Eight hands-on: Count-min sketch: range queries

Algorithm Design (2021/2022)

Gabriele Pappalardo
Email: g.pappalardo4@studenti.unipi.it
Department of Computer Science

April 2022

1 Introduction

Consider the counters $F[i]$ for $1 \leq i \leq n$, where n is the number of items in the stream of any length. At any time, we know that $||F||$ is the total number of items (with repetitions) seen so far, where each $F[i]$ contains how many times the item i has been so far. We saw that CM-sketches provide a FPTAS $F'[i]$ such that $F[i] \leq F'[i] \leq F[i] + \varepsilon ||F||$, where the latter inequality holds with probability at least $1 - \delta$.

Consider now a range query (a, b), where we want $F_{ab} = \sum_{a \leq i \leq b} F[i]$. Show how to adapt CM-sketch so that a FPTAS F'_{ab} is provided:

- Baseline is $\sum_{a \leq i \leq b} F'[i]$, but this has drawbacks as both time and error grows with $b - a + 1$.
- Consider how to maintain counters for just the sums when $b - a + 1$ is any power of 2 (less or equal to n):
  - Can we now answer quickly also when $b - a + 1$ is not a power of two?
  - Can we reduce the number of these power-of-2 intervals from $n \log n$ to $2n$?
  - Can we bound the error with a certain probability? *Suggestion*: it does not suffice to say that it is at most δ the probability of error of each individual counter; while each counter is still the actual wanted value plus the residual as before, it is better to consider the sum V of these wanted values and the sum X of these residuals, and apply Markov's inequality to V and X rather than on the individual counters.

2 Solution

2.1 Baseline Solution

The baseline solution is pretty straightforward to implement: we can use a `for` loop starting from the element a up to b and sum all the counters. Since each counter has some error, if we sum up all the counter in a large range, then also the final error will grow linearly as large as the range, that is $b - a + 1$.

```
def range(F: CountMinSketch, a: int, b: int): int:
    sum = 0
    for i in range(a, b + 1):
        sum += F[i]
    return sum
```

Listing 1: 'Range query for integer values'

2.2 Requested Solution

We can improve the baseline solution using the ranges of powers of two. In fact, any range (a, b) can be expressed as disjoint union of length with the powers of two, e.g., if we have the range $(4, 10)$:

$$(4, 10) = (4) \cup (5, 8) \cup (9, 10)$$

We can state the following fact that will allow us to compute all the possible range queries.

Fact 1 (Dyadic Ranges)**.** *Any range in the interval from* 1 *to* n *is expressible as the disjoint union of* $2 \log n$ *intervals in the set of dyadic ranges.*

Therefore, given a universe U we can build a collection of *dyadic ranges*, and we will need at most $2n$ of them. Having these ranges, we can bind the error of the range query up to $\log n$, thus a logarithmic error.

$$F_{ab} \leq \tilde{F}_{ab} \leq F_{ab} + 2\varepsilon \log n ||F||$$

This can be achieved using $\log n$ Count-Min Sketches. We build a "logic" binary tree where each range is split in two parts. Starting from the root we represent the range $(1, n)$ and going down we will have $(1, n/2)$ and $(n/2 + 1)$. We repeat the splitting process until we are not able to do so anymore, that is, when we obtain n ranges of the form $(1, 1), (2, 2), (3, 3), \ldots, (n-1, n-1), (n, n)$. In each level of this "logic" binary tree we add a new Count-Min Sketch, having in total $\log n$ sketches.

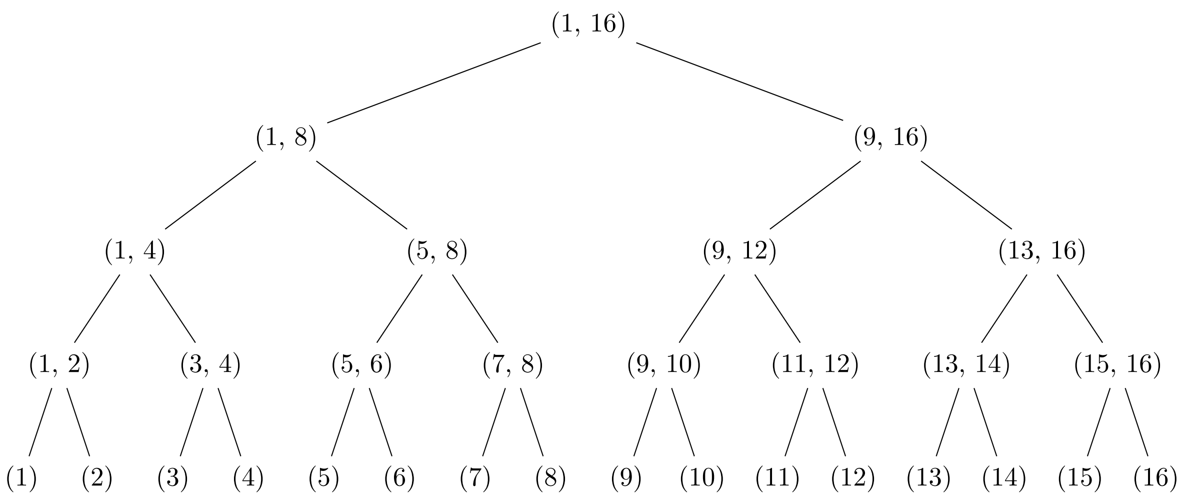

Figure 1: Generated ranges using $n = 16$.

