Lecture 1. Data Streams

Lecturer: S. Muthu Muthukrishnan   Scribes: Anıl Ada and Jacobo Torán

We start with a puzzle.
Puzzle 1: Given an array $A[1..n]$ of $\log n$ bit integers, sort them in place in $O(n)$ time.

1.1 Motivation

The algorithms we are going to describe act on massive data that arrive rapidly and cannot be stored. These algorithms work in few passes over the data and use limited space (less than linear in the input size). We start with three real life scenarios motivating the use of such algorithms.

Example 1: Telephone call. Every time a cell-phone makes a call to another phone, several calls between switches are being made until the connection can be established. Every switch writes a record for the call over approx. 1000 Bytes. Since a switch can receive up to 500 million calls a day, this adds up to something like 1 Terabyte per month information. This is a massive amount of information but has to be analyzed for different purposes. An example is searching for drop calls trying to find out under what circumstances such drop calls happen. It is clear that for dealing with this problem we do not want to work with all the data, but just want to filter with a few passes the useful information.

Example 2: The Internet. The Internet is made of a network of routers connected to each other, receiving and sending IP packets. Each IP packet contains a packet log including its source and destination addresses as well as other information that is used by the router to decide which link to take for sending it. The packet headers have to be processed at the rate at which they flow through the router. Each package takes about 8 nanoseconds to go through a router and modern routers can handle a few million packets per second. Keeping the whole information would need more than one Terabyte information per day and router. Statistical analysis of the traffic through the router can be done, but this has to be performed on line at nearly real time.

Example 3: Web Search. Consider a company for placing publicity in the Web. Such a company has to analyze different possibilities trying to maximize for example the number of clicks they would get by placing an add for a certain price. For this they would have to analyze large amounts of data including information on web pages, numbers of page visitors, add prices and so on. Even if the company keeps a copy of the whole net, the analysis has to be done very rapidly since this information is continuously changing.

Before we move on, here is another puzzle.
Puzzle 2: Suppose there are n chairs around a circular table that are labelled from 0 to $n-1$ in order. So chair i is in between chairs $i-1$ and $i+1$ mod n. There are two infinitely smart players

that play the following game. Initially Player 1 is sitting on chair 0. The game proceeds in rounds. In each round Player 1 chooses a number i from $\{1, 2, \ldots, n-1\}$, and then Player 2 chooses a direction left or right. Player 1 moves in that direction i steps and sits on the corresponding chair. Player 1's goal is to sit on as many different chairs as possible while Player 2 is trying to minimize this quantity. Let $f(n)$ denote the maximum number of different chairs that Player 1 can sit on. What is $f(n)$?

Here are the solutions for some special cases.

$$\begin{aligned}
f(2) &= 2 \\
f(3) &= 2 \\
f(4) &= 4 \\
f(5) &= 4 \\
f(7) &= 6 \\
f(8) &= 8 \\
f(p) &= p-1 \quad \text{for } p \text{ prime} \\
f(2^k) &= 2^k
\end{aligned}$$

1.2 Count-Min Sketch

In this section we study a concrete data streaming question. Suppose there are n items and let $F[1..n]$ be an array of size n. Index i of the array will correspond to item i. Initially all entries of F are 0. At each point in time, either an item i is added, in which case we increment $F[i]$ by one, or an item is deleted, in which case we decrement $F[i]$ by one. Thus, $F[i]$ equals the number of copies of i in the data, or in other words, the frequency of i. We assume $F[i] \geq 0$ at all times.

As items are being added and deleted, we only have $O(\log n)$ space to work with, i.e. logarithmic in the space required to represent F explicitly. Here we think of the entries of F as words and we count space in terms of number of words.

We would like to estimate $F[i]$ at any given time. Our algorithm will be in terms of two parameters ϵ and δ. With $1-\delta$ probability, we want the error to be within a factor of ϵ.

The algorithm is as follows. Pick $\log(\frac{1}{\delta})$ hash functions $h_j : [n] \to [e/\epsilon]$ chosen uniformly at random from a family of pair-wise independent hash functions. We think of $h_j(i)$ as a bucket for i corresponding to the jth hash function. We keep a counter for each bucket, count$(j, h_j(i))$. Initially all buckets are empty, or all counters are set to 0. Whenever an item i is inserted, we increment count$(j, h_j(i))$ by 1 for all j. Whenever an item i is deleted, we decrement count$(j, h_j(i))$ by 1 for all j (see Figure 1.1). Our estimation for $F[i]$, denoted by $\tilde{F}[i]$, will be $\min_j$ count$(j, h_j(i))$.

