Streaming: Sketching

Inge Li Gørtz

Today

- Sketching
- CountMin sketch

Sketching

Sketching

- **Sketching.** create compact sketch/summary of data.
- **Example.** Durand and Flajolet 2003.
  - Condensed the whole Shakespeares' work

```
ghfffghfghgghggggghghheehfhfhhgghghghhfgffffhhhiigfhhffgfiihfhhh
igigighfgihfffghigihghigfhhgeegeghgghhhgghhfhidiigihighihehhhfgg
hfgighigffghdieghhhgghhfghhfiiheffghghihifgggffihgihfggighgiiif
fjgfgjhhjiifhjgehgghfhhfhjhigggghghihigghhihihgiighgfhlgjfgjjjmfl
```

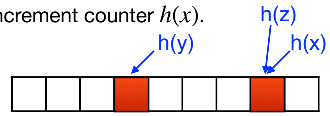
  - Estimated number of distinct words: 30897 (correct answer is 28239, ie. relative error of 9.4%).
- **Composable.**
  - Data streams S_1 and S_2 with sketches $sk(S_1)$ and $sk(S_2)$
  - There exists an efficiently computable function f such that

$$sk(S_1 \cup S_2) = f(sk(S_1), sk(S_2))$$

CountMin Sketch

Frequency Estimation

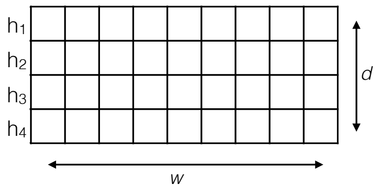
- Frequency estimation. Construct a sketch such that can estimate the frequency f_i of any element $i \in [n]$.
- First try.
  - array of counters of width w. Counters all initialized to be zero.
  - a pariwise independent hash function $h : [n] \to [w]$.
  - When item x arrives increment counter $h(x)$.



  - $E[\hat{f}_i] \leq f_i + m/w$
  - $P[\hat{f}_i \geq f_i + 2m/w] \leq 1/2$

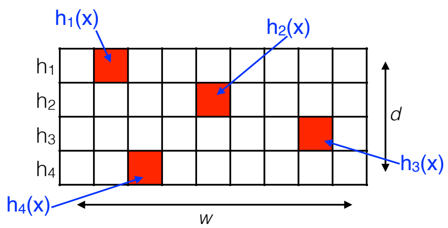
CountMin Sketch

- Fixed array of counters of width w and depth d. Counters all initialized to be zero.
- Pariwise independent hash function for each row $h_i : [n] \to [w]$.
- When item x arrives increment counter $h_i(x)$ of in all rows.



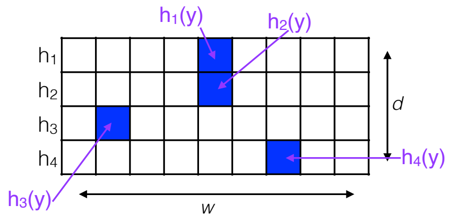
CountMin Sketch

- Fixed array of counters of width w and depth d. Counters all initialized to be zero.
- Pariwise independent hash function for each row $h_i : [n] \to [w]$.
- When item x arrives increment counter $h_i(x)$ of in all rows.



CountMin Sketch

- Fixed array of counters of width w and depth d. Counters all initialized to be zero.
- Pariwise independent hash function for each row $h_i : [n] \to [w]$.
- When item x arrives increment counter $h_i(x)$ of in all rows.
- Estimate frequency of y: return minimum of all entries y hash to.

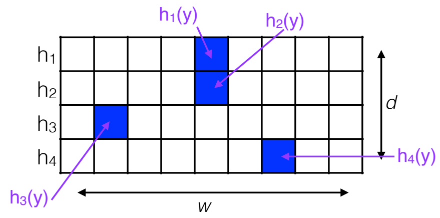


CountMin Sketch

Algorithm 1: CountMin

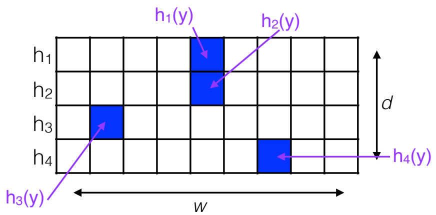
Initialize d independent hash functions $h_j : [n] \to [w]$.
Set counter $C_j(b) = 0$ for all $j \in [d]$ and $b \in [w]$.
while *Stream S not empty* **do**
  if *Insert(x)* **then**
    for $j = 1 \ldots d$ **do**
      $C_j(h_j(x)) = +1$
    end
  else if *Frequency(i)* **then**
    return $\hat{f}_i = \min_{j \in [d]} C_j(h_j(i))$.
  end
end

