Data Streams and Approximate Query Processing

Krzysztof Dembczyński

Intelligent Decision Support Systems Laboratory (IDSS)
Poznań University of Technology, Poland

Bachelor studies, eighth semester
Academic year 2018/19 (summer semester)

Review of the previous lectures

- Processing of massive datasets
- Evolution of database systems
- OLTP and OLAP systems
- ETL
- Dimensional modeling
- Data processing
- MapReduce in Spark

Motivation

Motivation

- Exploratory data analysis: original data can be too big to compute the results for unpredictable queries.

Motivation

- Exploratory data analysis: original data can be too big to compute the results for unpredictable queries.
- Data streams: fast rate of incoming data that cannot be entirely stored and analyze offline.

Motivation

- Exploratory data analysis: original data can be too big to compute the results for unpredictable queries.
- Data streams: fast rate of incoming data that cannot be entirely stored and analyze offline.
- Possible solution:

Motivation

- Exploratory data analysis: original data can be too big to compute the results for unpredictable queries.
- Data streams: fast rate of incoming data that cannot be entirely stored and analyze offline.
- Possible solution: fast, approximate answers based on small synopsis of database

Approximate query processing

- **Example**: Compute average salary:

Approximate query processing

- **Example**: Compute average salary:
  - Approximate answer: 65000 ± 2000 (with 95% confidence)
  - Return answer in 5 seconds
  - Exact answer: 65792.27
  - Return answer in 30 minutes

Approximate query processing

- **Example**: Compute average salary:
  - Approximate answer: 65000 ± 2000 (with 95% confidence)
  - Return answer in 5 seconds
  - Exact answer: 65792.27
  - Return answer in 30 minutes
- **Example**: How many users have visited a given Web site in a given month:

Approximate query processing

- **Example**: Compute average salary:
  - Approximate answer: 65000 ± 2000 (with 95% confidence)
  - Return answer in 5 seconds
  - Exact answer: 65792.27
  - Return answer in 30 minutes
- **Example**: How many users have visited a given Web site in a given month:
  - Efficient computation of the distinct count.

Synopsis data structure

- Synopsis data structure: any data structures that are substantively smaller than their base dataset.

Outline

Outline

1. Sampling
2. Filtering
3. Counting distinct elements
4. Summary

Sampling

- Simplest data synopsis = uniform random sample
  - Evaluate query over random sample of the data
  - Extrapolate from random sample to estimate overall result
  - Large body of knowledge from statistics: unbiased/biased estimators, variance of estimates, confidence intervals.

Sampling – confidence interval

- Confidence intervals for true average μ:

$$P(X_l \leq \mu \leq X_r) = 1 - \alpha$$

where X_l and X_r are random variables indicating the left and the right endpoints of the interval, and $1 - \alpha$ is the confidence level (usually, α is close to zero, e.g., $\alpha = 0.05$).

Sampling – confidence interval

- Let R be the original relation with the average μ, S the sample with n elements, and the confidence level 90%.
- Central limit theorem:
  - The confidence interval is defined through:

    $$P\left(\bar{X} - 1.65\frac{\sigma_S}{\sqrt{n}} \leq \mu \leq \bar{X} + 1.65\frac{\sigma_S}{\sqrt{n}}\right) = 0.9$$

    where $\bar{X}$ is the average computed over the sample, σ_S is the standard deviation of the values in S and 1.65 is the 0.95-quantile (since $1 - \alpha/2$) of the standardized normal distribution.
  - This confidence interval holds for $n \to \infty$ (in practice, when $n > 30$)

Sampling

- Two basic approaches to sampling

Sampling

- Two basic approaches to sampling
  - On-demand sampling:

Sampling

- Two basic approaches to sampling
  - On-demand sampling:
    - Generate sample when query is asked

Sampling

- Two basic approaches to sampling
  - On-demand sampling:
    - Generate sample when query is asked
    - Unfortunately can be quite slow (even more costly then scanning the whole relation)

Sampling

- Two basic approaches to sampling
  - On-demand sampling:
    - Generate sample when query is asked
    - Unfortunately can be quite slow (even more costly then scanning the whole relation)
  - Pre-computed samples:

Sampling

- Two basic approaches to sampling
  - On-demand sampling:
    - Generate sample when query is asked
    - Unfortunately can be quite slow (even more costly then scanning the whole relation)
  - Pre-computed samples:
    - Generate samples of big tables in advance

Sampling

- Two basic approaches to sampling
  - On-demand sampling:
    - Generate sample when query is asked
    - Unfortunately can be quite slow (even more costly then scanning the whole relation)
  - Pre-computed samples:
    - Generate samples of big tables in advance
    - Store the pre-computed samples separately

Sampling

- Two basic approaches to sampling
  - On-demand sampling:
    - Generate sample when query is asked
    - Unfortunately can be quite slow (even more costly then scanning the whole relation)
  - Pre-computed samples:
    - Generate samples of big tables in advance
    - Store the pre-computed samples separately
    - Query re-write for using sample tables

Sampling

- Two basic approaches to sampling
  - On-demand sampling:
    - Generate sample when query is asked
    - Unfortunately can be quite slow (even more costly then scanning the whole relation)
  - Pre-computed samples:
    - Generate samples of big tables in advance
    - Store the pre-computed samples separately
    - Query re-write for using sample tables
- Non-uniform (stratified) sampling

Sampling in data streams

- **A problem to solve**: sample a fraction r from a stream.

Sampling in data streams

- **A problem to solve**: sample a fraction r from a stream.
- **Solution**: Generate a random number and make a decision based on it about accepting or rejecting a given item.

Sampling in data streams

- **A problem to solve**: sample a fraction r from a stream.
- **Solution**: Generate a random number and make a decision based on it about accepting or rejecting a given item.
- **Example**: Store only $1/10$th of the stream; generate an integer from 0 to 9 and accept an item if the random number is, say, 0.

Sampling in data streams

- **A problem to solve**: sample a fraction r from a stream.
- **Solution**: Generate a random number and make a decision based on it about accepting or rejecting a given item.
- **Example**: Store only $1/10$th of the stream; generate an integer from 0 to 9 and accept an item if the random number is, say, 0.
- Instead of generating random numbers, we can use hash functions (applied to a key attribute of the item) to select the sample, for example, by hashing items to ten buckets and choosing the items only from the first bucket.

Sampling in data streams

- **A problem to solve**: sample a fraction r from a stream.
- **Solution**: Generate a random number and make a decision based on it about accepting or rejecting a given item.
- **Example**: Store only $1/10$th of the stream; generate an integer from 0 to 9 and accept an item if the random number is, say, 0.
- Instead of generating random numbers, we can use hash functions (applied to a key attribute of the item) to select the sample, for example, by hashing items to ten buckets and choosing the items only from the first bucket.
- We have to be very careful how we sample the data (i.e., select the key for the hash function).

