Advanced Algorithms 2024 2024

Sample Solutions GHW 02

Lecturer: Maximilian Probst *Teaching Assistant: Antti*

1 What's That Song? *Maximilian Probst*

1. Fix any X, Y. As we assume that no hash function h_i hashes any two distinct elements in its domain to either the same value or the same value save for the least significant bit, the minimum hash value $z^* \in \operatorname{argmin}_{z \in X \cup Y} h_i(z)$ is unique. Now, we consider two cases:

   - The minimum hash value appears in the intersection of the two sets, i.e. $z^* \in X \cap Y$. Conditioning on this event, we always have $g_i(X) = g_i(Y) \pmod 2$. As the hash function values are fully independent, the event $z^* \in X \cap Y$ occurs with probability $\mathrm{Sim}(X, Y)$.
   - The minimum hash value appears outside the intersection of the two sets: without loss of generality, $z^* \in X \setminus Y$. Conditioning on this event, let $z' = \operatorname{argmin}_{z \in Y}$.

     We want to argue that the least significant bits of $h_i(z^*)$ and $h_i(z')$ are uniform and independent, thus $g_i(X) = g_i(Y) \pmod 2$ holds with probability $\frac{1}{2}$. To see this, consider first fixing all but the least significant bits of the hash values. Since we assume that no hash function hashes two distinct elements to the same value or the same value save for the least significant bit, this is enough to uniquely determine z^* and z'. Since the hash function is uniform and its values are fully independent, the least significant bits of $h_i(z^*)$ and $h_i(z')$ are now uniform and independent.

   Thus, overall the probability of $g_i(X) = g_i(Y)$ is

   $$\mathrm{Sim}(X, Y) + (1 - \mathrm{Sim}(X, Y)) \cdot \frac{1}{2} = (1 + \mathrm{Sim}(X, Y))/2$$

   as desired.

   A note on independence: to (even approximately) obtain the property desired here, it is not sufficient for the hash function h_i to be merely pairwise independent. However, full independence of the hash function's values is not required. Observe that we meaningfully used the full independence of the function's values only to analyze the probability that $z^* \in X \cap Y$, as the least significant bits of z^* and z' being independent can be obtained by having a separate, pairwise independent hash function for just the least significant bit. The desired property $P(z^* \in X \cap Y) \in [\mathrm{Sim}(X, Y) - \epsilon, \mathrm{Sim}(X, Y) + \epsilon]$ follows immediately from the property that for all X and $x^* \in X$, $P(x^* = \arg\min_{x \in X} h_i(x)) \in \left[\frac{1-\epsilon}{|X|}, \frac{1+\epsilon}{|X|}\right]$, which is called *approximate min-wise independence*.

   For any constant $\epsilon > 0$, for some sufficiently large constant c, it can be shown that any uniform, k-wise independent family of hash functions for $k \geq c \log(1/\epsilon)$ is ϵ-approximately min-wise independent[1].

2. Consider query input X and some $Y \in S$ with similarity $\mathrm{Sim}(X, Y) \geq 0.9$. Let $d := 1 - \mathrm{Sim}(X, Y)$, we have $d \leq 0.1$.

[1] See "A small approximately min-wise independent family of hash functions", Piotr Indyk, J. Algorithms 38

Consider some fixed hash function f_i from the hash functions $f_1, f_2, \ldots, f_{\sqrt{n}}$. Using the claim about bitwise agreement from the last exercise, we obtain that

$$\mathbb{P}[f_i(Y) = f_i(X)] = ((1 + \mathrm{Sim}(X, Y))/2)^\ell = (1 - d/2)^\ell \geq 0.95^\ell$$

and for $\ell = \frac{1}{2}\log_{1/0.95}(n)$ this yields probability at least $1/\sqrt{n}$.

Thus, since functions $f_1, f_2, \ldots, f_{\sqrt{n}}$ are independent, and under each, there is a collision between X and Y with probability at least $1/\sqrt{n}$, we can conclude that the probability that no collision occurs under any such hash function is at most

$$(1 - 1/\sqrt{n})^{\sqrt{n}} \leq (e^{-1/\sqrt{n}})^{\sqrt{n}} = 1/e$$

and thus with probability at least $1 - 1/e$, we have a collision with Y.

Letting Y be the song in S with largest similarity to X ensures that the similarity between Y and X is outputted with at least this probability.

