Advanced Algorithms 2024 Deadline: 18/11/2024, 11:59 pm

Graded Homework 2

Lecturer: Bernhard Haeupler, Johannes Lengler, Maximilian Probst

Note 1: *Solutions must be typeset in LaTeX and uploaded via moodle by the deadline. Late submissions will not be graded. If you would like to add a drawing in your solution, you can simply include a picture of a hand-drawn figure in your LaTeX.*

Note 2: *This is a theory course and any claim should be substantiated with a proof. When asked to give an algorithm, the algorithm has to have polynomial running time, unless less efficient algorithms are explicitly permitted by the problem statement. You are required to prove the claimed properties of any algorithm you present.*

Note 3: *You can discuss the problems with the other students but you should write your own solutions independently, and you should be able to orally explain your submitted solution to the instructors, if asked. It is strictly prohibited to share any (hand)written or electronic (partial) solutions with fellow students. Moreover, if you discussed a problem with another student, you must list their names on your submitted solution.*

1 What's That Song? (25 points)

In this task, we consider one possible way to develop an application that is queried with a recording of a song and asked to return the title of the song played.

We are given a music library S with the songs that should be identifiable. Each song is represented by a set of features. Concretely, features come from a universe of size U and each song $A \in S$ is represented as a subset of features, i.e. $A \in 2^{[U]}$. The similarity between two songs $X, Y \in 2^{[U]}$ is measured by comparing the overlap in features:

$$\mathrm{Sim}(X, Y) = \frac{|X \cap Y|}{|X \cup Y|}.$$

Note that $\mathrm{Sim}(X, Y) \in [0, 1]$. For the purpose of this task, we define X as being *fairly similar* to Y if $\mathrm{Sim}(X, Y) \geq 0.9$.

The goal is now to construct a data structure that can be queried with a recording $X \in 2^{[U]}$ of a song and returns with at least a constant probability a song Y from its database that is fairly similar to X. We are guaranteed that such a Y always exists. If there multiple songs that are fairly similar to X, we can return an arbitrary one of them.

We start by describing a hash function that maps songs to $\mathcal{O}(\log n)$-bit bitstrings (where we denote $n = |S|$). Concretely, we construct a hash function $f : 2^{[U]} \to \{0, 1\}^{\ell}$ for $\ell = \frac{1}{2}\log_{(1/.95)}(n)$ as follows: we take ℓ uniform, 2-wise independent hash functions $h_1, h_2, \ldots, h_\ell$ where each h_j maps $[U]$ to $[2(nU)^{100}]$. We assume henceforth that for all j, no two distinct elements $x, y \in [U]$ are hashed to either the same value or the same value save for the least significant bit (i.e. $h_j(x) \neq h_j(y)$ and $h_j(x) \neq h_j(y)$ xor 1), which is true w.h.p.. For each j, we define $g_j : 2^{[U]} \to [2(nU)^{100}]$ as $g_j(X) = \min_{x \in X} h_j(x)$. Finally, we define the combined hash value $f(X)$ by

$$f(X) = (g_1(X) \bmod 2) \circ (g_2(X) \bmod 2) \circ \ldots \circ (g_\ell(X) \bmod 2).$$

That is, the combined hash value is an ℓ-bit string where the j-th bit indicates the parity of the minimum value hashed to among the components of X w.r.t. hash function h_j.

To implement the query for a song recording $X \in 2^{[U]}$, we construct independent hash functions $f_1, f_2, \ldots, f_{\sqrt{n}}$ as described above, each mapping $2^{[U]}$ to $\{0, 1\}^{\ell}$. Finally, for each such hash function

f_i, we build a map H_i such that for every $x \in \{0,1\}^\ell$, we can query $H_i[x]$ and get a list of all elements Y in S such that $f_i(Y) = x$.
Now, to perform a query on a song recording $X \in 2^{[U]}$, we use the maps $H_1, \dots, H_{\sqrt{n}}$ to find all elements $Y \in S$ such that for some i, $f_i(X) = f_i(Y)$. Finally, we return the song Y^* among these with the greatest similarity to X.

1. Show that for a hash function f as constructed above, the probability of any two songs $X, Y \in 2^{[U]}$ agreeing on the j-th bit, i.e. satisfying $g_j(X) = g_j(Y) \pmod 2$, is $(1 + \mathrm{Sim}(X,Y))/2$. (9 points)

   Note. Recall that we assume that there are no collisions when hashing via h_i for any i.

2. Show that with at least constant probability, upon query with $X \in 2^{[U]}$ for which a $Y \in S$ exists with $\mathrm{Sim}(X,Y) \geq .9$, we have that the algorithm returns an element $Z \in 2^{[U]}$ s.t. $\mathrm{Sim}(X,Z) \geq \mathrm{Sim}(X,Y)$. (8 points)

3. Assume that upon a query with recording X of a song in the library, the similarity to the recorded song $Y \in S$ is at least 0.9 while the similarity to any other song $Z \in S, Y \neq Z$ is at most 0.8.