Sampling in data streams

- **Example**: In the web traffic application we want to estimate the fraction of the typical user's queries that were repeated over the past month.

Sampling in data streams

- **Example**: In the web traffic application we want to estimate the fraction of the typical user's queries that were repeated over the past month.
  - The approach presented above will fail for this query!!!

Sampling in data streams

- **Example**: In the web traffic application we want to estimate the fraction of the typical user's queries that were repeated over the past month.
  - The approach presented above will fail for this query!!!
  - Suppose a user has issued s search queries one time and d search queries twice, and no search queries more than twice.

Sampling in data streams

- **Example**: In the web traffic application we want to estimate the fraction of the typical user's queries that were repeated over the past month.
  - The approach presented above will fail for this query!!!
  - Suppose a user has issued s search queries one time and d search queries twice, and no search queries more than twice.
  - The correct answer is: $d/(s+d)$.

Sampling in data streams

- **Example**: In the web traffic application we want to estimate the fraction of the typical user's queries that were repeated over the past month.
  - The approach presented above will fail for this query!!!
  - Suppose a user has issued s search queries one time and d search queries twice, and no search queries more than twice.
  - The correct answer is: $d/(s+d)$.
  - What is the answer if computed on 10% of queries in the sample?

Sampling in data streams

- **Example**: In the web traffic application we want to estimate the fraction of the typical user's queries that were repeated over the past month.
  - The approach presented above will fail for this query!!!
  - Suppose a user has issued s search queries one time and d search queries twice, and no search queries more than twice.
  - The correct answer is: $d/(s+d)$.
  - What is the answer if computed on 10% of queries in the sample?
    - The expected number of queries issued once:

Sampling in data streams

- **Example**: In the web traffic application we want to estimate the fraction of the typical user's queries that were repeated over the past month.
  - The approach presented above will fail for this query!!!
  - Suppose a user has issued s search queries one time and d search queries twice, and no search queries more than twice.
  - The correct answer is: $d/(s+d)$.
  - What is the answer if computed on 10% of queries in the sample?
    - The expected number of queries issued once: $s/10$

Sampling in data streams

- **Example**: In the web traffic application we want to estimate the fraction of the typical user's queries that were repeated over the past month.
  - The approach presented above will fail for this query!!!
  - Suppose a user has issued s search queries one time and d search queries twice, and no search queries more than twice.
  - The correct answer is: $d/(s+d)$.
  - What is the answer if computed on 10% of queries in the sample?
    - The expected number of queries issued once: $s/10$
    - The expected number of queries issued twice that also appear twice in the sample:

Sampling in data streams

- **Example**: In the web traffic application we want to estimate the fraction of the typical user's queries that were repeated over the past month.
  - The approach presented above will fail for this query!!!
  - Suppose a user has issued s search queries one time and d search queries twice, and no search queries more than twice.
  - The correct answer is: $d/(s+d)$.
  - What is the answer if computed on 10% of queries in the sample?
    - The expected number of queries issued once: $s/10$
    - The expected number of queries issued twice that also appear twice in the sample: $d/100$.

Sampling in data streams

- **Example**: In the web traffic application we want to estimate the fraction of the typical user's queries that were repeated over the past month.
  - The approach presented above will fail for this query!!!
  - Suppose a user has issued s search queries one time and d search queries twice, and no search queries more than twice.
  - The correct answer is: $d/(s+d)$.
  - What is the answer if computed on 10% of queries in the sample?
    - The expected number of queries issued once: $s/10$
    - The expected number of queries issued twice that also appear twice in the sample: $d/100$.
    - The expected number of queries issued twice that appear once in the sample:

Sampling in data streams

- **Example**: In the web traffic application we want to estimate the fraction of the typical user's queries that were repeated over the past month.
  - The approach presented above will fail for this query!!!
  - Suppose a user has issued s search queries one time and d search queries twice, and no search queries more than twice.
  - The correct answer is: $d/(s+d)$.
  - What is the answer if computed on 10% of queries in the sample?
    - The expected number of queries issued once: $s/10$
    - The expected number of queries issued twice that also appear twice in the sample: $d/100$.
    - The expected number of queries issued twice that appear once in the sample: $18d/100$.

Sampling in data streams

- **Example**: In the web traffic application we want to estimate the fraction of the typical user's queries that were repeated over the past month.
  - The approach presented above will fail for this query!!!
  - Suppose a user has issued s search queries one time and d search queries twice, and no search queries more than twice.
  - The correct answer is: $d/(s+d)$.
  - What is the answer if computed on 10% of queries in the sample?
    - The expected number of queries issued once: $s/10$
    - The expected number of queries issued twice that also appear twice in the sample: $d/100$.
    - The expected number of queries issued twice that appear once in the sample: $18d/100$.
    - The estimate is thus:

Sampling in data streams

- **Example**: In the web traffic application we want to estimate the fraction of the typical user's queries that were repeated over the past month.
  - The approach presented above will fail for this query!!!
  - Suppose a user has issued s search queries one time and d search queries twice, and no search queries more than twice.
  - The correct answer is: $d/(s+d)$.
  - What is the answer if computed on 10% of queries in the sample?
    - The expected number of queries issued once: $s/10$
    - The expected number of queries issued twice that also appear twice in the sample: $d/100$.
    - The expected number of queries issued twice that appear once in the sample: $18d/100$.
    - The estimate is thus: $d/(d+18d+10s)$.

Sampling in data streams

- To solve the above problem we would change the sampling method, for example, by selecting 10% of users.

Sampling in data streams

- To solve the above problem we would change the sampling method, for example, by selecting 10% of users.
- One way of selecting users is the following:

Sampling in data streams

- To solve the above problem we would change the sampling method, for example, by selecting 10% of users.
- One way of selecting users is the following:
  - maintain a list of users

Sampling in data streams

- To solve the above problem we would change the sampling method, for example, by selecting 10% of users.
- One way of selecting users is the following:
  - maintain a list of users
  - for each new query check whether a user is already in the list;

Sampling in data streams

- To solve the above problem we would change the sampling method, for example, by selecting 10% of users.
- One way of selecting users is the following:
  - maintain a list of users
  - for each new query check whether a user is already in the list;
  - if yes, store the query,

Sampling in data streams

- To solve the above problem we would change the sampling method, for example, by selecting 10% of users.
- One way of selecting users is the following:
  - maintain a list of users
  - for each new query check whether a user is already in the list;
  - if yes, store the query,
  - else, generate a random number to decide, whether to store or not the user.

Sampling in data streams

- To solve the above problem we would change the sampling method, for example, by selecting 10% of users.
- One way of selecting users is the following:
  - maintain a list of users
  - for each new query check whether a user is already in the list;
  - if yes, store the query,
  - else, generate a random number to decide, whether to store or not the user.
- To improve the method use hash functions (for the same user it gives the same value) and store queries for the users in a given range of hash values that corresponds to the desired fraction.