3. Upon input $X \in 2^{[U]}$, consider any $Z \in S$ with $\mathrm{Sim}(X, Z) \leq 0.8$.

   By the same calculations as before, we have for $d' = 1 - \mathrm{Sim}(X, Z) \geq 0.2$ that for any hash function function f_i, we have

   $$\mathbb{P}[f_i(Z) = f_i(X)] = ((1 + \mathrm{Sim}(X, Z))/2)^\ell = (1 - d'/2)^\ell \leq 0.9^\ell.$$

   By using the standard formula for converting bases of logarithms, we obtain the $\ell = \frac{1}{2}\log_{1/0.95}(n) = \frac{1}{2}\log_{1/0.9}(n)/(\log_{1/0.9}(0.95)) > \log_{1/0.9}(n)$ since $\log_{1/0.9}(0.95) < 0.5$. Thus, the probability of collision of X and Z is in this case at most $0.9^{\log_{1/0.9}(n)} = 1/n$.

   This yields that in expectation, the number of songs in $S \setminus \{Y\}$ that have similarity at most 0.8 to X is at most $n \cdot 1/n = 1$. Thus, for each hash function f_i, we expect to spend constant time evaluating candidate songs that hashed to the same value as X.

   Since there are $\sqrt{n}$ dictionaries, the expected query time is thus $O(\sqrt{n})$.

2 Estimating Frequencies

Antti Roeyskoe

1. Consider a stream where each integer in some subset $S \subseteq [n]$ of size $|S| = \Theta(1/\epsilon)$ appears strictly more than $2\epsilon N$ times, which is possible as $\epsilon < \frac{1}{2}$, and no element outside that subset appears. Running any streaming algorithm that approximates frequencies of elements of $[n]$ in the stream up to an additive ϵN-factor recovers this subset S. However, the space required to represent a subset of size $|S|$ of a set of size $n = |[n]|$ is at least

   $$\log \binom{n}{|S|} \geq \log \left((n/|S|)^{|S|} \right) = |S| \log(n/|S|) = \Theta(\log(n)/\epsilon),$$

   as long as $\epsilon = \Omega(n^{-0.99})$.

2a. Using linearity of expectation and that $\mathbb{E}[\sigma(j')\sigma(j'')] = 0$ for $j' \neq j''$, which follows from 2-wise independence of the hash function σ (and that $\mathbb{E}[ab] = \mathbb{E}[a]\mathbb{E}[b]$ for independent a, b), we get

$$\mathbb{E}[\sigma(j) \cdot Z] = \sum_{j' \in [n]} \mathbb{E}[\sigma(j)\sigma(j')] f_{j'} = \mathbb{E}[\sigma(j)^2] f_j = f_j$$

2b. Using the same properties and the fact that $\sigma(j) \in \{-1, 1\}$, we have

$$\begin{aligned}
\text{Var}[\sigma(j) \cdot Z] &= \mathbb{E}[(\sigma(j) \cdot Z - f_j)^2] \\
&= \mathbb{E}[(\sigma(j) \cdot \sum_{j'} \sigma(j') f_{j'} - f_j)^2] \\
&= \mathbb{E}[(\sum_{j'} \sigma(j') f_{j'} - \sigma(j) \cdot f_j)^2] \\
&= \mathbb{E}[(\sum_{j' \neq j} \sigma(j') f_{j'})^2] \\
&= \sum_{j' \neq j} \sum_{j'' \neq j} \mathbb{E}[\sigma(j')\sigma(j'')] f_{j'} f_{j''} \\
&= \sum_{j' \neq j} f_{j'}^2 \\
&\leq ||f||_2^2
\end{aligned}$$

3a. As in 2a., using the pairwise independence of h, we have

$$\mathbb{E}[\sigma(j) \cdot Z_{h(j)}] = \sum_{j' \in [n]} \mathbb{E}[\mathbb{I}[h(j) = h(j')] \cdot \sigma(j)\sigma(j')] f_{j'} = \mathbb{E}[\sigma(j)^2] f_j = f_j$$

3b. Using the fact that h is independent from σ, we have

$$\begin{aligned}
\text{Var}[\sigma(j) \cdot Z_{h(j)}] &= \mathbb{E}[(\sigma(j) \cdot Z_{h(j)} - f_j)^2] \\
&= \mathbb{E}[(\sum_{j' \neq j} \mathbb{I}[h(j) = h(j')]\sigma(j') f_{j'})^2] \\
&= \sum_{j' \neq j} \sum_{j'' \neq j} \mathbb{E}[\mathbb{I}[h(j) = h(j')]\mathbb{I}[h(j) = h(j'')]\sigma(j')\sigma(j'')] f_{j'} f_{j''} \\
&= \sum_{j' \neq j} \sum_{j'' \neq j} \mathbb{E}[\mathbb{I}[h(j) = h(j')]\mathbb{I}[h(j) = h(j'')]]\mathbb{E}[\sigma(j')\sigma(j'')] f_{j'} f_{j''} \\
&= \sum_{j' \neq j} \mathbb{E}[\mathbb{I}[h(j) = h(j')]^2]\mathbb{E}[\sigma(j')^2] f_{j'}^2 \\
&= \sum_{j' \neq j} \frac{f_{j'}^2}{k} \\
&\leq ||f||_2^2/k
\end{aligned}$$

