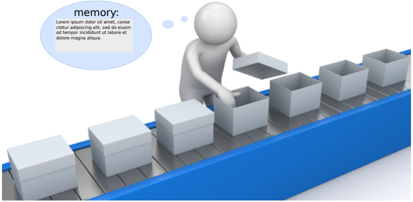
Streaming 2: (Distinct) element count

Philip Bille Inge Li Gørtz Eva Rotenberg

The streaming model

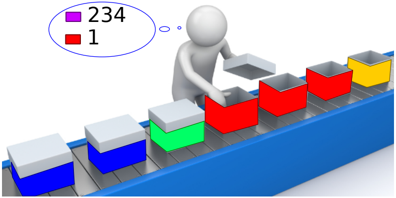


Stream, σ: $a_1, a_2, a_3, \ldots$ of elements $a_i \in U$ from some universe.

Maintain a small working memory. When seeing element a_i, update the memory depending only on a_i.

Goal: by the end of the stream, have completed some task.

Last week: Frequent elements



if $j \in keys(A)$ **then**
  A[j]++;
else if $|keys(A)| < k - 1$ **then**
  A[j] → 1;
else
  decrement all A[j].

Task: Detect very common colours.

Misra-Gries: Keep track of the $k - 1$ most common colours.

Communicating numbers

Here, Alice is thinking of a number between 0 and $m-1$.

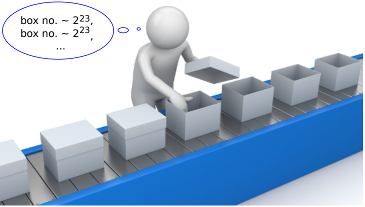
Alice wants to tell Bob this number using few bits.

Exact: $\lceil \lg m \rceil$ bits.

$\lceil \lg m \rceil - 1$ bits? (Exercise)

$\lceil \lg \lg m \rceil$ bits? (Exercise)

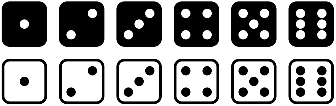
Counting



Imagine you want to count the elements.

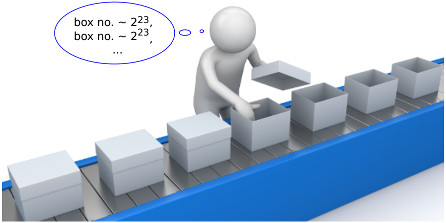
Space of exact count: $\log n$ bits memory needed.

Approx count: $\log \log n$ bits. Challenge: when to update?



!

Probabilistic counting



$X \leftarrow 0$;
for a_i in stream **do**
  w. prob. 2^{-X}:'
    X++;
end
return $2^X - 1$

Keep an approximate count: store c such that $2^c \simeq n$

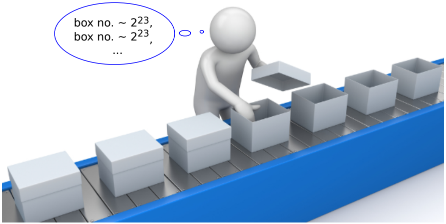
Update randomly with decreasing probability. Maintain 2^c is n in expectation.

Question: With which probability?

When c turns c_0, $n \simeq 2^{c_0}$, so it should stay there for circa 2^{c_0} turns. $\Rightarrow$ probability circa $1/2^{c_0}$.

(Question: smart way of rolling a 2^m-sided dice?)

Probabilistic counting



$X \leftarrow 0$;
for a_i in stream **do**
  w. prob. 2^{-X}:
    X++;
end
return $2^X - 1$

Space: $\log X$ bits $\leftarrow$ expected $\lg \lg m$ bits

Correctness:

- X_i value of X after processing a_i
- Set $Y_i = 2^{X_i}$
- Exercise: prove $\mathbb{E}[Y_m] = m + 1$ Hint: induction.

Probabilistic counting

Induction start:

$X_0 = 0,\ Y_0 = 2^{X_0} = 1$

Induction step:

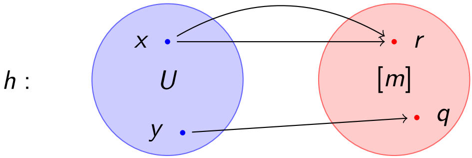
Assume $E[Y_{m-1}] = m$

$$E[Y_m] = E[2^{X_m}] = \sum_{j=0}^{\infty} 2^j P[X_m = j]$$
$$= \sum_j 2^j \left(P[X_{m-1} = j] \cdot (1 - \frac{1}{2^j}) + P[X_{m-1} = j-1] \cdot \frac{1}{2^{j-1}} \right)$$
$$= \sum_j 2^j P[X_{m-1} = j] + \sum_j (-P[X_{m-1} = j] + 2P[X_{m-1} = j-1])$$
$$= E[Y_{m-1}] + \sum_j P[X_{m-1} = j]$$
$$= m + 1$$

```
X ← 0;
for a_i in stream do
|   w. prob. 2^{-X}:
|   |   X++;
end
return 2^X − 1
```

Distinct element count

Reminder: hashing



~~Kiddie definition: A hash function is a function from U to $[m]$.~~

A hash function is a random variable in the set of functions $U \to [m]$.

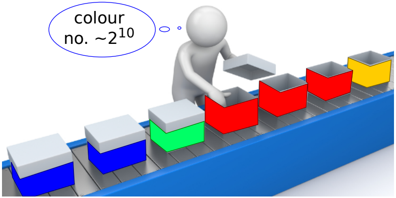
Question: If $|U| = u$ and $|[m]| = m$, how many functions $U \to [m]$?

