CS 761: Randomized Algorithms Fall 2019

Lecture 3 — September 20, 2019

Prof. Gautam Kamath *By: Clara Kang, Thi Xuan Vu*
Edited by Vedat Levi Alev

Disclaimer: These notes have not been subject to the usual scrutiny reserved for formal publications.

The following expressions and inequalities will be useful during today's lecture.

- The union bound:
$$\Pr[\bigcup_{i=1}^{n} X_i] \leq \sum_{i=1}^{n} \Pr[X_i], \tag{1}$$
- Binomial approximation/inequalities:
$$(1-x)^n \approx \exp^{-nx}, \tag{2}$$
$$(\frac{n}{k})^k \leq \binom{n}{k} \leq \frac{n^k}{k!} \leq (\frac{n\,e}{k})^k, \tag{3}$$
- Factorial inequality:
$$n! \leq e\sqrt{n}\left(\frac{n}{e}\right)^n. \tag{4}$$

1 MST from last lecture

This is a review of the algorithm for finding a minimum spanning tree (MST) of a graph. Recall that a MST is the subset of an edge-weighted undirected graph that connects all the vertices together with minimum total edge weights.

Algorithm: Naive MST

Generate $G(p)$ by sampling each edge with probability p
Find MSF of $G(p)$, call it F_p
Remove all F_p-heavy edges from G to form G'.
Find MST of G'.

The cost of Naive MST algorithm is $O(m + \sqrt{mn}\log n)$. This run time is linear when $\sqrt{mn}\log n \leq m \Leftrightarrow n(\log n)^2 \leq m \Leftrightarrow (\log n)^2 \leq \frac{m}{n}$ (the average degree).

We would like to have an algorithm with linear time, so we will compute the MST of these constructed graphs recursively.

Algorithm: Linear MST (Karger-Klein-Tarjan's algorithm)

Run 3 iterations of Boruvka's algorithm
Set $p = 1/2$
Generate $G(p)$ by sampling each edge with probability p
Use Linear MST on $G(p)$ to obtain MSF F_p
Remove all F_p-heavy edges from G to form G'.
Use Linear MST to find MST of G'.

The run time of this algorithm satisfies a recurrence $T(m, n) = c(m+n)+T(m/2, n/8)+T(n/4, n/8)$. Solving the recurrence, we get $T(m, n) \leq 2c(m + n)$.

2 Balls and Bins

Suppose we have m balls and n bins, and each ball is independently thrown into a uniformly random bin. What does the distribution of the balls in the bins look like? There are several interesting questions about this random process. For example, how many bins are empty? How many collisions will be occurred in the first m throws? Another interesting question concerns the maximum number of balls in a bin, or the maximum load. Many of these questions have applications to the design and analysis of algorithms.

We will also see in this section that the birthday paradox is an example of balls and bins.

2.1 Expected number of balls in a bin

Given an event A. An *indicator* random variable I_A is a special kind of random variable associated with the occurrence of A. That is

$$I_A = \begin{cases} 1, & \text{if A happens.} \\ 0, & \text{otherwise.} \end{cases}$$

Let p be the probability that the event happens. The variable I_A us a Bernoulli random variable. Recall that, for a Bernoulli random variable,

$$E[I_A] = p \cdot 1 + (1 - p) \cdot 0 = p = \Pr[A = 1].$$

For $1 \leq i \leq m, 1 \leq j \leq n$, the probability that the ball i falling into bin j is $1/n$; and let us denote by $B_{i,j}$ the indicator random variable of ball i falling into bin j. Then $L_j := \sum_{i\in[m]} B_{i,j}$ is the number of balls in bin j. We have

$$E[L_j] = \sum_{i\in[m]} E[B_{i,j}] = \sum_{i\in[m]} \Pr[\text{ball } i \text{ in bin } j] = \sum_{i\in[m]} \frac{1}{n} = m/n.$$

In particular, when $m = n$, the expected number of balls in a bin is one.

