CS 514: Advanced Algorithms II – Sublinear Algorithms **Rutgers: Spring 2020**

Problem set 2

Due: 11:59PM, March 24, 2020

Problem 1. In Lecture 5, we designed a local computation algorithm (LCA) for the maximal independent set (MIS) problem on graphs with maximum degree Δ. With high probability, the runtime of this algorithm in answering each probe was $\Delta^{O(\Delta \cdot \log \Delta)} \cdot \log n$ which is considered efficient (at least for "small" Δ). However, this LCA required to record $\Omega(n)$ random bits which is inefficient in terms of random bit complexity.

In this problem, we will examine two different (but similar) ways of reducing the number of random bits in this LCA using k-wise independent hash functions described in Lecture 7.

(a) Show that we can pick all the random bits used by the LCA using a k-wise independent hash function for some $k = \Delta^{O(\Delta \cdot \log \Delta)} \cdot \log n$, while still maintaining the correctness of the algorithm. Conclude that this gives an LCA for MIS with $\Delta^{O(\Delta \cdot \log \Delta)} \cdot \log^2 n$ probe time and $\Delta^{O(\Delta \cdot \log \Delta)} \cdot \log^2 n$ random bits.

(5 points)

(b) Recall that the LCA for MIS was based on simulating a distributed LOCAL algorithm for MIS and then use a deterministic post-processing. Suppose we pick the random bits in *each round* of the distributed LOCAL algorithm from a 2-wise independent hash function and choose the hash functions across the rounds *independently*. Show that this new LCA still outputs a correct MIS with high probability and that its runtime for each probe is $\Delta^{O(\Delta \cdot \log \Delta)} \cdot \log n$ and it needs $\text{poly}(\Delta, \log n)$ random bits.

(10 points)

Problem 2. A matching in a graph $G = (V, E)$ (not necessarily bipartite) is any collection of edges that do not share any vertices. A *maximal matching* M is then any matching that is not a proper subset of any other matching in G; in other words, for any edge $e \in E$, at least one of its endpoints is matched by M. An LCA for the maximal matching problem should maintain an implicit maximal matching to answer the following probe:

- Given an edge $e \in E$, is this edge part of the chosen maximal matching?

In this problem, we design an LCA for maximal matching on graphs with maximum degree Δ that answers each probe in *expected* $2^{O(\Delta)}$ time (note that this is a different guarantee from the LCA for MIS that works with high probability for every probe).

(a) Our LCA for this problem maintains a maximal matching M implicitly as follows:

> (a) Pick a random permutation π of edges of E.
>
> (b) Given the probe $e \in E$ do as follows:
>
> (i) Go over each edge $e' \in E$ that share a vertex with e and have $\pi(e') < \pi(e)$: *recursively* probe whether e' is part of the maximal matching M or not.
>
> (ii) If the answer to any of the probes incident on e was Yes, return e is *not* part of M; otherwise, return e is in M.

Prove that the set M of edges output by this LCA is always a maximal matching. **(5 points)**

(b) Let $T(e)$ be the set of edges that are recursively considered by LCA in response to a probe e. Prove each of the following statements: **(5 points)**

(i) Runtime of LCA in response to probe e is $O(\Delta \cdot |T(e)|)$;

(ii) Any path *examined by a recursive query sequence* in $T(e)$ starting from e has *decreasing* values of π over the edges. I.e., if the path is $(e =)e_1, e_2, \ldots, e_k$, then $\pi(e_1) > \pi(e_2) > \cdots > \pi(e_k)$.

(c) Prove each of the following statements: **(5 points)**

(i) For any fixed path P of length k in G, the probability that P has decreasing values of π over the edges is $\frac{1}{k!}$.

(ii) For any fixed vertex v, the number of edges on all paths of length at most k starting from v in G is at most Δ^k.

