Randomized Data Structures

Skip Lists, Bloom Filters, Count-Min Sketch

Jared Saia
University of New Mexico

Outline

- Skip Lists
- Bloom Filters
- Count-Min Sketch

Dictionary ADT

A dictionary ADT implements the following operations

- *Insert(x)*: puts the item x into the dictionary
- *Delete(x)*: deletes the item x from the dictionary
- *IsIn(x)*: returns true iff the item x is in the dictionary

Skip Lists

- An ordered dictionary with expected $O(\log n)$ search, insertion, and deletion time. Assume $n \geq 1$.
- Predecessor and successor queries use the same search path.
- The bounds hold for fixed keys and queries, independent of the random promotions.
- Rank and select (the i-th smallest key) additionally require each horizontal link to store how many bottom-level keys it spans.

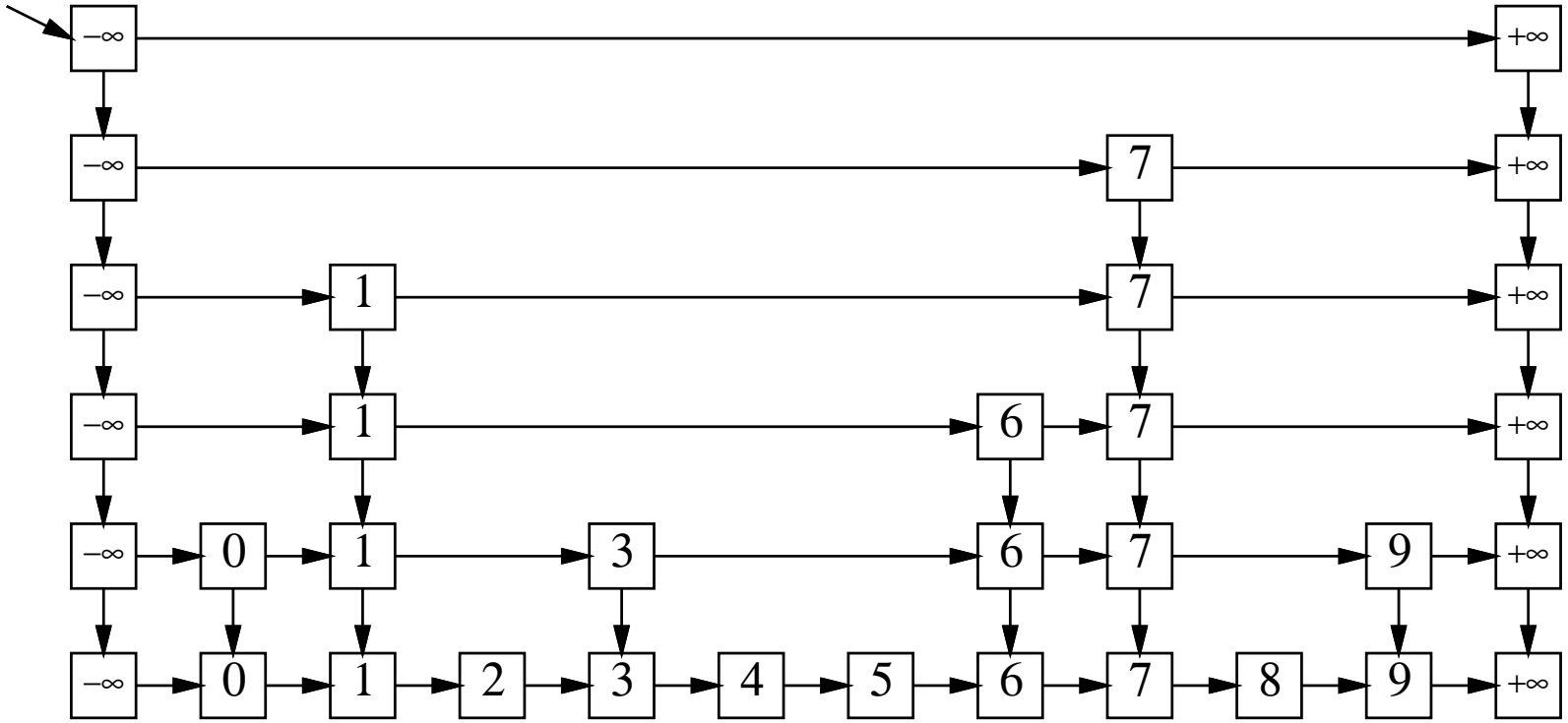
Skip-List Representation

- Distinct keys occupy sorted, doubly linked lists $L_0, L_1, \ldots, L_H$. Every key appears in L_0.
- Each level has leftmost and rightmost sentinels with keys $-\infty$ and $+\infty$.
- A key of height h has one copy at every level $0, \ldots, h$.
- H is the largest key height. An empty dictionary has only L_0.