- The estimator $\hat{f}_i$ has the following property:
  - $\hat{f}_i \geq f_i$
  - $\hat{f}_i \leq f_i + 2m/w$ with probability at least $1 - (1/2)^d$



CountMin Sketch: Analysis

- Use $w = 2/\varepsilon$ and $d = \lg(1/\delta)$.
- The estimator $\hat{f}_i$ has the following property:
  - $\hat{f}_i \geq f_i$
  - $\hat{f}_i \leq f_i + \varepsilon m$ with probability at least $1 - \delta$
- **Space.** $O(dw) = O(2\lg(1/\delta)/\varepsilon) = O(\lg(1/\delta)/\varepsilon)$ words.
- **Query and processing time.** $O(d) = O(\lg(1/\delta))$



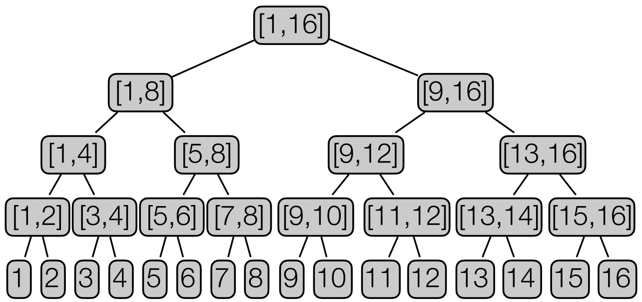
Applications of CountMin Sketch

- We can use the CountMin Sketch to solve e.g.:
  - **Heavy hitters:** List all heavy hitters (elements with frequency at least m/k).
  - **Range(a,b):** Return (an estimate of) the number of elements in the stream with value between a and b.
- Exercise.
  - How can we solve heavy hitters with a single CountMin sketch?
  - What is the space and query time?

Dyadic Intervals

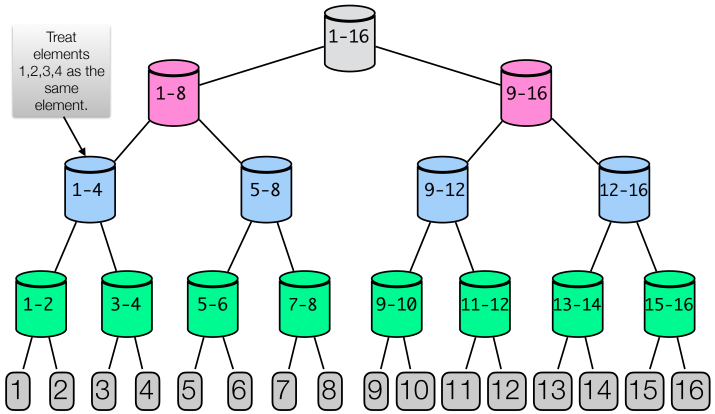
- **Dyadic intervals.** Set of intervals:

$$\{[j\frac{n}{2^i}+1,\ldots,(j+1)\frac{n}{2^i}] \mid 0 \le i \le \lg n,\ 0 \le j \le 2^{i-1}\}$$



