Randomized Algorithms

- Deterministic and (Las Vegas & Monte Carlo) Randomized Algorithms
- Probability Review
- Probabilistic Analysis of deterministic QUICK-SORT Algorithm
- RANDOMIZED-SELECT and RANDOMIZED-QUICK-SORT
- Max-Cut
- Min-Cut
- MAX-3-SAT and Derandomization
- Closest pair
- Hashing, Bloom filters, Streams, Sampling, Reservoir sampling, Sketch

IMDAD ULLAH KHAN

The dictionary ADT

- A dictionary maintains a set of n elements from a universe U
- Unique elements; element is known by its 'key' k
- Elements could be compound $(key, value)$ pairs
- Example: student ID as key and score as value

$$`16020102' : 17 \qquad `11010051' : 84$$

$$`11050001' : 22 \qquad `12060009' : 92$$

- Required operations: INSERT, LOOKUP, DELETE
- Dictionary can be implemented using the data structure
  - array ▷ sorted or unsorted
  - linked list ▷ sorted or unsorted
  - binary search trees ▷ balanced or unbalanced
  - hash tables

Dictionary Implementations

Unsorted Array:

- LOOKUP: Linear search - traverse array sequentially $\triangleright O(n)$
- INSERT: Insertion at the end of array (first empty slot) $\triangleright O(1)$
- DELETE: Given a position, shift left remaining elements $\triangleright O(n)$

Sorted Array:

- LOOKUP: Binary search $\triangleright O(\log n)$
- INSERT: Lookup to find position and shift to make space $\triangleright O(n)$
- DELETE: Given a position, shift left remaining elements $\triangleright O(n)$

Binary Search Tree:

- LOOKUP: Compare with root recursively in appropriate subtree $\triangleright O(h)$
- INSERT: Lookup for appropriate leaf position to insert node $\triangleright O(h)$
- DELETE: Given key, lookup to find node to remove and recursively link parent with one of the children $\triangleright O(h)$

Direct-Address Table

- How can all operations be done in $\mathcal{O}(1)$?
- Let each position in table correspond to a key in the universe U

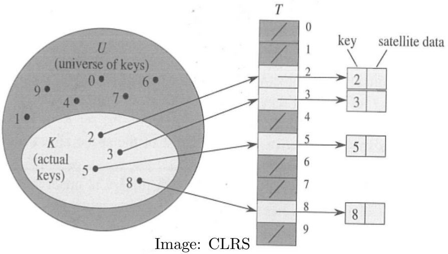


Image: CLRS

- Large universe $\implies$ large unused space
- If no satellite data, keys can be stored in a bit-vector
- For n elements, space taken by bit-vector $\ll$ array
- How can $\mathcal{O}(1)$ be achieved without wasting space?

Hash Table

- Let $m \in \mathbb{Z}^+$ and $h : U \to [m]$
- Make an array (or table) $T[1, \ldots, m]$
- LOOKUP: **return** $T[h(k)]$ $\triangleright$ $O(1)$
- INSERT: Store at $T[h(k)]$ $\triangleright$ $O(1)$
- DELETE: Remove from $T[h(k)]$ $\triangleright$ $O(1)$

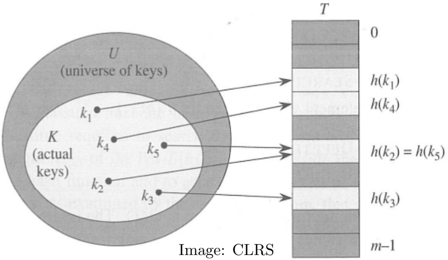


Image: CLRS

- What if $h(k_x) = h(k_y)$? Collisions occur
- How can k_x and k_y both be stored?

