CS 388R: Randomized Algorithms, Fall 2021 October 7th, 2021

Lecture 13: Sampling; Median Finding

Prof. Eric Price *Scribe: Nikos Mouzakis, Lucas Gretta*

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

1 Overview

In the last lecture we discussed fingerprinting, and applied the technique to verifying matrix multiplication, polynomial multiplication, and string matching, for example.

In this lecture we will discuss sampling algorithms, as well as how to use sampling to solve problems such as finding the median in a given list of numbers.

2 Example: estimating π

Suppose we wanted to estimate π, how would we do this? One way would be to use one of many series estimates for π, but we will do a more simple solution.

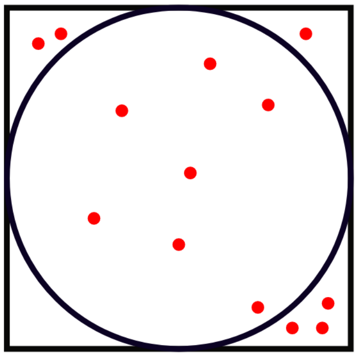

Figure 1: Estimating π by sampling

We know the area ratio of a circle with radius 1, and a square with side length 2, is $\pi/4$. Lets inscribe the circle inside the square, that is the square is $[-1,1]\times[-1,1]$, and the circle is of radius 1 centered on the origin. Now, if we uniformly sample in $[-1,1]\times[-1,1]$, each sample has a $\pi/4$ chance of landing in the circle (which is easily checked by $x^2+y^2\leq 1$), and so we can apply a Chernoff bound to this process, requiring $O(\frac{1}{\varepsilon^2}\log(1/\delta))$ samples, to estimate π within $1+\epsilon$.

More generally, for any high-dimensional polytope, and a given bounding box, we can estimate the volume of the polytope by sampling from the bounding box. The number of required samples depends on how good the given bounding box is.

3 Median-finding

There are 3 algorithms that come to mind:

- Sort the list, and then return the median, $O(n\log(n))$ time.
- Quickselect
- Median-of-Medians

3.1 Quickselect with random pivot

The idea is to use the same thing as quicksort, however instead of recursing on both ends, we only recurse on the side that contains the median. We will analyze the case where the pivot is selected randomly.

To analyze this we look at a recurrence $T(n) = O(n)+T(n')$, where n' is the size of our subproblem. Due to the same analysis as in quicksort, we know that $n' \leq 3/4n$ with probability $\geq 1/2$. Therefore, the expected number of rounds until our problem size is $\leq 3/4n$ is $O(1)$, and so in expectation it takes $O(1) * O(n)$ work until our problem size is $\leq 3/4n$, and so by a geometric sum our expected amount of work is $O(n + 3/4n + (3/4)^2 n + \cdots) = O(n)$, so in expectation this works in $O(n)$ time, which is obviously optimal as we need to look at all the values.

What about with high probability? We got such a bound for quicksort, but what about for quickselect? It turns out that we can't.

Suppose that we choose a pivot within the first n/k values (k to be determined later), and we repeatedly do this until our subproblem is of size $\leq n/2$. This would take $O(k/2 * n/2) = O(nk)$ work, so if we can make k greater than a constant, then this would show that we can't get a high probability bound.

Note that the probability this occurs is $\geq \frac{1}{k}^{k/2}$, so if we make $k = O(\frac{\log(n)}{\log\log(n)})$, then this occurs with $\geq 1/n$ probability, and so we can't get an $O(n)$ high probability bound.

3.2 Median-of-Medians

There is actually a *deterministic* algorithm for finding the median in $O(n)$ time!

Partition the values in our array into groups of size 5. For each group, find the median in $O(1)$ time, and then recursively find the median of medians. This gives us the recurrence

$$T(n) = O(n) + T(n/5) + T(7n/10) \tag{1}$$

The $7n/10$ term is because, after recursively finding the median of the $n/5$ middle values, we note that this value is $\geq 3n/10$ of the values, as it is greater than $1/2$ of the middle values, and each of these middle values is ≥ 2 other values, and so our median is greater than $1/2 * 3/5 = 3/10$ of the values.

By the master theorem this runs in $O(n)$.

Unfortunately, this has a rather large constant, so in the next section we will be using randomness to reduce the size of the constant.

3.3 Sampling

3.3.1 Sampling with small set

Another way to approach this problem would be to sample a set $S \subseteq \{x_1, x_2, ..., x_n\}$ (with replacement), and use its median as a pivot for QuickSelect.