Random Promotions

- For each new key, flip independent fair coins until the first tails. Its height is the number of heads.
- Promote the key one level for each heads. Coins for different keys are independent.
- Each copy stores its key and *left*, *right*, *up*, and *down* pointers. Missing vertical neighbors are NIL.
- Leftmost sentinels are also linked vertically.

Skip-List Example



The diagram includes an optional empty top level.

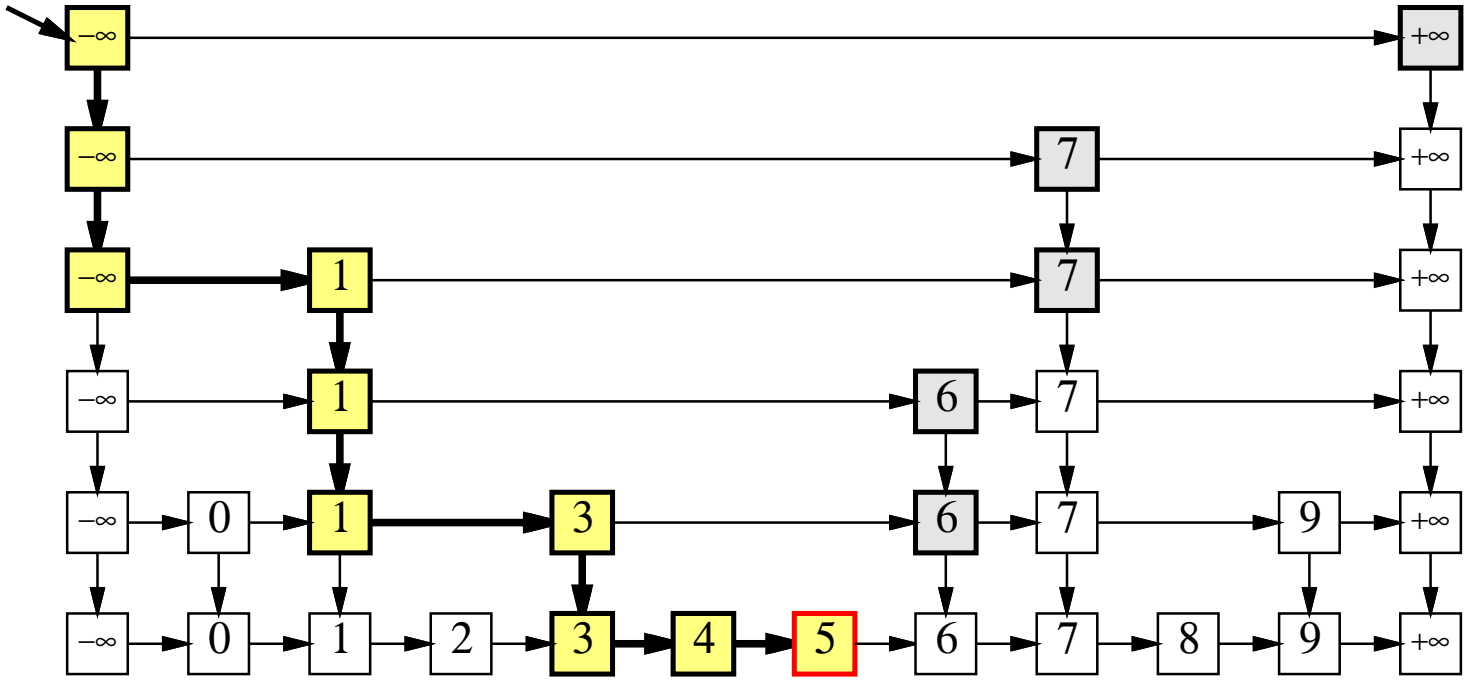
Search

- Start at the highest leftmost sentinel and move right while the next key is strictly less than x.
- When the next key is at least x, move down. Continue to level 0 even if x appears above it.
- At the bottom, let q be the next node to the right.
- Return q if its key equals x. Otherwise, return NIL.

