CS 388R: Randomized Algorithms, Fall 2023 October 14, 2023

Lecture 14: Heavy Hitters and Bipartite Perfect Matchings

Prof. Eric Price *Scribe: Bhargav Samineni*

NOTE: THESE NOTES HAVE NOT BEEN EDITED OR CHECKED FOR CORRECTNESS

1 Overview

In the previous lecture we discussed sampling algorithms for median finding in the offline setting. We also discussed algorithms to uniformly sample k elements and sampling distinct elements in the streaming setting.

In this lecture we study the Heavy Hitters problem and present algorithms for it in insertion-only and turnstile streams. We also study matchings and look at the Perfect Matching problem in the special case of d-regular bipartite graphs.

2 Heavy Hitters

Suppose you are given a stream $V = (v_1, \ldots, v_m)$ of elements from some universe with repetitions allowed. For an element v in this universe, let $x_v = |\{v_i \in V \mid v_i = v\}|$. That is, it is the count of the number of occurrences of element v in the stream V. Let $m = \sum_v x_v$ be the total count of elements in the universe in the stream V and let n the number of possible values of v, i.e., the size of the universe the elements are drawn from.

In the Heavy Hitters problem, we would like to find the most common elements in an arbitrary stream in sublinear space, i.e., space much less than $O(\min\{m, n\})$. In particular, we want to find a set S of elements in the universe such that S is a superset of all elements in the universe with count $\geq \alpha m$ and a subset of all elements in the universe with count $\geq \alpha m/2$ for some given parameter α. That is, S is a set that satisfies the following:

- For all v in the universe such that $x_v \geq \alpha m$, $v \in S$. We call such v "heavy" in the stream.
- For all v in the universe such that $x_v \leq \alpha m/2$, $v \notin S$. We call such v "light" in the stream.
- For all v in the universe such that $\alpha m/2 < x_v < \alpha m$, either $v \in S$ or $v \notin S$

To solve this problem, suppose we sample a set T of size k uniformly at random from the stream, where $k = O(1/\alpha \log 1/\delta)$ and δ is an error parameter we choose later. For an element v in the universe let C_v be its count in T. Consider the following cases for v:

- If v is heavy in the stream (i.e., $x_v \geq \alpha m$), then the expected count of v in T is

$$\mathbb{E}[C_v \mid v \text{ is heavy}] \geq \alpha k = O(\log 1/\delta).$$

Under this assumption, by multiplicative Chernoff we get that

$$\Pr\left[C_v \leq \frac{3}{4}\alpha k\right] = \Pr\left[C_v \leq \left(1-\frac{1}{4}\right)\alpha k\right] \leq \Pr\left[C_v \leq \left(1-\frac{1}{4}\right)\mathbb{E}[C_v \mid v \text{ is heavy}]\right]$$
$$\leq \exp\left(-\frac{O\left(\log 1/\delta\right)}{2(4^2)}\right) = O\left(\delta\right)$$

Hence, with probability $1 - O\left(\delta\right)$ the count of v in T will be $\geq 3\alpha k/4$ if v is heavy in the stream.

- If v is light in the stream (i.e., $x_v \leq \alpha m/2$), then the expected count of v in T is

$$\mathbb{E}[C_v \mid v \text{ is light}] \leq \frac{\alpha k}{2} = O\left(\log 1/\delta\right).$$

Under this assumption, by multiplicative Chernoff we get that

$$\Pr\left[C_v \geq \frac{3}{4}\alpha k\right] = \Pr\left[C_v \geq \left(1+\frac{1}{2}\right)\frac{\alpha k}{2}\right] \leq \Pr\left[C_v \geq \left(1+\frac{1}{2}\right)\mathbb{E}[C_v \mid v \text{ is light}]\right]$$
$$\leq \exp\left(-\frac{O\left(\log 1/\delta\right)}{(2+1/2)2^2}\right) = O\left(\delta\right)$$

Hence, with probability $1 - O\left(\delta\right)$ the count of v in T will be $< 3\alpha k/4$ if v is light in the stream.

