Maintaining k-MinHash signatures for fully dynamic data streams

Why k-MinHash Signatures?

- ***Efficiently*** detect similarities between large data sets

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

- Large sets → large computation

Recall: Key Property

- The probability that the MinHash values of two different sets are the same is the Jaccard similarity of those two sets

$$Pr[\textsc{MinHash}(A,h) = \textsc{MinHash}(B,h)] = J(A,B)$$

But how do we find this?

Recall: Key Property

- The probability that the MinHash values of two different sets are the same is the Jaccard similarity of those two sets

$$Pr[\textsc{MinHash}(A,h) = \textsc{MinHash}(B,h)] = J(A,B)$$

But how do we find this? We don't! We **estimate** it!

k-MinHash Signatures

- Compressed representations of sets
- *k* independent, uniformly random hash functions

k-MinHash signature of set *A*:

3	25	4	42	13	1	…	10	*k*

MinHash(A, h_1)

MinHash(A, h_k)

k-MinHash Signatures

- Don't compare a whole set, compare their signatures!

k-MinHash signature of set *A*:

3	25	4	42	13	1	…	10	*k*

k-MinHash signature of set *B*:

3	12	4	22	3	1	…	5	*k*

Problem

- Modern data-mining-intensive applications work with large sets of **continuously evolving data**
- Sets can have elements deleted and inserted
    - An entry of a set's MinHash signature can be deleted
    - A new element's hash can be smaller than the current MinHash

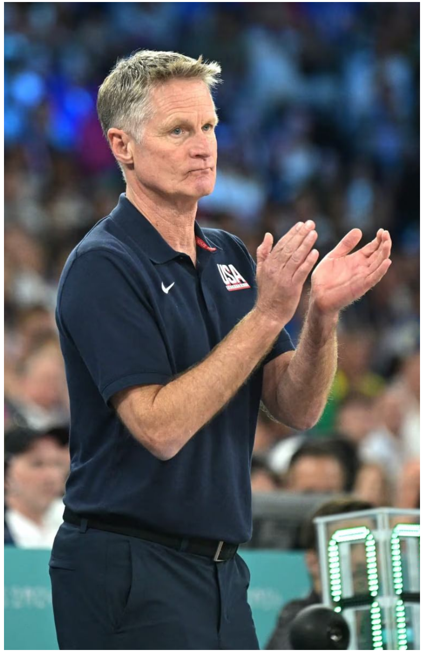
Solution: A New Data Structure

- ***The ℓ-buffered k-MinHash sketch***
    - Introduced by Clementi et al. in their paper "Maintaining *k*-MinHash Signatures over Fully-Dynamic Data Streams with Recovery"

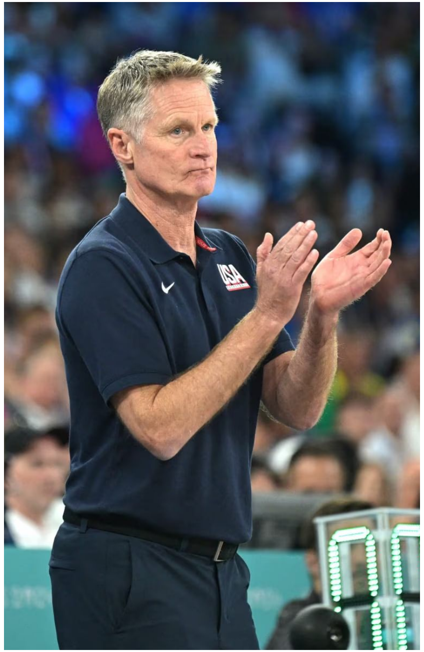
- Always returns the correct *k*-MinHash signature

- Deals with insertions and deletions pretty well!