(d) Use the two parts above to prove that the expected size of $T(e)$ is $2^{O(\Delta)}$ and conclude that the *expected* time it takes to answer any fixed probe is also $2^{O(\Delta)}$. **(10 points)**

Problem 3. Recall the problem of sparse recovery on $\mathbb{F}_2$ we studied in Lecture 6: Given a k-sparse vector $x \in \mathbb{F}_2^n$, i.e., $||x||_0 = k$, design a matrix A such that we can recover x *uniquely* from $A \cdot x$ (note that the computation is done over $\mathbb{F}_2$). In the class, we saw that we can find an A with $O(k \cdot \log(n/k))$ rows for this task and proved that $\Omega(k \cdot \log(n/k))$ rows are also necessary. However, our algorithm required *exponential* time (in k) for constructing A and also recovering x from $A \cdot x$.

Design a *polynomial-time randomized* algorithm that outputs a matrix A with $O(k \cdot \log(n/k))$ rows such that with probability at least $9/10$, for any fixed (unknown) k-sparse x, we can recover x from $A \cdot x$ – note that this is a weaker guarantee than our previous algorithm in terms of recovery that simultaneously worked for all k-sparse $x \in \mathbb{F}_2^n$. **(25 points)**

Hint: Consider the following matrix B with $\Theta(k)$ rows where each row $B^{(i)}$ of B is chosen by independently picking a random subset S_i of n with size (n/k), setting entry $B^{(i)}_j = 1$ for all $j \in S$ and zero otherwise. Combine this with the deterministic algorithm for 1-sparse recovery discussed in Lecture 6 to design a polynomial-time algorithm that can recover a *constant fraction* of entries of x. Let $y \in \mathbb{F}_2^n$ be the vector of recovered indices (so 1's of y are a subset of 1's of x). Show that another *independent* copy of the matrix in the previous part with a constant factor smaller number of rows can be used to recover a constant fraction of 1's in the vector $x - y$. Continue this to recover all the vector x.

Bonus part: Design a *deterministic* algorithm for this problem. You will still receive partial credit for algorithms that use sub-optimal number of measurements (e.g., a matrix with $O(k^2 \cdot \log(n/k))$ rows instead).

(+40 points)

Problem 4. Given a set of numbers S and a number $x \in S$, the *rank* of x is defined to be the number of elements in S that have value at most x:

$$\mathsf{rank}(x, S) = |\{y \in S : y \leq x\}|.$$

Given a parameter $\varepsilon \in (0, 1/2]$, we say that an element $x \in S$ is an ε-approximate element of rank r if

$$(1 - \varepsilon) \cdot r \leq \mathsf{rank}(x, S) \leq (1 + \varepsilon) \cdot r.$$

Suppose we are given a stream of numbers $S = s_1, s_2, \ldots, s_n$, where $s_i \in [m]$ for $1 \leq i \leq n$, and assume that all s_i's are *distinct*. Our goal is to design an $O(\varepsilon^{-2} \log m \log n)$ space streaming algorithm for retrieving an ε-approximate element for any given rank value.

(a) Consider the following algorithm for computing an ε-approximate median: sample $O(\varepsilon^{-2} \log n)$ numbers from the stream uniformly at random (with repetition) and return the median of the samples.

Show that this algorithm returns an ε-approximate median with probability at least $1 - 1/\text{poly}(n)$ and uses $O(\varepsilon^{-2} \cdot \log m \cdot \log n)$ bits of space. **(10 points)**

(b) We now extend the previous algorithm to compute an ε-approximate element of rank r for any $r \in [n]$. Consider the following algorithm for this problem. Let $t = \lceil 24\varepsilon^{-2} \log n \rceil$. If $r \leq t$, then simply maintain a list T of r smallest elements seen in the stream, and output the largest element in T at the end of the stream. Otherwise, choose each element in the stream with probability t/r, and maintain the t smallest sampled values in a list T. At the end of the stream, output the *largest* number in T.