For this to work, we would need the true rank of the sample median $\tilde{s}$ to be within $[\frac{3n}{10}, \frac{7n}{10}]$. We can bound the probability of this event by examining $Pr[rank(\tilde{s}) \leq \frac{3n}{10}]$ which is equivalent to bounding the probability of more than $\frac{|S|}{2}$ samples being from the $\frac{3n}{10}$ lowest elements. Let Y_i be the indicator variable of the event that the i-th element of the sampled set is of true rank at most $3n/10$. Since we sample with replacement, the Y_i are independent, and we can apply an additive Chernoff bound to their sum. Note that $E[Y_i] = 3/10$.

$$Pr\left[\sum_{i\in S} Y_i \geq \frac{|S|}{2}\right] \leq Pr\left[\sum Y_i \geq \frac{3|S|}{10} + \frac{|S|}{5}\right] \leq e^{-\frac{2|S|}{25}}$$

Hence, setting $|S| = log(n)$ is enough for this to work w.h.p. The overall time required for this algorithm is given by the recurrence $T(n) \leq O(|S| * log(|S|)) + O(n) + T(\frac{7n}{10})$, where each terms comes from sorting S, pivoting around the median of S, and recursively repeating on the at most $\frac{7n}{10}$ elements left, respectively. This gives us another $O(n)$ algorithm.

3.3.2 Sampling with large set

We notice that the above recurrence is particularly wasteful, in that the first term related to sorting S, is negligible compared to the rest. Hence we propose a better algorithm using a larger subset $|S| = \sqrt{n}$. Before we get into the details, let us prove a useful lemma:

Lemma 1. *Suppose an element has rank $\beta|S|$ in the sampled set. Then its rank in the entire set is in $\left[\beta - \sqrt{\frac{log(1/\delta)}{|S|}}, \beta + \sqrt{\frac{log(1/\delta)}{|S|}}\right]$ with probability $1 - \delta$.*

Proof. Let $\alpha = \sqrt{\frac{log(1/\delta)}{2|S|}}$. For the lower bound, we need at most $\beta|S|$ samples picked from the first $(\beta - \alpha)n$ elements.

Let Y_i be the indicator variable of whether the i-th sampled element is of rank at most $(\beta - \alpha)n$. Note that $E[Y_i] = \beta - \alpha$. We can bound their sum with a Chernoff bound, as they are independent (due to sampling with replacement).

$$Pr\left[\sum_{i\in S} Y_i \geq \beta|S|\right] \leq Pr\left[\sum Y_i \geq (\beta - \alpha)|S| + \alpha|S|\right] \leq e^{-2\alpha^2|S|} \leq \delta$$

The proof for the upper bound is the same. □

Applying this, we have that the median of the entire set $\tilde{X}$ is between that $\left[\frac{1}{2} - \sqrt{\frac{log(1/\delta)}{|S|}}\right]$-th and $\left[\frac{1}{2} + \sqrt{\frac{log(1/\delta)}{|S|}}\right]$-th elements of S, with probability $1-\delta$. We can use this bounded interval estimate for the true median in an algorithm as follows:

Algorithm 1 Sampling algorithm for median-finding

```
$S \leftarrow \sqrt{n}$ random elements of input set $X$          ▷ $\sqrt{n}$ time
$S \leftarrow$ Sort$(S)$                                            ▷ $\sqrt{n} log(n)$ time
$L \leftarrow S[\frac{S}{2} - \sqrt{Slog(1/\delta)}]$
$H \leftarrow S[\frac{S}{2} + \sqrt{Slog(1/\delta)}]$
$T_{low} \leftarrow \emptyset$
$T_{mid} \leftarrow \emptyset$
$T_{high} \leftarrow \emptyset$
for $x \in X$ do                                                    ▷ $O(n)$ time
    if $x < L$ then
        $T_{low} \leftarrow T_{low} \cup \{x\}$
    else if $x \leq H$ then
        $T_{mid} \leftarrow T_{mid} \cup \{x\}$
    else
        $T_{high} \leftarrow T_{high} \cup \{x\}$
    end if
end for
$T_{mid} \leftarrow$ Sort$(T_{mid})$                                ▷ $|T_{mid}| * log(|T_{mid}|)$ time
return $T_{mid}[\frac{n}{2} - |T_{low}|]$
```

This algorithm is correct as long as the true median ends up in T_{mid}, which happens with probability at least $1-\delta$.

We can show with high probability that the number of elements that end up in T_{mid} is at most $\sqrt{\frac{log(1/\delta)}{|S|}} * n$, which for $|S| = \sqrt{n}$ is $\sqrt{log(1/\delta)} * n^{3/4}$. Moreover, regarding the dominant term $O(n)$ we can notice that approximately half of the time we will only need to do one comparison, or two for the elements in T_{mid}. Hence, the number of comparisons in the for loop can be bounded by $1.5n + O(T_{mid}) \leq 1.5n + \sqrt{log(1/\delta)} * n^{3/4}$.

This gives an overall time bound of $1.5n + O(\sqrt{log(1/\delta)} * n^{3/4})$, which allows us to make δ small enough such that the required time is bounded by $1.5n + o(n)$ with high probability.