Y \in S$ then this returns TRUE. If $Y \notin S$ then each test has probability less than $1/2$ of succeeding by chance, and therefore the test overall has a probability less than $1/2^8$ of succeeding.

To generalize: If you have $N$ objects and use $K$ hash functions then:

- The space required is $2 \cdot K \cdot N$ bits.
- The time required to initialize the bit vector is $\Theta(K \cdot N \cdot Q)$.
- The probability of a false negative is 0. The probability of a false positive is less than $2^{-K}$.

Actually, you can do a little better, because of collisions. If you use $K$ hash functions and a table of size $\alpha KN$, then the probability that A[1] is empty is $(1 - 1/\alpha KN)^{KN} = e^{-1/\alpha}$. (For $\alpha = 2$, this value is 0.6.) This same number is the expected fraction of empty slots in the table. You optimize the trade off of probability vs. space if the table is half full, so you want to choose $\alpha = \log_2(e) = 1.4427$. So with this tuning:

- The space required is $1.4427 \cdot K \cdot N$ bits; 11.5 MBytes in our example.
- The time required to initialize the bit vector is $\Theta(K \cdot N \cdot Q)$.
- The probability of a false negative is 0. The probability of a false positive is $2^{-K}$.