2.2 Expected number of empty bins

Let E_2 denote the expected number of empty bins. Let Y_j, for $j = 1, \ldots, m$, be the indicator random variable that bin j is empty. The probability of a particular ball not falling into a particular bin is $1 - \frac{1}{n}$. Then, $\Pr[Y_j = 1] = (1 - 1/n)^m$, which is the probability of throwing m balls and none of them fall into bin j. This implies that $E[Y_j] = \big(1 - \frac{1}{n}\big)^m$. Therefore,

$$E_2 = \sum_{j \in [n]} E[Y_j] = \sum_{j \in [n]} \big(1 - \frac{1}{n}\big)^m \approx \sum_{j \in [n]} \exp(-m/n) = n \exp(-m/n).$$

In particular, when $m = n$, $E_2 \approx n/e \approx 0.37\, n$, i.e., a $\frac{1}{e}$-fraction of the bins are empty.

Question 1. *How large does n have to be before we have no empty bins?*

This falls into the framework of the coupon collector problem, which we studied in previous lectures. We would like at least one ball in each bin, which is the equivalent of collecting n coupons, with the probability of getting a new coupon equals to $\frac{1}{n}$. We know from the study of the coupon collector problem that the expected number of balls that we need to throw is $m = nH_n \approx n \log n$. We can verify this result by substituting $m = n \log n$ into the expression of empty bins. We get $E_2 = ne^{-\frac{m}{n}} = ne^{-\frac{n \log n}{n}} = 1$. If we throw slightly more balls than m, there will be no empty bin.

Remark 1. *The above bound is expectation bound, which means that the expected number of balls that we need to throw for there to be no empty bin is $n \log n$. This does not happen with probability 1.*

2.3 Probability of no collision in the first m throws

We would like to calculate the probability that no two balls land in the same bin in the first m throws. Let us denote this probability by P.

Observe that, for $2 \le i \le m$, the probability of the i^{th} ball does not get thrown into the same bin as any of the $i - 1$ preceding balls, conditioned on the first $i - 1$ balls do not collide, is $1 - \frac{i-1}{n}$. Then P is the probability of this happening for all $1 \le i \le m$. Therefore,

$$\begin{aligned} P &= 1\big(1 - \frac{1}{n}\big)\big(1 - \frac{2}{n}\big) \cdots \big(1 - \frac{m-1}{n}\big) \\ &\le \prod_{i=1}^{m-1} \exp(-i/n) = \exp(-\frac{1}{n} \sum_{i=1}^{m-1} i) \\ &= \exp(-\frac{1}{n} \frac{m(m-1)}{2}) \approx \exp(-\frac{m^2}{2n}). \end{aligned}$$

In particular, when $m = \sqrt{2n \log 2}$, $P = \exp(\frac{-2n \log 2}{2n}) = \frac{1}{2}$. In other words, when $m = \Theta(\sqrt{n})$, there is $1/2$ probability that a collision will happen.

Remark 2 (Birthday paradox)**.** *Consider $n = 365$, the number of days per year. The birthday problem asks, how many people must there be in a room for there to be at least a 50% chance to*

have two of them share the same birthday. Our calculation above answer a related question: how many people must there to expect at least one birthday collision? We can see people as balls and days as bin; then, $m = \sqrt{2n \log 2} \approx 23$.

In other words, given a group of 23 people, there is a 50% *chance that there exists a pair of people among them that shares a birthday.*

2.4 Max load when $m = n$

We would like to know the max load of any bin. The strategy of solving this problem is to first bound the probability that a bin has k balls, then take union bound of n bins. If this quantity is small then it is unlikely that any bin has more than k balls, and the maximum load is thus k.

Theorem 3. *No bin has more than* $k = \frac{3 \ln n}{\ln \ln n}$ *balls in it, with probability at least* $1 - 1/n$

Proof. Let us denote by P_j, for $1 \leq j \leq n$, the probability that bin j has at least k balls, and by P the probability that there exists a bin with at least k balls.