4. The sketch setup as above (with hash functions σ, h and computed values $Z_1, \ldots, Z_k$) gives an unbiased estimator for each f_j with variance $||f||_2^2/k$. Using Chebychev, we get that the probability that this estimate has absolute error of more than $\epsilon||f||_2$ is at most

$$\mathbb{P}\left(|\sigma(j)Z_{h(j)} - f_j| \geq \epsilon||f||_2\right) = \mathbb{P}\left(|\sigma(j)Z_{h(j)} - f_j| \geq \epsilon\sqrt{k} \cdot \frac{||f||_2}{\sqrt{k}}\right) \leq \frac{1}{\epsilon^2 k}.$$

Thus, selecting as k the first prime number larger than $\lceil 4/\epsilon^2 \rceil = \Theta(1/\epsilon^2)$, we get that the probability the estimate has absolute error more than allowed is at most $\frac{1}{4}$. Note, by Bertrand's Postulate we have $k = \Theta(1/\epsilon^2)$.

Now, we use the median trick. Our streaming algorithm will maintain $t = \Theta(\log(n))$ independent copies of the above sketch, and when queried for the frequency of an element, returns the median of the estimates from the sketches.

Let $X_{j,i}$ be a random variable with value 1 if the i-th independent copy of the sketch has absolute error more than $\frac{1}{4}$ estimating the frequency of element j. The median of the estimates then has absolute error at most $\epsilon||f||_2$ as long as $\sum_{i\in[t]} X_{j,i} \leq \frac{t}{2}$. By Chernoff, for X a sum of i.i.d. Bernoulli random variables and $c \geq 2$, we have $\mathbb{P}(X \geq c\mathbb{E}[X]) \leq e^{-c\mathbb{E}[X]/6}$. Using this, we get

$$\mathbb{P}\left(\sum_{i\in[t]} X_{j,i} \geq \left(\frac{t/2}{\mathbb{E}[\sum_{i\in[t]} X_{j,i}]}\right) \cdot \mathbb{E}[\sum_{i\in[t]} X_{j,i}]\right) \leq e^{-t/12}.$$

Thus, for $t = \Omega(\ln n)$, we get that the median of the estimates for a specific j has the desired additive error with high probability. Union bounding over all $j \in [n]$ obtains that all estimates have the desired additive error with high probability.

Finally, to analyze the space complexity, each of the $\Theta(\log n)$ independent sketches maintains $k = \Theta(1/\epsilon^2)$ integers in $[-N, N]$. Thus, as $N = \text{poly}(n)$, the total space complexity is $O(\log^2(n)/\epsilon^2)$.

3 Hashing With Worst-Case Access

Antti Roeyskoe

1. The following condition is equivalent to there existing an orientation with outdegree at most one: ever subset S of vertices contains at most $|S|$ edges (where an edge is contained if both its endpoints are in the subset). To see the equivalence, note that a connected component contains more edges than vertices if and only if it contains at least two cycles, and a connected component can be oriented with outdegree at most one if and only if it contains at most one cycle.

   To bound the probability such a subset S exists, we union bound over all subsets of vertices. For any given size $|S| = k$, an edge has both endpoints in the subset with probability $(k/n)^2$. Thus, using $\binom{a}{b} \leq (ae/b)^b$, we obtain the following upper bound on the probability at least $k+1$ edges have both endpoints in a fixed subset of size k:

   $$\binom{m}{k+1}\left(k^2/n^2\right)^{k+1} \leq \left(\frac{em}{k+1}\right)^{k+1}\left(k^2/n^2\right)^{k+1} \leq \left(\frac{k}{2en}\right)^{k+1}.$$

   There are $\binom{n}{k}$ subsets of size $|S| = k$, thus union bounding over all sizes and subsets of that size, we get the upper bound

   $$\sum_{k=1}^{n}\binom{n}{k}\left(\frac{k}{2en}\right)^{k+1} \leq \sum_{k=1}^{n}\left(\frac{en}{k}\right)^{k}\left(\frac{k}{2en}\right)^{k+1} = \frac{1}{en}\cdot\sum_{k=1}^{n}\frac{k}{2^{k+1}} \leq \frac{1}{en} = O(1/n).$$

2. We perform induction on d. For $d = 1$, there is a path of length 1 between two vertices if and only if there is an edge between them. The expected number of edges between any pair of distinct vertices is $2m/n^2 \leq \frac{1}{e^2 n}$, thus by Markov the probability a path of length 1 exists between the two is at most e^{-2}/n.

