Algorithms and Probability (FS2025)
Week 9

Rui Zhang

April 16, 2025

Contents

Questions :

1. Monte-Carlo-Theorem holds for $p_{correct} < \frac{1}{2}$ as well
2. Hopefully cheatsheet 2w after end of sem.

1 Content

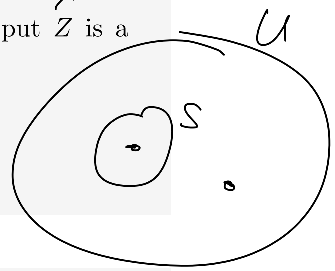
1.1 Target Shooting

$$Z = \frac{1}{N}\sum_{i=1}^{N} I_S(u_i)$$

Theorem: For the Target Shooting Algorithm which performs N iterations, the output Z is a random variable, which has expectation $|S|/|U|$ and variance

$$\frac{1}{N}\left(\frac{|S|}{|U|} - \left(\frac{|S|}{|U|}\right)^2\right)$$

$$\frac{|S|}{|U|}$$



Theorem: Let $\delta, \varepsilon > 0$. If we let Target Shooting run with at least $N \geq 3\frac{|U|}{|S|}\varepsilon^{-2}\ln\left(\frac{2}{\delta}\right)$, iterations, then the result Z is within the interval

$$\left[(1-\varepsilon)\frac{|S|}{|U|}, (1+\varepsilon)\frac{|S|}{|U|}\right]$$

with probability at least $1 - \delta$

$$\mathbb{P}\left[\left|Z - \frac{|S|}{|U|}\right| \leq \varepsilon \cdot \frac{|S|}{|U|}\right] \geq 1-\delta$$

$$\Rightarrow \mathbb{P}\left[\left|Z - \frac{|S|}{|U|}\right| > \varepsilon \cdot \frac{|S|}{|U|}\right] \leq \delta$$

$$\Rightarrow \mathbb{P}\left[\left|Z - E[Z]\right| > \varepsilon \cdot E[Z]\right] \leq \delta$$

(lin. Exp.

$$\Rightarrow \mathbb{P}\left[\left|NZ - E[NZ]\right| > \varepsilon \cdot E[NZ]\right] \leq \delta$$

$$\Rightarrow \mathbb{P}\left[\{NZ > (1+\varepsilon)E[NZ]\} \cup \{NZ < (1-\varepsilon)E[NZ]\}\right]$$

$$= \mathbb{P}\left[NZ > (1+\varepsilon)E[NZ]\right] + \mathbb{P}\left[NZ < (1-\varepsilon)E[NZ]\right]$$

$$\overset{\text{Chernoff}}{\leq} 2e^{-\varepsilon^2\frac{E[NZ]}{3}} = 2e^{-\varepsilon^2 N \cdot \frac{|S|}{|U|} \cdot \frac{1}{3}} \leq 2e^{-\varepsilon^2 3 \cdot \frac{|U|}{|S|} \cdot \varepsilon^{-2} \ln\left(\frac{2}{\delta}\right)\frac{|S|}{|U|} \cdot \frac{1}{3}}$$

$$= 2e^{-\ln\left(\frac{2}{\delta}\right)}$$

$$= 2 \cdot \frac{\delta}{2} = \delta$$

$$Z = \frac{1}{N}\sum_{i=1}^{N} I_S(u_i)$$

$$NZ = \sum_{i=1}^{N} I_S(u_i)$$

↪ bernoulli variables

$$\begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \\ (1, & 2, & 3, & 1, & 1) \end{array}$$

$(0,3),(0,4),(3,4)$

1.2 Hashing

Hashing is a technique used often in programming to speed up datastructures and processes. We will look at one such example here: Assume you have an array of n elements $D = (s_1, s_2, \ldots, s_n)$. A pair is a duplicate (i, j) if $s_i = s_j$. Our job now is to find all duplicates in this array.