For any set of k balls, the probability of all k balls fall into bin j is $\left(\frac{1}{n}\right)^k$; and there are $\binom{n}{k}$ sets of k balls from n balls. So one has $P_j \leq \binom{n}{k}\left(\frac{1}{n}\right)^k$. From (3), we have $P_j \leq \left(\frac{n\,e}{k}\right)^k \left(\frac{1}{n}\right)^k = \left(\frac{e}{k}\right)^k$. Therefore, by using union bound over all such sets,

$$P = \sum_{j=1}^{n} P_j \leq \sum_{j=1}^{n} \left(\frac{e}{k}\right)^k = \exp(\ln n + k - k \ln k).$$

We would like to estimate the smallest value of k such that P is small enough, i.e., we want to minimum k such that $k \ln k > \ln n$. This can be done by setting $k = \frac{3 \ln n}{\ln \ln n}$, so that $P \leq 1/n$. □

2.5 Max load lower bound when $m = n$

Let the most loaded bin have at least r balls, we would like to know what r is. We will first calculate the probability that no bin has more than r balls, which is an expression containing r. By setting r such that this probability is small, we will know that there will be some bin with at least r balls.

Let us denote by P the probability that some specific bin has exactly r balls. For any set of m balls, the probability that exactly r balls fall into a bin is $\left(\frac{1}{n}\right)^r \left(1 - \frac{1}{n}\right)^{n-r}$; and there are $\binom{n}{r}$ sets of r balls. Then $P = \binom{n}{r}\left(\frac{1}{n}\right)^r \left(1 - \frac{1}{n}\right)^{n-r}$. When $n \gg r$,

$$\begin{aligned} P &= \frac{1}{r!} \frac{n(n-1)\cdots(n-r+1)}{n^r} \left(1 - \frac{1}{n}\right)^{n-r} \\ &\approx \frac{1}{r!} \frac{n^r}{n^r} \left(1 - \frac{1}{n}\right)^n = \frac{1}{e\, r!}. \end{aligned}$$

Heuristic argument: All bins are independent.

We use the assumption that all bins are independent to simplify our calculation. Please note that this is not true. As a simple example, consider the case where there are 2 bins. Let event A be

the first bin has r balls, event B be the second bin has $(m-r)$ balls. Then, Pr[A$\cap$ B] = Pr[A] $\neq$ Pr[A]Pr[B].

With this assumption, we can calculate the probability that no bin has more than r balls as the following.

$$\begin{aligned}\Pr[\text{No bin has } \geq r \text{ balls}] &= \Pr[\text{All bins have } \leq r \text{ balls}] \\ &\leq \big(1 - \frac{1}{e\,r!}\big)^n \leq \exp\big(-\frac{n}{e\,r!}\big).\end{aligned}$$

If this probability is small, e.g. $\exp\big(-\frac{n}{e\,r!}\big) \leq n^{-2}$, then with high probability, there will be some bin with at least r balls.

Lemma 4. *For any positive integer n and $r = \frac{\ln n}{\ln\ln n}$, one has*

$$\exp\big(-\frac{n}{e\,r!}\big) \leq n^{-2}. \tag{5}$$

Proof. We have

$$\begin{aligned}(5) &\Leftrightarrow -\frac{n}{e\,r!} \leq -2\ln n \\ &\Leftrightarrow \quad r! \ \leq \frac{n}{2e}\,\ln n \\ &\Leftrightarrow \ln(r!) \leq \ln n - \ln\ln n - \ln(2e). \end{aligned} \tag{6}$$

From (4), $\ln(r!) \leq \ln r + r(\ln r - \ln e) = \ln r + r = m.\ln r - r$. Then, if $r = \frac{\ln n}{\ln\ln n}$, one has

$$\begin{aligned}\ln(r!) &\leq \ln\Big(\frac{\ln n}{\ln\ln n}\Big) + \frac{\ln n}{\ln\ln n}\ln r - \frac{\ln n}{\ln\ln n} \\ &\leq \ln n - \frac{\ln n}{\ln\ln n} \\ &\leq \ln n - \ln\ln n - \ln(2e),\end{aligned}$$

the identity (6) which we wanted to prove. □

As a consequence, there exists some bin with load $\Omega(\ln n/\ln\ln n)$.

