Bloom filters vs. space-efficient perfect hash

Recall that classic Bloom filters use roughly $1.44log_2(1/f)$ bits per key, as seen in class (where $f = (1-p)^k$ is the failure probability minimized for $p \approx e^{-\frac{kn}{m}=1/2}$). The problem asks to extend the implementation required in Problem 10 by employing an additional random universal hash function $s : U \to [m]$ with $m = \lceil \frac{1}{f} \rceil$, called signature, so that s(x) is also stored (in place of x, which is discarded). The resulting space-efficient perfect hash table T has now a one-side error with failure probability of roughly f, as in Bloom filters: say why. Design a space-efficient efficient implementation of T, and compare the number of bits per key required by T with that required by Bloom filters.

SOLUTION

Now the data structure has got one-side error with probability because if we use the hash table s, which has got m elements, the probability of a collision is $\frac{1}{m}$. Since, $m = \frac{1}{f}$ we have: $Pr[error] = \frac{1}{\frac{1}{f}} = f$.

We consider first of all to have the same data structure of EX10. Therefore, we have H, T, A, B, P, and then till now if we have n keys we occupy a space equal to $18n + o(n)$ bits. Now we add another data structure T' where we store the hash generated by s. T' has got n slots, as the number of keys, in which every element occupy $log_2(m) = log_2(1/f)$ bits, since $s : U \to [m]$ with $m = \lceil \frac{1}{f} \rceil$. Now, since this data structure is static, ones we build the H, T, A, B, P we can insert each $s(k)$ in T' in the same position as it is in T. For example, the first element of bucket 1 (if it exist) is going to be the fist element of T', and so on. Then now to find the position of the hash in T' given a key k we use $rank_1(\text{BASE} + \text{OFFSET})$, where BASE + OFFSET is the position calculate to check the if there is a 1 in T ($T[\text{BASE} + \text{OFFSET}]$, look EX10).

Now we need to check how many bits are used for each key. T' uses $log_2(1/f)$ bit per key as explained before, and, by the previous analysis, we have 18 bit for each key + o(1)($18n + o(n)$). Therefore, we have roughly $log_2(1/f) + 18$ per key, instead Bloom filters use roughly $1.44log_2(1/f)$. Let's compare them, to see which is more convenient:

$$\begin{aligned} log_2(1/f) + 18 &< 1.44log_2(1/f) \\ -log_2(f) + 18 &< -1.44log_2(f) \\ -log_2(f) + 1.44log_2(f) &< -18) \\ log_2(f) &< -\frac{18}{0.44}) \\ f &< \frac{1}{2^{\frac{18}{0.44}}} \end{aligned}$$

Then perfect hash is convenient for f less than $\approx 4.84 \times 10^{-13}$.