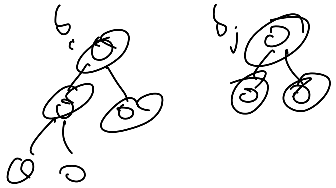
Naive Solution:

1. sort array $O(n \log n)$
2. scan array $O(n + |Dup(D)|)$

$0 \leq |Dup(D)| \leq n^2$

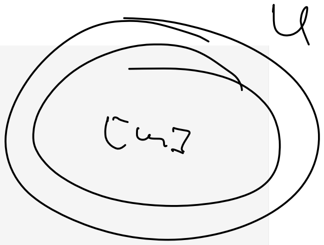
$(b_1, b_2, \ldots, b_n)$

b_i is binary tree



Hashing Solution:

But what if accessing elements / comparing elements is very expensive?



Theorem: Formally, a hash function is a function

$$h : U \to [m]$$

such that

$$\forall u \in U \ \forall i \in [m] \quad \Pr[h(u) = i] = \frac{1}{m}$$

$0\ 1\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ -\ -\ -\ -\ -\ 1\ 0 \mid 0\ 0 \mod 2$

$0\ 1\ -\ -\ -\ -\ -\ -\ -\ -\ - \mid 1\ 0 \mod 2$

Theorem: Let $s, t \in U$, then $s = t \Rightarrow h(s) = h(t)$

Note that the other direction does not hold.

Now, our solution consists of calculating the hash for all elements in the array, sorting the hashes and finding the duplicates via a scan just like before, this time on the hashes. However, there is a problem: We might have collisions: $(b_1, b_2, \ldots, b_n) \Rightarrow ((h(b_1), 1), (h(b_2), 2), \ldots, h(b_n), n))$

Definition: Collisions are the unwanted false positives we get from this approach: i.e. when we have two elements s, t such that $s \neq t$ but $h(s) = h(t)$ still holds because we got unlucky. The algorithm would then say "oh no there's a duplicate here!" But in reality there is only a duplicate on the level of the hashes, not the original elements.

We will now analyze how often we get collisions:
Let $K_{i,j}$ be the indicator variable for the event "(i, j) is a collision". We have

$$\Pr[K_{i,j} = 1] = \begin{cases} 1/m & \text{if } s_i \neq s_j \\ 0 & \text{else} \end{cases} \quad \Rightarrow \mathbb{E}[K_{i,j}] \leq 1/m$$

$$\frac{1}{m} \cdot \frac{1}{\cancel{m}} \cdot \cancel{m} = \frac{1}{m}$$

$$\begin{array}{c|c} h(s_i) & h(s_j) \\ \hline 1 & 1 \\ 2 & 2 \\ \vdots & \vdots \\ m & m \end{array} \quad \leftarrow \frac{1}{m} \cdot \frac{1}{m} \quad \Bigg\} \, m$$

$E[\# collisions] \leq O(1)$

$$E[K] = \sum_{1 \le i < j \le n} E[K_{i,j}] \le \sum_{\dots} \frac{1}{m} = \binom{n}{2}\frac{1}{m}$$

And with the number of collisions being $K = \sum_{1\leq i<j\leq n} K_{i,j}$, we apply linearity of expectation and get

$$\mathbb{E}[K] \leq \binom{n}{2}\frac{1}{m} = \frac{n(n-1)}{2} \cdot \frac{1}{n^2} \le \frac{1}{2} \le O(1)$$

And if we choose $m = n^2$, we get that the expected number of collisions is less than 1, and we can deal with that!

With this solution, we still keep a relatively good runtime and $O(n \log n)$ additional space needed for saving the hash values, which is doable.

$$\underbrace{\text{additional space}} \to O(n(\log_2(m) + \log_2(n)))$$

(h, i), $h \to 1, \dots, m$, $i \to 1, \dots, n$

runtime:
hashing: $O(n)$
sorting: $O(n \log n)$
Scan: $O(n + |Dupl(D)| + \#\text{collisions})$ $* O(\text{size } b_i)$

1.3 Bloom Filters

> **Definition:** A repeated entry in $D = (s_1, \ldots, s_n)$ is an index $j \in [n]$ such that there exists a previous index $i \in [j-1]$ which has the same element $s_i = s_j$