Chained Hash Table

- Let $T[i]$ be an array or list for $1 \leq i \leq m$
- LOOKUP: Lookup in list $T[h(k)]$
- INSERT: Insert in list $T[h(k)]$
- DELETE: Delete from list $T[h(k)]$

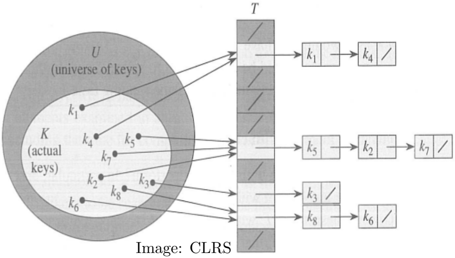


Image: CLRS

- Runtime of all operations: O(length of list in $T[k]$)
- How can we ensure the length of lists in T is not too large?

- Can the hash function h involve randomness?

  ▷ **No!** An element must always hash to the same list in T
- Radomly choose a hash function to use
- For $z \in U$, $h(z)$ is chosen uniformly at random from $\{0, \cdots, m-1\}$
- For any $x_i \in U$, let random variable $C_i = \begin{cases} 1 & \text{if } h(x_i) = h(z) \\ 0 & \text{otherwise} \end{cases}$
- Let X be the number of elements in the same list as z
- $X = \sum_{x_i \neq z} C_i$ Then,

$$E[X] = E[\sum_{x_i \neq z} C_i] = \sum_{x_i \neq z} E[C_i] = \sum_{x_i \neq z} Pr[h(x_i) = h(z)] = \sum_{x_i \neq z} \frac{1}{m} \leq \frac{n}{m}$$

- Expected runtime of operations is $\mathcal{O}(1 + E[X]) = \mathcal{O}(1 + n/m)$
- Space-time tradeoff: larger $m \implies$ lower expected runtime

A family of hash functions $\mathcal{H}$ is 2-universal iff for any $x, y \in_{x \neq y} U$, if $h \in \mathcal{H}$ is chosen uniformly at random, then $Pr[h(x) = h(y)] \leq 1/m$

Desired properties from hashing

- Small range (m) and fewer collisions
- Easy to evaluate hash value for any key with small space complexity

Linear Congruential Generators for $U = \mathbb{Z}$

- Pick a prime number $p > m$
- For any two integers a and b $(1 \leq a \leq p-1)$, $(0 \leq b \leq p-1)$
- A hash function $h_{a,b} : U \mapsto [m]$ is defined as

$$h_{a,b}(x) = [(ax + b) \mod p] \mod m$$

$\mathcal{H} := \{h_{a,b} \ : \ 1 \leq a \leq p-1 \ , \ 0 \leq b \leq p-1\}$ is 2-universal

Picking a random $h \in \mathcal{H}$ amounts to picking random a and b

Data Streams

- A data stream is a massive sequence of data
- Too large to store (on disk, memory, cache, etc.)
  - Social media (twitter feed, foursquare checkins)
  - Web click stream analysis
  - Search Query Stream Analysis
  - Sensor data (weather, radars, cameras, IoT devices, energy data)
  - Network traffic (trajectories, source/destination pairs)
  - Financial Data
  - Satellite data feed
- How to deal with such data?
- What are the issues?

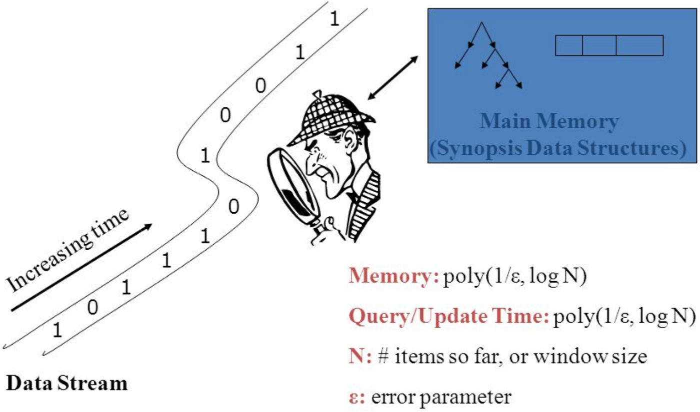
Characteristics of Data Stream

- Huge volumes of continuous data, possibly infinite
- Fast changing and requires fast, real-time response
- Data stream captures nicely our data processing needs of today
- Random access is expensive
- Single scan algorithm (can only have one look)
- Store only the summary of the data seen so far
- Most stream data are pretty low-level or multidimensional in nature, needs multi-level and multi-dimensional processing

