Lecture 8: Operations on Data Streams
CSCI 700 - Algorithms I

Andrew Rosenberg

October 14, 2010
Last Time

- Balanced Search Trees
Data Streams

- Maximum (and Minimum)
- Mean
- Median
- Sampling – select $k$ elements at random
A data stream is a sequence of data whose size is not known ahead of time, or is infinite.

- Stock information
- Sports stats
- Health information
**Mean**

- Find the Mean of an Array of numbers (floats, integers).
- Mean = $\mu = \frac{1}{n} \sum_{i=1}^{n} A[i]$
Find the Mean of an Array of numbers (floats, integers).

Mean = \( \mu = \frac{1}{n} \sum_{i=1}^{n} A[i] \)

Mean(A)

\[
\begin{align*}
sum & \leftarrow 0 \\
\text{for } i & \leftarrow 1 \ldots \text{size}(A) \text{ do} \\
& \quad sum \leftarrow sum + A[i] \\
\text{end for} \\
\text{return } \frac{sum}{\text{size}(A)}
\end{align*}
\]
Given a stream of $n$ numbers find the Mean.
Data Stream Mean

- Given a stream of \( n \) numbers find the Mean.

UpdateMean(A,x)

```plaintext
static sum ← 0
ENQUEUE(A,x)
return Mean(A)
```
Data Stream Mean

- Given a stream of $n$ numbers find the Mean.

**UpdateMean(A,x)**

```plaintext
static sum ← 0
ENQUEUE(A,x)
return Mean(A)
```

- $O(n^2)$. 
Data Stream Mean

- Given a stream of $n$ numbers find the Mean.

**UpdateMean(A,x)**

```plaintext
static sum ← 0
ENQUEUE(A,x)
return Mean(A)
```

- $O(n^2)$.
- But do we need to calculate the Mean every time?
Given a stream of \( n \) numbers find the Mean.
Given a stream of $n$ numbers find the Mean.

RunningMean($x$)

```plaintext
class RunningMean:
    def __init__(self):
        self.sum = 0
        self.count = 0

    def update(self, x):
        self.sum += x
        self.count += 1
        return self.sum / self.count
```

```plaintext
static sum ← 0
count sum ← 0
sum ← sum + x
count ← count + 1
return sum/count
```
Given a stream of $n$ numbers find the Mean.

**RunningMean(x)**

```
static sum ← 0
count sum ← 0
sum ← sum + x
count ← count + 1
return sum/count
```

$O(n)$. But this calculates the mean of the whole stream.
Given a stream of $n$ numbers find the Mean of the last $k$. 
Given a stream of $n$ numbers find the Mean of the last $k$.

**UpdateMean**(A,x)

```
static sum ← 0
ENQUEUE(A,x)
if size(A) > k then
    tail ← DEQUEUE(A,x)
end if
return Mean(A)
```
Given a stream of $n$ numbers find the Mean of the last $k$.

**UpdateMean(A,x)**

```plaintext
static sum ← 0
ENQUEUE(A,x)
if size(A) > k then
    tail ← DEQUEUE(A,x)
end if
return Mean(A)
```

$O(n \cdot k)$. 
Given a stream of $n$ numbers find the Mean of the last $k$.

**UpdateMean**(A,x)

```plaintext
static sum ← 0
Enqueue(A,x)
if size(A) > k then
tail ← Dequeue(A,x)
end if
return Mean(A)
```

- $O(n \cdot k)$.
- But do we need to recalculate the Mean every time?
Given a stream of $n$ numbers find the Mean of the last $k$.

**UpdateMean(A,x)**

```plaintext
static sum ← 0
ENQUEUE(A,x)
sum ← sum + x
if size(A) > k then
    tail ← DEQUEUE(A,x)
    sum ← sum − tail
end if
return sum/size(A)
```

$O(1)$. 
Given a stream of \( n \) numbers find the Standard Deviation of the last \( k \). Where the mean is \( \mu = \frac{1}{n} \sum_{i=1}^{n} A[i] \).

\[
\sigma = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (A[i] - \mu)^2}
\]  

(1)
Mean and Standard Deviation

99.7% between ±3 s.d.
95.4% between ±2 s.d.
68.3% between ±1 s.d.

