Exercise: Linear data structures for storing sets

Rasmus Pagh

July 15, 2014

Note. This exercise expands the material on IBLTs presented by Michael on Monday, and makes use of the material on retrieval presented by Rasmus on Tuesday.

Background. In some applications it is desirable to have *linear* data structures for storing sets, such that adding and subtracting data structures correspond to set operations. For example, suppose that $S_1 \subseteq S_2$ and these sets are represented by linear data structures $D(S_1)$ and $D(S_2)$. Then the difference $D(S_2)-D(S_1)$ will be a data structure for $S_2 \backslash S_1$. Computing $S_2 \backslash S_1$ could for example be important in a distributed setting where data sets periodically need to be kept in sync. The surprise is that the capacity of $D(S_1)$ and $D(S_2)$ can possibly be much smaller than the sets S_1 and S_2, as long as it is big enough for $S_2 \backslash S_1$. That is, each of $D(S_1)$ and $D(S_2)$ individually may not tell us anything about S_1 and S_2, while their difference can!

Simple linear dictionaries based on peeling. Following Goodrich and Mitzenmacher (Allerton 2011), we will derive a simple linear dictionary based on the idea of multiple-choice hashing and peeling. We will consider keys to be positive integers, such that it makes sense to do arithmetic using them. The first idea is to use the same setup as for a split Bloom filter with two hash functions (disjoint ranges inside $\{1,\ldots,\mathrm{m}\}$), but rather than placing a bit in $A[h_1(x)]$ and $A[h_2(x)]$, we **add** the number x in both places. If we let $S_i = \{x \in S \mid h_1(x) = i \vee h_2(x) = i\}$ be the set of keys in S having i as a hash value, then $A[i] = \sum_{x \in S_i} x$. To analyze this we can consider the bipartite *choice graph* with vertices corresponding to hash values, and edges $\{(h_1(x), h_2(x)) \mid x \in S\}$.

a) Argue that this yields a linear data structure (in the sense that $D(S_2) = D(S_1) + D(S_2 \backslash S_1)$ when $S_1 \subseteq S_2$), but will not always allow us to compute S, even if the choice graph is acyclic.

b) Argue that if we extend the data structure with a table of counts, where $C[i] = |S_i|$, and the choice graph $\{\{h_1(x), h_2(x)\} \mid x \in S\}$ is acyclic, we can determine all keys in S. **Hint:** Once a key is determined, "peel" it from the data structure.

The solution of storing counts is not entirely satisfying because it may silently fail if we do not have $S_1 \subseteq S_2$, more precisely when a key in $S_1 \backslash S_2$ collides with a key in $S_2 \backslash S_1$. To get a more robust solution we want to be able to decide whether $S_i = \{|A[i]|\}$. As we will see, this can be achieved, with a small error probability, by replacing C by a table A' with sums of *hash values* of the sets S_i. That is, for some hash function $H : U \to \{0, \ldots, p-1\}$ we let $A'[i] = \sum_{x \in S_i} H(x)$, which is again linear.

c) Show that if values of H are *random and independent*, the probability that $\sum_{y \in A} H(y) = \sum_{y \in B} H(y)$ where $A \neq B$ is $1/p$. (This holds even modulo p.)

d) For prime p suppose that $A, B \subseteq \{0, \ldots, p-1\}$, and take a, b from the field of size p^2. Recall that elements of the field can be represented as symbolic polynomials $x \mapsto ux + v$, where $u, v \in \{0, \ldots, p-1\}$. Define $H(y) = (a^y b) \mod x$ (where all operations are in the field, and in particular the mod operation takes the constant part of the polynomial corresponding to $a^y b$). Argue that for this choice of H the following property holds for $A \neq B$:

$$\Pr[\sum_{y \in A} H(y) = \sum_{y \in B} H(y)] \leq 2/p \ .$$

Hint: Consider $H'(y) = a^y b$, and use that two degree $p-1$ polynomials can agree in at most p values.