Thus, if we output the set $S = \{v \in T \mid C_v \geq 3\alpha k/4\}$, then each individual $v \in S$ is correct with probability $1 - O\left(\delta\right)$ and hence taking a union bound gives that S is correct with probability $1 - O\left(\delta\right)$. If we set $\delta = 1/\min\{m,n\}$, then S is correct whp and the space usage of the algorithm is $O\left(1/\alpha \log(\min\{m,n\})\right)$ since we only have to sample k elements.

2.1 Count-Min Sketch

In the previous section, we considered Heavy Hitters in the setting where the counts of elements in the universe only every increased, i.e., we had an insertion-only stream. We can also consider Heavy Hitters in a *turnstile* stream where the stream now consists of updates of the form $(x_v, +1)$ or $(x_v, -1)$ for an element v in the universe. That is, we now allow insertions and deletions of elements in the stream, with the caveat that deletion updates bringing the count of an element below 0 at any point are not allowed. To solve Heavy Hitters in this model using sublinear space, we use something called the Count-Min Sketch [CM05].

We first choose $r = O\left(\log n\right)$ hash functions such that $h_i \colon [n] \to [c]$ for each $i \in [r]$ and some value c we will choose later. Intuitively, we can think of each hash function representing a row in a table mapping elements in the universe amongst c columns. For a row i and column j, let $y_{i,k} = \sum_{u \colon h_i(u)=j} x_u$. That is, $y_{i,k}$ is the sum of the counts of elements in the universe that get mapped to column k under hash function h_i. Since this is value is linear, we can easily maintain this value under insertions and deletions on the x_u values.

For an element u in the universe, for each $i \in [r]$ we can think of $\tilde{x}_u^{(i)} = y_{i,h_i(u)}$ as an estimate of x_u. We are guaranteed that $\tilde{x}_u^{(i)}$ is at least as large as x_u, but each estimate is likely to be an overestimate due to hash collisions with other elements in the universe. To get the best estimate on x_u, we can simply take the minimum value $\hat{x}_u = \min_i\{\tilde{x}_u^{(i)}\}$.

To analyze the correctness of this algorithm, pick some row $i \in [r]$ and consider the value $0 \leq \tilde{x}_u^{(i)} - x_u = \sum_{u' \neq u} x_{u'} \cdot \mathbb{1}_{h_i(u')=h_i(u)}$, where $\mathbb{1}_{h_i(u')=h_i(u)}$ is a binary variable that equals 1 if u' and u hash to the same column and 0 otherwise. This is the amount that x_u is overestimated by $\tilde{x}_u^{(i)}$. Then,

$$\mathbb{E}[\tilde{x}_u^{(i)} - x_u] \leq \left(\sum_{u' \neq u} x_{u'} \right) \frac{1}{c} \leq \frac{m}{c}$$

where $m = \sum_v x_v$ is the sum of all counts of all elements at the end of the stream. Applying Markov's inequality, we get that

$$\Pr\left[\tilde{x}_u^{(i)} - x_u \geq \frac{2m}{c}\right] \leq \Pr\left[\tilde{x}_u^{(i)} - x_u \geq 2\,\mathbb{E}[\tilde{x}_u^{(i)} - x_u]\right] \leq \frac{1}{2},$$

so with prob at least $1/2$ we have that $\tilde{x}_u^{(i)} - x_u \leq 2m/c$. This is independent over the choices of i, so with probability at least $1 - 1/2^r$, we get that $\hat{x}_u - x_u \leq 2m/c$. Recall that the set S we want to return must include all elements v in the universe such that $x_v \geq \alpha m$ and not contain any elements v such that $x_v \leq \alpha m/2$. Hence, we can return the set $S = \{v \mid \hat{x}_v \geq \alpha m\}$. To ensure that this set is valid whp, we can take $c = 4/\alpha$ and $r = 2 \log n$. These values give that with probability $1 - 1/n$, for all elements u in the universe, we have that $\hat{x}_u - x_u \leq \alpha m/2$, i.e., that the count of any element is not overestimated by more than $\alpha m/2$. The total space usage of this algorithm is $O(1/\alpha \log n)$, which follows since we need to maintain an $r \times c = 8/\alpha \log n$ size table.