   Now, for larger d, fix u and v and the penultimate (second-to-last) vertex v' on the path. For there to exist a vertex-simple path of length d from u to v with v' as the penultimate vertex, there has to exist a vertex-simple path from u to v' of length $d-1$ not visiting v (which by induction occurs with probability at most $e^{-2(d-1)}/n$) and an edge between v' and v (which occurs with probability at most e^{-2}/n). These two events are not independent, but they are negatively associated, thus the probability a vertex-simple path from u to v with v' as its penultimate vertex exists is at most e^{-2d}/n^2. Union bounding over v', we get that the probability a vertex-simple path from u to v of length d exists is at most e^{-2d}/n, as desired.

3. We store each edge at the vertex it is directed away from.

   Whenever a new edge is added, if one of its endpoints has outdegree zero, we direct the new edge from that vertex to its other endpoint. Otherwise, consider the maximal directed paths from both vertices. If both of these paths end in a cycle (possibly, but not necessarily, the same one), the graph after adding the new edge cannot have an orientation where each vertex has indegree at most one, as it now contains a connected component with more than one cycle. Thus, at least one of the paths ends in a vertex with outdegree zero. Reverse the orientation of every edge on that path, resulting in the vertex that used to be its source now having outdegree zero. We can now direct the new edge from that vertex to the edge's other endpoint.

   From part 2, union bounding over v, we know that for any fixed vertex u, in an undirected graph with m edges with independently and uniformly sampled endpoints, the probability there exists a vertex-simple path of length d starting at u (and ending anywhere) is at most e^{-2d}. Thus, for this fixed u, the expected maximum length of a vertex-simple path from u is $\sum_{d=1}^{n} d \cdot e^{-2d} = O(1)$. Reducing the number of edges in the graph can only reduce the maximum length of a vertex-simple path from u. Thus, in a graph with up to m edges with independently and uniformly sampled endpoints, the expected length of the longest vertex-simple path from a uniformly randomly sampled vertex is $O(1)$.

   We return to analyzing the time complexity of the update step. The time required to find the path and reverse it is linear in that path's length, as we can simply follow the edges stored at the vertices on the path. From the above, we see that the expected length of the maximum vertex-simple path from either of the sampled endpoints of the new edge is $O(1)$. Thus, the update takes $O(1)$ time in expectation.

4. The hashing technique here is known as Cuckoo Hashing.

   Fix any sequence of operations involving at most $m = \lfloor n/2e^2 \rfloor$ insertions. Consider some prefix of the sequence, and the multigraph (possibly with self-loops) on vertex set $[n]$ containing for each element $x \in [U]$ stored in the hash table the edge $(h_1(x), h_2(x))$. Since the hash functions h_1 and h_2 are fully independent, the endpoints of each of these up to m edges are sampled independently and uniformly at random.

   We maintain an orientation on the edges of this graph. We call this orientation *valid* if each vertex has outdegree at most one. From part 1, we know that the probability a valid orientation exists at all times is at least $1 - O(1/n)$, as there are $m = \lfloor n/2e^2 \rfloor$ insertions in total, and deleting edges cannot make a valid orientation invalid. From part 3, we obtain an algorithm with expected update time $O(1)$ for maintaining the orientation of the edges as new edges are inserted.

   Now, if a valid orientation can be maintained, which can be done with probability $1 - O(1/n)$, we can simply store element x at the position its corresponding edge is directed away from. Whenever a new element is inserted, we only need to reverse the orientation of $O(1)$ edges in expectation, and for each reversed edge, we move a single element of the array from one given index to another, which can be performed in $O(1)$ time. Deleting an element x requires deleting one edge from the graph and one element from the array, which takes $O(1)$ time as the edge and element are both stored at one of the positions $h_1(x)$ or $h_2(x)$. Finally, lookup similarly takes $O(1)$ time, as we only need to check positions $h_1(x)$ and $h_2(x)$ of the array.