Only 3 points in 1000 will fall outside the area 3 standard deviations either side of the center line.

s.d. = standard deviation

From http://syque.com/quality_tools/toolbook/Variation/Image375.gif
Given a stream of $n$ numbers find the Standard Deviation of the last $k$. Where the mean is $\mu$.

\[
\sigma = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (A[i] - \mu)^2} \quad (2)
\]

\[
= \sqrt{\left(\frac{1}{n} \sum_{i=1}^{n} A[i]^2\right) - \mu^2} \quad (3)
\]

\[
= \sqrt{\left(\frac{1}{n} \sum_{i=1}^{n} A[i]^2\right) - \left(\frac{1}{n} \sum_{i=1}^{n} A[i]\right)^2} \quad (4)
\]

When we calculated the mean we maintained $\text{sum}(A)$ as elements were inserted and removed from the buffer. Here we also need to maintain $\text{sum}(A^2)$.
Given a stream of \( n \) numbers find the Standard Deviation of the last \( k \).

**UpdateStandardDeviation(A,x)**

```plaintext
static sum ← 0
static sumOfSquares ← 0
ENQUEUE(A,x)
sum ← sum + x
sumOfSquares ← sumOfSquares + x^2
if size(A) > k then
tail ← DEQUEUE(A,x)
sum ← sum − tail
sumOfSquares ← sumOfSquares − tail^2
end if
return \( \sqrt{\frac{\text{sumOfSquares}}{\text{size}(A)} - \left(\frac{\text{sum}}{\text{size}(A)}\right)^2} \)
```

\( O(1) \).
Find the maximum element observed in the stream so far.
Find the maximum element observed in the stream so far.
Call UpdateMaximum(x) for each observed token, x.

**UpdateMaximum(x)**

```
static max ← −∞
if x > max then
    max ← x
end if
return max
```
Maximum

- Given a stream of $n$ numbers find the Maximum of the last $k$. 
Given a stream of \( n \) numbers find the Maximum of the last \( k \).

**UpdateMaximum**\((A,x) : O(n \cdot k)\). We can make it more efficient though:

\[
\text{static } max \leftarrow -\infty \\
\text{ENQUEUE}(A, x) \\
\text{if } \text{size}(A) > k \text{ then} \\
\quad \text{tail} \leftarrow \text{DEQUEUE}(A) \\
\text{end if} \\
\text{max} \leftarrow \text{MAXIMUM}(A) \\
\text{return } \text{max}
\]
Given a stream of $n$ numbers find the Maximum of the last $k$.

UpdateMaximum($A, x$) : $O(n \cdot k)$. We can make it more efficient though

```plaintext
static max ← −∞
ENQUEUE($A, x$)
if size($A$) > $k$ then
tail ← DEQUEUE($A$)
end if
max ← MAXIMUM($A$)
return max
```

Maximum($A$)

```plaintext
max ← −∞
for $i ← 1..size(A)$ do
    if $A[i] > max$ then
        max ← $A[i]$
    end if
end for
return max
```
Given a stream of $n$ numbers find the Maximum of the last $k$. 
Given a stream of \( n \) numbers find the Maximum of the last \( k \).

### UpdateMaximum\((A,x)\)

```plaintext
static max ← −∞
ENQUEUE\((A, x)\)
if size\((A) > k\) then
tail ← DEQUEUE\((A)\)
end if
if \( x > max \) then
  max ← x
end if
if \( \text{tail} \geq max \) then
  max ← MAXIMUM\((A)\)
end if
return max
```
Data Stream Maximum

- Given a stream of $n$ numbers find the Maximum of the last $k$.

- `UpdateMaximum(A,x)`

```plaintext
static max ← −∞
ENQUEUE(A, x)
if size(A) > k then
    tail ← DEQUEUE(A)
end if
if x > max then
    max ← x
end if
if tail ≥ max then
    max ← MAXIMUM(A)
end if
return max
```

- Still $O(n \cdot k)$ though. Is it possible to do better?
What is the smallest running time of \texttt{MAXIMUM} over an array \( A \) of size \( k \)?
What is the smallest running time of $\text{MAXIMUM}$ over an array $A$ of size $k$?

**Adversary Method**

Assume an adversary who tries to make the algorithm do more work, or show a contradiction.

Think of the adversary as creating input as the algorithm is running, such that the input is consistent and the algorithm must do as much work as possible.
What is the smallest running time of \textsc{Maximum} over an array \( A \) of size \( k \)?

\textbf{Adversary Method}

Assume an adversary who tries to make the algorithm do more work, or show a contradiction.

Think of the adversary as creating input as the algorithm is running, such that the input is consistent and the algorithm must do as much work as possible.

For maximum, if fewer than \( k - 1 \) comparisons are made, and the algorithm says \( A[j] \) is the maximum value in the array, the adversary can say \( A[i] = A[j] + 1 \) and maximum is wrong.
What it the smallest running time of \textsc{Maximum} over an array $A$ of size $k$?

