CS 224 Advanced Algorithms — Spring 2017

Problem Set 2

Due: 11:59pm, Wednesday, February 15th

Submit solutions to Canvas, one PDF per problem:
`https://canvas.harvard.edu/courses/21996`

Solution max page limits: Two pages each for problems 1 and 2, one page for problem 3

See homework policy at `http://people.seas.harvard.edu/~cs224/spring17/hmwk.html`

Problem 1: (15 points) In class we introduced cuckoo hashing as a solution to the dynamic dictionary problem, and we also showed how to solve the "approximate static dictionary problem" via a cuckoo-hashing based approach to Bloomier filters. Our analyses in class assumed that the two hash functions $h, g : [u] \to [m]$ used by cuckoo hashing were fully random, where u is such that all keys are in $[u]$, and $m = cn$ is the the number of cells in our hash table for some constant c (in class we set $c = 4$, but you are allowed to assume any different constant value of c to make your proofs go through when solving this problem). In this problem, you will show that for both applications (dynamic dictionary, and approximate static dictionary), it suffices for h, g to be independently drawn from a k-wise independent family for $k \in \Theta(\lg n)$.

(a) (5 points) Show that if h, g are drawn independently from a $\Theta(\lg n)$-wise independent family, then the expected time per insertion using cuckoo hashing is still $O(1)$.

(b) (5 points) Devise a solution to the approximate static dictionary problem using $O(nr + \lg n \cdot \lg u)$ bits of memory and with $O(\lg n)$ query time, where keys are in $[u]$ and the value associated with each key is in $\{0,1\}^r$. Unlike in class, you should *not* assume that hash functions can be stored for free. You can use the fact, however, that for any integers $a, b \geq 2$ a k-wise independent hash family $\mathcal{H}$ exists mapping $[a]$ to $[b]$ such that any $h \in \mathcal{H}$ can be specified by a bistring of length $O(k \lg(ab))$ bits for any $a, b \geq 2$, and such that evaluation of $h(x)$ for any $x \in [a]$ takes time $O(k)$.

(c) (5 points) The solution in part (b) is not ideal: the space has an extra additive term of $\lg n \cdot \lg u$, and the query time is $O(\lg n)$ instead of $O(1)$. You will now show both how to improve the space, and to completely remove the second shortcoming. Recent work of [3] (improving upon [6, 7]) for any integer $t \geq 2$ constructs a k-wise independent family $\mathcal{H}_t$ mapping $[a]$ to $[b]$ such that $h \in \mathcal{H}$ can be represented using $O(k \cdot a^{1/t} \cdot t \cdot \log(ab))$ bits, and for any $x \in [a]$, $h(x)$ can be computed in time $O(t \log t)$. You do not need to understand how this construction works, but show that it implies an approximate static dictionary data structure using space $O(nr + \lg u)$ bits and with $O(1)$ query time. **Hint:** You may first want to reduce the universe size of the keys from u to something smaller. Show that if $g : [u] \to [m]$ is drawn from a 2-wise independent family, then for any $X \subset [u]$ of size n, g is likely to act injectively on X as long as $m = \Omega(n^2)$. That is, $\forall X \subset [u]$ s.t. $|X| = n, \mathbb{P}_{h \in \mathcal{H}}(\exists x \neq y \in X : g(x) = g(y)) \ll 1$ for some $m \in \Omega(n^2)$.

Problem 2: (20 points) In *simple tabulation hashing*, we write a key $x \in [u]$ in base $u^{1/c}$ (assume $u^{1/c}$ is an integer and power of 2) so that $x = \sum_{i=0}^{c-1} x_i \cdot u^{i/c}$. We call the x_i "characters". We then allocate c completely random lookup tables $H_0, \ldots, H_{c-1}$ each of size $[u^{1/c}]$. Then for each $y \in [u^{1/c}]$ we set $H_i(y)$ uniformly at random from $[m]$ (assume also m is a power of 2). Then to hash x, we define (where $\oplus$ denotes bitwise XOR)

$$h(x) = H_0(x_0) \oplus H_1(x_1) \oplus \cdots \oplus H_{c-1}(x_{c-1}).$$

(a) (5 points) Fix $c, m \geq 1$. Show that the family $\mathcal{H}$ of all such hash functions is 3-wise independent.

(b) (5 points) Show that for any $c > 1$, $\mathcal{H}$ is not 4-wise independent.