Sampling from infinite streams

- How to sample s elements from an infinite stream with equal probability?

Sampling from infinite streams

- How to sample s elements from an infinite stream with equal probability?
- The answer is: **Reservoir sampling**

Sampling from infinite streams

- How to sample s elements from an infinite stream with equal probability?
- The answer is: **Reservoir sampling**
  - Take s first elements from the stream

Sampling from infinite streams

- How to sample s elements from an infinite stream with equal probability?
- The answer is: **Reservoir sampling**
  - Take s first elements from the stream
  - For each next ith element do the following:

Sampling from infinite streams

- How to sample s elements from an infinite stream with equal probability?
- The answer is: **Reservoir sampling**
  - Take s first elements from the stream
  - For each next ith element do the following:
    - Accept the ith element with probability s/i.

Sampling from infinite streams

- How to sample s elements from an infinite stream with equal probability?
- The answer is: **Reservoir sampling**
  - Take s first elements from the stream
  - For each next ith element do the following:
    - Accept the ith element with probability s/i.
    - If the element is accepted then remove one of the previously drawn elements, with equal probability $1/s$.

Sampling from infinite streams

- How to sample s elements from an infinite stream with equal probability?
- The answer is: **Reservoir sampling**
  - Take s first elements from the stream
  - For each next ith element do the following:
    - Accept the ith element with probability s/i.
    - If the element is accepted then remove one of the previously drawn elements, with equal probability $1/s$.
- Is the probability of choosing any element equal?

Sampling from infinite streams

- How to sample s elements from an infinite stream with equal probability?
- The answer is: **Reservoir sampling**
  - Take s first elements from the stream
  - For each next ith element do the following:
    - Accept the ith element with probability s/i.
    - If the element is accepted then remove one of the previously drawn elements, with equal probability $1/s$.
- Is the probability of choosing any element equal?
- In other words, is the probability of each element being selected equal s/i in each iteration?

Sampling from infinite streams

- The proof is by induction:

Sampling from infinite streams

- The proof is by induction:
  - The new element is chosen with s/i as we would like to have.

Sampling from infinite streams

- The proof is by induction:
  - The new element is chosen with s/i as we would like to have.
  - By inductive hypothesis, the probability of drawing each previous element is $s/(i-1)$.

Sampling from infinite streams

- The proof is by induction:
  - The new element is chosen with s/i as we would like to have.
  - By inductive hypothesis, the probability of drawing each previous element is $s/(i-1)$.
  - Recall that in case of accepting the new element, an old one is discarded with probability $1/s$.

Sampling from infinite streams

- The proof is by induction:
  - The new element is chosen with s/i as we would like to have.
  - By inductive hypothesis, the probability of drawing each previous element is $s/(i-1)$.
  - Recall that in case of accepting the new element, an old one is discarded with probability $1/s$.
  - The probability of each previous element to be selected is then:

Sampling from infinite streams

- The proof is by induction:
  - The new element is chosen with s/i as we would like to have.
  - By inductive hypothesis, the probability of drawing each previous element is $s/(i-1)$.
  - Recall that in case of accepting the new element, an old one is discarded with probability $1/s$.
  - The probability of each previous element to be selected is then:

$$\left(1-\frac{s}{i}\right)\left(\frac{s}{i-1}\right)+\left(\frac{s}{i}\right)\left(\frac{s}{i-1}\right)\left(1-\frac{1}{s}\right) \;=$$

Sampling from infinite streams

- The proof is by induction:
  - The new element is chosen with s/i as we would like to have.
  - By inductive hypothesis, the probability of drawing each previous element is $s/(i-1)$.
  - Recall that in case of accepting the new element, an old one is discarded with probability $1/s$.
  - The probability of each previous element to be selected is then:

$$\left(1-\frac{s}{i}\right)\left(\frac{s}{i-1}\right)+\left(\frac{s}{i}\right)\left(\frac{s}{i-1}\right)\left(1-\frac{1}{s}\right) \;=\; \frac{s}{i-1}\left(\frac{i-s}{i}+\frac{s(s-1)}{is}\right)$$

Sampling from infinite streams

- The proof is by induction:
  - The new element is chosen with s/i as we would like to have.
  - By inductive hypothesis, the probability of drawing each previous element is $s/(i-1)$.
  - Recall that in case of accepting the new element, an old one is discarded with probability $1/s$.
  - The probability of each previous element to be selected is then:

$$\begin{aligned}\left(1-\frac{s}{i}\right)\left(\frac{s}{i-1}\right)+\left(\frac{s}{i}\right)\left(\frac{s}{i-1}\right)\left(1-\frac{1}{s}\right) &= \frac{s}{i-1}\left(\frac{i-s}{i}+\frac{s(s-1)}{is}\right)\\ &= \frac{s}{i-1}\left(\frac{i-s}{i}+\frac{s-1}{i}\right)\end{aligned}$$

- The proof is by induction:
  - The new element is chosen with s/i as we would like to have.
  - By inductive hypothesis, the probability of drawing each previous element is $s/(i-1)$.
  - Recall that in case of accepting the new element, an old one is discarded with probability $1/s$.
  - The probability of each previous element to be selected is then:

$$\begin{aligned}\left(1-\frac{s}{i}\right)\left(\frac{s}{i-1}\right)+\left(\frac{s}{i}\right)\left(\frac{s}{i-1}\right)\left(1-\frac{1}{s}\right) &= \frac{s}{i-1}\left(\frac{i-s}{i}+\frac{s(s-1)}{is}\right)\\ &= \frac{s}{i-1}\left(\frac{i-s}{i}+\frac{s-1}{i}\right)\\ &= \frac{s}{i-1}\frac{i-1}{i}=\frac{s}{i}\end{aligned}$$

Outline

Filtering

- Checking whether a given item does not belong to a given set.

Filtering

- Checking whether a given item does not belong to a given set.
- **Example**: Filtering of spam email addresses

Filtering

- Checking whether a given item does not belong to a given set.
- **Example**: Filtering of spam email addresses
  - Suppose we have a set S of one billion allowed email addresses — those that we will allow through because we believe them not to be spam.

Filtering

- Checking whether a given item does not belong to a given set.
- **Example**: Filtering of spam email addresses
  - Suppose we have a set S of one billion allowed email addresses — those that we will allow through because we believe them not to be spam.
  - The problem is to quickly verify whether an email address is spam or not.