> **Definition:** Let $m, k \in \mathbb{N}$ and $h_1, \ldots, h_k : U \to [m]$ be k random hash functions and M a boolean array of length m initially set to all 0's.
> Then, a bloom filter operates as follows:
>
> 1. Iterate through the array of elements D
> 2. For each element $s \in D$, calculate the hash vector $v = (h_1(s), \ldots h_k(s))$
> 3. For each entry $h_i(s) \in v$ of the hash vector, check if $M[h_i(s)]$ is set to 1. If not, set to 1.
> 4. If all $h_i(s)$ was set to 1 in M, then add s to the list of repeated entries L

$m = 4$

$2 \xrightarrow{h} (\overset{h_1(2)}{1}, \overset{h_2(2)}{2}) = v$ $\quad M = (\underset{0}{0}\ \underset{1}{1}\ \underset{2}{1}\ \underset{3}{0})$

$1 \xrightarrow{h} (2, 3) = v$ $\quad M\ (0\ 1\ 1\ 1)$

$3 \xrightarrow{h} (1, 3) = v$ $\quad M\ ($

Note that with this procedure, just like with hashing, we only have false positives yet not false negatives:

> **Theorem:** If s_i is a repeated entry, then its index i will be in list L:
>
> $$s_i \in \{s_1, \ldots, s_{i-1}\} \Rightarrow i \in L$$

We will from now refer to the false positives (entries in L but which are not repeated entries) as "incorrect entries" and would like to calculate $\mathbb{E}$[number of incorrect entries in L]. To this end define the indicator variable

$$X_i = \text{"indicator variable for when } i \text{ is in incorrect entry"}$$

$$X_i = 1 \Leftrightarrow \begin{cases} s_i \notin (s_1, \ldots s_{i-1}) \text{ and} \\ M[h_1(s_i)] = M[h_2(s_i)] = \cdots = M[h_k(s_i)] = 1 \end{cases}$$

l: num of non-rep. entries in $\{1, \dots, i-1\}$

$l \leq i-1$

$$\Pr\left[M[h_1(s_i)] = 0\right] = (1 - \frac{1}{m})^{kl} \geq (1 - \frac{1}{m})^{k(i-1)}$$

where l is the number of non-repeated entries in the $i-1$ previous elements the algorithm had to go through before reaching s_i. Note that this inequality is true as the base is less than 1 and the exponent is larger on the right hand side. It leaves us with the information that

$$\Pr\left[M[h_1(s_i)] = 1\right] = 1 - \Pr\left[M[h_1(s_i)] = 0\right] \leq 1 - (1 - \frac{1}{m})^{k(i-1)}$$

$$\Pr[X_i = 1] = \Pr[\forall j \in [\overset{[k]}{\cancel{k-1}}]\ M[h_j(s_i)] = 1] \lessapprox (1 - (1 - \frac{1}{m})^{k(i-1)})^k$$

and with $\mathbb{E}[X_i] = \Pr[X_i = 1]$ and number of incorrect entries in $L = X_1 + X_2 + \cdots + X_n$, we get

$1 - x \leq e^{-x}$

$1 - \frac{1}{m} \leq e^{-\frac{1}{m}}$

$$\mathbb{E}[\text{number of incorrect entries}] \leq n \cdot (1 - (1 - \frac{1}{m})^{k(i-1)})^k \leq n\left(1 - \left(e^{-\frac{kn}{m}}\right)\right)^k$$

$$\leq n \cdot (1 - e^{-kn/m})^k$$

$$n \cdot \left(1 - e^{-\frac{n \ln n}{n \ln n}}\right)^{\ln n} = n(1 - e^{-1})^{\ln n} = n \cdot (0,632)^{\ln n} = n\, e^{\ln(0,632) \cdot \ln n} = n\left(e^{\ln n}\right)^{\ln(c)} = n \cdot n^{\ln(c)}$$

$$= n \cdot n^{-0,46}$$

$$= n^{0,54}$$

$n = e^k$

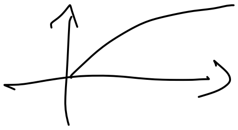
And choosing $k = \ln n$ and $m = n \ln n$ gives us a constant number on the right hand side. This gives us a runtime of kn hashing operations and an extra required space of m for the array M

$\approx \sqrt{n}$

$\approx \sqrt{e^k}$

2 Kahoot!!

https://quizizz.com/admin/quiz/67ffa780c785c72c8d5c6259



???

can sb. ask why?