1 Balls, Bins and Random Graphs: Hashing bit strings

Most of us faced the scenario that when we create a account on a some website it says "your password is weak, please try some other password". So why it happens? This happens because it has been observed that most people used same common words and phrases as their password. So it becomes easy to crack such passwords. So in order to avoid such scenarios we are asked to enter a complicated password which is not a common .So let us see what exactly happens there.

Let's say we have a set of commonly used passwords as S, such that
$S = \{s_1, s_2, s_3, ..., s_m\}$ where $s_1, s_2, s_3, ..., s_m$ are the commonly used passwords. Now given a random password x, find if $x \in S$ or $x \ni S$. (Note: x is a big string and there are huge number of commonly used password) What should we do?
One method is to store S in a sorted order and whenever x comes perform a binary search. The time complexity of this will be $O(\log(n))$.
Another method is to use hashing. Basically, hash the passwords in S into a hash table and when x comes perform a lookup. The time complexity of this will be the size of the max linked list which will again be of $O(\log(n))$.
So can we do better?

2 Fingerprinting Method

Now let's say there are 2^b slots in the hash table. This hash table contain all the m commonly used passwords. Now when a random string x comes it can be hashed to one of the slots that have some sting in that or to an empty slot. When it is hashed to an empty slot we will allow that string to be used as passwords otherwise not. Now let us analyse this algorithm.

When x is hashed to an empty slot we can say for sure that x does not belong to S as if it does then there should be some not acceptable password s_i that should already be present in the slot where x is hashed. But if it is hashed to a slot that already have some non acceptable password then it may or may not belong to S. So it can give us false positive(ie saying x is a non acceptable password while actually it is acceptable) which is alright as this just makes our algorithm overly conservative.

Now next question is how large our hash table should be such that there are acceptable number of false positive? Basically we have to find an appropriate value of b so that there are not too many false positive. For that let us see the probability of acceptable password to be false positive(let us denote it with P). The probability that an acceptable password has hashed to a slot already containing a particular not allowable password is $(1 - 1/2^b)$. There are are m such passwords. So the probability that it will hash to none of them is $(1 - 1/2^b)^m$. So value of P will be,

$P = 1 - (1 - 1/2^b)^m \geq 1 - \exp^{-m/2^b}$

Now let's say we want this probability to be less then equal to c,ie

$P \leq c$
$1 - \exp^{-m/2^b} \leq c$

$\exp^{-m/2^b} \geq 1 - c$

$-m/2^b \geq \ln(1 - c)$

$2^b \geq m/\ln(1/(1 - c))$

$b \geq \log_2(m/\ln(1/(1 - c)))$, where $\ln(1/(1 - c))$ is a constant,

Therefore the value of $b = \Omega(\log_2(m))$. Now we will use $b = 2\log_2(m)$ and see the value P. The P will become,

$P = 1 - (1 - 1/m^2)^m$
$= 1 - \exp(-1/m)$
$= 1 - (1 - 1/m - 1/(2!m^2) + ...)$
$\leq 1/m$

Hence if we choose $b = 2\log_2(m)$ then there is very less probability of we having an acceptable answer as false positive. Also as m tends to infinity the probability of having an acceptable string as false positive tends to zero.

An alternative proof for the same can be as follows. We have m passwords in S which we are not allowed to use. Now when we hash these passwords the total number of slots that are occupied by these is less than equal to m. So the probability that hash function hash our password x to already occupied slot is less than equal to $m/2^b$, ie

$P \leq m/2^b$

Again let's say we want this probability to be less then equal to c,

$P \leq c$

$m/2^b \leq c$

$2^b \geq m/c$

$b \geq \log_2(m/c)$,

Again the value of $b = \Omega(\log_2(m))$. Now for various values of b the probability P will be,
for $b = 2\log_2 m, P \leq 1/m$

for $b = 3\log_2 m, P \leq 1/m^2$

and so on. Basically the probability decreases exponentially as we increase the number of bits.