- Data items can be complex types
  - Documents (tweets, news articles)
  - Images
  - geo-located time-series
  - $\cdots$
- To study basic algorithmic ideas we abstract away application-specific details
- Consider the data stream as a sequence of numbers

Stream Model of Computation

Motwani, PODS (2002)



Stream Model of Computation

Stream $\mathcal{S} := a_1, a_2, a_3, \ldots, a_m$ $\triangleright$ m may be unknown

Each $a_i \in [n]$

Goal: Compute a function of the stream $\mathcal{S}$ (e.g. mean, median, number of distinct elements, frequency moments..)

Subject to

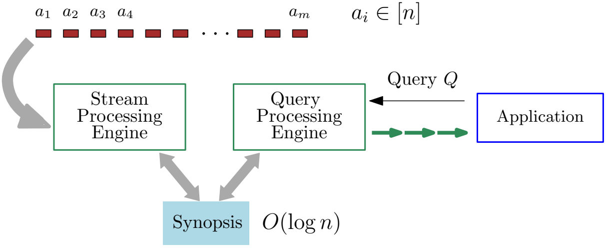
- Single pass, read each element of $\mathcal{S}$ only once sequentially
- Per item processing time $O(1)$
- Use memory polynomial in $O(1/\epsilon, 1/\delta, \log n)$
- Return (ϵ, δ)-randomized approximate solution

Fundamental Methodology: Keep a synopsis of the stream and answer query based on it. Update synopsis after examining each item in $O(1)$

Synopsis: Succinct summary of the stream (so far) (poly-log bits)

Families of Synopsis

- Sliding Window
- Random Sample
- Histogram
- Wavelets
- Sketch



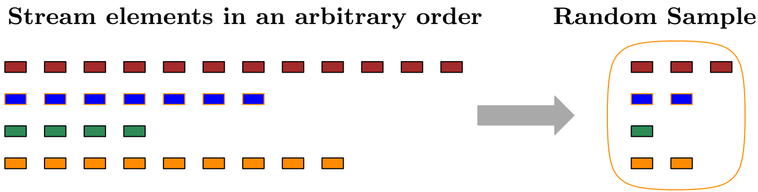
How to Tackle Massive Data Streams

- A general and powerful technique: Sampling
- Idea:
  1. Keep a random sample of the data stream
  2. Perform the computation on the sample
  3. Extrapolate
- Example: Compute the median of a data stream (How to extrapolate in this case?)
- Sampling Techniques: How to keep a random sample of a data stream?

Random Sample

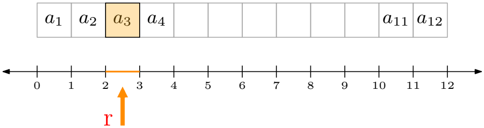
Keep a "representative" subset of the stream

Approximately compute query answer from sample (with appropriate scaling etc.)



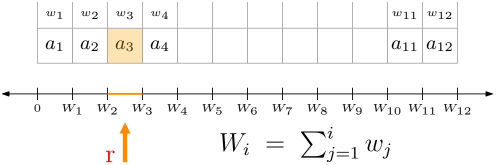
Sample a random element from array A of length n $\triangleright$ $A[i]$ with prob $1/n$

- Generate a random number $r \in [0, n]$ $\triangleright$ $r \leftarrow \text{RAND}() \times n$
- Return $A[\lceil r \rceil]$



Sample random element (by weight) from array A $\triangleright$ $A[i]$ with prob. w_i/W

- Generate a random number $r \in [0, \sum_{j=1}^{n} w_i]$ $\triangleright$ $r \leftarrow \text{RAND}() \times W_n$
- Return $A[i]$ if $W_{i-1} \leq r < W_i$



$$W_i = \sum_{j=1}^{i} w_j$$

Sample a random element from the stream S ▷ a_i with prob. $1/m$

- If m is known, use algorithm for sampling from array. For unknown m

Algorithm : Reservoir Sampling ($\mathcal{S}$)