Search

Search(x)
$v \leftarrow$ leftmost sentinel at the highest level
while true
  while key(right(v)) $< x$
    $v \leftarrow$ right(v)
  if at the bottommost level **then break**
  $v \leftarrow$ down(v)
$q \leftarrow$ right(v)
if key(q) $\neq x$ **then** $q \leftarrow$ NIL
return q

Search Example



For $x = 5$, the bottom predecessor is 4. The final check finds 5. The optional empty top level adds one downward step.

Insertion

For insertion, augment Search to fill pred[ℓ] with the rightmost node visited at level ℓ, whose key is less than x.

Insert(x)
$q \leftarrow$ **Search**(x)
if $q \neq$ NIL **then return**
$h \leftarrow 0$
while a fair coin is heads: $h \leftarrow h + 1$
Create any missing levels through h with linked sentinels
for $\ell \leftarrow 0, \ldots, h$
  Splice a copy of x after pred[ℓ]
  Link it to the copy below, if $\ell > 0$
$H \leftarrow \max(H, h)$

On any new levels, insert immediately after the leftmost sentinel.

Deletion

Delete(x)
$v \leftarrow$ **Search**(x)
while $v \neq$ NIL
    $u \leftarrow$ up(v)
    right(left(v)) $\leftarrow$ right(v)
    left(right(v)) $\leftarrow$ left(v)
    Delete this copy and set $v \leftarrow u$
while $H > 0$ and level H has no keys
    Remove its sentinels and set $H \leftarrow H - 1$

The search returns the bottom copy. Repair surviving sentinel links when removing a level.

Analysis Plan

- Each level retains half the keys of the previous level in expectation.
- First bound the maximum height using a union bound.
- Next compute expected space using the tail-sum identity.
- Finally bound expected horizontal and vertical search steps separately. A high-probability height bound alone does not bound expected running time.

Height of One Key

Let H_u be the number of times the coin comes up heads when u is inserted. For every integer $h \geq 0$,

$$\Pr(H_u \geq h) = 2^{-h}.$$

Reaching level h requires the first h coin flips to be heads.

For $n \geq 2$, let $h = \lceil 2 \log n \rceil$. Then

$$\Pr(H_u \geq h) = 2^{-h} \leq n^{-2}.$$

Key u occupies $H_u + 1$ levels, starting at level 0.

Union Bound

For any events $A_1, \ldots, A_n$,

$$\Pr\left(\bigcup_{u=1}^{n} A_u\right) \leq \sum_{u=1}^{n} \Pr(A_u).$$

Reminder: The indicator random variable $I(A)$ is 1 when event A occurs and 0 otherwise. Thus $\mathbb{E}[I(A)] = \Pr(A)$.

Proof: For every outcome,

$$I\left(\bigcup_{u=1}^{n} A_u\right) \leq \sum_{u=1}^{n} I(A_u).$$

The left side says whether any event occurs. The right side counts how many occur. Take expectations to obtain the bound.

Maximum Height

Let $H = \max_u H_u$. For every integer $h \geq 0$, by a union bound:

$$\Pr(H \geq h) \leq \sum_{u=1}^{n} \Pr(H_u \geq h) = n2^{-h}.$$

For any constant $c > 1$ and $n \geq 2$,

$$\Pr(H \geq \lceil c \log n \rceil) \leq n^{1-c}.$$

In particular, $c = 2$ gives failure probability at most $1/n$.

Height with High Probability

- An event holds *with high probability* if its probability is $1 - o(1)$ as $n \to \infty$.
- The maximum height is $O(\log n)$ with high probability.
- This is a statement about the height. We will prove the expected search-time bound by summing tail probabilities.
- All expectations are over independent fair coins, with the stored keys and query fixed in advance.

Tail-Sum Identity

If X is a nonnegative integer-valued random variable, then

$$\mathbb{E}[X] = \sum_{r=1}^{\infty} \Pr(X \geq r).$$

Proof: $X = \sum_{r \geq 1} I(X \geq r)$. Taking expectations of both sides gives the identity.

Expected Space Used

In-class exercise: Let $C_u = H_u + 1$ be the number of copies of key u.

- What are $\Pr(C_u \geq 1)$, $\Pr(C_u \geq 2)$, and $\Pr(C_u \geq 3)$?
- What is $\Pr(C_u \geq r)$ for an integer $r \geq 1$?
- Use the tail-sum identity to compute $\mathbb{E}[C_u]$.
- What is the expected total number of nodes in the skip list, excluding sentinels?