Claim 1. *Let* $\|F\| = \sum_i F[i]$.

1. $\tilde{F}[i] \geq F[i]$.
2. $\tilde{F}[i] \leq F[i] + \epsilon\|F\|$ *with probability at least* $1-\delta$.

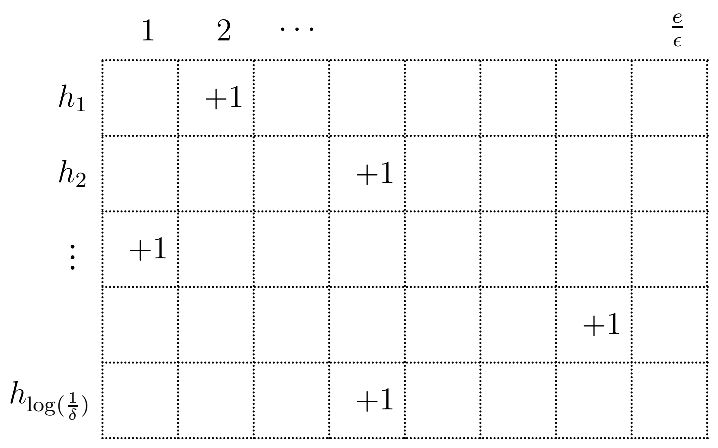

Figure 1.1: Each item is hashed to one cell in each row.

Proof. The first part is clear. For the second part, denote by X_{ji} the contribution of items other than i to the $(j, h_j(i))$th cell (bucket) of Figure 1.2. It can be easily shown that

$$\mathbf{E}[X_{ji}] = \frac{\epsilon}{e}\|F\|.$$

Then by Markov's inequality,

$$\begin{aligned}\Pr\left\{\tilde{F} > F[i] + \epsilon\|F\|\right\} &= \Pr\left\{\forall j\ F[i] + X_{ji} > F[i] + \epsilon\|F\|\right\} \\ &= \Pr\left\{\forall j\ X_{ji} > e\mathbf{E}[X_{ji}]\right\} \\ &\leq \left(\frac{1}{2}\right)^{\log 1/\delta}\end{aligned}$$

□

Thus, we conclude that we can estimate $F[i]$ within an error of $\epsilon\|F\|$ with probability at least $1-\delta$ using $O((1/\epsilon)\log(1/\delta))$ space. Observe that this method is not effective for estimating $F[i]$ for small $F[i]$. On the other hand, in most applications, one is interested in estimating the frequency of high frequency objects.

It is possible to show that the above is tight, with respect to the space requirements, using a reduction from the communication complexity problem Index. In this problem, Alice holds an n bit string $x \in \{0,1\}^n$ and Bob holds a $\log n$ bit string $y \in \{0,1\}^{\log n}$. We view Bob's input as an integer $i \in [n]$. We consider the one-way probabilistic communication model. Therefore only Alice is allowed to send Bob information. Given the information from Alice, Bob needs to determine the value x_i. In this model, it is well known that Alice needs to send $\Omega(n)$ bits in order for Bob to determine x_i with constant probability greater than 1/2.

Lemma 1. *In order to estimate $F[i]$ within an error of $\epsilon\|F\|$ with constant probability, one needs to use $\Omega(1/\epsilon)$ space.*

Proof. Given an instance of the Index problem (x, y), where x denotes Alice's input, y denotes Bob's input and $|x| = n$, choose ϵ such that $n = \frac{1}{2\epsilon}$. Construct the array $F[0..\frac{1}{2\epsilon}]$ as follows. If $x_i = 1$ then set $F[i] = 2$ and if $x_i = 0$ then set $F[i] = 0$ and increment $F[0]$ by 2 (initially $F[0] = 0$). With this construction, clearly we have $\|F\| = 1/\epsilon$. Suppose we can estimate $F[i]$ within error $\epsilon\|F\| = 1$ with constant probability and s space. This means we can determine the value of x_i: if the estimate for $F[i]$ is above 1 then $x_i = 1$ and $x_i = 0$ otherwise. Now the $\Omega(n) = \Omega(1/\epsilon)$ lower bound on the communication complexity of Index implies a lower bound of $\Omega(1/\epsilon)$ for s. $\square$

Homework: Given a data stream as an array $A[1..n]$, how can we estimate $\sum_i A[i]^2$? If we are given another data stream $B[1..n]$, how can we estimate $\sum_i A[i]B[i]$?