$R \leftarrow a_1$ ▷ R (reservoir) maintains the sample

for $i \geq 2$ **do**

Pick a_i with probability $1/i$

Replace with current element in R

Prob. that a_i is in the sample R_m (m: stream length or query time)

$$= \underbrace{\text{Pr that } a_i \text{ was selected at time } i}_{\frac{1}{i}} \times \underbrace{\text{Pr that } a_i \text{ survived in } R \text{ until time } m}_{\prod_{j=i+1}^{m}\left(1-\frac{1}{j}\right)}$$

$$= \frac{1}{\cancel{i}} \times \frac{\cancel{i}}{\cancel{i+1}} \times \frac{\cancel{i+1}}{\cancel{i+2}} \times \frac{\cancel{i+2}}{\cancel{i+3}} \times \ldots \times \frac{\cancel{m-2}}{\cancel{m-1}} \times \frac{\cancel{m-1}}{m} = \frac{1}{m}$$

Sample k random elements from the stream S ▷ a_i with prob. k/m

Algorithm : Reservoir Sampling $(\mathcal{S}, k)$

$R \leftarrow a_1, a_2, \ldots, a_k$ ▷ R (reservoir) maintains the sample
for $i \geq k+1$ **do**
  Pick a_i with probability k/i
  If a_i is picked, replace with it a randomly chosen element in R

Prob. that a_i is in the sample R_m (m: stream length or query time)

$$= \underbrace{\text{Pr that } a_i \text{ was selected at time } i}_{\frac{k}{i}} \times \underbrace{\text{Pr that } a_i \text{ survived in } R \text{ untill time } m}_{\prod_{j=i+1}^{m} \left(1 - \left(\frac{k}{j} \times \frac{1}{k}\right)\right)}$$

$$= \frac{k}{\cancel{i}} \times \frac{\cancel{i}}{\cancel{i+1}} \times \frac{\cancel{i+1}}{\cancel{i+2}} \times \frac{\cancel{i+2}}{\cancel{i+3}} \times \ldots \times \frac{\cancel{m-2}}{\cancel{m-1}} \times \frac{\cancel{m-1}}{m} = \frac{k}{m}$$

Data Stream: Linear Sketch

- Sample is a general purpose synopsis
- Process sample only – no advantage from observing the whole stream
- Sketches are specific to a particular purpose (query)
- Sketches benefit from the whole stream (though can't save all)

A linear sketch interprets the stream as defining the frequency vector

Often we are interested in functions of the frequency vector from a stream

$\mathcal{S}: \ a_1, a_2, a_3, a_4, \dots, a_m$

$a_i \in [n]$

$\mathbf{F}:$

1	2	3		n
f_1	f_2	f_3	$\dots \quad \dots$	f_n

$f_j = |\{a_i \in \mathcal{S} : a_i = j\}|$ (frequency of j in $\mathcal{S}$)

$\mathcal{S}: \ 2, 5, 6, 7, 8, 2, 1, 2, 7, 5, 5, 4, 2, 8, 8, 9, 5, 6, 4, 4, 2, 5, 5$

$\mathbf{F}:$

1	2	3	4	5	6	7	8	9
1	5	0	3	6	2	2	3	1

$$\mathcal{S} = < a_1, a_2, a_3, \ldots, a_m > \qquad a_i \in [n]$$

f_i : frequency of i in $\mathcal{S}$ $\qquad$ $\mathbf{F} = \{f_1, f_2, \ldots, f_n\}$

$F_0 := \sum_{i=1}^{n} f_i^0$ $\qquad$ ▷ number of distinct elements

$F_1 := \sum_{i=1}^{n} f_i$ $\qquad$ ▷ length of stream, m

$F_2 := \sum_{i=1}^{n} f_i^2$ $\qquad$ ▷ second frequency moment

Count-Min Sketch

- Count-Min sketch (Cormode & Muthukrishnan 2005) for frequency estimates
- Cannot store frequency of every elements
- Store total frequency of random groups (elements in hash buckets)

Algorithm : Count-Min Sketch (k, ϵ, δ)