Show this algorithm returns an ε-approximate element of rank r with probability at least $1 - 1/\text{poly}(n)$ and uses $O(\varepsilon^{-2} \cdot \log m \cdot \log n)$ bits of space. **(10 points)**

Problem 5. Suppose we are given a stream of numbers $e_1, \ldots, e_n$ from a universe $[m]$ which defines a *frequency vector* $f \in \mathbb{N}^m$. In Lecture 7, we saw streaming algorithms for estimating frequency moments of the vector f. In this problem, we consider algorithms for another problem related to the frequency vector, namely, point-wise estimation of f. In particular, the streaming algorithm needs to output a data structure such that at the end of the stream, for any given $i \in [m]$, with probability at least $1 - \delta$, we can recover $\widetilde{f_i}$ from this data structure where:

$$f_i \leq \widetilde{f_i} \leq f_i + \varepsilon \cdot ||f||_1.$$

(a) The standard solution for this problem is called the *count-min sketch*. Let $a = 10 \ln(1/\delta)$ and $b = \frac{4}{\varepsilon}$. Pick a pairwise independent hash functions $h_1, \ldots, h_a : [m] \to [b]$. Throughout the stream, compute $a \cdot b$ counters:

$$c_{p,q} = |\{e_i \text{ in the stream} \mid h_p(e_i) = q\}|.$$

At the end of the stream, given any $i \in [m]$, return

$$\widetilde{f_i} = \min_{p \in [a]} (c_{p,q} \text{ where } q = h_p(i)).$$

Prove that count-min sketch described above solves the given problem and analyze its space complexity. **(10 points)**

(b) For a frequency vector f and $\phi \in (0, 1)$, a *ϕ-heavy hitter* of f is any element $i \in [m]$ such that $f_i \geq \phi \cdot ||f||_1$. Design a streaming algorithm that given a stream $e_1, \ldots, e_n$ from universe $[m]$ (defining the frequency vector f) and parameters $\phi, \varepsilon, \delta \in (0, 1/2)$, outputs a list of at most $\frac{2}{\phi}$ numbers such that with probability $1 - \delta$, every ϕ-heavy hitter of f belongs to this list, and no element which is *not* a $(\phi - \varepsilon)$-heavy hitter in f is reported in this list. The space complexity of your algorithm should be poly-logarithmic in $n, m, (1/\delta)$, and polynomial in ϕ and ε. **(5 points)**

Problem 6 (**Bonus Problem**). Recall the uniform distribution testing problem from Lecture 4: We are given sample access to a distribution μ on domain $[n]$ and our goal is to decide whether μ is the uniform distribution U_n on $[n]$ or it is ε-far from U_n in total variation distance, i.e., $\Delta_{\text{tvd}}(\mu, U_n) \geq \varepsilon$. We showed that for any *constant* $\varepsilon \in (0, 1)$, this problem can be solved with constant probability using $O(\sqrt{n})$ samples. Our goal now is to prove $\Omega(\sqrt{n})$ samples are also necessary for solving this problem for some *constant* $\varepsilon \in (0, 1)$.

(a) Define $\mathcal{D}_{\mathsf{no}}$ as the distribution over family of distributions on $[n]$ as follows (elements of $\mathcal{D}_{\mathsf{no}}$ are itself distributions, i.e., when we sample from $\mathcal{D}_{\mathsf{no}}$, we obtain a distribution):

- A sample distribution μ from $\mathcal{D}_{\mathsf{no}}$ is obtained by sampling a subset $S \subseteq [n]$ of size $(n/2)$ uniformly at random, and letting $\mu(i) = \frac{2}{n}$ for all $i \in S$ and $\mu(j) = 0$ for $j \notin S$ (i.e., distribution μ is uniform over S and has no mass in $[n] - S$).

Prove that any distribution $\mu \sim \mathcal{D}_{\mathsf{no}}$ is ε-far from uniform in total variation distance for some fixed constant $\varepsilon \in (0, 1)$. **(+5 points)**

(b) Consider any algorithm A that uses $o(\sqrt{n})$ samples for uniformity testing: Prove that A *cannot* distinguish between U_n or a distribution $\mu \sim \mathcal{D}_{\mathsf{no}}$ with probability of success at least $2/3$. Conclude that any algorithm for uniformity testing with some constant $\varepsilon \in (0, 1)$ with probability of success at least $2/3$ requires $\Omega(\sqrt{n})$ samples. **(+15 points)**