Filtering

- Checking whether a given item does not belong to a given set.
- **Example**: Filtering of spam email addresses
  - Suppose we have a set S of one billion allowed email addresses — those that we will allow through because we believe them not to be spam.
  - The problem is to quickly verify whether an email address is spam or not.
- Standard approaches:

Filtering

- Checking whether a given item does not belong to a given set.
- **Example**: Filtering of spam email addresses
  - Suppose we have a set S of one billion allowed email addresses — those that we will allow through because we believe them not to be spam.
  - The problem is to quickly verify whether an email address is spam or not.
- Standard approaches:
  - Bitmap of all items (too big, hard to implement if the domain changes)

Filtering

- Checking whether a given item does not belong to a given set.
- **Example**: Filtering of spam email addresses
  - Suppose we have a set S of one billion allowed email addresses — those that we will allow through because we believe them not to be spam.
  - The problem is to quickly verify whether an email address is spam or not.
- Standard approaches:
  - Bitmap of all items (too big, hard to implement if the domain changes)
  - List (or array) of elements in a set (slow performance, storage is also costly).

Filtering

- Checking whether a given item does not belong to a given set.
- **Example**: Filtering of spam email addresses
  - Suppose we have a set S of one billion allowed email addresses — those that we will allow through because we believe them not to be spam.
  - The problem is to quickly verify whether an email address is spam or not.
- Standard approaches:
  - Bitmap of all items (too big, hard to implement if the domain changes)
  - List (or array) of elements in a set (slow performance, storage is also costly).
  - Efficient implementation of a set, e.g., hash-based (much more efficient, but no improvement in space complexity).

Bloom filter

- The problem can be defined as:

Bloom filter

- The problem can be defined as:
  - Find a representation that allows efficient membership queries of the n-element set $S = \{s_1, s_2, \ldots, s_n\}$ from a very large universe U, $|U| = u$, with $u \gg n$.

Bloom filter

- The problem can be defined as:
  - Find a representation that allows efficient membership queries of the n-element set $S = \{s_1, s_2, \ldots, s_n\}$ from a very large universe U, $|U| = u$, with $u \gg n$.
- A Bloom filter is a space-efficient probabilistic data structure for set membership.

Bloom filter

- The problem can be defined as:
  - Find a representation that allows efficient membership queries of the n-element set $S = \{s_1, s_2, \ldots, s_n\}$ from a very large universe U, $|U| = u$, with $u \gg n$.
- A Bloom filter is a space-efficient probabilistic data structure for set membership.
  - To maximize space efficiency, correctness is sacrificed: if a given key is not in the set, then a Bloom filter may give the wrong answer (this is called a false positive), but the probability of such a wrong answer can be made small.

Bloom filter

- The problem can be defined as:
  - Find a representation that allows efficient membership queries of the n-element set $S = \{s_1, s_2, \ldots, s_n\}$ from a very large universe U, $|U| = u$, with $u \gg n$.
- A Bloom filter is a space-efficient probabilistic data structure for set membership.
  - To maximize space efficiency, correctness is sacrificed: if a given key is not in the set, then a Bloom filter may give the wrong answer (this is called a false positive), but the probability of such a wrong answer can be made small.
  - The more elements that are added to the set, the larger the probability of false positives.

Bloom filter

- An empty Bloom filter is a bitarray of m ($m > n$, say $m = 8n$) bits, all set to 0.

0	0	0	0	0	0	0	0	0	0	0

- There must also be k different hash functions defined, each of which maps or hashes some set element to one of the m array positions with a uniform random distribution.

Bloom filter

- To add an element, feed it to each of the k hash functions to get k array positions. Set the bits at all these positions to 1.

0	1	0	1	0	1	0	0	0	0	0

Bloom filter

- To add an element, feed it to each of the k hash functions to get k array positions. Set the bits at all these positions to 1.

0	1	0	1	0	1	0	0	0	0	0

- To query for an element (test whether it is in the set), feed it to each of the k hash functions to get k array positions.

Bloom filter

- To add an element, feed it to each of the k hash functions to get k array positions. Set the bits at all these positions to 1.

0	1	0	1	0	1	0	0	0	0	0

- To query for an element (test whether it is in the set), feed it to each of the k hash functions to get k array positions.
  - If any of the bits at these positions are 0, the element is definitely not in the set – if it were, then all the bits would have been set to 1 when it was inserted.

Bloom filter

- To add an element, feed it to each of the k hash functions to get k array positions. Set the bits at all these positions to 1.

0	1	0	1	0	1	0	0	0	0	0

- To query for an element (test whether it is in the set), feed it to each of the k hash functions to get k array positions.
  - If any of the bits at these positions are 0, the element is definitely not in the set – if it were, then all the bits would have been set to 1 when it was inserted.
  - If all are 1, then either the element is in the set, or the bits have by chance been set to 1 during the insertion of other elements, resulting in a false positive.

Bloom filter

- Consider a Bloom filter with $m = 11$ and $k = 3$:

Bloom filter

- Consider a Bloom filter with $m = 11$ and $k = 3$:
  - $h_1(x) = x \mod 11$

Bloom filter

- Consider a Bloom filter with $m = 11$ and $k = 3$:
  - $h_1(x) = x \mod 11$
  - $h_2(x) = (2x + 3) \mod 11$

Bloom filter

- Consider a Bloom filter with $m = 11$ and $k = 3$:
  - $h_1(x) = x \mod 11$
  - $h_2(x) = (2x + 3) \mod 11$
  - $h_3(x) = (3x + 5) \mod 11$

Bloom filter

- Consider a Bloom filter with $m = 11$ and $k = 3$:
  - $h_1(x) = x \mod 11$
  - $h_2(x) = (2x + 3) \mod 11$
  - $h_3(x) = (3x + 5) \mod 11$
- Add to the filter the following numbers: 7, 11, 25:

Bloom filter

- Consider a Bloom filter with $m = 11$ and $k = 3$:
  - $h_1(x) = x \mod 11$
  - $h_2(x) = (2x + 3) \mod 11$
  - $h_3(x) = (3x + 5) \mod 11$
- Add to the filter the following numbers: 7, 11, 25:
  - $h_1(7) = 7$, $h_2(7) = 6$, $h_3(7) = 4$

0	0	0	0	1	0	1	1	0	0	0

Bloom filter

- Consider a Bloom filter with $m = 11$ and $k = 3$:
  - $h_1(x) = x \mod 11$
  - $h_2(x) = (2x + 3) \mod 11$
  - $h_3(x) = (3x + 5) \mod 11$
- Add to the filter the following numbers: 7, 11, 25:
  - $h_1(7) = 7$, $h_2(7) = 6$, $h_3(7) = 4$

0	0	0	0	1	0	1	1	0	0	0

  - $h_1(11) = 0$, $h_2(11) = 3$, $h_3(11) = 5$

1	0	0	1	1	1	1	1	0	0	0

Bloom filter

- Consider a Bloom filter with $m = 11$ and $k = 3$:
  - $h_1(x) = x \mod 11$
  - $h_2(x) = (2x + 3) \mod 11$
  - $h_3(x) = (3x + 5) \mod 11$
- Add to the filter the following numbers: 7, 11, 25:
  - $h_1(7) = 7$, $h_2(7) = 6$, $h_3(7) = 4$

0	0	0	0	1	0	1	1	0	0	0

  - $h_1(11) = 0$, $h_2(11) = 3$, $h_3(11) = 5$

1	0	0	1	1	1	1	1	0	0	0

  - $h_1(25) = 3$, $h_2(25) = 9$, $h_3(25) = 3$

1	0	0	1	1	1	1	1	0	1	0

Bloom filter

- For the Bloom filter:

1	0	0	1	1	1	1	1	0	1	0

check whether 5 and 15 is in the set:

Bloom filter

- For the Bloom filter:

1	0	0	1	1	1	1	1	0	1	0

check whether 5 and 15 is in the set:

  - $h_1(5) = 5$, $h_2(5) = 2$, $h_3(5) = 9$

0	0	1	0	0	1	0	0	0	1	0

There is no 5 in the set.