References: [CM04], [Mut09]

Lecture 2. Streaming Algorithms via Sampling

Lecturer: S. Muthu Muthukrishnan Scribes: Faith Ellen and Toniann Pitassi

2.1 Estimating the number of distinct elements

This lecture presents another technique for streaming algorithms, based on sampling.

Definition 1. *Let $a_1, a_2, \ldots, a_n$ denote a stream of items from some finite universe $[1..m]$. Let $D_n = |\{a_1, a_2, \ldots, a_n\}|$ be the number of distinct elements in the stream, and let U_n be the number of unique items in the stream, i.e. the number of items that occur exactly once.*

Let F be the frequency vector, where $F[i]$ is the number of times that item i occurs in the stream, for each $i \in [1..m]$. Then D_n is the number of nonzero entries in the frequency vector F and U_n is the number of entries of F with value 1.

Our goal is to get good estimates for U_n/D_n and D_n.

2.1.1 Estimating U_n/D_n

First we will try to estimate U_n/D_n.

We assume that n is known. We can easily choose an item uniformly at random from the stream, by choosing each item with probability $1/n$. Doing this k times in parallel gives us a sample of size k. The problem with this approach (uniform sampling from the data stream) is that heavy items, i.e. those with high frequency, are likely to appear many times. Since each such item doesn't contribute to U_n and only contributes to D_n once, such a sample is not helpful for estimating U_n/D_n.

Instead, we would like to be able to sample nearly uniformly from the set of (distinct) items in the stream, i.e. element x is chosen with probability close to $1/D_n$.

To do this, let h be a permutation of the universe chosen uniformly at random from among all such permutations. The idea is to remember the item s in the stream with the smallest value of $h(s)$ seen so far and the number of times this item has occurred. Specifically, as we see each item a_i in the stream, we compute $h(a_i)$. If $h(a_i) < h(s)$ (or $i = 1$), then s is set to a_i and $c(s)$ is set to 1. If $h(a_i) = h(s)$, then increment $c(s)$. If $h(a_i) > h(s)$, then do nothing. Note that, for any subset S of the universe, each item in S is equally likely to be mapped to the smallest value by h among all the elements in S. In particular, each element in the set of items in the stream has probability $1/D_n$ of being chosen (i.e. mapped to the smallest value by h) and, thus, being the value of s at the end of the stream. At any point in the stream, $c(s)$ is the number of times s has occurred so far in the stream, since we start counting it from its first occurrence.

Doing this k times independently in parallel gives us a collection of samples $s_1, \ldots, s_k$ of size k. We will choose $k = O(\log(1/\delta)/\epsilon^2)$. Let $c_1, \ldots, c_k$ be the number of times each of these items occurs in the stream. Our estimate for U_n/D_n will be $\#\{i \mid c_i = 1\}/k$. Since Prob[$c_i = 1$] $= U_n/D_n$, using Chernoff bounds it can be shown that with probability at least $(1-\delta)$, $(1-\epsilon)U_n/D_n \leq \#\{i \mid c_i = 1\}/k \leq (1+\epsilon)U_n/D_n$. Thus, $\#\{i \mid c_i = 1\}/k$ is a good estimator for U_n/D_n.

It's not necessary to have chosen a random permutation from the set of all permutations. In fact, simply storing the chosen permutation takes too much space. It suffices to randomly choose a hash function from a family of functions such that, for any subset of the universe, every element in the subset S has the smallest hashed value for the same fraction $(1/|S|)$ of the functions in the family. This is called minwise hashing and it was defined in a paper by Broder, et al. They proved that any family of hash functions with the minwise property must be very large.