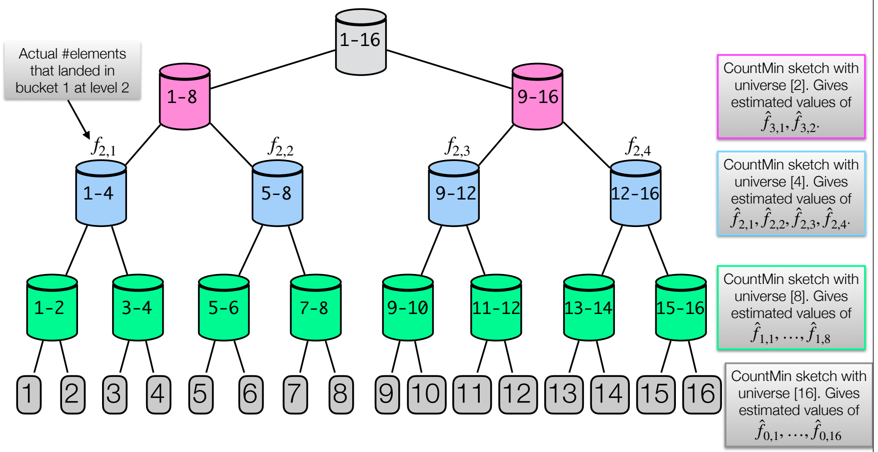
Heavy Hitters

- **Heavy Hitters.** Store a CountMin Sketch for each level in the tree of dyadic intervals (same d and w for all sketches).
  - On a level: Treat all elements in same bucket/interval as the same element.



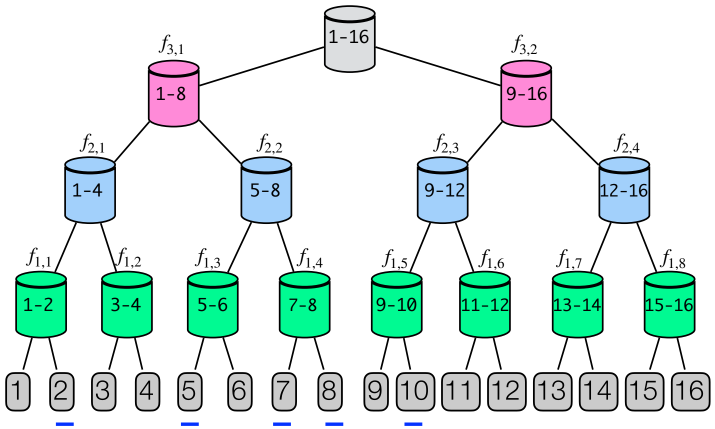
Heavy Hitters

- **Heavy Hitters.** Store a CountMin Sketch for each level in the tree of dyadic intervals (same d and w for all sketches).
  - On a level: Treat all elements in same bucket/interval as the same element.



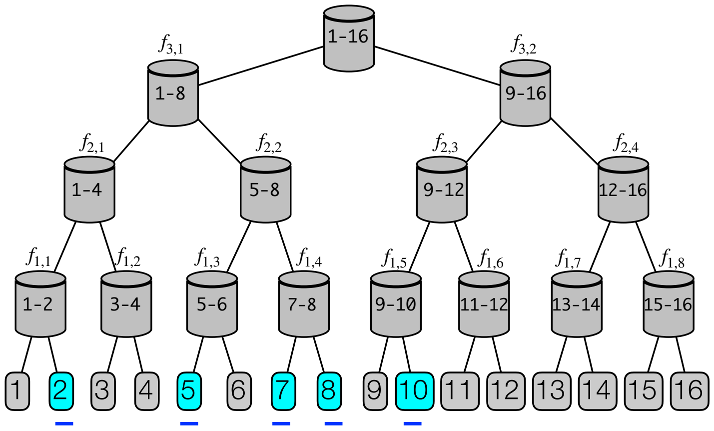
Heavy Hitters

- **Heavy Hitters.** Store a CountMin Sketch for each level in the tree of dyadic intervals (same d and w for all sketches).
  - On a level: Treat all elements in same bucket/interval as the same element.



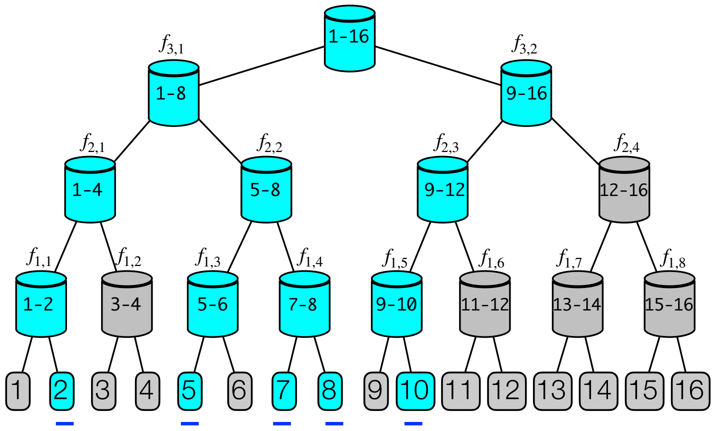
Heavy Hitters

- **Heavy Hitters.** Store a CountMin Sketch for each level in the tree of dyadic intervals (same d and w for all sketches).
  - On a level: Treat all elements in same bucket/interval as the same element.



