CS 787: Advanced Algorithms — Homework 5

Out: 4/24/19 — **Due: 5/9/19**

Note: Homework should be submitted in pairs on Canvas. We will only accept PDF files.

1. **(Heavy hitters.)** Recall that given a parameter $\theta \in (0,1)$, a heavy hitter in a stream of length m is any element that appears at least θm times. In class we developed various sketches for estimating which elements are heavy hitters as well as estimating their frequencies. In this problem you will put these sketches together to develop an algorithm that reliably returns a list of all heavy hitters.

   Specifically, consider a stream of length m where each element belong to a universe U of size n. Fix parameters $\epsilon, \delta, \theta \in (0,1)$. Develop an algorithm to process this stream and return a list of the heavy hitters. With probability at least $1-\delta$, your algorithm should return a list of all the elements in U that appear at least $(1+\epsilon)\theta m$ times in the stream and should not return any elements that appear fewer than $(1-\epsilon)\theta m$ times. Your algorithm should perform a single pass over the stream and use a data structure of size $\text{poly}(1/\epsilon, 1/\theta, \log(1/\delta))$.[1]

2. **(Counting in small space.)** Counting the length (m) of a stream is simple using $O(\log m)$ bits – we just keep a counter that we increment every time we see an element in the stream. Is it possible to keep an approximate count in smaller space?

   Consider the following estimator. We initialize a counter $X = 0$. Every time we see an element in the stream, we increment X with probability 2^{-X}. Let $Y = 2^X$.

   (a) Compute $\mathrm{E}[Y]$.

   (b) Compute $\mathrm{Var}[Y]$.

   (c) Develop an (ϵ, δ) estimator for the count m based on the random variable Y. What are the space requirements of your estimator?

3. **(Spanners.)** For any $t > 1$, a t-spanner of an unweighted undirected graph $G = (V, E)$ is a subgraph $H = (V, E')$ such that for every pair of vertices $u, v \in V$, we have $d_G(u,v) \le d_H(u,v) \le t d_G(u,v)$ where $d_G(u,v)$ (resp. $d_H(u,v)$) is the distance between u and v in G (resp. H).

   Consider the following simple algorithm for constructing a t-spanner of a given graph G. Start with the empty graph $H = (V, \emptyset)$. Consider the edges of G in arbitrary order. For each edge $(u,v) \in E$ in sequence, if $d_H(u,v) > t$, then add (u,v) to H, otherwise discard the edge. In this problem you will prove that the above algorithm produces a t-spanner of small size.

   (a) Prove that the graph H generated above is a t-spanner.

   (b) The length of the shortest cycle in an unweighted graph is called the girth of the graph. Prove that the graph H has girth at least $t+2$.

   (c) Prove that any graph on n nodes with girth g has at most $n^{1+O(1/g)}$ edges.

   Hint: Let m denote the number of edges in the graph. Suppose every node in the graph has degree $2m/n$. How would you then prove this statement? Now consider the general case. Can you remove all "low" degree nodes from the graph without removing too many edges, and then run your previous argument?

4. **(Counting Min Cuts.)** The following randomized algorithm by Karger finds small cuts in unweighted graphs. Let $G = (V, E)$ be an unweighted graph. At every step, the algorithm picks a uniformly random edge in the graph, say (u,v). It then merges the vertices u and v into a single meta-node "uv". In effect, it removes the vertices u and v from the graph and instead adds in uv, connecting the new meta-node to all of the previous

[1] Note that the counts in your data structure may be numbers of size $O(\log m + \log n)$, but the size of your data structure should not otherwise depend on the parameters m and n.

neighbors of u and v.[2] It then recurses in the remaining (multi-)graph. The algorithm continues until only two meta-nodes remain. Each of these nodes corresponds to some subset of the nodes in the original graph, and together the meta-nodes form a partition of the graph into two parts. The algorithm returns the corresponding cut.

In this problem you will prove that Karger's algorithm produces small cuts with "good" probability.

(a) Let C^* be a minimum cut in the graph, and let C be any other cut with $|C| = \alpha|C^*|$ for some $\alpha > 1$. Consider some iteration i of the algorithm. We say that the cut C has survived up to iteration i if at the beginning of this iteration, no meta-node in the graph contains vertices from both sides of the partition defined by C. Compute the probability that the algorithm picks an edge in $(u, v) \in C$ to merge during iteration i conditioned on the cut surviving up to iteration i. *Hint: how many edges does the graph have remaining at iteration i?*

(b) Use your calculation in part (a) to bound from below the probability that the algorithm returns cut C. Your answer should be a function of n and α.

(c) Use your answer to part (b) to bound the number of distinct cuts in graph G with fewer than $\alpha|C^*|$ edges.

5. **(List Coloring.)** Consider an undirected graph $G = (V, E)$, where each vertex v has a list $S(v)$ of allowed colors with $|S(v)| = k$ for some parameter k. A **list-coloring** χ of G assigns each vertex $v \in V$ a color from its list $S(v)$. A proper list-coloring is one that ensures that all edges are bichromatic, that is, their end points are assigned different colors.

Suppose that for each $v \in V$ and each color $c \in S(v)$, there are at most $k/10$ neighbors u of v that contain c in their color sets $S(u)$. (There is no bound on the degree of the underlying graph, though.) Show that in this case there always exists a proper list-coloring of G.

Hint: this is a simple application of the Lovasz Local Lemma.

[2]Note that after such a merger, some pairs of nodes may have multiple edges between them. For example, if u and v both share a neighbor w, then the meta-node uv has two edges to w.