Indyk observed that an approximate version of the minwise property is sufficient. Specifically, for any subset S of the universe, each element in the subset has the smallest hashed value for at least a fraction $1/((1+\epsilon)|S|)$ of the functions in the family. There is a family of approximately minwise hash functions of size $n^{O(\log n)}$, so $(\log n)^2$ bits are sufficient to specify a function from this family.

An application for estimating U_n/D_n comes from identifying distributed denial of service attacks. One way these occur is when an adversary opens many connections in a network, but only sends a small number of packets on each. At any point in time, there are legitimately some connections on which only a small number of packets have been sent, for example, for newly opened connections. However, if the connections on which only a small number of packets have been sent is a large fraction of all connections, it is likely a distributed denial of service attack has occurred.

2.1.2 Estimating D_n

Now we want to estimate D_n, the number of distinct elements in $a_1, \ldots, a_n \in [1..m]$. Suppose we could determine, for any number t, whether $D_n < t$. To get an approximation to within a factor of 2, we could estimate D_n by determining whether $D_n < 2^i$ for all $i = 1, \ldots, \log_2 m$. Specifically, we estimate D_n by 2^k, where $k = \min\{i | c_i = 0\}$. If we do these tests in parallel, the time and space both increase by a factor of $\log_2 m$.

To determine whether $D_n < t$, randomly pick a hash function h from $[1..m]$ to $[1..t]$. Let c be the number of items that hash to bucket 1. We'll say that $D_n < t$ if $c = 0$ and say that $D_n \geq t$ if $c > 0$. To record this as we process the stream requires a single bit that tells us whether c is 0 or greater than 0. Specifically, for each item a_i, if $h(a_i) = 1$, then we set this bit to 1.

If $D_n < t$, then the probability that no items in the stream hash to bucket 1 (i.e. that $c = 0$) is $(1 - 1/t)^{D_n} > (1 - 1/t)^t \approx 1/e$. If $D_n > 2t$, then the probability no items in the stream hash to bucket 1 (i.e. that $c = 0$) is $(1-1/t)^{D_n} < (1-1/t)^{2t} \approx 1/e^2$. More precisely, using a Taylor series approximation, $Pr[c = 0 | D_n \geq (1+\epsilon)t] \leq 1/e - \epsilon/3$ and $Pr[c = 0 | D_n < (1-\epsilon)t] \geq 1/e + \epsilon/3$.

To improve the probability of being correct, repeat this several times in parallel and take majority answer. This give the following result.

Theorem 1. *It is possible to get an estimate t for D_n using $O[(1/\epsilon^2)\log(1/\delta)\log m]$ words of space such that Prob$[(1-\epsilon)t \leq D_n < (1+\epsilon)t] \geq 1-\delta$.*

2.2 Extensions for inserts and deletes

Exercise Extend the algorithm for approximating the number of distinct items to allow the stream to include item deletions as well as item insertions.

The algorithm described above for sampling nearly uniformly from the set of (distinct) items in the stream doesn't extend as easily to allow deletions. The problem is that if all occurrences of the item with the minimum hash value are deleted at some point in the stream, we need to replace that item with another item. However, information about other items that have appeared in the stream and the number of times each has occurred has been thrown away. For example, suppose in our sampling procedure all of the samples that we obtain happen to be items that are inserted but then later deleted. These samples will clearly be useless for estimating the quantities of interest.

We'll use a new trick that uses sums in addition to counts. Choose $\log_2 m$ hash functions $h_j : [1..m]$ to $[1..2^j]$, for $j = 1, \ldots, \log_2 m$. For the multiset of items described by the current prefix of the stream, we will maintain the following information, for each $j \in [1.. \log_2 m]$:

1. D'_j, which is an approximation to the number of distinct items that are mapped to location 1 by h_j,
2. S_j, which is the exact sum of all items that are mapped to location 1 by h_j, and
3. C_j, which is the exact number of items that are mapped to location 1 by h_j.

For each item a_i in the stream, if $h_j(a_i) = 1$, then C_j is incremented or decremented and a_i is added to or subtracted from S_j, depending on whether a_i is being inserted or deleted.

The number of distinct elements is dynamic: at some point in the stream it could be large and then later on it could be small. Thus, we have $\log_2 m$ hash functions and maintain the information for all of them.

If there is a single distinct item in the current multiset that is mapped to location 1 by h_j, then S_j/C_j is the identity of this item. Notice that, because S_j and C_j are maintained exactly, this works even if the number of distinct items in the current multiset is very large and later becomes 1.