Search Viewed Backwards

Fix the stored keys and a query x, independently of the coins.

- Search ends at the bottom predecessor of x, then checks its right neighbor for equality.
- Trace the predecessor path backwards: go up when an upper copy exists, otherwise go left. At a leftmost sentinel, go up until reaching the starting level.
- At each level, the forward path scans the keys below x that lie after the last promoted key below x.

Horizontal Steps at One Level

Let R_i be the number of rightward steps at level i. Condition on the keys in L_i and scan those below x from right to left.

- Stop at the first promoted key or the leftmost sentinel. The skipped suffix has length R_i.
- Promotions from L_i to L_{i+1} are independent fair coins, even after conditioning on L_i.

Thus, for $r \geq 1$,

$$\Pr(R_i \geq r \mid L_i) \leq 2^{-r}, \qquad \mathbb{E}[R_i] \leq \sum_{r \geq 1} 2^{-r} = 1.$$

Expected Horizontal Steps

The previous slide gives $\mathbb{E}[R_i] \leq 1$. Also, R_i cannot exceed the number of keys in L_i, so

$$\mathbb{E}[R_i] \leq n2^{-i}.$$

Set $K = \lceil \log n \rceil$ for $n \geq 1$. Use the bound 1 for levels $0, \ldots, K$ and $n2^{-i}$ for higher levels:

$$\begin{aligned}\mathbb{E}\left[\sum_{i\geq 0} R_i\right] &\leq (K+1) + n \sum_{i=K+1}^{\infty} 2^{-i} \\ &= (K+1) + n2^{-K} \leq K+2.\end{aligned}$$

Expected Search Time

Levels $0, \ldots, H$ give $H+1$ levels and H downward steps. By slide 15, $\Pr(H \geq i) \leq n2^{-i}$. Applying the tail-sum identity (slide 17),

$$\begin{aligned}\mathbb{E}[H] &= \sum_{i=1}^{\infty} \Pr(H \geq i) \\ &\leq K + n \sum_{i=K+1}^{\infty} 2^{-i} = K + n2^{-K} \leq K + 1.\end{aligned}$$

Hence the expected total number of steps is at most

$$(K+2) + (K+1) = 2K + 3 = O(\log n).$$

Insertion and deletion add expected $O(\log n)$ work for copies and level maintenance.

Skip Lists: Summary

For $n \geq 1$ stored keys:

- **Correctness:** Search, insertion, and deletion implement an exact ordered dictionary.
- **Expected time:** $O(\log n)$ per search, insertion, or deletion.
- **Expected space:** $O(n)$ nodes.
- **Height:** $O(\log n)$ with high probability.

The bounds use independent fair coins, with stored keys and queries fixed independently of the coins.

Bloom Filters

- A randomized data structure for representing a set using little space.
- **Insert**(x): add x to the represented set.
- **IsMember**(x): report whether x may belong to the set.
- Membership queries can produce false positives.
- Applications include databases, networking, and packet routing.

Bloom Filters

- Have m slots and k hash functions; assume hash functions are all independent
- Each slot stores 1 bit, initially all bits are 0
- Insert(x) : Set the bit in slots $h_1(x), h_2(x), ..., h_k(x)$ to 1
- IsMember(x) : Return yes iff the bits in $h_1(x), h_2(x), ..., h_k(x)$ are all 1

Bloom Filter: Insertions

Use $m = 8$ bits and $k = 2$ hash functions. Insert a, then b.

Key	h_1	h_2
a	1	4
b	4	6

Bit position	1	2	3	4	5	6	7	8
Initially	0	0	0	0	0	0	0	0
After a	1	0	0	1	0	0	0	0
After b	1	0	0	1	0	1	0	0

Both insertions set bit 4. Setting a bit again leaves it at 1.

Bloom Filter: Queries

After inserting a and b, bits $1, 4, 6$ are set.

Query	Hash positions	Bits read	Result
a	$1, 4$	$1, 1$	True positive
c	$1, 6$	$1, 1$	False positive
d	$2, 4$	$0, 1$	Definite negative

Neither c nor d was inserted. The filter accepts c because other keys set both of its positions.