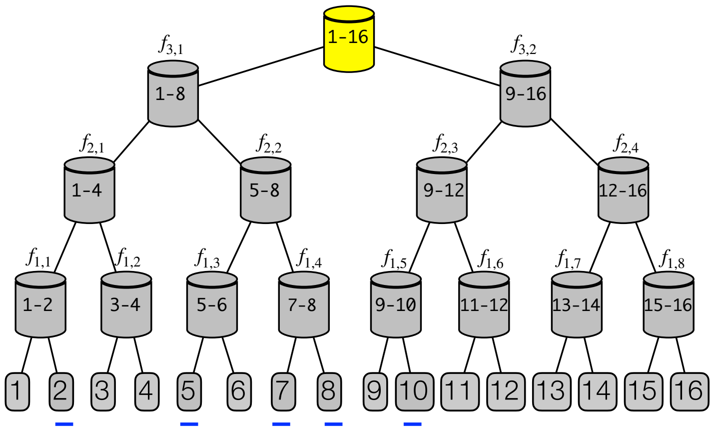
Heavy Hitters

- **Heavy Hitters.** Store a CountMin Sketch for each level in the tree of dyadic intervals (same d and w for all sketches).
  - On a level: Treat all elements in same bucket/interval as the same element.



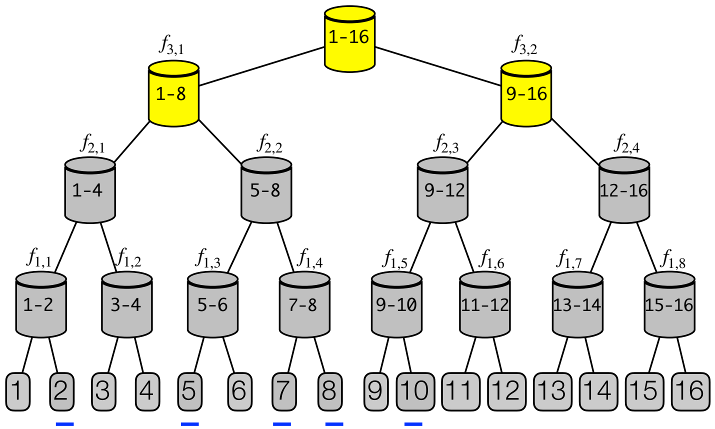
Heavy Hitters

- **Heavy Hitters.**
  - traverse tree from root.
  - only visit children with estimated frequency ≥ m/k



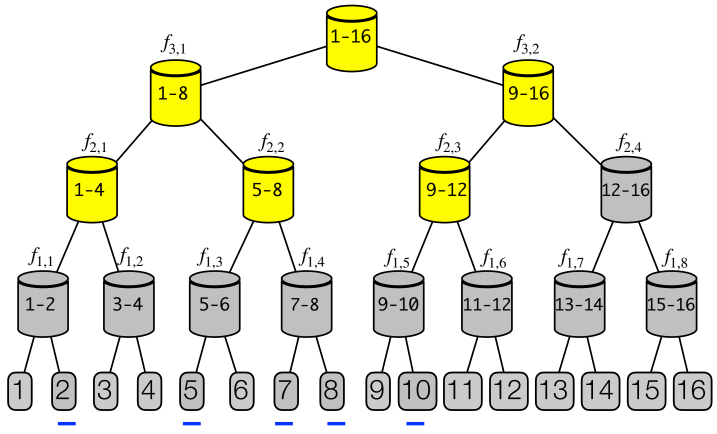
Heavy Hitters

- **Heavy Hitters.**
  - traverse tree from root.
  - only visit children with estimated frequency ≥ m/k