Suppose that D'_j is always bounded below and above by $(1 - \epsilon)$ and $(1 + \epsilon)$ times the number of distinct items hashed to location 1 by h_j, respectively, for some constant $\epsilon < 1$. Then there is only 1 distinct item hashed to location 1 by h_j, if and only if $D'_j = 1$.

If $D'_j = 1$, then S_j/C_j can be returned as the sample. If there is no j such that $D_j = 1$, then no sample is output. If the hash functions are chosen randomly (from a good set of hash functions), then each distinct item is output with approximately equal probability.

Instead of getting just one sample, for many applications, it is better to repeat this $(1/\epsilon^2)\log(1/\delta)$ times in parallel, using independently chosen hash functions. We'll call this the sampling data structure.

Yesterday, we had an array $F[1..m]$ keeping track of the number of occurrences of each of the possible items in the universe $[1..m]$. We calculated the heavy hitters (i.e. items i whose number of occurrences, $F[i]$, is at least some constant fraction of the total number of occurrences, $\sum_{i=1}^{m} F[i]$) and estimated $F[i]$, $\sum_{i=1}^{m} F[i]$, $\sum_{i=1}^{m} F[i]^2$, and quantiles. Today, we estimated the number of distinct elements, i.e., $\#\{i \mid F(i) > 0\}$. The following definition gives a more succinct array for answering many of the questions that we've looked at so far (i.e., distinct elements, quantiles, number of heavy hitters.)

Definition 2. *Let $I[1..k]$ be an array, where $I[j]$ is the number of items that appear j times, i.e. the number of items with frequency j, and $k \leq n$ is the maximum number of times an item can occur. For example, $I[1]$ is the number of unique items, items that appear exactly once. Heavy hitters are items that have frequency at least $\phi \sum_{i=1}^{k} I[i]$, for some constant ϕ.*

We'd like to apply the CM sketch directly to the I array. The problem is how to update I as we see each successive item in the stream. If we know how many times this item has previously been seen, we could decrement that entry of I and increment the following entry. However, we don't know how to compute this directly from I.

The sampling data structure as described above, which can be maintained as items are added and deleted, allows the entries of the I array to be approximated.

2.3 Homework Problems

1. A (directed or undirected) graph with n vertices and $m < n^2$ distinct edges is presented as a stream. Each item of the stream is an edge, i.e. a pair of vertices (i, j). Each edge may occur any number of times in the stream. Edge deletions do not occur. Let d_i be the number of distinct neighbors of vertex i. The goal is to approximate $M_2 = \sum_i d_i^2$. It is called M_2 since it is analogous to F_2 from yesterday. The key difference is that M_2 only counts a new item if it is distinct, i.e. it hasn't appeared before.

   The best known algorithm for this problem uses space $O((1/\epsilon^4)\sqrt{n}\log n)$. It can be obtained by combining two sketches, for example, the CM sketch and minwise hashing. (In general, the mixing and matching of different data structures can be useful.) The solution to this problem doesn't depend on the input being a graph. The problem can be viewed as an array of values, where each input increments two array entries.

   Although the space bound is sublinear in n, we would like to use only $(\log n)^{O(1)}$ space. This is open.

2. **Sliding window version of sampling**:
   Input a sequence of items, with no deletions. Maintain a sample uniformly chosen from among the set of distinct items in the last w items. The space used should be $O(\log w)$.

   Note that if minwise hashing is used and the last copy of the current item with minimum hashed value is about to leave the window, a new item will need to be chosen.

Lecture 3. Some Applications of CM-Sketch

Lecturer: S. Muthu Muthukrishnan   Scribes: Arkadev Chattopadhyay and Michal Koucký

3.1 Count-Min Sketch

Prelude: *Muthu is a big fan of movies. What we will see today is like the movie "The Usual Suspects" with Kevin Spacey: 12 years of research fit into one sketch. It will also take some characteristics of another great movie "Fireworks" by Takashi Beat Kitano. That movie has three threads which in the end meet. This lecture will have three threads.*

Problem 1 (from yesterday): Sort an array $A[1,\ldots,n]$ of $\log_2 n$-bit integers in place in linear time.