\textbf{Adversary Method}

Assume an adversary who tries to make the algorithm do more work, or show a contradiction.

Think of the adversary as creating input as the algorithm is running, such that the input is consistent and the algorithm must do as much work as possible.

For maximum, if fewer than $k - 1$ comparisons are made, and the algorithm says $A[j]$ is the maximum value in the array, the adversary can say $A[i] = A[j] + 1$ and maximum is wrong.

Thought problem: Try to design a divide and conquer algorithm to calculate \textsc{Maximum}. Does it make fewer than \( k - 1 \) comparisons?
When we want an efficient maximum what data structure should we use?
Update Maximum Using a Heap

- When we want an efficient maximum what data structure should we use?
- A Heap.
Update Maximum Using a Heap

UpdateMaximum(A,x)

\begin{verbatim}
static max ← −∞
static Heap
ENQUEUE(A, x)
HEAPINSERT(Heap, x)
if size(A) > k then
tail ← DEQUEUE(A)
    HEAPDELETE(Heap, tail)
end if
max ← HEAPMAXIMUM(Heap)
return max
\end{verbatim}
Update Maximum Using a Heap

**UpdateMaximum(A, x)**

```plaintext
static max ← −∞
static Heap
Enqueue(A, x)
HeapInsert(Heap, x)
if size(A) > k then
    tail ← Dequeue(A)
    HeapDelete(Heap, tail)
end if
max ← HeapMaximum(Heap)
return max
```

- **HeapInsert** is $\Theta(\log n)$.
- **HeapDelete** is $\Theta(\log n)$.
- **HeapMaximum** is $\Theta(1)$.
- Takes $\Theta(n \log k)$ which is faster than previous $\Theta(nk)$. 
Update Maximum Using a Heap

UpdateMaximum(A,x)

```plaintext
static max ← −∞
static Heap
ENQUEUE(A, x)
HEAPINSERT(Heap, x)
if size(A) > k then
    tail ← DEQUEUE(A)
    HEAPDELETE(Heap, tail)
end if
max ← HEAPMAXIMUM(Heap)
return max
```

- **HEAPINSERT** is $\Theta(\log n)$.
- **HEAPDELETE** is $\Theta(\log n)$.
- **HEAPMAXIMUM** is $\Theta(1)$.
- Takes $\Theta(n \log k)$ which is faster than previous $\Theta(nk)$.
- Does this violate the minimum runtime of MAXIMUM proof?
Randomly select an element

- Select an element at random from a stream of size $n$.
- At any point in the processing of the stream, the likelihood that any element is selected should be $\frac{1}{n}$ where $n$ is the number of elements that have been seen.
Randomly select an element

- Select an element at random from a stream of size $n$.
- The probability that any element is selected should be $\frac{1}{n}$.
Randomly select an element

- Select an element at random from a stream of size $n$.
- The probability that any element is selected should be $\frac{1}{n}$.
- Note: $\text{Random}$ returns a random number $x$ such that $0 < x < 1$.

```
UpdateSample(x)
```

```plaintext
static n ← 0
static sample
n ← n + 1
p ← Random(0, 1)
if $p < \frac{1}{n}$ then
    sample ← x
end if
return sample
```
Need to show the likelihood that any element is selected is $\frac{1}{n}$.

**Base Case** When $n = 1$, $p < \frac{1}{n} = \frac{1}{1} = 1$, is always true. Therefore, the element is always selected. That is, the likelihood of selection is 1.
Proof of UpdateSample

- Need to show the likelihood that any element is selected is $\frac{1}{n}$

- **Inductive Step** Assume that each of the first $n$ elements were selected with likelihood $\frac{1}{n}$. Show that when another element is seen the likelihood of any element being selected becomes $\frac{1}{n+1}$.
Proof of UpdateSample

- Need to show the likelihood that any element is selected is \( \frac{1}{n} \).

- **Inductive Step** Assume that each of the first \( n \) elements were selected with likelihood \( \frac{1}{n} \). Show that when another element is seen the likelihood of any element being selected becomes \( \frac{1}{n+1} \).

- For the \( n+1 \)-th element, the likelihood that \( p \) is selected is \( \frac{1}{n+1} \) because \( p \) is randomly selected.
Need to show the likelihood that any element is selected is $\frac{1}{n}$.

**Inductive Step** Assume that each of the first $n$ elements were selected with likelihood $\frac{1}{n}$. Show that when another element is seen the likelihood of any element being selected becomes $\frac{1}{n+1}$.