$\text{COUNT} \leftarrow \text{ZEROS}(k)$ ▷ sketch consists of k integers

Pick a random $h : [n] \mapsto [k]$ from a 2-universal family $\mathcal{H}$

On input a_i

  $\text{COUNT}[h(a_i)] \leftarrow \text{COUNT}[h(a_i)] + 1$ ▷ increment count at index $h(a_i)$

On query j ▷ query: $\mathbf{F}[j] =?$

  return $\text{COUNT}[h(j)]$

Algorithm : Count-Min Sketch (k, ϵ, δ)

$\text{COUNT} \leftarrow \text{ZEROS}(k)$ ▷ sketch consists of k integers

Pick a random $h : [n] \mapsto [k]$ from a 2-universal family $\mathcal{H}$

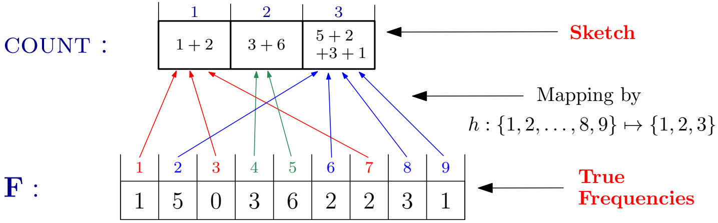
On input a_i

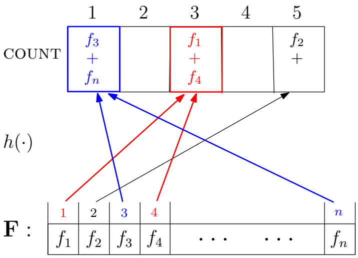
  $\text{COUNT}[h(a_i)] \leftarrow \text{COUNT}[h(a_i)] + 1$ ▷ increment count at index $h(a_i)$

On query j ▷ query: $\mathbf{F}[j] = ?$

  return $\text{COUNT}[h(j)]$

$$\mathcal{S} : \quad 2, 5, 6, 7, 8, 2, 1, 2, 7, 5, 5, 4, 2, 8, 8, 9, 5, 6, 4, 4, 2, 5, 5$$

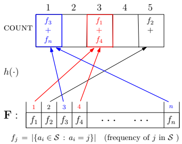




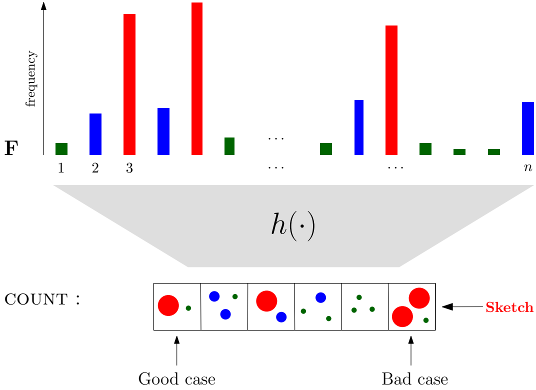
$f_j = |\{a_i \in \mathcal{S} : a_i = j\}|$ (frequency of j in $\mathcal{S}$)

- $k = \frac{2}{\epsilon}$
- Large k means better estimate (smaller groups) but more space
- $\tilde{f}_j$: estimate for f_j – output of algorithm

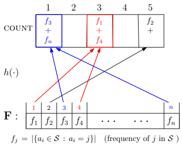
- $k = {}^{2}/_{\epsilon}$
- Large k means better estimate but more space
- $\tilde{f}_j$: estimate for f_j – output of algorithm



Bounds on $\tilde{f}_j$: (idea)



- $k = {}^{2}/_{\epsilon}$
- Large k means better estimate but more space
- $\tilde{f}_j$: estimate for f_j – output of algorithm



Bounds on $\tilde{f}_j$: (idea)