(c) (10 points) Imagine using such an $\mathcal{H}$ to implement hashing with chaining, as in Lecture 3. Show that if $m > n^{1.01}$ then with probability at least 2/3, every linked list in the hash table has size $O(1)$. **Hint:** show that if a subset T of the n items is "large", then $\mathcal{H}$ behaves completely randomly on some "somewhat large" subset T' of T.

Problem 3: (10 points) A problem that was encountered by the Altavista search engine (and still arises today) is that of near-duplicate document detection in information retrieval. When a user issues a query several web pages might seem to be good matches, however many of those pages might be (nearly) mirrors of some primary source. Search engines for obvious reasons don't then want to flood you with a page of search results all mirroring the same content. They thus wanted a quick-and-dirty way to compare two web pages for similarity.

Altavista's solution, invented by Broder [1], worked as follows. Treat each web page A as a "bag of words", that is, a set of elements in some dictionary D (those elements are simply the words that appear in the document). Given two documents (sets) A and B, define the *Jaccard similarity coefficient* $J(A, B)$ as $|A \cap B|/|A \cup B|$. Notice this is 0 if they contain disjoint words, and 1 if they contain exactly the same set of words. Computing $J(A, B)$ between all pairs of search result pages A, B could be slow. Also storing $J(A, B)$ for every pair of web pages on the Internet would take too much space (the square of the number of web pages on the Internet!). Let $|D| = n$. Broder proposed picking a uniformly random permutation $\pi : [n] \to [n]$ and, for each page A, storing $h(A) = \min_{x \in A} \pi(x)$. It is easy to then see $J(A, B) = \mathbb{P}(h(A) = h(B))$. The storage overhead of storing $h(A)$ for each A is quite small, since it is one extra word, whereas we are storing the entire web page already.

(a) (5 points) Suppose in pre-processing we pick k such random permutations pairwise independently, $\pi_1, \ldots, \pi_k$. That is to say, if s_i is the binary representation of π_i as a bitstring of length $t = \lceil \lg(n!) \rceil$, we can view s_i as an integer in $[2^t]$. Then $s_i = h(i)$ for h drawn from a 2-wise independent family mapping $[n]$ to $[2^t]$. We then store $h_i(A) = \min_{x \in A} \pi_i(A)$ for every web page A and for each $1 \leq i \leq k$. Given this setup, devise a scheme that, given pages A, B, produces a value Z such that

$$\mathbb{P}(|Z - J(A, B)| > \varepsilon) < 1/3. \tag{1}$$

The smaller $k = k(\varepsilon)$ your scheme requires, the better.

(b) (5 points) Suppose we want high probability of success and not just 1/3. That is, for $0 < \delta < 1$ we want $\mathbb{P}(|Z - J(A,B)| > \varepsilon) < \delta$. What must k be in your scheme from part (a) to guarantee this? What if the s_i were fully independent? Now imagine a different scheme: let $Z_1, Z_2, \ldots, Z_t$ be independent estimators each satisfying Eq. (1). Show that for some $t = O(\log(1/\delta))$,

$$\mathbb{P}(|(\underset{1 \le i \le t}{median}\, Z_i) - J(A,B)| > \varepsilon) < \delta.$$

If you want to read more about how to relax the assumption that π is a totally random permutation, see [2, 4, 5] for definitions and constructions of "min-wise hash families".

References

[1] Andrei Z. Broder. On the resemblance and containment of documents. In *Proceedings of Compression and Complexity of SEQUENCES*, pages 21–29, 1997.

[2] Andrei Z. Broder, Moses Charikar, Michael Mitzenmacher. A derandomization using min-wise independent permutations. *J. Discrete Algorithms* 1(1), pages 11–20, 2003.

[3] Tobias Christiani, Rasmus Pagh, Mikkel Thorup. From independence to expansion and back again. In *Proceedings of the 47th Annual ACM Symposium on Theory of Computing (STOC)*, pages 813–820, 2015.

[4] Piotr Indyk. A Small Approximately Min-Wise Independent Family of Hash Functions. *J. Algorithms* 38(1), pages 84–90, 2001.

[5] Mihai Pătraşcu, Mikkel Thorup. The Power of Simple Tabulation Hashing. *J. ACM* 59(3): 14, 2012.

[6] Alan Siegel. On universal classes of extremely random constant-time hash functions. *SIAM J. Comput.*, 33(3):505–543, 2004.

[7] Mikkel Thorup. Simple tabulation, fast expanders, double tabulation, and high independence. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 90–99, 2013.