For the $n+1$-th element, the likelihood that $p$ is selected is $\frac{1}{n+1}$ because $p$ is randomly selected.

Need to show that the likelihood of the other elements being selected is $\frac{1}{n+1}$. 
Proof of UpdateSample

- Need to show the likelihood that any element is selected is $\frac{1}{n}$

- **Inductive Step** Assume that each of the first $n$ elements were selected with likelihood $\frac{1}{n}$. Show that when another element is seen the likelihood of any element being selected becomes $\frac{1}{n+1}$.

- For the $n + 1$-th element, the likelihood that $p$ is selected is $\frac{1}{n+1}$ because $p$ is randomly selected.

- Need to show that the likelihood of the other elements being selected is $\frac{1}{n+1}$.

- The likelihood of the $n + 1$-th element **not** being selected is $1 - \frac{1}{n+1} = \frac{n}{n+1}$. 
Proof of UpdateSample

- Need to show the likelihood that any element is selected is \( \frac{1}{n} \).

- **Inductive Step** Assume that each of the first \( n \) elements were selected with likelihood \( \frac{1}{n} \). Show that when another element is seen the likelihood of any element being selected becomes \( \frac{1}{n+1} \).

- For the \( n+1 \)-th element, the likelihood that \( p \) is selected is \( \frac{1}{n+1} \) because \( p \) is randomly selected.

- Need to show that the likelihood of the other elements being selected is \( \frac{1}{n+1} \).

- The likelihood of the \( n+1 \)-th element not being selected is \( 1 - \frac{1}{n+1} = \frac{n}{n+1} \).

- The likelihood of one of the first \( n \) elements being selected is \( \frac{1}{n} \).
Proof of UpdateSample

- Need to show the likelihood that any element is selected is \( \frac{1}{n} \).

- **Inductive Step** Assume that each of the first \( n \) elements were selected with likelihood \( \frac{1}{n} \). Show that when another element is seen the likelihood of any element being selected becomes \( \frac{1}{n+1} \).

- For the \( n + 1 \)-th element, the likelihood that \( p \) is selected is \( \frac{1}{n+1} \) because \( p \) is randomly selected.

- Need to show that the likelihood of the other elements being selected is \( \frac{1}{n+1} \).

- The likelihood of the \( n + 1 \)-th element **not** being selected is \( 1 - \frac{1}{n+1} = \frac{n}{n+1} \).

- The likelihood of one of the first \( n \) elements being selected is \( \frac{1}{n} \).

- The likelihood of both of these things happening is \( \frac{1}{n} \cdot \frac{n}{n+1} = \frac{1}{n+1} \).
Sampling $k$ elements

- Select $k$ elements at random from a stream of unknown size $n$.
- The probability that any element is selected should be $\min(1, \frac{k}{n})$. 
Sampling $k$ elements

- Select $k$ elements at random from a stream of unknown size $n$.
- The probability that any element is selected should be $\min(1, \frac{k}{n})$.

```
UpdateSample(A, x)

static n ← 0
n ← n + 1
p ← RANDOM(0, 1)
if p < $\frac{k}{n}$ then
    AddToSample(A, x)
end if
return A
```
Sampling $k$ elements

- Select $k$ elements at random from a stream of unknown size $n$.
- The probability that any element is selected should be $\min(1, \frac{k}{n})$.

UpdateSample($A, x$)

```python
static n ← 0
n ← n + 1
p ← Random(0, 1)
if $p < \frac{k}{n}$ then
    AddToSample($A, x$)
end if
return $A$
```

AddToSample($A, x$)

```python
if size($A$) < $k$ then
    $A[size(A)] ← x$
else
    $p ← Random(0, 1)$
    $idx ← \lceil k \cdot p \rceil$
    $A[idx] ← x$
end if
```
Need to show that the likelihood of a sample being selected is $\min(1, \frac{k}{n})$.

**Base Case:** For $n \leq k$, every element is included in the sample. Therefore the likelihood is 1. Since $\frac{k}{n} \geq 1$ for all $n \leq k$, $\min(1, \frac{k}{n}) = 1$. Done.
■ Need to show that the likelihood of a sample being selected is \( \min(1, \frac{k}{n}) \).