   Show that in this case, the expected number of similarity computations performed by the algorithm is $O(\sqrt{n})$. (8 points)

2 Estimating Frequencies (25 points)

In exercise 4 of exercise set 6, we saw a special case with $\epsilon = 0.5$ of a deterministic streaming algorithm for estimating the frequency of integers in an N-length stream of integers from $[n]$ ($N = \mathrm{poly}(n)$) up to an additive error of ϵN with space complexity of $O(\log(n)/\epsilon)$.

1. Show that this is asymptotically optimal: there cannot exist a streaming algorithm in the same setting that estimates the frequency of each integer up to the same additive error with space complexity $o(\log(n)/\epsilon)$, i.e. better dependency on either n or ϵ and better or the same dependency on the other. (4 points)

Even though this tradeoff between additive error and space complexity is asymptotically optimal, surprisingly, it is possible to, for example, find with high probability an integer that occurs at least $\sqrt{N}$ times in a stream where each other integer occurs at most once, using just $O(\log^2 n)$ bits of memory. In this exercise, we obtain an algorithm achieving a more general and powerful result.

2. Recall the AMS algorithm from the lecture notes: an uniform, 4-wise independent hash function σ mapping $[n]$ to $\{-1, 1\}$ is sampled, then a value $Z = \sum_{j \in [n]} \sigma(j) f_j$ is computed. The AMS algorithm uses Z^2 as an estimator for $F_2 = \sum_{j \in [n]} f_j^2$. We repurpose this value Z.

   2a. Show that $\sigma(j) \cdot Z$ is an unbiased estimator for f_j (i.e. $\mathbb{E}[\sigma(j) \cdot Z] = f_j$). (2 points)

   2b. Show that the variance of $\sigma(j) \cdot Z$ is at most $||f||_2^2 = \sum_{j' \in [n]} f_{j'}^2$. (5 points)

3. Suppose we introduce an additional uniform, pairwise independent hash function $h : [n] \to [k]$ for some prime k, and compute values $Z_1, \dots, Z_k$, where Z_v is the value Z computed by an AMS sketch of stream elements that hash to v, i.e.,

$$Z_v := \sum_{\substack{j \in [n] \\ h(j) = v}} \sigma(j) f_j.$$

   Show that $\sigma(j) \cdot Z_{h(j)}$ is still an unbiased estimator for f_j, and that the variance of $\sigma(j) \cdot Z_{h(j)}$ is now at most $||f||_2^2/k$ (4 points).

4. Give a randomized streaming algorithm with space complexity $O(\log(n)^2/\epsilon^2)$ that with high probability, for each $j \in [n]$, gives an unbiased estimate of f_j with additive error at most $\epsilon||f||_2$. (10 points)

3 Hashing With Worst-Case Access (25 points)

Consider a random undirected n-vertex multigraph (possibly with self-loops) constructed by independently sampling the endpoints for each of $m = \lfloor n/2e^2 \rfloor$ edges.

1. Show that with probability at least $1 - O\left(\frac{1}{n}\right)$, there exists an orientation of the edges of the graph where each vertex has outdegree at most one. (4 points)

2. Show that for any fixed $u, v \in V$, the probability that there exists a path of length d from u to v that does not visit any vertex more than once is at most e^{-2d}/n. (5 points)

3. Suppose the m edges are sampled one by one. Given that there always exists an orientation of the edges of the graph where each vertex has outdegree at most one, give an algorithm with expected update time complexity $O(1)$ that maintains such an orientation. (6 points)

Now, consider the following hash table design: two hash functions h_1 and h_2 from $[U]$ to $[n]$ for prime n are selected. Elements inserted into the table are stored directly in an array, with at most one element stored at any index, and the following guarantee: if an element $x \in [U]$ appears, it must be stored either at index $h_1(x)$ or $h_2(x)$ of the table.

4. Suppose the hash functions h_1 and h_2 are fully independent and can be evaluated in constant time. Show how to implement a hash table with this layout that can, with probability at least $1 - O\left(\frac{1}{n}\right)$, process up to $m = \Omega(n)$ insertions with expected insertion time complexity $O(1)$ and **worst-case** lookup and deletion time complexity $O(1)$. (10 points)

For each part of this exercise, you may assume the results of earlier parts, even if you did not complete those parts.

4 Verifying 3-Colouring (25 points)

Consider an undirected graph $G = (V, E)$ with n vertices and $m = \Omega(n^{1.1})$ edges, where each vertex has a colour $c(v) \in \{\text{red}, \text{green}, \text{blue}\}$. This graph is given to you in a streaming form: the stream starts with the integers n and m, then m pairs of integers (u_i, v_i) identifying the i-th edge e_i of the graph as connecting vertices u_i and v_i, and finally, the stream concludes with the colour of each vertex $c(v)$, $v \in V$. The goal is to output whether the colouring is a valid 3-colouring, i.e., VALID if there is no edge e_i such that $c(u_i) = c(v_i)$, and INVALID otherwise.

1. Show that there cannot exist a randomized streaming algorithm with space complexity $o(m)$ that with high probability correctly answers if the 3-colouring is valid. (5 points)

   Hint: consider a bipartite graph.

2. Suppose that for each edge e_i, there are edges $e_j, e_{j'}$ with $|j - i|, |j' - i| \leq n$ such that the three edges form a triangle. Give a randomized streaming algorithm with space complexity $O(n \log n)$ that with high probability correctly answers if the 3-colouring is valid. (20 points)