Bloom filter

- For the Bloom filter:

1	0	0	1	1	1	1	1	0	1	0

  check whether 5 and 15 is in the set:

  - $h_1(5) = 5$, $h_2(5) = 2$, $h_3(5) = 9$

0	0	1	0	0	1	0	0	0	1	0

    There is no 5 in the set.

  - $h_1(15) = 4$, $h_2(15) = 0$, $h_3(15) = 6$

1	0	0	0	1	0	1	0	0	0	0

    There might be 15 in the set (false positive in this case)

Probability of false positives

- The probability that one hash fails to set a given bit is:

$$1 - \frac{1}{m}$$

Probability of false positives

- The probability that one hash fails to set a given bit is:

$$1 - \frac{1}{m}$$

- Hence, after all n elements have been inserted into the Bloom filter, the probability that a specific bit is still 0 is

$$\left(1 - \frac{1}{m}\right)^{kn}$$

assuming that the hash functions are independent and perfectly random.

Probability of false positive

- Since:

$$\left(1 - \frac{1}{x}\right)^{x} \approx e^{-1}, \quad \text{we have:} \left(1 - \frac{1}{m}\right)^{\frac{m(kn)}{m}} \approx e^{-\frac{kn}{m}}$$

Probability of false positive

- Since:

$$\left(1-\frac{1}{x}\right)^{x} \approx e^{-1}, \quad \text{we have: } \left(1-\frac{1}{m}\right)^{\frac{m(kn)}{m}} \approx e^{-\frac{kn}{m}}$$

- The probability of a false positive is the probability that a specific set of k bits are 1, which is

$$\left(1-\left(1-\frac{1}{m}\right)^{kn}\right)^{k} \approx \left(1-e^{-\frac{kn}{m}}\right)^{k}$$

Probability of false positive

- For a ratio $m/n = 8$ and $k = 1$ we get

$$\left(1 - e^{-\frac{kn}{m}}\right)^k = 1 - e^{-1/8} = 0.1175$$

Probability of false positive

- For a ratio $m/n = 8$ and $k = 1$ we get

$$\left(1 - e^{-\frac{kn}{m}}\right)^k = 1 - e^{-1/8} = 0.1175$$

- For $k = 2$ we can get:

$$\left(1 - e^{-\frac{kn}{m}}\right)^k = \left(1 - e^{-2/8}\right)^2 = 0.0489$$

Optimizing the number of hash functions

- Suppose we are given the ratio $\frac{m}{n}$ and want to optimize the number k of hash functions to minimize the false positive rate.

Optimizing the number of hash functions

- Suppose we are given the ratio $\frac{m}{n}$ and want to optimize the number k of hash functions to minimize the false positive rate.
- Note that more hash functions increase the precision but also the number of 1s in the filter, thus making false positives both less and more likely at the same time.

Optimizing the number of hash functions

- Suppose we are given the ratio $\frac{m}{n}$ and want to optimize the number k of hash functions to minimize the false positive rate.
- Note that more hash functions increase the precision but also the number of 1s in the filter, thus making false positives both less and more likely at the same time.
- Formally, the problem can be stated as:

$$k^* = \arg\min_{k \in \mathbb{N}^+} \left(1 - e^{-\frac{kn}{m}}\right)^k$$

Optimizing the number of hash functions

- The solution is:
$$k^* = (\ln 2)\frac{m}{n}$$

Optimizing the number of hash functions

- The solution is:

$$k^* = (\ln 2)\frac{m}{n}$$

- For the optimal value of k, the false positive rate is:

$$\begin{aligned}\left(1-e^{-\frac{k^* n}{m}}\right)^{k^*} &= \left(1-e^{-(\ln 2)\frac{m}{n}\frac{n}{m}}\right)^{(\ln 2)\frac{m}{n}} \\ &= \left(\frac{1}{2}\right)^{(\ln 2)\frac{m}{n}} = (0.6185)^{\frac{m}{n}}\end{aligned}$$

Optimizing the number of hash functions

- The solution is:

$$k^* = (\ln 2)\frac{m}{n}$$

- For the optimal value of k, the false positive rate is:

$$\begin{aligned}\left(1 - e^{-\frac{k^* n}{m}}\right)^{k^*} &= \left(1 - e^{-(\ln 2)\frac{m}{n}\frac{n}{m}}\right)^{(\ln 2)\frac{m}{n}} \\ &= \left(\frac{1}{2}\right)^{(\ln 2)\frac{m}{n}} = (0.6185)^{\frac{m}{n}}\end{aligned}$$

- Already $m = 8n$ reduces the chance of error to roughly 2%, and $m = 10n$ to less than 1%.

Optimizing Bloom filters

- The optimal number of hash functions can also be expressed in terms of false positive rate ϵ.

Optimizing Bloom filters

- The optimal number of hash functions can also be expressed in terms of false positive rate ϵ.
- Remark that:

$$\epsilon = \left(\frac{1}{2}\right)^{(\ln 2)\frac{m}{n}} = \left(\frac{1}{2}\right)^{k^*}$$

so, we get:

$$k^* = \log_2(1/\epsilon)$$

Optimizing Bloom filters

- The optimal number of hash functions can also be expressed in terms of false positive rate ϵ.
- Remark that:

$$\epsilon = \left(\frac{1}{2}\right)^{(\ln 2)\frac{m}{n}} = \left(\frac{1}{2}\right)^{k^*}$$

  so, we get:

$$k^* = \log_2(1/\epsilon)$$

- The required space in turn can be given as

$$\frac{m}{n} = (\ln 2)^{-1} \log_2(1/\epsilon) = 1.44 \log_2(1/\epsilon)$$

Outline

Distinct Count of Elements

- A quite simple instruction in SQL:

```
SELECT COUNT (DISTINCT A) FROM R
```

Distinct Count of Elements

- A quite simple instruction in SQL:

```
SELECT COUNT (DISTINCT A) FROM R
```

- Standard approach:

Distinct Count of Elements

- A quite simple instruction in SQL:

```
SELECT COUNT (DISTINCT A) FROM R
```

- Standard approach:
  - Sorting: requires $O(n \log n)$ operations and $O(n)$ space,

Distinct Count of Elements

- A quite simple instruction in SQL:

```
SELECT COUNT (DISTINCT A) FROM R
```

- Standard approach:
  - Sorting: requires $O(n \log n)$ operations and $O(n)$ space,
  - Hashing: requires $O(n)$ operations and $O(n)$ space,

Distinct Count of Elements

- A quite simple instruction in SQL:

```
SELECT COUNT (DISTINCT A) FROM R
```

- Standard approach:
  - Sorting: requires $O(n \log n)$ operations and $O(n)$ space,
  - Hashing: requires $O(n)$ operations and $O(n)$ space,
  - Sampling: we need to be careful with the estimate.