Analysis Sketch

- m slots and k hash functions; assume hash functions are all independent. Assume n elements inserted.
- Then $P(\text{fixed slot is still 0}) = (1 - 1/m)^{kn}$
- Useful fact from Taylor expansion of e^{-x}: $e^{-x} - x^2/2 \leq 1 - x \leq e^{-x}$ for $x < 1$
- Then if $x \leq 1$

$$e^{-x}(1 - x^2) \leq 1 - x \leq e^{-x}$$

Analysis

- Thus we have the following to good approximation.

$$\begin{aligned} Pr(\text{fixed slot is still 0}) &= (1 - 1/m)^{kn} \\ &\approx e^{-kn/m} \end{aligned}$$

- Let $p = e^{-kn/m}$ and let ρ be the fraction of 0 bits after n elements inserted then

$$Pr(\text{false positive}) = (1 - \rho)^k \approx (1 - p)^k$$

- Where this last approximation holds because ρ is very close to p (by a Martingale argument beyond the scope of this class)

Choosing the Number of Hashes

Use calculus to minimize the approximate false-positive probability

$$(1-p)^k = \left(1 - e^{-kn/m}\right)^k$$

as a function of k.

Treating k as real, the minimum occurs at

$$k = \frac{m}{n} \ln 2.$$

Union and Resizing

- **Union:** OR the two bit vectors. Both filters must use the same size, hash functions, and hash seeds.
- **Halving:** For an even size m, index bits by $0, \ldots, m-1$. Replace each pair $B[j], B[j+m/2]$ by their OR.
- Queries in the folded filter must use indices $h_a(x)$ mod $(m/2)$.
- Folding saves space but can increase the false-positive rate.

From Membership to Counting

- Bloom filters answer membership queries, but do not count occurrences: inserting the same item again leaves the bits unchanged.
- To estimate how often each item occurs, we will use the **Count-Min Sketch**.

Data Streams

- A router forwards a stream of packets.
- An administrator may want to identify fixed-length substrings that occur unusually often, for example in spam or worm traffic.
- The stream can contain too many distinct substrings to keep an exact counter for each one.
- We want compact frequency estimates and a way to identify frequent items.

Streaming Model

- The algorithm processes a stream sequentially in one pass.
- Available memory is much smaller than the stream.
- Each arrival contributes one occurrence of its item. There are no deletions or weighted updates.
- We allow approximate frequency estimates with a quantified error probability.

Frequency Queries

The stream is $j_1, j_2, \ldots, j_T$. Define

$$f_i(T) = \text{Count}(i, T) = \sum_{t=1}^{T} I(j_t = i).$$

- A query supplies an item i and asks for an estimate of $f_i(T)$.
- The sum of all true frequencies is T.
- For the probability bounds, fix the stream, queried item, and query time independently of the sketch's random hashes.

Count-Min Sketch

- Use d rows of w counters each, for $m = dw$ counters in total.
- Choose row hashes $h_1, \ldots, h_d$ independently, each mapping keys to $\{1, \ldots, w\}$.
- Within a row, assume the collision bound

$$\Pr(h_a(i) = h_a(j)) \leq 1/w \qquad \text{for } i \neq j.$$

- A universal hash family supplies this bound. Rows use independently chosen functions from the family.

Counter Array

- Let $C_{a,b}$ be the counter in row a and column b.
- Row a uses hash h_a. Item i maps to counter $C_{a,h_a(i)}$.
- Different items can share a counter, causing overcounting.
- Independent rows provide different collision patterns.

Unit Updates

All counters start at zero.

Update(i)
for $a \leftarrow 1, \ldots, d$
$\quad C_{a,h_a(i)} \leftarrow C_{a,h_a(i)} + 1$

Each arriving item adds exactly 1 in each row. An update takes $O(d)$ time, assuming constant-time hash evaluation and counter operations.

Count Estimates

After T unit updates, return

$$\widehat{f}_i(T) = \min_{1 \le a \le d} C_{a,h_a(i)}(T).$$

- Every selected counter includes all $f_i(T)$ occurrences of i.
- Taking the minimum reduces collision overcounting.
- A query takes $O(d)$ time.

Count-Min Example

Use $d = 3$ rows and $w = 4$ columns. Process a, b, a, c, b, a.

- Hash positions (h_1, h_2, h_3): $a \mapsto (1, 2, 3)$, $b \mapsto (1, 3, 1)$, and $c \mapsto (2, 2, 1)$.