2.6 Coupon collector

We want to estimate the probability of having some empty bin after $n\ln n + c\,n$ balls, for some constant c.

Using the same argument as in Section 2.5, the probability that some specific bin has exactly r balls is $P = \binom{m}{r}\big(\frac{1}{n}\big)^r\big(1-\frac{1}{n}\big)^{m-r}$. When $m, n \gg r$,

$$\begin{aligned}P &= \frac{1}{r!}\,\frac{m(m-1)\cdots(m-r+1)}{m^r}\,\big(1-\frac{1}{n}\big)^{m-r} \\ &\approx \frac{1}{r!}\,\big(\frac{m}{r}\big)^r \exp(-\frac{m}{n}).\end{aligned}$$

When $r = 0$, i.e. when some bins have no balls, $P_0 = 1 \cdot 1 \cdot e^{-\frac{m}{n}}$. In particular, when $m = n \ln n + cn$, $P_0 \approx \exp(-\ln n - c) = \frac{e^{-c}}{n}$. Therefore,

$$\Pr[\text{there exists empty bin}] = 1 - \big(1 - \frac{e^{-c}}{n}\big)^n \approx 1 - \exp(-e^{-c}) = 1 - \frac{1}{e^{1/(e^c)}}.$$

Observation 5. *If $c > 10$, then there are no empty bins with high probability; if $c < -10$, then there are some empty bins with high probability. There is a sharp threshold near $m = n \log n$.*

3 Poisson distribution

Recall from the previous section that $P = \frac{1}{r!}\big(\frac{m}{r}\big)^r \exp(-\frac{m}{n})$, think of m/n as the mean.

A discrete random variable X is said to have a *Poisson distribution* with parameter λ if for $j \in \mathbb{N}$, the PMF of X is given by

$$\Pr[X = j] = \exp(-\lambda)\frac{\lambda^j}{j!}.$$

Fact 6. *$X \sim Poisson(\lambda) \to E[X] = \lambda$.*

Fact 7. *Given $Binomial(n, p)$, if $np = \lambda$ and λ is not dependent on n, then $\lim_{n\to\infty} Bin(n, p) = Poisson(\lambda)$.*

In Balls and Bins, with n balls and n bins, each marginal $= Binomial(n, \frac{1}{n})$ is $Poisson(1)$.

For $1 \le i \le n$, let $X^{(i)}$ be the number of balls in bin i under the regular balls and bins process, and $Y^{(i)}$ be a Poisson random variable with mean m/n. The main difference between $\overrightarrow{X} = (X^{(1)}, \ldots, X^{(m)})$ and $\overrightarrow{Y} = (Y^{(1)}, \ldots, Y^{(m)})$ is $\sum_{i=1}^n X^{(i)} = m$ while in general, $\sum_{i=1}^n Y^{(i)} \neq m$.

Theorem 8. *[3, Theorem 5.6, Theorem 5.7] With all notation as being above, we have*

1. $\Pr[\sum_{i=1}^n Y^{(i)} = m] \ge \frac{1}{e\sqrt{m}}$,
2. *conditioned on $\sum_{i=1}^n Y^{(i)} = m$, then $\overrightarrow{X}$ and $\overrightarrow{Y}$ have the same distribution.*

From this Theorem, if we can give a good upper bound in the Poisson distribution, we can give a slightly bigger bound for the original distribution. In other words, for some event E_1, if $\Pr_{\overrightarrow{Y}}[E_1] \le p$, then $\Pr_{\overrightarrow{X}}[E_1] \le p\, e\sqrt{m}$.

For the maximum load problem and the coupon collector problem, i.e., E_1 is the event that all bins have at least $\frac{\ln n}{\ln \ln n}$ balls, then $\Pr_{\overrightarrow{Y}}[E_1] \le n^{-2}$ implies $\Pr_{\overrightarrow{X}}[E_1] \le n^{-1.5}$.

