1 Streaming

1.1 Computing Frequency Moments

- \( n \) = length of stream
- \( N \) = size of alphabet

1.1.1 Definition of Frequency Moments

\( k \)-th moment is \( \sum_{X_i \in X} (m_i)^k \), where \( m_i \) is the frequency of \( X_i \) in \( S \).

1. Example

- Stream \( S_1 \): a, b, a, c, a, d, a, c
- Stream \( S_2 \): b, c, d, a, a, a, a, a
- \( k = 0 \): \( 4^0 + 1^0 + 2^0 + 1^0 = 4 \) (\( S_1 \)). \( k = 0 \) would mean distinct objects
- \( k = 1 \): \( 4^1 + 1^1 + 2^1 + 1^1 = 8 \) (\( S_1 \)). \( k = 1 \) would mean total length of stream
- \( k = 2 \): \( 4^2 + 1^2 + 2^2 + 1^2 = 16 \), (\( S_1 \)).
- \( k = 2 \): \( 5^2 + 1^2 + 1^2 + 1^2 = 28 \), (\( S_2 \)).

1.2 Finding Second Moments


- \( \text{Ans} = n(2X(i).\text{count} - 1) \), where \( X(i).\text{count} \) is the number of occurrences of element \( X(i) \) in positions \( i, i+1, \cdots, n \).

1. Example: a, b, c, b, d, a, c, a, a, b, d, c, a, a, b

- \( n = 15 \)
- True answer
  - \( m_a = 5, m_b = 4, m_c = 3, m_d = 3 \)
  - \( 25 + 16 + 9 + 9 = 59 \)
- Take a look at the underlined letters
  - c.count = 3, d.count = 2, a.count = 2
  - c: 15 * (2 * 3 - 1) = 75
  - c: 75, d: 45, a: 45. So average is 55.
2. Algorithm: Imagine we try all choices for \( i = 1, \ldots, n \), take average.
\[
\sum m_i^2 = \frac{1}{n} \sum_{i=1}^{n} n(2X(i).\text{count} - 1)
\]

3. Correctness: Denote the right hand side of the previous equation with \( \text{Ans} \), and we want to prove \( E[\text{Ans}] = \sum m_i^2 \):

\[
E[\sum_{\text{distinct } v} m_v^2] = E[\sum_{\text{distinct } v} \sum_{j=1}^{m_v} (2j - 1)]
\]
\[
= E[\sum_{\text{distinct } v} (2m_v - 2j + 1)]
\]
\[
= E[\sum_{i} 2X(i).\text{count} - 1]
\]
\[
= nE[2X(i).\text{count} - 1]
\]

Note for the \( j \)-th occurrence of a certain value \( v \) at location \( i \), \( m_v - j + 1 \) is the number of occurrences of this element in position \( i, i+1, \ldots, n \), in other words, this equals to \( X(i).\text{count} \). So \( 2m_v - 2j + 1 = 2X(i).\text{count} - 1 \).

1.3 Finding majority elements

1.3.1 Find the element that appears more than half (when there is no such element, there is no guarantee on the output)

Suppose the stream \( S = X(1), \ldots, X(n) \). We use \( v \) to denote our candidate, and initialize it with some special symbol NULL, and \( \text{count} = 0 \) to denote its count

1. Algorithm

   - For \( i := 1 \) to \( n \) do
     - If \( v = \text{NULL} \), then set \( v := X(i) \), and \( \text{count} := 1 \)
     - Else If \( X(i) = v \), \( \text{count} := \text{count} + 1 \)
     - Else \( \text{count} = \text{count} - 1 \)
     - If \( \text{count} = 0 \), \( v = \text{NULL} \)

   - Return \( v \)

2. Note: This algorithm does not guarantee correctness if the most frequent value does not appear strictly more than half of the stream.

3. Analysis: [Wikipedia Ref]

   **Claim**: If there is a strict majority element, this algorithm correctly finds it.

   **Proof**: If there is a majority element, the algorithm will always find it. Supposing that the majority element is \( m \), let \( c \) be a number defined at any step of the algorithm to be either the counter, if the stored element is \( m \), or the negation of the counter otherwise. Then at each step in which the algorithm encounters \( m \), the value of \( c \) will increase by one, and at each step at which it encounters a different value, the value of \( c \) may either increase or decrease by one. If \( m \) truly is the majority, there will be more increases than decreases, and \( c \) will be positive
at the end of the algorithm. But this can be true only when the final stored element is $m$, the majority element.

4. Generalization to $T$ elements with frequency at least $\frac{1}{T+1}$
   - For $i = 1$ to $n$ do
     - If $X(i) \in K$, count[$X(i)$] += 1
     - Else add $X(i)$ to $K$, count[$X(i)$] = 1
     - If $|K| > T$, subtract 1 from all counters, throw out zero elements.

5. Analysis: A proof can be found in handouts link.

1.4 Estimating #of distinct elements in a stream
Flajolet-Martin (1984)
For this algorithm, we need a hash function $h(x)$ which hash $x$ into a $k$ bit binary number, where $2^k$ is at least larger than $n$.
   - $r = \max_{j,i}(\#\text{zero's at the right end of } h_j(x_i))$,
   - Ans: $2^r$.

1.5 Analysis
Suppose $m$ is the number of distinct elements. Then
   - Chance that none reaches special node: $(1 - 2^{-r})^m = (1 - \frac{1}{2^r})^{m-2^r} \simeq e^{-\frac{m}{2^r}}$
   - If $m > 2^r$, say $m = 2^r+1$, then $e^{-\frac{m}{2^r}} = e^{-2}$, which is a low chance.
   - If $m \ll 2^r$, say $m = 2^r-2$, then $e^{-\frac{2^r-2}{2^r}} = e^{-\frac{1}{4}}$, which is close to 1.