l-buffered k-MinHash: Intuition



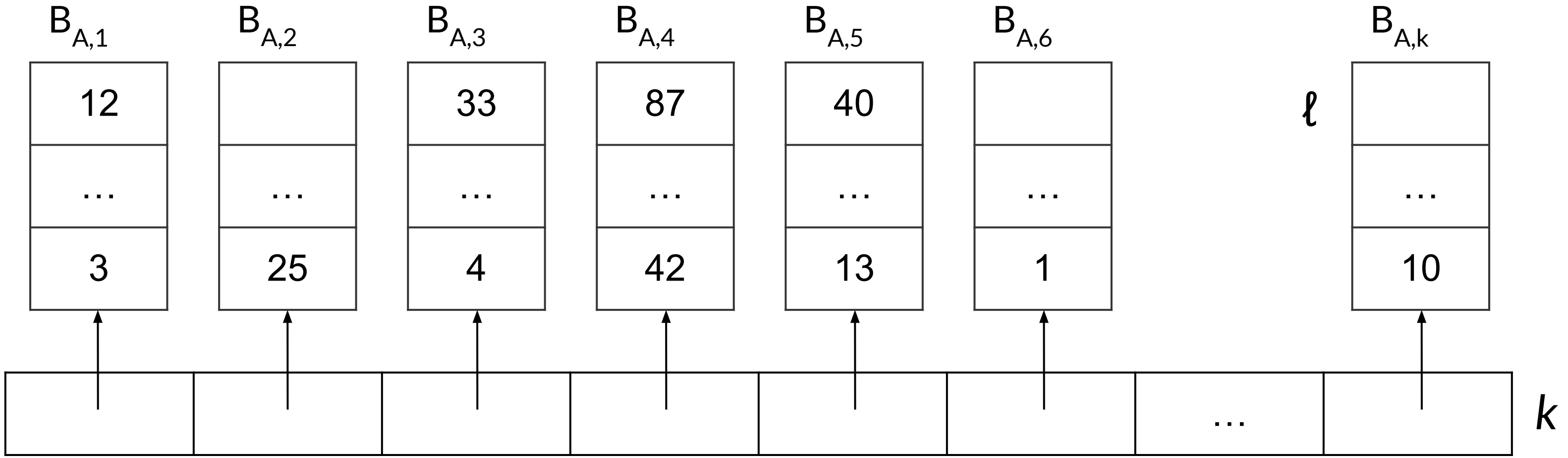
l-buffered k-MinHash: Intuition



l-buffered k-MinHash: Intuition

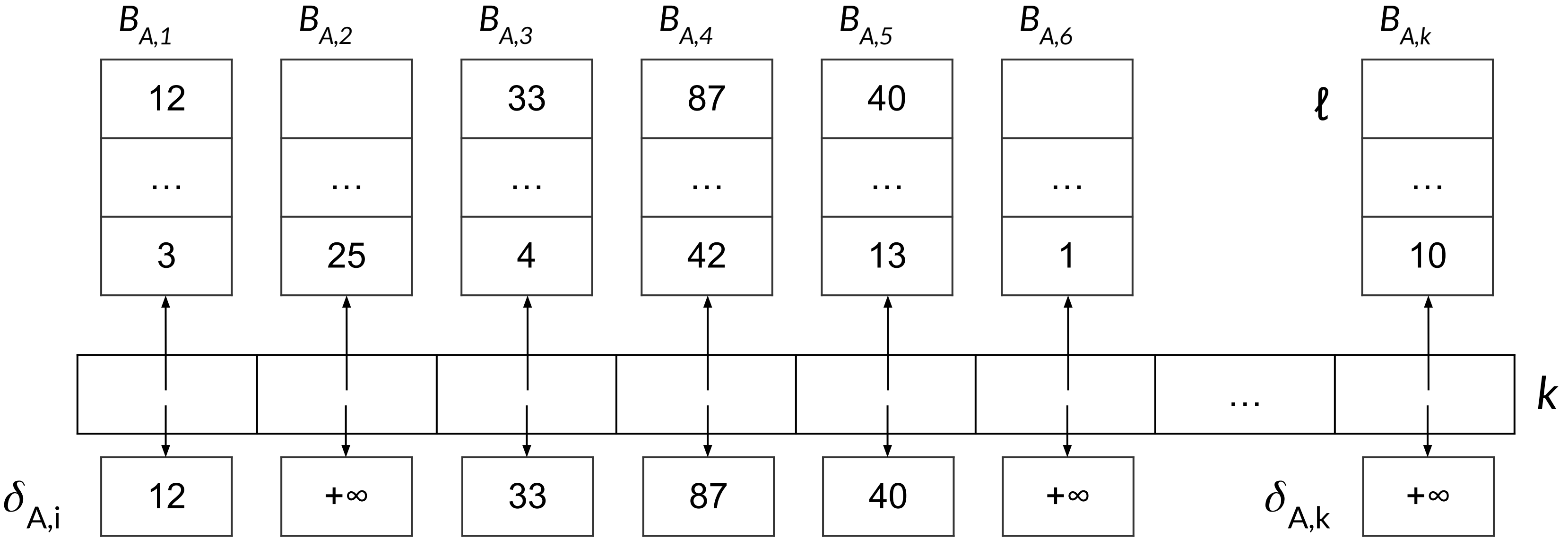
l-buffered *k*-MinHash Sketch

- Buffers *at most* the ℓ smallest hashes (implemented as balanced BSTs)



