CS5234: Combinatorial and Graph Algorithms Lecture 4

Tutorial: Week 4

Lecturer: Seth Gilbert *September 6, 2018*

1 Twitter Hashtags

You have access to the main Twitter feed, which consists of about 350,000 tweets per minute. Tweets, consisting of at most 150 characters often contain hashtags, i.e., strings that begin with a hash ('#') character. Your goal is to determine, approximately, how many hashtags are used in each tweet. Unfortunately, you do not know how many tweets there are per day, though it is approximately 500 million. Since there are so many tweets, you cannot process every tweet. Instead, you can ask your Twitter client to skip tweets. For example, you might ask to skip the next 524 *tweets, deliver tweet number* 525.

In order to process as few tweets as possible, we are going to use a sampling approach. The first problem is how to sample from the stream of tweets. If we knew that there were M tweets, then we could simply take each tweet with probability $1/M$. Unfortunately, we do not know how many tweets there are. Thus we are going to use reservoir sampling.

Reservoir sampling. Imagine we want to sample k items so that each item is included in our sample with probability k/T, where T is the number of elements in the stream. We begin by saving the first k items into a set S. Starting with item $k+1$, we save item i with probability k/i. Whenever we choose to save an item, we randomly delete an item from the set S so that we have space to add the new item, ensuring that we always have k items in the set.

We can prove that this works by induction. Notice that immediately after item k, the set S contains each of the first k elements with probability $k/k = 1$. For the inductive step, consider element i. By inductive assumption, we know that the set S contains a uniform random subset of size k of the first $i-1$ elements in the stream.

Now we show that after element i, each item in the stream from 1 to i is included in the set S with probability k/i: Element i is included in the set with probability k/i by definition. Each element from 1 to $i-1$ was in the set S with probability $k/(i-1)$ and remains in the set with probability $1-(k/i)(1/k) = (i-1)/i$. Thus, the probability that the element remains in the set S is $(k/(i-1))((i-1)/i) = k/i$, as needed.

Finally, observe that we do not need to look at every tweet. We only need to look at tweets that are added to the set S. We can skip all the intervening tweets. (That is, we can simulate the probabilistic choices without seeing the tweets, only requesting that the twitter client deliver a tweet if we decide to save it.) The expected number of tweets that we will save is $\sum_{i=1}^{T}(k/i) = O(k \log T)$. Thus, we only need to process $O(k \log T)$ tweets in order to find a sample of size k.

Average hashtags. Next, we want to use our sample to determine the average number of hashtags per tweet. Notice that each hashtag consists of at least two characters: the hash character and one other character. So there are at most 70 hashtags in a tweet. Our basic algorithm is to use reservoir sampling to collect a sample S of size s; we then calculate $(1/s)\sum_{x \in S} hashtags(x)$, where the function $hashtags(x)$ counts the number of hashtags in x.

Now, we analyze this algorithm and show that the answer is close to A, the real average number of hashtags per tweet. In doing so, we will also determine the sample size s. Let X_i be the number of hashtags in the ith sampled tweet, and let $X = \sum(X_i)$. The expected value $\mathrm{E}[X_i] = A$ and $\mathrm{E}[X] = sA$.

Notice that the answer returned is X/s. We want to show that $|(X/s) - A| \geq \epsilon$ is small, i.e., $|X - sA| \geq \epsilon s$ is small.

This would imply that X/s is an additive ϵ approximation of A. Since $\mathrm{E}[X] = sA$, we can show:

$$\begin{aligned} \Pr[|X - sA| \geq \epsilon s] &= \\ \Pr[|X - \mathrm{E}[X]| \geq \epsilon s] &\leq 2e^{-\epsilon^2 s^2/(3 \cdot 70 \cdot s)} \\ &\leq 2e^{-s\epsilon^2/210} \end{aligned}$$

This implies that if we set $s = 420/\epsilon^2$, we will find an answer that is within ϵ of A with probability at least $2/3$.

2 Morris Counters

Assume that there are n items in a stream. In this problem, we looked at the question of how to count, approximately, the total number of items in a stream using only $\log \log n$ bits. The basic algorithm (which is known as the Morris Counter) suggested was as follows:

1. $X = 0$
2. For each item in the stream, increment X with probability $p(x)$.
3. Return $f(X)$.

The goal was to choose functions $p(x)$ and $f(x)$ to make this work well. For example, if you choose $p(x) = 1/2$ and $f(x) = 2x$, then the expected value of the algorithm is exactly equal to the number of items in the stream, and the total sizeis $\log(n/2) = \log(n) - 1$, i.e., it saves one bit.

It turns out that you can set $p(x) = 1/2^x$, i.e., increment the counter with exponentially decreasing probabilities. As x gets bigger, you will increment it less often.

Intuitively, you might guess that the size of the counter is about $\log \log(n)$: imagine that $X = \log(n)$; then the probability of incrementing it is only $1/n$ and hence you would only expect to increment the counter at most one more time. That is, the counter is unlikely to be bigger than $\log(n) + 1$.

In fact, we can show that the expected value of the counter is $\log(n + 1)$, and so the total space used is $\log \log(n + 1)$ in expectation. And that implies that the answer we want to return $f(x) = 2^X - 1$.

Claim 1 $\mathrm{E}\left[2^X\right] = n + 1$

Let X_j be the value of X after the jth item in the stream is processed. We will prove by induction that $\mathrm{E}\left[2^{X_j}\right] = j+1$. Notice that this is immediately true for the base case: if $j = 0$, then $X_j = 0$ and hence $2^{X_j} = 1$. As our inductive hypothesis, assume that $\mathrm{E}\left[2^{X_{j-1}}\right] = j - 1 + 1 = j$. We will now analyze $\mathrm{E}\left[2^{X_j}\right]$.

In order to determine the expected value of 2^{X_j}, we use conditional expectation to divide the cases up based on the previous value of X_{j-1}. Then we can calculate the expectation based on the two cases: either we increment X or we

do not increment X.

$$\begin{aligned}
\mathrm{E}\left[2^{X_j}\right] &= \sum_{i=0}^{n} \Pr\left[X_{j-1}=i\right] \mathrm{E}\left[2^{X_j} | X_{j-1}=i\right] \\
&= \sum_{i=0}^{n} \Pr\left[X_{j-1}=i\right] \left[(1/2^i)2^(i+1) + (1-1/2^i)2^i\right] \\
&= \sum_{i=0}^{n} \Pr\left[X_{j-1}=i\right] \left[2+2^i-1\right] \\
&= \sum_{i=0}^{n} \Pr\left[X_{j-1}=i\right] \left[1+2^i\right] \\
&= \sum_{i=0}^{n} \Pr\left[X_{j-1}=i\right] + \sum_{i=0}^{n} \Pr\left[X_{j-1}=i\right] 2^i \\
&= 1 + \mathrm{E}\left[2^{X_{j-1}}\right] \\
&= 1+j
\end{aligned}$$

Notice that the second-to-last line follows because: (i) summing the probabilities over all the possible values of i yields 1, since X_{j-1} has to be one of these values, and (ii) the right term is exactly the definition of the expected value of $2^{X_{j-1}}$.

So we have shown that the expected value of the counter is good. As an exercise, you might calculate the variance of the counter. Much like with the FM algorithm, you will find it is useful to design an algorithm Morris+ wherein you average the count returned by a collection of Morris Counters. And then you might design an algorithm Morris++ wherein you take the median of the counts returned by a collection of Morris+ Counters.