3 Perfect Matchings in Bipartite Graphs

Recall that a *matching* in a graph $G = (V, E)$ is a subset of edges $M \subseteq E$ such that each vertex $v \in V$ is incident to at most one edge in M. In this lecture, we will consider the case where G is a bipartite graph and look at the Perfect Matching (PM) problem, which is to find a perfect matching in G, i.e., a matching M where each vertex is incident to *exactly* one edge in M. Note that trivially a perfect matching in G is also one of maximum cardinality.

Given a bipartite graph $G = (L, R, E)$, where $|L| = n = |R|$ and $m = |E|$ and assuming G has a perfect matching, we can convert a PM problem on G into an equivalent Max Flow problem by adding a source vertex s with edges (s, l) for each $l \in L$, a sink vertex t with edges (r, t) for each $r \in R$, and directing all edges from s to t. Using Ford-Fulkerson [FF56], we can solve this new flow problem in $O(mn)$ time and retrieve a perfect matching from it by taking all the saturated edges in E. Alternatively, another algoithm by Hopcroft and Karp [HK73] based on augmenting path techniques can solve PM in $O(m\sqrt{n})$ time. When G is d-regular for some $d > 0$ (i.e., each vertex has degree exactly d), there are also the following algorithms:

- For the special case where $d = 2^k$ for some $k \in \mathbb{Z}_{\geq 0}$, there is a deterministic $O(m)$ time algorithm by Gabow and Kariv [GK82].
- For general d, there is a randomized algorithm by Goel, Kapralov, and Khanna [GKK10] that has expected running time $O(n \log n)$.

We fill focus on these latter two algorithms.

3.1 Hall's Theorem

For a vertex v, let $N(v)$ denote the set of neighbors of v, and similarly for a subset of vertices U, let $N(U)$ denote the set of neighbors of all vertices in U (i.e., $N(U) = \bigcup_{u \in U} N(u)$). G being a d-regular graph is important as we can guarantee that it always has a perfect matching. This is based on Hall's Theorem.

Theorem 1 (Hall's Theorem)**.** *A perfect matching in a bipartite graph $G = (L, R, E)$ where $|L| = n\,|R|$ exists iff $|S| \leq |N(S)|$ for all subsets $S \subseteq L$.*

Proof. We prove the forward (perfect matching exists implies subset condition) and reverse direction (subset condition implies perfect matching exists):

- $\implies$: Suppose that there is some subset $S \subseteq L$ where $|S| > |N(S)|$. Then this would imply that all the vertices in S can't be matched, which is a contradiction as we assume that a perfect matching exists.
- $\impliedby$: We will prove this via the max-flow min-cut theorem. Suppose we run Ford-Fulkerson on the flow network induced by G until termination. By definition, upon termination there must be no s-t path in the residual graph. Let $\mathcal{L} \subseteq L$ and $\mathcal{R} \subseteq R$ be the set of vertices reachable by s in L and R, respectively, in the residual graph. Then, the s-t min-cut $C = \{s, \mathcal{L}, \mathcal{R}\}$ and its capacity is

$$\begin{aligned} |C| &= (n - |\mathcal{L}|) + |\mathcal{R}| + \left|N(\mathcal{L}) \cap \overline{\mathcal{R}}\right| \\ &= (n - |\mathcal{L}|) + |\mathcal{R}| + |N(\mathcal{L})| + |\mathcal{R}| \\ &= n + |N(\mathcal{L})| - |\mathcal{L}| \\ &\geq n \end{aligned}$$

  where $\overline{\mathcal{R}} = R \setminus \mathcal{R}$ and the last inequality follows by the assumption on the subsets $S \subseteq L$. The first inequality follows since there will be $n - |L|$ edges that cross C between s and L, $|\mathcal{R}|$ edges that cross C between R and t, and $\left|N(\mathcal{L}) + \overline{\mathcal{R}}\right|$ edges that cross C between L and R.