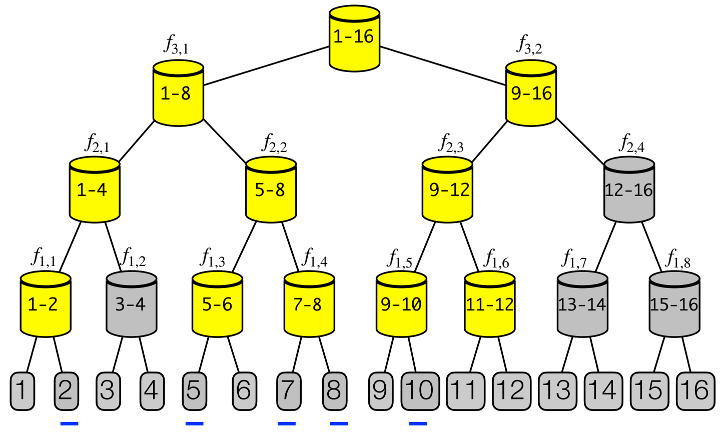
Heavy Hitters

- **Heavy Hitters.**
  - traverse tree from root.
  - only visit children with estimated frequency ≥ m/k



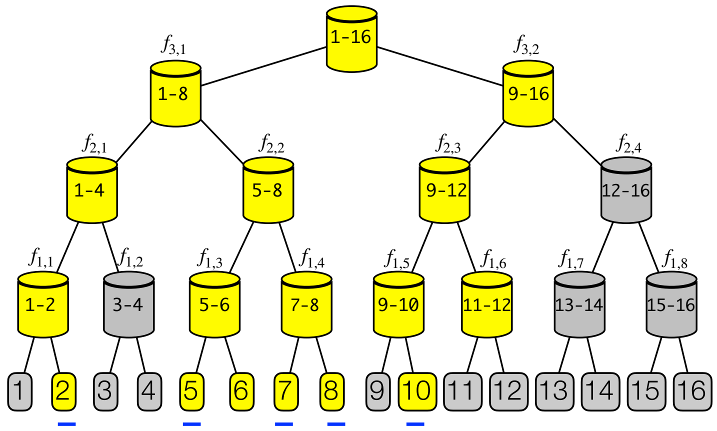
Heavy Hitters

- **Heavy Hitters.**
  - traverse tree from root.
  - only visit children with estimated frequency ≥ m/k



Heavy Hitters

- **Heavy Hitters.**
  - traverse tree from root.
  - only visit children with estimated frequency ≥ m/k

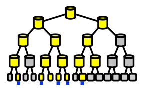


Heavy Hitters

- **Heavy Hitters.**
  - Store a CountMin sketch for each level in the tree of dyadic intervals (same d and w for all sketches).
  - On a level: Treat all elements in same bucket/interval as the same element.
  - **To find heavy hitters:**
    - traverse tree from root.
    - only visit children with estimated frequency ≥ m/k

- **Analysis.**
  - **Time.** *Assume CountMin sketch makes no large errors.*
    - Number of intervals queried: $O(k \lg n)$.
    - Query time: $O(k \lg n \cdot \lg(1/\delta))$
  - **Space.**

$$O\left(\lg n \cdot \frac{1}{\epsilon} \lg\left(\frac{1}{\delta}\right)\right) \text{ words.}$$



Count Sketch

Algorithm 2: CountSketch

Initialize d independent hash functions $h_j : [n] \to [w]$.
Initialize d independent hash functions $s_j : [n] \to \{\pm 1\}$.
Set counter $C[j, b] = 0$ for all $j \in [d]$ and $b \in [w]$.
while *Stream S not empty* **do**
  if *Insert(x)* **then**
    for $j = 1 \dots d$ **do**
      $C[j, h_j(x)] =+ s_j(i)$
    end
  else if *Frequency(i)* **then**
    $\hat{f}_{ij} = C(h_j(i)) \cdot s_j(i)$
    return $\widetilde{f}_{ij} = \text{median}_{j \in [d]} \hat{f}_{ij}$
  end
end

	Space	Error
Count-Min	$O\left(\frac{1}{\epsilon} \log n\right)$	ϵF_1 (one-sided)
Count-Sketch	$O\left(\frac{1}{\epsilon^2} \log n\right)$	$\epsilon \sqrt{F_2}$ (two-sided)