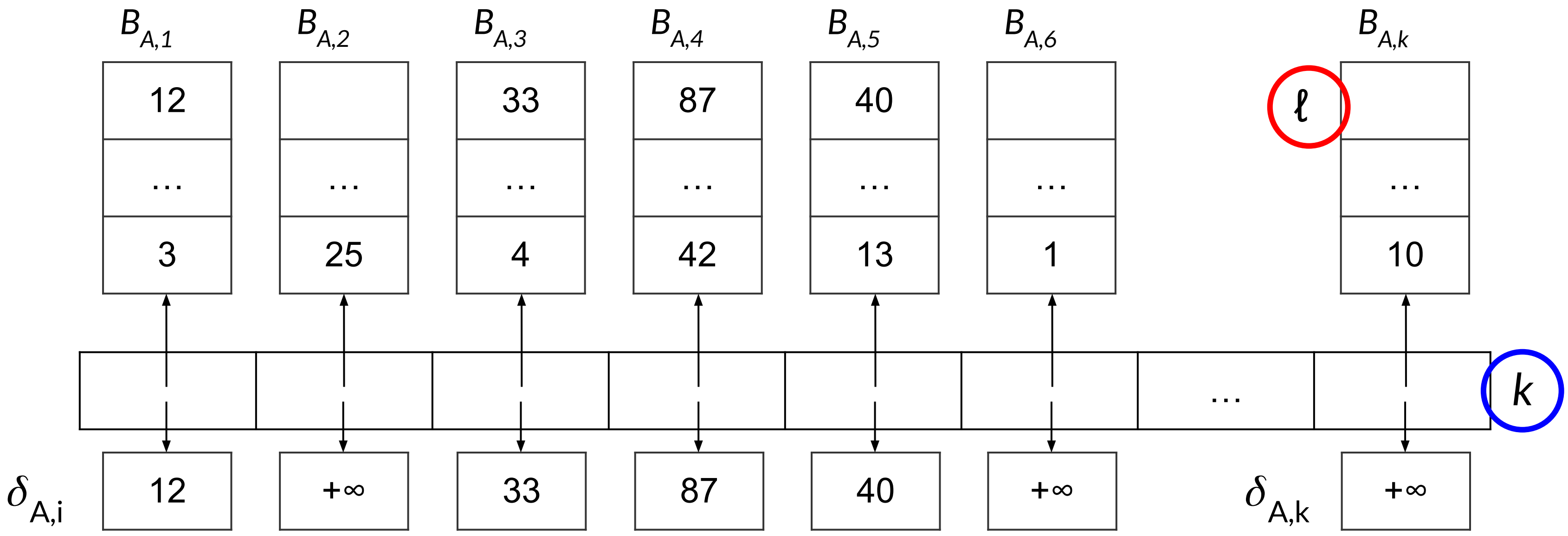
l-buffered *k*-MinHash Sketch

- Stores thresholds $\delta_{A,i}$



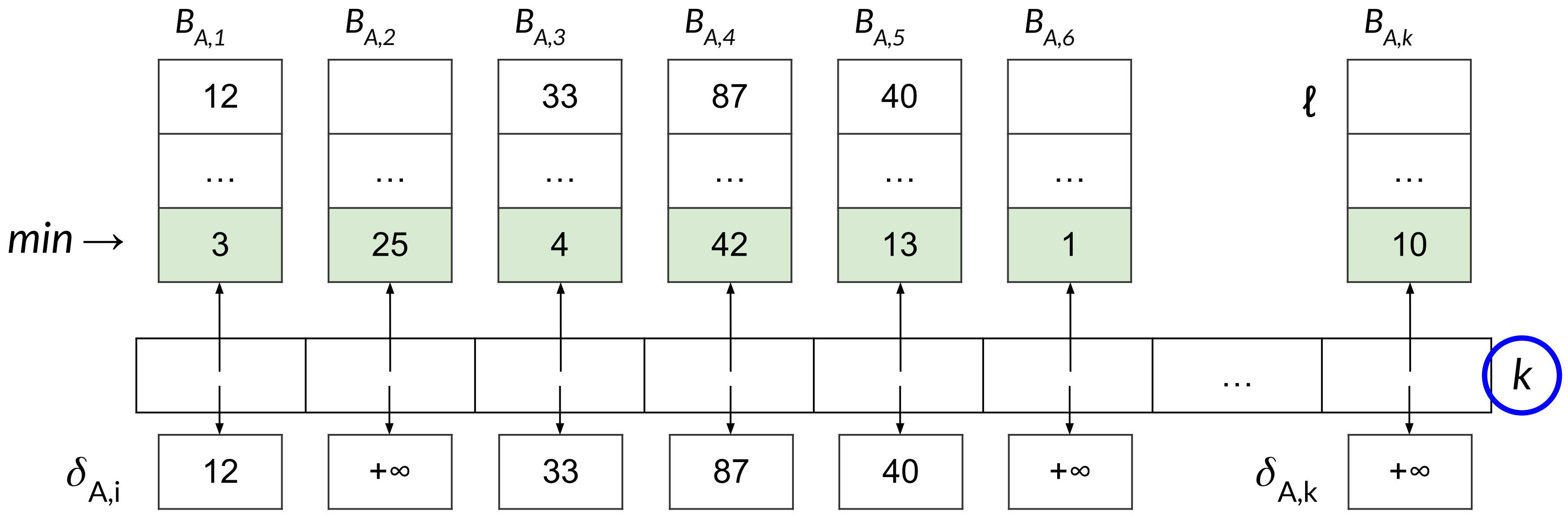
Memory Consumption

- Memory usage is **O(*k* *ℓ*)**



Signature Queries

- Get the minimum value of each buffer (*min* operation in a balanced BST)
- Runs in **O(*k*)** time



Sketch Initialization

Suppose $\ell = 4$ and we have a set $A = \{43, 72, 2, 300, 8, 10\}$

For each entry in the sketch:

1. We hash every element in A with h_i and get $\{2, 54, 4, 100, 55, 102\}$
2. We take the ℓ smallest hashes and put them in buffer $B_{A,i}$

$B_{A,i}$				
	2	4	54	55

Sketch Initialization

$B_{A,i}$	2	4	54	55

3. The buffer is full, so we set the threshold $\delta_{A,i} = max(B_{A,i}) = 55$
   a. If the buffer was not full, we would set $\delta_{A,i} = +\infty$