■ **Inductive Step**: Assume true for some \( n \leq k \). Need to prove that for \( n + 1 \) the likelihood of selecting any element is \( \frac{k}{n+1} \). When processing the \( n + 1 \)-th element the likelihood that it is selected as part of the sample is \( \frac{k}{n+1} \). By the random selection of \( p \) and the following if statement. Just need to show that the likelihood of any other element being selected is also \( \frac{k}{n+1} \).
Proving the correctness of UpdateSample(A,x)

- Need to show that the likelihood of a sample being selected is \( \min(1, \frac{k}{n}) \).
- **Inductive Step:** Assume true for some \( n \leq k \). Need to prove that for \( n + 1 \) the likelihood of selecting any element is \( \frac{k}{n+1} \). When processing the \( n + 1 \)-th element the likelihood that it is selected as part of the sample is \( \frac{k}{n+1} \). By the random selection of \( p \) and the following if statement. Just need to show that the likelihood of any other element being selected is also \( \frac{k}{n+1} \).

- The likelihood of any of the previous \( n \) elements being in the sample is \( \frac{k}{n} \).
- Want to calculate the likelihood of these elements staying in the sample.
Need to show that the likelihood of a sample being selected is \( \min(1, \frac{k}{n}) \).

**Inductive Step:** Assume true for some \( n \leq k \). Need to prove that for \( n + 1 \) the likelihood of selecting any element is \( \frac{k}{n+1} \). When processing the \( n + 1 \)-th element the likelihood that it is selected as part of the sample is \( \frac{k}{n+1} \). By the random selection of \( p \) and the following if statement. Just need to show that the likelihood of any other element being selected is also \( \frac{k}{n+1} \).

The likelihood of any of the previous \( n \) elements being in the sample is \( \frac{k}{n} \).

Want to calculate the likelihood of these elements staying in the sample.

The likelihood of an element, \( i \), being replaced is 1) the likelihood that the \( n+1 \)-th element is selected, \( \frac{k}{n+1} \), times 2) the likelihood of element \( i \) being selected for replacement, \( \frac{1}{k} \).
Proving the correctness of UpdateSample(A,x)

- Need to show that the likelihood of a sample being selected is \( \min(1, \frac{k}{n}) \).

- **Inductive Step**: Assume true for some \( n \leq k \). Need to prove that for \( n + 1 \) the likelihood of selecting any element is \( \frac{k}{n+1} \). When processing the \( n+1 \)-th element the likelihood that it is selected as part of the sample is \( \frac{k}{n+1} \). By the random selection of \( p \) and the following if statement. Just need to show that the likelihood of any other element being selected is also \( \frac{k}{n+1} \).

- The likelihood of any of the previous \( n \) elements being in the sample is \( \frac{k}{n} \).

- Want to calculate the likelihood of these elements staying in the sample.

- The likelihood of an element, \( i \), being **replaced** is 1) the likelihood that the \( n+1 \)-th element is selected, \( \frac{k}{n+1} \), times 2) the likelihood of element \( i \) being selected for replacement, \( \frac{1}{k} \).

- \( \frac{k}{n+1} \times \frac{1}{k} = \frac{1}{n+1} \). Therefore the likelihood of an element **staying** in the sample is \( 1 - \frac{1}{n+1} = \frac{n}{n+1} \).
Need to show that the likelihood of a sample being selected is $\min(1, \frac{k}{n})$.

**Inductive Step:** Assume true for some $n \leq k$. Need to prove that for $n + 1$ the likelihood of selecting any element is $\frac{k}{n+1}$. When processing the $n + 1$-th element the likelihood that it is selected as part of the sample is $\frac{k}{n+1}$. By the random selection of $p$ and the following if statement. Just need to show that the likelihood of any other element being selected is also $\frac{k}{n+1}$.

The likelihood of any of the previous $n$ elements being in the sample is $\frac{k}{n}$.

Want to calculate the likelihood of these elements staying in the sample.

The likelihood of an element, $i$, being replaced is 1) the likelihood that the $n+1$-th element is selected, $\frac{k}{n+1}$, times 2) the likelihood of element $i$ being selected for replacement, $\frac{1}{k}$.

$\frac{k}{n+1} \times \frac{1}{k} = \frac{1}{n+1}$. Therefore the likelihood of an element staying in the sample is $1 - \frac{1}{n+1} = \frac{n}{n+1}$.

So the likelihood of one of the previous $n$ elements being in the sample is the likelihood that it was in the sample after $n$ elements, $\frac{k}{n}$, and the likelihood that it survived, $\frac{n}{n+1}$.

$\frac{k}{n} \times \frac{n}{n+1} = \frac{k}{n+1}$
Bye

- Next time (10/19)
  - Midterm Review
- For Next Class
  - Bring any questions, Homework problems that you don’t understand, etc.