Solution idea: With a bit of extra space, say $O(\sqrt{n})$, one could run $\sqrt{n}$-way radix sort to sort the array in $O(n)$ time. Where do we get this extra space? Sort/re-arrange the elements according to the highest order bit. Now, we can save a bit per element by representing the highest order bit implicitly. This yields $O(n/\log n)$ space to run the radix sort. The details are left to the reader. There are also other solutions.

Problem 2: We have a stream of items from the universe $\{1,\ldots,n\}$ and we want to keep a count $F[x]$ of every single item x. We relax the problem so that we do not have to provide a precise count but only some approximation $\tilde{F}[x]$:

$$F[x] \le \tilde{F}[x] \le F[x] + \epsilon \sum_{i=1}^{n} F[i].$$

Solution: For t that will be picked later, let $q_1,\ldots,q_t$ are the first t primes. Hence, $q_t \approx t \ln t$. We will keep t arrays of counters $F_j[1,\ldots,q_j]$, $j = 1,\ldots,t$. All the counters will be set to zero at beginning and whenever an item x arrives we will increment all counters $F_j[x \mod q_j]$ by one. Define $\tilde{F}[x] = \min_{j=1,\ldots,t} F_j[x \mod q_j]$.

Claim 2. *For any* $x \in \{1,\ldots,n\}$,

$$F[x] \le \tilde{F}[x] \le F[x] + \frac{\log_2 n}{t} \sum_{i=1}^{n} F[i].$$

Proof. The first inequality is trivial. For the second one note that for any $x' \in \{1,\ldots,n\}$, $x' \neq x$, $x' \mod q_j = x \mod q_j$ for at most $\log_2 n$ different j's. This is implied by Chinese Reminder Theorem. Hence, at most $\log_2 n$ counters corresponding to x may get incremented as a result of an arrival of x'. Since this is true for all $x' \neq x$, the counters corresponding to x may get over-counted by at most $\log_2 n \cdot \sum_{x' \in \{1,\ldots,n\} \setminus \{x\}} F[x']$ in total. On average they get over-counted by at most $\frac{\log_2 n}{t} \cdot \sum_{x' \in \{1,\ldots,n\} \setminus \{x\}} F[x']$, so there must be at least one of the counters corresponding to x that gets over-counted by no more than this number. $\square$

We choose $t = \frac{\log_2 n}{\epsilon}$. This implies that we will use space $O(t^2 \log t) = O(\frac{\log^2 n}{\epsilon^2} \log \log n)$, where we measure the space in counters. This data structure is called Count-Min Sketch (CM sketch) and was introduced in *G. Cormode, S. Muthukrishnan: An improved data stream summary: the count-min sketch and its applications. J. Algorithms 55(1):58-75 (2005).* It is actually used in Sprinter routers.

Intermezzo: *Al Pacino's second movie: Godfather (depending on how you count).*

Problem 3: We have two independent streams of elements from $\{1,\ldots,n\}$. Call the frequency (count) of items in one of them $A[1,\ldots,n]$ and $B[1,\ldots,n]$ in the other one. Estimate $X = \sum_{i=1}^{n} A[i] \cdot B[i]$ with additive error $\epsilon \cdot ||A||_1 \cdot ||B||_1$.

Solution: Again we use CM sketch for each of the streams: $T^A = (T_j^A[1,\ldots,q_j])_{j=1,\ldots,t}$ and $T^B = (T_j^B[1,\ldots,q_j])_{j=1,\ldots,t}$, and we output estimate

$$\tilde{X} = \min_{j=1,\ldots,t} \sum_{k=1} T_j^A[k] \cdot T_j^B[k].$$

Claim 3.

$$X \leq \tilde{X} \leq X + \frac{\log_2 n}{t} ||A||_1 \cdot ||B||_1.$$

Proof. The first inequality is trivial. For the second one note again that for any $x, x' \in \{1,\ldots,n\}$, $x' \neq x$, $x' \mod q_j = x \mod q_j$ for at most $\log_2 n$ different j's. This means that the term $A[x] \cdot B[x']$ contributes only to at most $\log_2 n$ of the sums $\sum_{k=1} T_j^A[k] \cdot T_j^B[k]$. Hence again, the total over-estimate is bounded by $\log_2 n \cdot ||A||_1 \cdot ||B||_1$ and the average one by $\frac{\log_2 n}{t} ||A||_1 \cdot ||B||_1$. Clearly, there must be some j for which the over-estimate is at most the average. $\square$

Choosing $t = \frac{\log_2 n}{\epsilon}$ gives the required accuracy of the estimate and requires space $O(\frac{\log^2 n}{\epsilon^2} \log \log n)$.