$\delta_{A,i}$	55

4. Put $(B_{A,i}, \delta_{A,i})$ in the sketch at entry i

Insertions

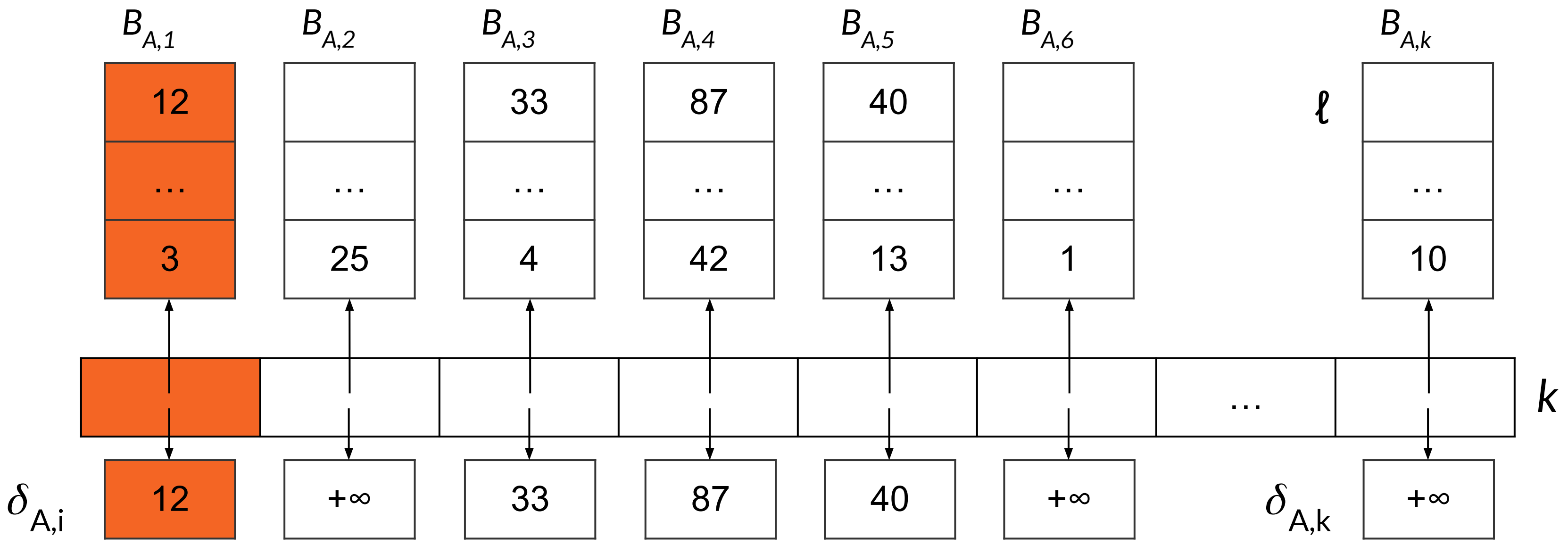
Suppose element x is inserted into set A.

For each buffer $B_{A,i}$, $i = \{1,..,k\}$:

1. Compute $h_i(x)$
2. If it's smaller than a buffer's threshold, recompute the ℓ smallest
3. Insert the new ℓ smallest hashes into the buffer
    a. If the buffer's length is full, recompute $\delta_{A,i} = max(B_{A,i})$

Insertions

For example, let's just look at one entry



Insertions

$B_{A,1}$

3	4	6	12

ℓ $\delta_{A,i}$ 12

- Suppose we insert a new element x, where $h_1(x) = 2$

$B_{A,1}$

2	3	4	6

ℓ $\delta_{A,i}$ 6

- Runs in **O(k log ℓ)** time
  - Balanced BSTs of size ℓ perform insert and delete operations in O(logℓ) time

Deletions

Suppose an element x is deleted from A.

For each buffer $B_{A,i}$, $i = \{1,..,k\}$:

1. Delete $h_i(x)$ from the buffer
2. If the buffer becomes empty (**fault**):
    a. Get the current state of set A with a **recovery query**
    b. Re-initialize the sketch with A

Deletions

Fault (**worst**) case:

Suppose an element x is deleted from A, where $h_1(x) = 3$, and we have this buffer:

$$B_{A,1}\ \begin{array}{|c|c|c|c|}\hline 3 & \quad & \quad & \quad \\ \hline\end{array}\ \ell \longrightarrow B_{A,1}\ \begin{array}{|c|c|c|c|}\hline \quad & \quad & \quad & \quad \\ \hline\end{array}\ \ell$$

- **Fault** $\rightarrow$ Re-initialize the sketch
- We can fine-tune ℓ to make faults very rare

Deletions

No fault (**average**) case:

Suppose an element x is deleted from A, where $h_1(x) = 3$, and we have this buffer:

$B_{A,1}$

3	4	6	12

ℓ $\longrightarrow$ $B_{A,1}$

	4	6	12

ℓ

- Runs in **O(*k* log ℓ)**
  - Balanced BSTs of size ℓ perform insert and delete operations in O(logℓ) time

Conclusion

- The **ℓ-buffered *k*-MinHash sketch** can maintain *k*-Minhash signatures for fully-dynamic data sets
- Memory usage: $O(k\,\ell)$
- Insertions: $O(k \log \ell)$
- Deletions: $O(k \log \ell)$ (amortized)

References

[1] A. Clementi, L. Gual`a, L. Pep`e Sciarria, and A. Straziota. Maintaining k-MinHash Signatures over Fully-Dynamic Data Streams with Recovery. In Proc. of the 18th ACM International Conference on Web Search and Data Mining (WSDM), 2025.

[2] J. Leskovec, A. Rajaraman, and J. D. Ullman. Mining of Massive Datasets (2nd ed.). Cambridge University Press, 2014. ISBN: 9781139924801.