Distinct count of elements

- Other linear space solutions:
  - Bloom filter
  - Linear counting

Distinct count of elements

- Bloom filter for distinct count:

Distinct count of elements

- Bloom filter for distinct count:
  - For each element test whether it is already present in the Bloom filter.

Distinct count of elements

- Bloom filter for distinct count:
  - For each element test whether it is already present in the Bloom filter.
  - If the item is not present in the filter, then insert it into the filter, and increase the current count of distinct elements.

Distinct count of elements

- Bloom filter for distinct count:
  - For each element test whether it is already present in the Bloom filter.
  - If the item is not present in the filter, then insert it into the filter, and increase the current count of distinct elements.
- Because of the one-sided error nature of the Bloom filter

Distinct count of elements

- Bloom filter for distinct count:
  - For each element test whether it is already present in the Bloom filter.
  - If the item is not present in the filter, then insert it into the filter, and increase the current count of distinct elements.
- Because of the one-sided error nature of the Bloom filter
  - The distinct count never overestimates the true count,

Distinct count of elements

- Bloom filter for distinct count:
  - For each element test whether it is already present in the Bloom filter.
  - If the item is not present in the filter, then insert it into the filter, and increase the current count of distinct elements.
- Because of the one-sided error nature of the Bloom filter
  - The distinct count never overestimates the true count,
  - But may underestimate due to collisions.

Distinct count of elements

- Bloom filter for distinct count:
  - For each element test whether it is already present in the Bloom filter.
  - If the item is not present in the filter, then insert it into the filter, and increase the current count of distinct elements.
- Because of the one-sided error nature of the Bloom filter
  - The distinct count never overestimates the true count,
  - But may underestimate due to collisions.
- To ensure a small constant rate of undercounting, the size of the Bloom filter has to be proportional to the cardinality being estimated.

Distinct count of elements

- Bloom filter for distinct count:
  - For each element test whether it is already present in the Bloom filter.
  - If the item is not present in the filter, then insert it into the filter, and increase the current count of distinct elements.
- Because of the one-sided error nature of the Bloom filter
  - The distinct count never overestimates the true count,
  - But may underestimate due to collisions.
- To ensure a small constant rate of undercounting, the size of the Bloom filter has to be proportional to the cardinality being estimated.
- Due to the compactness of the Bloom filter bit vector, this requires less space than storing the full representation of the set, but only by constant factors.

Distinct count of elements

- Linear counting:

Distinct count of elements

- Linear counting:
  - Can be seen as a Bloom filter with a single hash function.

Distinct count of elements

- Linear counting:
  - Can be seen as a Bloom filter with a single hash function.
  - The number of distinct items is estimated based on the fraction of bits in the filter which remain as 0

Distinct count of elements

- Linear counting:
  - Can be seen as a Bloom filter with a single hash function.
  - The number of distinct items is estimated based on the fraction of bits in the filter which remain as 0
  - If this fraction is z, then the number of distinct items is estimated as

    $$m \ln \frac{1}{z},$$

    where m is the number of bits in the filter.

Distinct count of elements

- Linear counting:
  - Can be seen as a Bloom filter with a single hash function.
  - The number of distinct items is estimated based on the fraction of bits in the filter which remain as 0
  - If this fraction is z, then the number of distinct items is estimated as

    $$m \ln \frac{1}{z},$$

    where m is the number of bits in the filter.
  - For an accurate estimation, the m is required to be proportional to the number of distinct items.

Analysis of linear counting

- The probability of a bit b_i to be set to 0 after n insertion is:

$$P(b_i^n = 0) = \left(1 - \frac{1}{m}\right)^n \simeq \exp\left(-\frac{n}{m}\right) .$$

Analysis of linear counting

- The probability of a bit b_i to be set to 0 after n insertion is:

$$P(b_i^n = 0) = \left(1 - \frac{1}{m}\right)^n \simeq \exp\left(-\frac{n}{m}\right) .$$

- Let U_n be the random variable being a sum of 0 bits after n insertions:

$$U_n = \sum_{i=1}^{m} [\![b_i^n = 0]\!]$$

Analysis of linear counting

- The probability of a bit b_i to be set to 0 after n insertion is:

$$P(b_i^n = 0) = \left(1 - \frac{1}{m}\right)^n \simeq \exp\left(-\frac{n}{m}\right) .$$

- Let U_n be the random variable being a sum of 0 bits after n insertions:

$$U_n = \sum_{i=1}^{m} [\![b_i^n = 0]\!]$$

- The expectation of U_n is given by:

$$\mathbb{E}(U_n) = \sum_{i=1}^{m} P(b_i^n = 0) = m \exp\left(-\frac{n}{m}\right)$$

Analysis of linear counting

- Since $z = U_n/m$, we have:

$$\mathbb{E}(z) = \exp\left(-\frac{n}{m}\right)$$

and we can obtain:

$$n = -m \ln \mathbb{E}(z) = m \ln \frac{1}{\mathbb{E}(z)}$$

Analysis of linear counting

- Since $z = U_n/m$, we have:

$$\mathbb{E}(z) = \exp\left(-\frac{n}{m}\right)$$

  and we can obtain:

$$n = -m \ln \mathbb{E}(z) = m \ln \frac{1}{\mathbb{E}(z)}$$

- Plugging-in the observed variables we get:

$$\hat{n} = m \ln \frac{1}{z} \,.$$

Analysis of linear counting

- Bloom filter and linear counting need a priori knowledge of the cardinality being estimated.

Analysis of linear counting

- Bloom filter and linear counting need a priori knowledge of the cardinality being estimated.
- If the filter size is underestimated, then the filter will saturate (be almost entirely full of 1s), and the estimation will be useless.

Analysis of linear counting

- Bloom filter and linear counting need a priori knowledge of the cardinality being estimated.
- If the filter size is underestimated, then the filter will saturate (be almost entirely full of 1s), and the estimation will be useless.
- On the other hand, if the filter is mostly empty then the estimate will be very accurate, but the unused space will be wasted.