4 Verifying 3-Colouring

Antti Roeyskoe

1. Suppose n is a multiple of 4, let $V = U \cup V \cup U' \cup V'$ with $|U| = |V| = |U'| = |V'|$, and consider the distribution of graphs where

- Every edge (u', v') between $u' \in U'$ and $v' \in V'$ exists
- For any pair (u, v) with $u \in U$ and $v \in V$, the edge (u, v) exists with probability $\frac{1}{2}$

Such graphs satisfy $m = \Omega(n^{1.1})$.

Let G be a graph from this distribution, and let $u \in U$, $v \in V$ be some vertices. Consider a colouring where every vertex in $U \cup U'$ except u is coloured red, every vertex in $V \cup V'$ except v is coloured blue, and the vertices u and v are coloured green. Then, the colouring is a valid 3-colouring if and only if the edge (u, v) does not appear in G.

Using this reduction, we can thus obtain from a $o(m)$-memory streaming algorithm for verifying a 3-colouring with high probability a $o(n)$-memory algorithm that succeeds with high probability for the following streaming problem: read in a n-bit bitstring x, then afterwards, output a n-bit bitstring x', such that the bitstrings x and x' match in at least $\frac{3}{4}n$ positions, i.e. $x_i = x'_i$ holds for at least $\frac{3}{4}n$ indices $i \in [n]$.

This, of course, is impossible. To show this, let $f(R, s) = x'$ be any map from a sequence R of random bits and a $o(n)$-bit advice bitstring s into n-bit bitstrings x'. For any f, s and R, the probability that for a uniformly random n-bit bitstring x at least $\frac{3}{4}n$ bits of x and $x' = f(R, s)$ match, is, by Chernoff, at most

$$\begin{aligned} P(\sum_i I[x_i = x'_i] \geq (1 + \frac{1}{2})\frac{n}{2}) &= P(\sum_i I[x_i = x'_i] \geq (1 + \frac{1}{2})\mathbb{E}[\sum_i I[x_i = x'_i]]) \\ &\leq e^{-\mathbb{E}[\sum_i I[x_i = x'_i]]/(4(2+\frac{1}{2}))} \\ &= e^{-\Omega(n)}. \end{aligned}$$

as the bits of x are uniform and independent of x'. Thus in particular, for any R, we have

$$P\left(\max_s \sum_i I[x_i = f(R, s)] \geq (1 + \frac{1}{2})\frac{n}{2}\right) \leq 2^{o(n)} e^{-\Omega(n)} = e^{-\Omega(n)}.$$

Thus, even if s is an arbitrary $o(n)$-bit bitstring function of x, the probability more than $\frac{3}{4}n$ bits of x and x' match is at most $e^{-\Omega(n)}$.

2. With this guarantee on the stream, we can remember each of the last $2n$ edges to appear in the stream using $O(n \log n)$ space. Whenever a new edge appears, we check all triplets of the last $2n$ edges to see if they form a triangle. The union of triangles found this way covers every edge of the original graph: for any edge in the original graph there is a triangle containing it, and for any triangle obtained each of its edges appears in the original graph.

   Consider a sketch that maintains for each vertex a single bit. Initially, all bits are set to zero. When processing a triangle, the algorithm flips a fair coin, and if it lands on heads, it does nothing, otherwise it flips the bit of each of the triangle's endpoint vertices. Let $r, g, b \in \{0, 1\}$ denote, respectively, the xor of all the red, green and blue vertex bits. After reading the colouring, the algorithm computes r, g, b and outputs VALID if and only if $r = g = b$.

   First, we show that if the coloring is valid, the algorithm always outputs VALID. Initially, $r = g = b = 0$. Any valid colouring has each of the 3 colours appear exactly once in any triangle, thus in a valid colouring, processing any triangle either xors r, g and b by 1 or xors none of them.

   Now we show that if the colouring is invalid, the algorithm outputs INVALID with probability at least $\frac{1}{2}$: as the triangles cover each edge of the graph, there must be a triangle that does not have each colour appear exactly once as its endpoints. Processing that

triangle either xors one colour's value by 1 and leaves the others unchanged, or xors none of the values. Fixing all other coinflips we see that one of these scenarios must force the colours not not have equal values, and as both have probability $\frac{1}{2}$, invalidity is detected with probability at least $\frac{1}{2}$.

To check validity with high probability, we simply maintain $O(\log n)$ independent copies of this sketch. Each sketch stores 1 bit for each vertex, which takes $n \cdot O(\log n)$ total space. As mentioned before, remembering the last n edges to appear takes $O(n \log n)$ space, and finally remembering the colours of the vertices takes $O(n)$ space. Thus, the final space complexity is $O(n \log n)$.