Intermezzo: *Mario is not happy: for vectors $A = B = (1/n, 1/n, \ldots, 1/n)$ the error is really large compare to the actual value. Well, the sketch works well for vectors concentrated on few elements.*

Problem 4: A single stream $A[1,\ldots,n]$ of elements from $\{1,\ldots,n\}$. Estimate $F_2 = \sum_{i=1}^{n} (A[i])^2$.

Solution: The previous problem provides a solution with an additive error $\epsilon ||A||_1^2$. We can do better. So far, our CM sketch was deterministic, based on arithmetic modulo primes. In general one can take hash functions $h_1, \ldots, h_t : \{1,\ldots,n\} \rightarrow \{1,\ldots,w\}$ and keep a set of counters $T_j[1,\ldots,w], j = 1,\ldots,t$. On arrival of item x one increments counters $T_j[h_j(x)]$, $j = 1,\ldots,t$. The hash functions h_j are picked at random from some suitable family of hash functions. In such a case one wants to guarantee that for a given stream of data, the estimates derived from this CM sketch are good with high probability over the random choice of the hash functions. For the problem of estimating F_2 we will use a family of four-wise independent hash functions. Our sketch will consists of counters $T_j[1,\ldots,w]$, $j=1,\ldots,t$, for even w. To estimate F_2 we calculate for each j

$$Y_j = \sum_{k=1}^{w/2} (T_j[2k-1] - T_j[2k])^2,$$

and we output the median $\tilde{X}$ of Y_j's.

Claim 4. *For $t = O(\ln \frac{1}{\delta})$ and $w = \frac{1}{\epsilon^2}$,*

$$|\tilde{X} - F_2| \leq \epsilon F_2$$

with probability at least $1-\delta$.

Proof. Fix $j \in \{1,\ldots,t\}$. First observe that

$$E[Y_j] = F_2.$$

To see this let us look at the contribution of terms $A[x]\cdot A[y]$ for $x,y \in \{1,\ldots,n\}$ to the expected value of Y_j. Let us define a random variable $f_{x,y}$ so that for $x \neq y$, $f_{x,y} = 2$ if $h_j(x) = h_j(y)$, $f_{x,y} = -2$ if $h_j(x) = 2k = h_j(y)+1$ or $h_j(y) = 2k = h_j(x)+1$ for some k, and $f_{x,y} = 0$ otherwise. For $x = y$, $f_{x,y} = 1$ always. Notice for $x \neq y$, $f_{x,y} = 2$ with probability $1/w$ and also $f_{x,y} = -2$ with probability $1/w$. It is straightforward to verify that $Y_j = \sum_{x,y} f_{x,y} A[x]\cdot A[y]$. Clearly, if $x \neq y$ then $E[f_{x,y}] = 0$. By linearity of expectation,

$$E[Y_j] = \sum_{x,y} E[f_{x,y}]\cdot A[x]\cdot A[y] = F_2.$$

Now we show that

$$Var[Y_j] \leq \frac{8}{w} F_2^2.$$

$$\begin{aligned}
Var[Y_j] &= E\left[\left(\sum_{x,y} f_{x,y} A[x]\cdot A[y] - \sum_{x} A[x]\cdot A[x]\right)^2\right] \\
&= E\left[\left(\sum_{x\neq y} f_{x,y} A[x]\cdot A[y]\right)^2\right] \\
&= E\left[\sum_{x\neq y, x'\neq y'} f_{x,y}\cdot f_{x',y'}\cdot A[x]\cdot A[y]\cdot A[x']\cdot A[y']\right].
\end{aligned}$$