Analysis of linear counting

- Bloom filter and linear counting need a priori knowledge of the cardinality being estimated.
- If the filter size is underestimated, then the filter will saturate (be almost entirely full of 1s), and the estimation will be useless.
- On the other hand, if the filter is mostly empty then the estimate will be very accurate, but the unused space will be wasted.
- How to generalize linear counting to k hash functions?

Analysis of linear counting

- Bloom filter and linear counting need a priori knowledge of the cardinality being estimated.
- If the filter size is underestimated, then the filter will saturate (be almost entirely full of 1s), and the estimation will be useless.
- On the other hand, if the filter is mostly empty then the estimate will be very accurate, but the unused space will be wasted.
- How to generalize linear counting to k hash functions?

$$\hat{n} = \frac{m}{k} \ln \frac{1}{z}\,.$$

Distinct count of elements

- The Flajolet-Martin algorithm:

Distinct count of elements

- The Flajolet-Martin algorithm:
  - Also based on hashing.

Distinct count of elements

- The Flajolet-Martin algorithm:
  - Also based on hashing.
  - Needs several repetition to get a good estimate.

Distinct count of elements

- The Flajolet-Martin algorithm:
  - Also based on hashing.
  - Needs several repetition to get a good estimate.
  - The size of the data synopsis is double logarithmic in the largest possible cardinality being estimated.

Distinct count of elements

- The Flajolet-Martin algorithm:
  - Also based on hashing.
  - Needs several repetition to get a good estimate.
  - The size of the data synopsis is double logarithmic in the largest possible cardinality being estimated.
  - The idea: the more different elements in the data, the more different hash-values we shall see:

Distinct count of elements

- The Flajolet-Martin algorithm:
  - Also based on hashing.
  - Needs several repetition to get a good estimate.
  - The size of the data synopsis is double logarithmic in the largest possible cardinality being estimated.
  - The idea: the more different elements in the data, the more different hash-values we shall see:
    - As we see more different hash-values, it becomes more likely that one of these values will be **unusual**.

Distinct count of elements

- The Flajolet-Martin algorithm:
  - Also based on hashing.
  - Needs several repetition to get a good estimate.
  - The size of the data synopsis is double logarithmic in the largest possible cardinality being estimated.
  - The idea: the more different elements in the data, the more different hash-values we shall see:
    - As we see more different hash-values, it becomes more likely that one of these values will be **unusual**.
    - For example, the unusual property can be the value ends in many 0's (although many other options exist).

The Flajolet-Martin algorithm

- Whenever we apply a hash function h to an element a, the bit string $h(a)$ will end in some number of 0's, possibly none.

The Flajolet-Martin algorithm

- Whenever we apply a hash function h to an element a, the bit string $h(a)$ will end in some number of 0's, possibly none.
- Call this number the tail length for a and h.

The Flajolet-Martin algorithm

- Whenever we apply a hash function h to an element a, the bit string $h(a)$ will end in some number of 0's, possibly none.
- Call this number the tail length for a and h.
- Let R be the maximum tail length of any a seen so far in the stream.

The Flajolet-Martin algorithm

- Whenever we apply a hash function h to an element a, the bit string $h(a)$ will end in some number of 0's, possibly none.
- Call this number the tail length for a and h.
- Let R be the maximum tail length of any a seen so far in the stream.
- The FM estimate: We will use 2^R to estimate the number of distinct elements seen in the stream.

The Flajolet-Martin algorithm

- Consider a 32-bit hash function (not perfect, but good for an example):

$$h(a) = a \mod 2^{32}\,.$$

The Flajolet-Martin algorithm

- Consider a 32-bit hash function (not perfect, but good for an example):
$$h(a) = a \mod 2^{32}\,.$$
- For each $h(a)$ we take its binary representation and count the tail length.

The Flajolet-Martin algorithm

- Consider a 32-bit hash function (not perfect, but good for an example):
$$h(a) = a \mod 2^{32}\,.$$
- For each $h(a)$ we take its binary representation and count the tail length.
- The sequence of data is: 2, 16, 5, 9, 11, 192, 5, 150, 96:

The Flajolet-Martin algorithm

- Consider a 32-bit hash function (not perfect, but good for an example):

$$h(a) = a \mod 2^{32} \,.$$

- For each $h(a)$ we take its binary representation and count the tail length.
- The sequence of data is: 2, 16, 5, 9, 11, 192, 5, 150, 96:
  - $h(2) = 10b, r = 1, R = 1, count = 2^1 = 2$

The Flajolet-Martin algorithm

- Consider a 32-bit hash function (not perfect, but good for an example):
$$h(a) = a \mod 2^{32}\,.$$
- For each $h(a)$ we take its binary representation and count the tail length.
- The sequence of data is: $2,\ 16,\ 5,\ 9,\ 11,\ 192,\ 5,\ 150,\ 96$:
  - $h(2) = 10b, r = 1, R = 1, count = 2^1 = 2$
  - $h(16) = 10000b, r = 4, R = 4, count = 2^4 = 16$

The Flajolet-Martin algorithm

- Consider a 32-bit hash function (not perfect, but good for an example):
$$h(a) = a \mod 2^{32}\,.$$
- For each $h(a)$ we take its binary representation and count the tail length.
- The sequence of data is: 2, 16, 5, 9, 11, 192, 5, 150, 96:
  - $h(2) = 10b, r = 1, R = 1, count = 2^1 = 2$
  - $h(16) = 10000b, r = 4, R = 4, count = 2^4 = 16$
  - $h(5) = 101b, r = 0, R = 4, count = 2^4 = 16$

The Flajolet-Martin algorithm

- Consider a 32-bit hash function (not perfect, but good for an example):
$$h(a) = a \mod 2^{32}\,.$$
- For each $h(a)$ we take its binary representation and count the tail length.
- The sequence of data is: $2, 16, 5, 9, 11, 192, 5, 150, 96$:
  - $h(2) = 10b, r = 1, R = 1, count = 2^1 = 2$
  - $h(16) = 10000b, r = 4, R = 4, count = 2^4 = 16$
  - $h(5) = 101b, r = 0, R = 4, count = 2^4 = 16$
  - $h(9) = 1001b, r = 0, R = 4, count = 2^4 = 16$

The Flajolet-Martin algorithm

- Consider a 32-bit hash function (not perfect, but good for an example):
$$h(a) = a \mod 2^{32}\,.$$
- For each $h(a)$ we take its binary representation and count the tail length.
- The sequence of data is: $2,\ 16,\ 5,\ 9,\ 11,\ 192,\ 5,\ 150,\ 96$:
  - $h(2) = 10b, r = 1, R = 1, count = 2^1 = 2$
  - $h(16) = 10000b, r = 4, R = 4, count = 2^4 = 16$
  - $h(5) = 101b, r = 0, R = 4, count = 2^4 = 16$
  - $h(9) = 1001b, r = 0, R = 4, count = 2^4 = 16$
  - $h(11) = 1011b, r = 0, R = 4, count = 2^4 = 16$