The Poisson process make the independent assumption of Heuristic argument in subsection 2.5 to be more precise.

The key point is that we can work with independent Poission random variables and apply the Chernoff bound to see that bad events happen with very small probability.

4 The power of two choices

We have seen that when n balls are thrown into n bin, then the maximum load is $\Theta(\ln n / \ln \ln n)$ with high probability.

Let us now consider a variant problem when each ball is thrown, we pick two random bins and put the ball in less loaded bin. We will see in this section that, this variant process can significantly reduce the maximum load to $O(\ln \ln n)$.

The intuition is the following. There are $\leq \frac{n}{4}$ bins with ≥ 4 balls, otherwise the total number of balls exceeds n, which is a contradiction to us having only n balls. Creating a bin with 5 balls requires us to choose between two bins that both have 4 balls. The probability of having 2 bins with 4 balls is $\leq \frac{1}{4} \times \frac{1}{4}$ Using the same reasoning, we get

- Pr[creating bin with 6 balls] $\leq$ Pr[selecting 2 bins with ≥ 5 balls] $\leq \frac{1}{2^{2^2}} \times \frac{1}{2^{2^2}} = \frac{1}{2^{2^3}}$.
- $\cdots$
- Pr[creating bin with k balls] $\leq$ Pr[selecting 2 bins with $\geq k-1$ balls] $\leq \frac{1}{2^{2^{k-3}}}$.

Therefore, if $k = O(\ln \ln n)$, the probability of having a bin with k balls is less than $\frac{1}{n}$. With the power of 2 choices, the max load is $O(\ln \ln n)$

5 Hashing

5.1 Motivation

Hashing allows us to access data efficiently. We would like to retrieve data quickly and save the storage space at the same time.

Our problem is the following: given an element x and a set $S \subset U$ with $|S| = n$ and $|U| = m$. Check if $x \in S$.

A trivial solution is to perform a linear scan. This needs $O(n)$ time, which is very slow for many queries.

A less trivial solution is to create table T with $|T| = m$. The insertion and lookup operations are performed as follows:

Insert: set $T[x] = 1$ for all $x \in S$, and $T[x] = 0$ otherwise.

Lookup: check $T[x] = 1$, if yes, then $x \in S$

This method requires $O(1)$ time for lookup, but $O(m)$ space, note that m can get arbitrarily large.

Our goal is to have a data structure that requires $O(1)$ time lookup and $O(n)$ space.

5.2 Hash tables and hash functions

A *hash function* h is used to map the elements of the universe to locations in a smaller table, i.e., $h : U \to \{0, \ldots, n-1\}$. A *hash table* is a data structure that consists of a table T with n cells indexed by $\{0, \ldots, n-1\}$ and a hash function h. Ideally, h is a bijection, mapping different keys into different locations. However, this is impossible to achieve unless we know S in advance (and even then, this is not trivial).

Instead of random mapping, we would like to have

1. $\Pr[h(x) = j] = \frac{1}{n}$ for all $j \in \{0, \ldots, n-1\}$ and $h \sim D$ (D : distribution over family of all hash functions from U to $\{0, \ldots, n-1\}$).
2. for all $x_1 \ldots x_t$, the values $h(x_i)$ are independent.

For example, one can take D to be uniform over all functions from U to $\{0, \ldots, n-1\}$. The insertion and lookup operations are now performed as follows:

Insert: add x to linked list L at $T[h(x)]$.

Lookup: scan through L at $T[h(x)]$.

With uniform hash functions, the lookup time is equivalent to the load of bins in the "balls and bin" problem. The expected time for lookup is thus $O(1)$ (from section 2.1), and the worse case lookup is $O\left(\frac{\log n}{\log \log n}\right)$ (from section 2.4).

We can improve the lookup time by using "two choices". We select two functions h_1 and h_2.

Insert: add x to less loaded location of $T[h_1(x)], T[h_2(x)]$.

Lookup: scan $T[h_1(x)], T[h_2(x)]$.