When we have to update a counter, we traverse the tree updating the counters for each traversed range. For instance, if we have to update the counter for the element 7 in Figure 1, we are going to update the counters in each range containing the element, that are: $(1, 16), (1, 8), (5, 8), (7, 8), (7)$. The `update` operation takes $O(\log n)$ time, the pseudo-Python code is shown in Listing ?.

```
def update(x: int) -> None:
    # The first range to update is the root
    range = (1, n)
    level = 1
    # When the range has size one we stop
    while range != (x, x):
        # Update the CMS in the current level
        update(CMSS[level], range)
        # Go a level below in the logical tree
        level += 1
        # Compute the new range to update
        if x in (range[0], int(range[1]/2)):
            range = (range[0], int(range[1] / 2))
        else:
            range = (int(range[0] / 2) + 1, range[1])
    return
```

The `query` operation relies on **Fact** 1, also this computation takes $O(\log n)$ time. We have to select each dyadic range needed to build the original range.

```
def query(r: (int, int)) → int:
    sum = 0
    # Assuming we have a function that computes
    # dyadic ranges.
    ranges = dyadicRanges(r)
    # Query each CMS in the respective level with
    # the computed ranges
    for (level, range) in ranges:
        sum += query(CMSS[level], range)
    return 0
```

These algorithms work also with one Count-min Sketch, the universe of the counters will be the dyadic ranges.

2.3 Bounding the failure probability

The error analysis will require some definitions to work with. First, we are going to define the set of dyadic ranges as:

$$D = \{(1,n),(1,n/2),(n/2+1,n),\dots,(1,1),(2,2),\dots,(n,n)\}, |D| = 2n$$

The analysis makes use of just one Count-Min Sketch, and it is restricted to the ranges contained in the set D. Furthermore, we introduce the function:

$$\text{dy} : \mathscr{P}(\mathbb{N}\times\mathbb{N}) \to \mathscr{P}(D)$$

which computes the dyadic ranges of a given range, as described in the previous section. For any range r given to the function dy, thanks to the **Fact** 1, we have that:

$$\forall r \in \mathscr{P}(\mathbb{N}\times\mathbb{N}).|\text{dy}(r)| \leq 2\log n$$

We express the counter F_{ab} as $F_{(a,b)}$ where the range $(a,b) \in D$. The approximate counter $\tilde{F}_{(a,b)}$ is obtained as follows in Equation 1.

$$\tilde{F}_{(a,b)} = F_{(a,b)} + \sum_{(\alpha,\beta)\in D,(\alpha,\beta)\neq(a,b)} X_{(\alpha,\beta)} \tag{1}$$

The random variable $X_{(a,b)}$ is a value representing the "garbage" (according to the definition given during the lectures). We would like to know the expected value of this random variable. To do so, we define a new indicator random variable, as seen in class:

$$I_{j,i,k} = \begin{cases} 1 & h_j(i) = h_j(k) \\ 0 & \text{othwerwise} \end{cases} \; i,k \in D, j \in [d]$$

Therefore, when the j-th hash function has a collision with different ranges in D the variable will be set to 1. Fixed a $j \in [d]$ and given an $i \in D$ we can define $X_i^{(j)}$ as:

$$X_i^{(j)} = \sum_{k\in\text{dy}(k)} I_{j,i,k} \cdot F_k$$

We can compute its expectation as follows:

$$
\begin{aligned}
E\left[X_i^{(j)}\right] &= E\left[\sum_{k\in \mathrm{dy}(k)} I_{j,i,k}F_k\right] \\
&= \sum_{k\in \mathrm{dy}(k)} E[I_{j,i,k}\cdot F_k] \\
&= \sum_{k\in \mathrm{dy}(k)} \Pr(\{I_{j,i,k}=1\})F_k \\
&= \sum_{k\in \mathrm{dy}(k)} \frac{\varepsilon}{e}\cdot F_k \\
&= 2\log n\frac{\varepsilon}{e}\sum_{k\in \mathrm{dy}(k)} F_k \\
&\leq 2\log n\frac{\varepsilon}{e}||F||
\end{aligned}
$$

Knowing that $E\left[X_i^{(j)}\right] \leq 2\log n\frac{\varepsilon}{e}||F||$, we can bind the error probability in a single row using the Markov's inequality:

$$
\begin{gathered}
\Pr(\{\tilde{F}_{(a,b)} \geq F_{(a,b)} + 2\varepsilon\log n||F||\}) \leq \delta \iff \\
\Pr\left(\left\{F_{(a,b)} + \sum_{(\alpha,\beta)\neq(a,b)} X_{(\alpha,\beta)} \geq F_{(a,b)} + 2\varepsilon\log n||F||\right\}\right) \leq \delta \iff \\
\Pr\left(\left\{\sum_{(\alpha,\beta)\neq(a,b)} X_{(\alpha,\beta)} \geq 2\varepsilon\log n||F||\right\}\right) \leq \frac{E\left[\sum X_{(\alpha,\beta)}\right]}{2\varepsilon\log n||F||} = \frac{2\log n\frac{\varepsilon}{e}||F||}{2\varepsilon\log n||F||} = \frac{1}{e}
\end{gathered}
$$

Since we have r row, where each hash function is independent, we will have:

$$
\prod_r \Pr\left(\left\{\sum_{(\alpha,\beta)\neq(a,b)} X_{(\alpha,\beta)} \geq 2\varepsilon\log n||F||\right\}\right) \leq \left(\frac{1}{e}\right)^r = \delta
$$