  Since trivially the maximum possible flow value is n (as there are only n edges out of s), this implies that the min-cut has capacity n and therefore there exists a flow with value n. Such a flow must also induce a perfect matching amongst the edges in E.

□

Corollary 2. *A perfect matching exists in any d-regular bipartite graph $G = (L, R, E)$.*

Proof. Consider any arbitrary set $S \subseteq L$. Since G is d-regular, there must be $d\,|S|$ edges from S to $N(S)$. By the same logic, there must also be $d\,|N(S)|$ edges from $N(S)$ to $N(N(S))$. Since $S \subseteq N(S)$, this implies that $d\,|N(S)| \geq d\,|S|$, which gives that $|N(S)| \geq |S|$. By Hall's Theorem, this implies that there always exists a perfect matching in G. □

3.2 A Deterministic $O(m)$ Time Algorithm

We now describe the simple Gabow-Kariv [GK82] algorithm for a special family of d-regular bipartite graphs. Given a d-regular bipartite graph $G = (L, R, E)$ where $d = 2^k$ for some $k \in \mathbb{Z}_{\geq 0}$, the algorithm is as follows.

Algorithm 1 GABOW-KARIV

Input: A d-regular bipartite graph G where $d = 2^k$ for some integer k

1: **if** $d = 1$ **then**
2: **return** G
3: Construct a Eulerian tour on G
4: Let G' be the subgraph induced by the edges in the tour that go in the forward direction. By construction, G' will be a $d/2$-regular graph.
5: GABOW-KARIV(G') *// Recurse with G' until $d = 1$*

A Eulerian tour is a directed circuit that visits every edge of G once and can be constructed in $O(m)$ time. Since G' is $d/2$-regular, it must have lost half the edges of G. Hence, the running time of this algorithm is $T(n) = O(m) + T(m/2) = O(m)$.

3.3 A Randomized $O(n \log n)$ Expected Time Algorithm

We now describe the randomized Goel-Kapralov-Khanna [GKK10] algorithm. Given a d-regular bipartite graph $G = (L, R, E)$ where $d > 0$ and $|L| = n = |R|$, the algorithm is to essentially do a randomized version of Ford-Fulkerson on the flow network induced by G. Specifically instead of using BFS or DFS to find an augmenting path in the residual network, a random walk is used, with the caveats that if t can be reached in one step, then this step is always taken, and that the walk does not travel back to s at any point.

In standard Ford-Fulkerson, there are n iterations, and it takes $O(m)$ time to do each iteration (using, for example, BFS), so the total runtime is $O(nm)$. In this randomized version of Ford-Fulkerson, there are still n iterations, but we will show that the $(n-k+1)^{\text{th}}$ iteration for $k \in [n]$ will take $O(n/k)$ time in expectation.

Lemma 3. *The $(n-k+1)^{th}$ iteration of the algorithm $k \in [n]$ will take $O(n/k)$ time in expectation.*

Proof. Choose some arbitrary $k \in [n]$ and consider the state of the flow network in its corresponding iteration. There will be exactly k vertices in both L and R that will be unmatched at the start of this iteration, i.e., they will have no flow coming in or going out of them. For the left bipartition L, let X_m and X_u denote the sets of matched vertices and unmatched vertices in L, respectively. Let Y_m and Y_u be defined similarly for the right bipartition R. Note that $|X_u| = k = |Y_u|$ and $|X_m| = n - k = |Y_m|$. Additionally, let $M(\cdot)$ denote the current matching. That is, $M(x) = y$ and $M(y) = x$ for $x \in X_m$, $y \in Y_m$ iff x and y are matched to each other.