The expected time for lookup is still $O(1)$, but the worse case lookup is now: $O(\log \log n)$. (from section 4)

5.3 Bloom filters

Instead of performing a lookup every time a key is given, we would like to tell quickly (in constant time) whether an element is present in the hash table. To do this, we will use a bit vector.

Let us start with a simple example. We start with a table T with $|T| = \ell$ ($m \gg \ell > n$, e.g., $\ell = 5n$). Initially, $T = \overrightarrow{0}$.

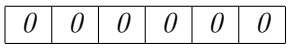
Insert: Set $T[h(x)] = 1$.

Lookup: If $T[h(x)] = 1$, then say $x \in S$. Otherwise, say $x \notin S$.

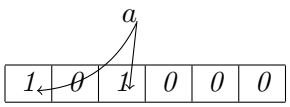
The pros of this method is that worse case lookup is $O(1)$, and the space taken by this table is $O(\ell)$. However, this method produce a lot of false positives. Since $\Pr[\mathrm{T}[h(x)] = 0, x \notin S] = \left(1 - \frac{1}{n}\right)^n$, $\Pr[\text{false positives}] = \Pr[\mathrm{T}[h(x)] = 1, x \notin S] = 1 - \left(1 - \frac{1}{\ell}\right)^n \approx 1 - \exp(-\frac{n}{\ell})$. For example, if $\ell = 8n$, then we have an 11.8% false positive rate.

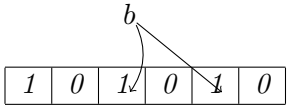
Example 9. *Consider $k = 2$,*

Initial:

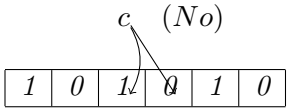


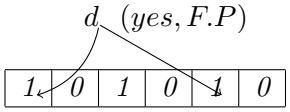
Insert :





Lookup :





The solution is to use k hash functions $h_1, \ldots, h_k$, instead of 1 hash function h.

Insert: Set $T[h_i(x)] = 1$ for all $i = 1, \ldots, k$.

Lookup: check if $T[h_i(x)] = 1$ for all $i = 1, \ldots, k$.

We have Pr[some entry of T equal zero] $= \left(1 - \frac{1}{\ell}\right)^{k\,n} \approx \left(\frac{kn}{\ell}\right) := p$. Pretend settings of T are independent with probability p (this is not true but it is close to Poisson analysis). Then,

$$\Pr[\text{false positives}] = \left(1 - \left(1 - \frac{1}{\ell}\right)^{k\,n}\right)^k \approx \left(1 - \exp(-\frac{kn}{\ell})\right)^k.$$

In particular, if $k = \ln 2 \times (\ell/n)$, i.e., $\ell = O(n \log n)$, Pr[false positives] $\approx (0.6185)^{\ell/n}$. Then, if $\ell = 8n$, $k = 5$ or 6, then Pr[false positives] ≈ 0.02.

Recent work: Learning in Index [1, 2] Often, classification tasks are studied in machine learning. If we use machine learning algorithms to predict whether an element is in the set, then we could reduce the size of the table by a lot. This task is called "learning to index". The goal is to get a function $f : U \to [0, 1]$, which takes an element in the universe and returns 1 if it is in the set, and 0 otherwise. To get f, the algorithm is trained on a set of $(x, 1)$ for $x \in S$, and a set of $(y, 0)$ for $y \notin S$.

Insert: If $f(x) \leq \tau$, insert into the "backup" bloom filter, where τ is a threshold.

Lookup: If $f(x) > \tau$, say "yes", otherwise search in bloom filter.

References

[1] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, pages 489–504, New York, NY, USA, 2018. ACM.

[2] Michael Mitzenmacher. A model for learned bloom filters, and optimizing by sandwiching. In *Proceedings of the 32Nd International Conference on Neural Information Processing Systems*, NIPS'18, pages 462–471, USA, 2018. Curran Associates Inc.

[3] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, New York, NY, USA, 2005.