In practise, h is chosen uniform at random from a subset of $f : U \to [m]$.

2-independent hashing: For all $x \neq y \in U$, $q, r \in [m]$,

$P[h(x) = r \wedge h(y) = q] = \frac{1}{m^2}$.

Distinct element count



$z \leftarrow 0,$

for a_i in stream **do**

  $z = \max\{z, 0s(h(a_i))\}$

end

return $2^{z+0.5}$

Imagine you want to count element types (e.g. colours, see figure).

Challenge: A random dice roll that depends on the input.

Solution: Hashing.

Take a 2-independent hash function h.

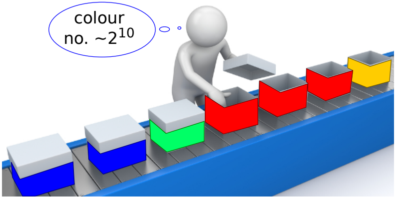
Use $z =$ the number of trailing 0s in the hash values $h(x)$ seen so far.

Note: h is uniform, so $\frac{1}{2}$ end with 0, $\frac{1}{4}$ end with 00, $\frac{1}{8}$ with 000 etc.

Estimate: count $\simeq 2^{z+\frac{1}{2}}$. (we denote this $\hat{d}$, estimator of d)

Exercise: Bound $P[\hat{d} \geq 3d]$ and $P[\hat{d} \leq d/3]$.

Distinct element count



$z \leftarrow 0$,

for a_i in stream **do**

$z = \max\{z, 0s(h(a_i))\}$

end

return $2^{z+0.5}$

Use $z =$ the max n.o. trailing 0s in the hash values $h(x)$ seen so far.

Estimate: count $\simeq \hat{d} = 2^{z+\frac{1}{2}}$.

Exercise: Bound $P[\hat{d} \geq 3d]$ and $P[\hat{d} \leq d/3]$.

a: smallest integer s.t. $2^{a+\frac{1}{2}} \geq 3d$

b: largest integer s.t. $2^{b+\frac{1}{2}} \leq d/3$

Y_r: nunber of distinct elements a_i with $0s(h(a_i)) \geq r$

Hint: $P[\hat{d} \geq 3d] = P[z \geq a] = P[Y_a > 0] =?$

$P[\hat{d} \leq d/3] = P[z \leq b] = P[Y_{b+1} = 0] =?$

The Median Trick

Lemma: $\hat{d}$ deviates from d by a factor 3 with prob. $\leq 2\frac{\sqrt{2}}{3}$.

Not very impressive. Still interesting!

What if we run k independent copies of the algorithm and return the median, m?

$m > 3d$ means $k/2$ of the copies exceed $3d$.

Expected: only $k\frac{\sqrt{2}}{3}$ exceed $3d$.

Since they are independent, we can use Chernoff. $\Rightarrow$ prob. $2^{-\Omega(k)}$.

Distinct element count: Analysis.

How well does $\hat{d} = 2^{z+\frac{1}{2}}$ estimate d?

$X_{r,j}$: indicator variable for $\geq r$ zeros in the hash value $h(j)$.

$\mathbb{E}[X_{r,j}] = P[r \text{ coinflips turn head}] = \left(\frac{1}{2}\right)^r$.

$Y_r = \sum_{j \in \text{stream}} X_{r,j}$: number of seen elements with $\geq r$ 0s.

$\mathbb{E}[Y_r] = d \cdot \mathbb{E}[X_{r,*}] = \frac{d}{2^r}$

$Var[Y_r] = \sum_j Var[X_{r,j}] \leq \sum_j \mathbb{E}[X_{r,j}^2] = \sum_j \mathbb{E}[X_{r,j}] = \frac{d}{2^r}$ $(j \in \text{stream})$

$P[Y_r > 0] = P[Y_r \geq 1] \overset{\text{Markov}}{\leq} \frac{\mathbb{E}[Y_r]}{1} = \frac{d}{2^r}$

$P[Y_r = 0] \leq P[|Y_r - \mathbb{E}[Y_r]| \geq \frac{d}{2^r}] \overset{\text{Chebysh.}}{\leq} \frac{\mathbb{E}[Y_r]}{(d/2^r)^2} \leq \frac{1}{(d/2^r)}$

Now, the probability of $\hat{d}$ being within a factor 3 of d.

$P[\hat{d} \geq 3d] = P[z \geq a]$ for some a with $2^{a+1/2} \geq 3d$.

$= P[Y_a > 0] \leq \frac{d}{2^a} = \frac{3 \cdot d \cdot \sqrt{2}}{3 \cdot 2^a \cdot \sqrt{2}} = \frac{\sqrt{2}}{3} \cdot \frac{3d}{2^{a+\frac{1}{2}}} \leq \frac{\sqrt{2}}{3}$.

Similarly, $P[\hat{d} \leq d/3] \leq \frac{\sqrt{2}}{3}$.

A Lower Bound

Assume we have an algorithm taking up s bits space and deterministically, exactly able to report the number of distinct elements.
Then, given any binary sequence x of length n, we can do the following:
Let the algorithm stream through a sequence consisting of $i : x_i = 1$.
Example: $x = 1001101$ Stream: $1, 4, 5, 7$.
Then, the state of the algorithm must be some configuration reflecting this information.
Now, regardless of what x was, we can recover x by streaming the following sequence: $1, 2, 3, 4, \ldots$, each time noticing whether the number of distinct elements goes up.
Thus, the state of the algorithm must have been able to distinguish between all different strings of length n.$\Rightarrow s = n$.
Exercise: spend 2 minutes convincing yourself/your neighbour about this.