For $(x,y) \neq (x',y'), x \neq y, x' \neq y'$

$$E\left[f_{x,y}\cdot f_{x',y'}\cdot A[x]\cdot A[y]\cdot A[x']\cdot A[y']\right] = 0$$

because of the four-wise independence of h_j. For $(x, y) = (x', y'), x \neq y, x' \neq y'$

$$\begin{aligned} E\left[f_{x,y} \cdot f_{x',y'} \cdot A[x] \cdot A[y] \cdot A[x'] \cdot A[y']\right] &= \left(\frac{1}{w} \cdot 4 + \frac{1}{w} \cdot 4\right) A[x]^2 \cdot A[y]^2 \\ &= \frac{8}{w} \cdot A[x]^2 \cdot A[y]^2. \end{aligned}$$

Hence,

$$Var[Y_j] \leq \frac{8}{w}\left(\cdot \sum_x A[x]^2\right)^2 = \frac{8}{w} F_2^2.$$

Applying Chebyshev's inequality, and the fact that $w = \frac{1}{\epsilon^2}$ we get,

$$\Pr\left[|Y_j - F_2| \geq \epsilon F_2\right] \leq \frac{1}{8}$$

Since each hash function is chosen independently, we can apply Chernoff bound to conclude that taking $O(\log(1/\delta))$ hash functions is enough to guarantee that the median of the Y_j's gives an approximation of F_2 with additive error less than ϵF_2 with probability at least $1 - \delta$.

□

Definition: Let $A[1, \ldots, n]$ be the count of items in a stream. For a constant $\phi < 1$, item i is called a *ϕ-heavy hitter* if $A[i] \geq \phi \sum_{j=1}^{n} A[j]$.

Problem 5: Find all ϕ-heavy hitters of a stream.

Solution: First we describe a procedure that finds all ϕ-heavy hitters given access to any sketching method. In this method, we form $\log n$ streams $B_0, \ldots B_{\log n - 1}$ in the following way:

$$B_i[j] = \sum_{k=(j-1)2^i+1}^{j2^i} A[k]$$

This means that $B_i[j] = B_{i-1}[2j-1] + B_{i-1}[2j]$. When a new element arrives in stream A, we update simultaneously the sketch of each B_i. Finally, in order to find ϕ-heavy hitters of A, we do a binary search by making hierarchical point queries on the $\log n$ streams that we created, in the following way: we start at $B_{\log n-1}$. We query $B_{\log n-1}[1]$ and $B_{\log n-1}[2]$. If $B_{\log n-1}[1] \geq \phi \sum_{k=1}^{n} A[k] = T$ (say), then we recursively check the two next level nodes $B_{\log n-2}[1]$ and $B_{\log n-2}[2]$ and so on.

In other words, the recursive procedure is simply the following: if $B_i[j] \geq T$, then descend into $B_i[2j-1]$ and $B_i[2j]$. If $B_i[j] < T$, then this path of recursion is terminated. If $i = 0$, and $B_i[j] \geq T$, then we have found a heavy hitter.

Clearly, this procedure finds all heavy hitters if the point queries worked correctly. The number of queries it makes can be calculated in the following way: for each i, B_i can have at most $1/\phi$ heavy hitters and the algorithm queries at most twice the number of heavy-hitters of a stream. Thus, at most $(2 \log n/\phi)$ point queries are made.

If we implement the probabilistic version of CM-sketch, as described in the solution to Problem 4 above, it is not hard to see that each point-query can be made to return an answer with positive additive error bounded by ϵ, with probability $1 - \delta$, by using roughly $\log(1/\delta)$ pairwise[1] independent hash functions, where each hash function has about $O(1/\epsilon)$ hash values. Such a sketch uses $O\big(\frac{1}{\epsilon}\log(1/\delta)\big)$ space.

For the application to our recursive scheme here for finding heavy hitters, we want that with probability at least $(1 - \delta)$, none of the at most $(2\log n/\phi)$ queries fail[2]. Thus, using probabilistic CM-sketch with space $O\big(\frac{1}{\epsilon}\log n \log\big(\frac{2\log n}{\phi\delta}\big)\big)$ and probability $(1 - \delta)$, we identify all ϕ-heavy hitters and not return any element whose count is less than $(\phi - \epsilon)$-fraction of the total count.

Reference: [CM04]

[1]Note for making point queries we just need pairwise independence as opposed to 4-wise independence used for estimating the second moment in the solution to Problem 4 before.

[2]A query fails if it returns a value with additive error more than an ϵ-fraction of the total count.