For all vertices $v \in L \cup R$, let $b(v)$ be the expected number of back edges in the random walk starting from v until a node in Y_u is reached. Then, the expected time of this iteration will be at most the expected length of a random walk starting from s and finishing at some node in Y_u, which

is $O\left(b(s)\right)$ (more specifically it is something like $1 + 2b(s)$). Thus, if we can show that $b(s) \leq n/k$, then the proof is done.

To show this, note that by definition of b, it must be that

$$b(s) = \frac{1}{k} \sum_{x \in X_u} b(x)$$

as the random walk starting from s can only travel forward to a node in X_u and there are k nodes in X_u. Additionally,

$$b(y) = \begin{cases} 0 & \text{if } y \in Y_u \\ 1 + b(M(y)) & \text{if } y \in Y_m \end{cases} \quad (1) \qquad b(x) = \begin{cases} \frac{1}{d} \sum_{y \in N(x)} b(y) & \text{if } x \in X_u \\ \frac{1}{d-1} \sum_{y \in N(x) \setminus M(x)} b(y) & \text{if } x \in X_m \end{cases} \quad (2)$$

We can equivalently express Eq. (2) in the following form: for $x \in X_u$, $db(x) = \sum_{y \in N(x)} b(y)$ and for $x \in X_m$,

$$\begin{aligned} (d-1)b(x) &= \sum_{y \in N(x) \setminus M(x)} b(y) \\ &= -b(M(x)) + \sum_{y \in N(x)} b(y) \\ &= -(1 + b(x)) + \sum_{y \in N(x)} b(y) \\ db(x) &= \sum_{y \in N(x)} b(y) - 1. \end{aligned}$$

Hence, in general for any $x \in L$, we get that $db(x) = \sum_{y \in N(x)} b(y) - \mathbb{1}_{x \in X_m}$, where $\mathbb{1}_{x \in X_m}$ is a binary indicator variable that is 1 if $x \in X_m$ and 0 otherwise. Summing over all $x \in L$, we get that

$$\begin{aligned} d \sum_{x \in L} b(x) &= -\left|X_m\right| + \sum_{y \in R} db(y) \\ &= -\left|X_m\right| + d \sum_{y \in Y_m} \left(1 + b(M(y))\right) \\ &= \left(d-1\right)\left|X_m\right| + d \sum_{x \in X_m} b(x) \\ d \sum_{x \in X_u} b(x) &= \left(d-1\right)\left|X_m\right| \end{aligned}$$

Thus, we get that

$$b(s) = \frac{1}{k} \sum_{x \in X_u} b(x) = \frac{\left|X_m\right|}{k} \left(\frac{d-1}{d}\right) = \frac{n-k}{k} \left(\frac{d-1}{d}\right) \leq \frac{n}{k}$$

□

Summing over all the iterations, Lemma 3 implies that the runtime for this algorithm will be $O\left(n/n + n/n-1 + \ldots + n/2 + n/1\right) = O\left(nH_n\right) = O\left(n \log n\right)$ in expectation.

References

[CM05] Graham Cormode and Shan Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005. URL: https://dsf.berkeley.edu/cs286/papers/countmin-latin2004.pdf.

[FF56] Lester R. Ford and Delbert R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956. URL: https://www.cambridge.org/core/services/aop-cambridge-core/content/view/5D6E55D3B06C4F7B1043BC1D82D40764/S0008414X00036890a.pdf/maximal-flow-through-a-network.pdf.

[GK82] Harold N. Gabow and Oded Kariv. Algorithms for edge coloring bipartite graphs and multigraphs. *SIAM Journal on Computing*, 11(1):117–129, 1982. URL: https://dl.acm.org/doi/pdf/10.1145/800133.804346.

[GKK10] Ashish Goel, Michael Kapralov, and Sanjeev Khanna. Perfect matchings in $O(n \log n)$ time in regular bipartite graphs. In *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC 2010)*, pages 39–46, 2010. URL: https://arxiv.org/abs/0909.3346.

[HK73] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973. URL: https://web.eecs.umich.edu/~pettie/matching/Hopcroft-Karp-bipartite-matching.pdf.