1. $\tilde{f} \geq f_j$
   - Other elements that hash to $h(j)$ contribute to $\tilde{f}_j$
2. $Pr\left[\, \tilde{f}_j \leq f_j + \epsilon \|F\|_1 \,\right] \geq \frac{1}{2}$
   - $X_j = \tilde{f}_j - f_j$ ▷ Excess in $\tilde{f}_j$ (error)
   - $X_j = \sum_{i \in [n] \setminus j} f_i \cdot 1_{h(i)=h(j)}$ ▷ $1_{condition}$ is indicator of condition

$$\mathbb{E}(X_j) = \mathbb{E}\left(\sum_{i \in [n] \setminus j} f_i \cdot 1_{h(i)=h(j)} \right) = \sum_{i \in [n] \setminus j} f_i \cdot \frac{1}{k} \leq \sum_{i \in [n] \setminus j} \|F\|_1 \cdot \frac{\epsilon}{2}$$

   - By Markov inequality we get the bound

Idea: Amplify the probability of the basic count-min sketch

Keep t over-estimates, $t = \log(1/\delta)$, $k = 2/\epsilon$ and return their minimum

Unlikely that all t functions hash j with very frequent elements

Algorithm : Count-Min Sketch (k, ϵ, δ)

$\textsc{count} \leftarrow \textsc{zeros}(t \times k)$ ▷ sketch consists of t rows of k integers

Pick t random functions $h_1, \ldots, h_t : [n] \mapsto [k]$ from a 2-universal family

On input a_i

for $r = 1$ to t **do**

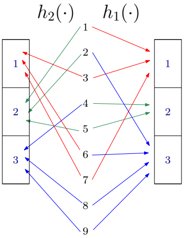
$\textsc{count}[r][h_r(a_i)] \leftarrow \textsc{count}[r][h_r(a_i)] + 1$

▷ increment $\textsc{count}[r]$ at index $h_r(a_i)$

On query j ▷ query: $\mathbf{F}[j] = ?$

return $\min\limits_{1 \leq r \leq t} \textsc{count}[r][h_r(j)]$

Count-Min Sketch



$\mathcal{S}$: 2, 5, 6, 7, 8, 2, 1, 2, 7, 5, 5, 4, 2, 8, 8, 9, 5, 6, 4, 4, 2, 5, 5

COUNT : ← **Sketch**

	1	2	3
$h_1(\cdot)$	$0+1+2$	$3+6$	$5+2$ $+3+1$
$h_2(\cdot)$	$0+2+2$	$1+5+6$	$3+3+1$

F : ← **True Frequencies**

1	2	3	4	5	6	7	8	9
1	5	0	3	6	2	2	3	1

On input a

$h_1(a)$ $h_t(a)$

	1	2	3	...	...	k
COUNT[1][$\cdot$]		+1				
COUNT[2][$\cdot$]					+1	
COUNT[3][$\cdot$]				+1		
$\vdots$						
COUNT[t][$\cdot$]						+1

On query a $\quad$ MIN_i COUNT[i][$h_i(a)$]

Count-Min Sketch

1. $\tilde{f}_j \; \geq \; f_j$
   - For every r, other elements that hash to $h_r(j)$ contribute to $\tilde{f}_j$
2. $\tilde{f}_j \; \leq \; f_j + \epsilon\|F\|_1$ with probability at least $1 - \delta$
   - X_{jr} : contribution of other elements to $Count[r][h_r(j)]$
   - $\Pr\left[X_{jr} \geq \epsilon\|F\|_1\right] \; \leq \; \frac{1}{2}$ for $k = 2/\epsilon$
   - The event $\tilde{f}_j \; \geq \; f_j + \epsilon\|F\|_1$ is $\forall\, 1 \leq r \leq t \quad X_{jr} \geq \epsilon\|F\|_1$
   - $\Pr\left[\, \forall\, r\; X_{jr} \geq \epsilon\|F\|_1\right] \; \leq \; \left(\frac{1}{2}\right)^t$
   - $t = \log(\frac{1}{\delta}) \implies \Pr\left[\, \forall\, r\; X_{jr} \geq \epsilon\|F\|_1\right] \; \leq \; \left(\frac{1}{2}\right)^{\log 1/\delta} \; = \; \delta$

- Count-Min sketch is an $(\epsilon\|F\|_1, \delta)$-additive approximation algorithm
- Space required is $k \cdot t$ integers $= O(1/\epsilon \log(1/\delta) \log n)$ (plus constant)