The final counter array is

	Column 1	Column 2	Column 3	Column 4
Row 1	5	1	0	0
Row 2	0	4	2	0
Row 3	3	0	3	0

- Query a: $\widehat{f}_a = \min(5, 4, 3) = 3 = f_a$. The third row removes the overcount in the first two rows.
- Suppose an unseen key z hashes to $(1, 2, 1)$. Then $\widehat{f}_z = \min(5, 4, 3) = 3$, although $f_z = 0$.

Count-Min Guarantee

Theorem. Fix a unit-update stream, item i, and time T, independently of the random hashes. Given $0 < \varepsilon, \delta < 1$, choose

$$w = \left\lceil \frac{e}{\varepsilon} \right\rceil, \qquad d = \left\lceil \ln \frac{1}{\delta} \right\rceil.$$

Then $\widehat{f}_i(T) \geq f_i(T)$ for every choice of hashes, and with probability at least $1 - \delta$,

$$\widehat{f}_i(T) \leq f_i(T) + \varepsilon T.$$

Space is dw counters. Update and query time are $O(d)$.

Proof: Overcount in One Row

For a fixed query (i, T), let Z_a be the number of arrivals of other items that collide with i in row a.

$$C_{a,h_a(i)}(T) = f_i(T) + Z_a, \qquad Z_a \geq 0.$$

Consequently,

$$\widehat{f}_i(T) = f_i(T) + \min_a Z_a \geq f_i(T).$$

It remains to bound the probability that all rows have large overcount.

Collision Indicators

Let j be the item arriving at time t. For row a, define

$$X_{a,t} = I(j \neq i \text{ and } h_a(j) = h_a(i)) .$$

- If $j = i$, then $X_{a,t} = 0$.
- If $j \neq i$, then $\mathbb{E}[X_{a,t}] \leq 1/w$ by the collision bound.
- Thus $Z_a = \sum_{t=1}^{T} X_{a,t}$.

Repeated arrivals may make indicators dependent within a row. Linearity of expectation does not require independence.

Expected Overcount

By linearity of expectation,

$$\begin{aligned}\mathbb{E}[Z_a] &= \sum_{t=1}^{T} \mathbb{E}[X_{a,t}] \\ &\leq \frac{T - f_i(T)}{w} \leq \frac{T}{w}.\end{aligned}$$

Since $w \geq e/\varepsilon$, we obtain

$$\mathbb{E}[Z_a] \leq \frac{\varepsilon T}{e}.$$

Next use Markov's inequality to turn this expectation into a probability bound.

Markov's Inequality

For a nonnegative random variable X and $\lambda > 0$,

$$\Pr(X \geq \lambda) \leq \frac{\mathbb{E}[X]}{\lambda}.$$

Proof: $X \geq \lambda \cdot I(X \geq \lambda)$. Take expectations.

For $T > 0$, apply this with $X = Z_a$ and $\lambda = \varepsilon T$:

$$\Pr(Z_a > \varepsilon T) \leq \frac{\mathbb{E}[Z_a]}{\varepsilon T} \leq \frac{1}{e}.$$

At $T = 0$, every frequency and estimate is zero.

Independent Rows Reduce Error

The row overcounts $Z_1, \ldots, Z_d$ are independent because each is a function of its independently chosen row hash and the fixed stream.

$$\begin{aligned}\Pr(\widehat{f}_i(T) > f_i(T) + \varepsilon T) &= \Pr\left(\bigcap_{a=1}^{d} \{Z_a > \varepsilon T\}\right) \\ &= \prod_{a=1}^{d} \Pr(Z_a > \varepsilon T) \\ &\leq e^{-d} \leq \delta.\end{aligned}$$

The minimum fails only if every row fails.

Heavy Hitters

- **Heavy hitters** are items that account for a large fraction of the stream.
- With an additional method for identifying candidate keys, Count-Min can help find heavy hitters using little memory.

Recap

- Skip lists: exact ordered-dictionary operations with expected logarithmic time and expected linear space.
- Bloom filters: compact membership queries that may return false positives.
- Count-Min: compact frequency estimates with one-sided additive error, using unit updates here.
- Heavy hitters: extensions can identify frequent items using little memory.

References

- W. Pugh. *Skip Lists: A Probabilistic Alternative to Balanced Trees*. Communications of the ACM, 33(6), 1990.
- G. Cormode and S. Muthukrishnan. *An Improved Data Stream Summary: The Count-Min Sketch and Its Applications*. Journal of Algorithms, 55(1), 2005.