6.897 Advanced Data Structures (Spring'05)

Prof. Erik Demaine TA: Mihai Pătraşcu

Problem 1 – Solution

Proof of the lemma. Let's compute the probability that there exists a cycle of length t. There are $\frac{1}{t}n(n-1)(n-2)\cdots \le n^t$ choices for the edges on the cycle, in order — remember that an edge is an element from S; the $\frac{1}{t}$ factor comes from the symmetry of choosing a cycle. The vertices on the cycle are chosen from $\{1,2,\ldots,cn\}$, so there are $(cn)^t$ choices. Note that only t vertices determine t edges, because consecutive edges share a vertex.

Now, for a t-subset of S, the probability that h_1 and h_2 will map the subset to precisely the cycle is $2^t(cn)^{-2t}$, because we choose a uniformly random edge for everything in the subset. Be careful not to forget the factor of 2^t: each edge has two ways of having the desired end-points (we want $\{h_1(x), h_2(x)\} = \{a, b\}$, which can achieved either by $h_1(x) = a, h_2(x) = b$ or $h_1(x) = b, h_2(x) = a$). In total, the probability is $\le n^t(cn)^t 2^t(cn)^{-2t} = (2/c)^t$.

Summing over all t, the probability that there exist any cycles is $\le \sum_{t=1}^{\infty}(2/c)^t$. The series converges for $c > 2$, and for sufficiently large c, it is at most $\frac{1}{2}$.

Construction of the Bloomier filter. Let $T[1\,..\,cn]$ be an array of r-bit elements. It is easy to pack the array into $O(nr)$ bits of space. A query to some value x simply returns $T[h_1(x)] \oplus T[h_2(x)]$, where $\oplus$ is bitwise xor.

Why can this work? The above essentially shows that we can find an injective mapping (a matching) of S into T, where x can either be mapped to $T[h_1(x)]$ or $T[h_2(x)]$. Say that x "owns" the position of T it is mapped to. Now assume x owns $T[h_1(x)]$. Due to other elements, $T[h_2(x)]$ might have to be fixed to some arbitrary r-bit value. But we can always fix $T[h_1(x)]$ (which x owns, so nobody else will fix), to make $T[h_1(x)] \oplus T[h_2(x)]$ equal to whatever value we want. The only trouble with this argument is cycles, but we already know that cycles don't appear with constant probability.

More formally, we proceed as follows. We choose random h_1 and h_2. If G contains cycles, we repeat until it does not; $O(1)$ trials are sufficient in expectation. Now G is a forest. Root every tree component of G arbitrarily. For every edge given by an element $x \in S$, we make x own the node which is lower in the tree. We set the position of T corresponding to a root to zero. Now, we perform a level traversal of each tree, from the root down. When we consider each edge, the upper vertex has been fixed in T; we fix the lower vertex (the one owned by the edge) to make the xor be what we want.

A very interesting alternative solution was found by Igor Ganichev and Brian Jacokes. Every edge has a lower endpoint, and a higher endpoint, where lower and higher are determined by the distance to the root (aka depth in the tree). As above, we can always store the data value in the lower endpoint. To determine which is lower, we can store the depth of each vertex modulo 3.

Several people tried to solve the problem by holding another array of data values, which should somehow tell you which of $T[h_1(x)]$ and $T[h_2(x)]$ you want. None of these attempts worked. The most common mistake was assuming that the data values are unique to each element; this cannot possibly be true when $r < \lg n$. So different elements with the same data could conspire to create a bad configuration in your disambiguation table.