The Flajolet-Martin algorithm

- Consider a 32-bit hash function (not perfect, but good for an example):
$$h(a) = a \mod 2^{32}\,.$$
- For each $h(a)$ we take its binary representation and count the tail length.
- The sequence of data is: $2,\ 16,\ 5,\ 9,\ 11,\ 192,\ 5,\ 150,\ 96$:
  - $h(2) = 10b, r = 1, R = 1, count = 2^1 = 2$
  - $h(16) = 10000b, r = 4, R = 4, count = 2^4 = 16$
  - $h(5) = 101b, r = 0, R = 4, count = 2^4 = 16$
  - $h(9) = 1001b, r = 0, R = 4, count = 2^4 = 16$
  - $h(11) = 1011b, r = 0, R = 4, count = 2^4 = 16$
  - $h(193) = 11000001b, r = 0, R = 4, count = 2^4 = 16$

The Flajolet-Martin algorithm

- Consider a 32-bit hash function (not perfect, but good for an example):
$$h(a) = a \mod 2^{32}\,.$$
- For each $h(a)$ we take its binary representation and count the tail length.
- The sequence of data is: 2, 16, 5, 9, 11, 192, 5, 150, 96:
  - $h(2) = 10b, r = 1, R = 1, count = 2^1 = 2$
  - $h(16) = 10000b, r = 4, R = 4, count = 2^4 = 16$
  - $h(5) = 101b, r = 0, R = 4, count = 2^4 = 16$
  - $h(9) = 1001b, r = 0, R = 4, count = 2^4 = 16$
  - $h(11) = 1011b, r = 0, R = 4, count = 2^4 = 16$
  - $h(193) = 11000001b, r = 0, R = 4, count = 2^4 = 16$
  - $h(5) = 101b, r = 0, R = 4, count = 2^4 = 16$

The Flajolet-Martin algorithm

- Consider a 32-bit hash function (not perfect, but good for an example):
$$h(a) = a \mod 2^{32}\,.$$
- For each $h(a)$ we take its binary representation and count the tail length.
- The sequence of data is: 2, 16, 5, 9, 11, 192, 5, 150, 96:
  - $h(2) = 10b, r = 1, R = 1, count = 2^1 = 2$
  - $h(16) = 10000b, r = 4, R = 4, count = 2^4 = 16$
  - $h(5) = 101b, r = 0, R = 4, count = 2^4 = 16$
  - $h(9) = 1001b, r = 0, R = 4, count = 2^4 = 16$
  - $h(11) = 1011b, r = 0, R = 4, count = 2^4 = 16$
  - $h(193) = 11000001b, r = 0, R = 4, count = 2^4 = 16$
  - $h(5) = 101b, r = 0, R = 4, count = 2^4 = 16$
  - $h(150) = 10010110b, r = 1, R = 4, count = 2^4 = 16$

The Flajolet-Martin algorithm

- Consider a 32-bit hash function (not perfect, but good for an example):

$$h(a) = a \mod 2^{32}.$$

- For each $h(a)$ we take its binary representation and count the tail length.
- The sequence of data is: $2, 16, 5, 9, 11, 192, 5, 150, 96$:
  - $h(2) = 10b, r = 1, R = 1, count = 2^1 = 2$
  - $h(16) = 10000b, r = 4, R = 4, count = 2^4 = 16$
  - $h(5) = 101b, r = 0, R = 4, count = 2^4 = 16$
  - $h(9) = 1001b, r = 0, R = 4, count = 2^4 = 16$
  - $h(11) = 1011b, r = 0, R = 4, count = 2^4 = 16$
  - $h(193) = 11000001b, r = 0, R = 4, count = 2^4 = 16$
  - $h(5) = 101b, r = 0, R = 4, count = 2^4 = 16$
  - $h(150) = 10010110b, r = 1, R = 4, count = 2^4 = 16$
  - $h(96) = 1100000b, r = 5, R = 5count = 2^5 = 32$

The Flajolet-Martin algorithm

- Consider a 32-bit hash function (not perfect, but good for an example):
$$h(a) = a \mod 2^{32}\,.$$
- For each $h(a)$ we take its binary representation and count the tail length.
- The sequence of data is: 2, 16, 5, 9, 11, 192, 5, 150, 96:
  - $h(2) = 10b, r = 1, R = 1, count = 2^1 = 2$
  - $h(16) = 10000b, r = 4, R = 4, count = 2^4 = 16$
  - $h(5) = 101b, r = 0, R = 4, count = 2^4 = 16$
  - $h(9) = 1001b, r = 0, R = 4, count = 2^4 = 16$
  - $h(11) = 1011b, r = 0, R = 4, count = 2^4 = 16$
  - $h(193) = 11000001b, r = 0, R = 4, count = 2^4 = 16$
  - $h(5) = 101b, r = 0, R = 4, count = 2^4 = 16$
  - $h(150) = 10010110b, r = 1, R = 4, count = 2^4 = 16$
  - $h(96) = 1100000b, r = 5, R = 5count = 2^5 = 32$
- The estimate is 32

The Flajolet-Martin algorithm[1]

$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\dots$	$\frac{1}{2^{32}}$

- A $BITMAP$ interpretation:

[1] https://en.wikipedia.org/wiki/Flajolet-Martin_algorithm

The Flajolet-Martin algorithm[1]

$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\dots$	$\frac{1}{2^{32}}$

- A $BITMAP$ interpretation:
  - Suppose the probability of mapping item i to table $BITMAP[r]$ is $\frac{1}{2^{r+1}}$

[1] `https://en.wikipedia.org/wiki/Flajolet-Martin_algorithm`

The Flajolet-Martin algorithm[1]

$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\dots$	$\frac{1}{2^{32}}$

- A $BITMAP$ interpretation:
  - Suppose the probability of mapping item i to table $BITMAP[r]$ is $\frac{1}{2^{r+1}}$
  - If there are n distinct items, then $BITMAP[0]$ is accessed approximately $n/2$ times, $BITMAP[1]$ is accessed approximately $n/4$ times and so on.

[1] https://en.wikipedia.org/wiki/Flajolet-Martin_algorithm

The Flajolet-Martin algorithm[1]

$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\dots$	$\frac{1}{2^{32}}$

- A $BITMAP$ interpretation:
  - Suppose the probability of mapping item i to table $BITMAP[r]$ is $\frac{1}{2^{r+1}}$
  - If there are n distinct items, then $BITMAP[0]$ is accessed approximately $n/2$ times, $BITMAP[1]$ is accessed approximately $n/4$ times and so on.
  - If $r \gg \log_2 n$, then $BITMAP[r]$ is almost certainly 0, and if $r \ll \log_2 n$, then $BITMAP[r]$ is almost certainly 1.

[1] `https://en.wikipedia.org/wiki/Flajolet-Martin_algorithm`

The Flajolet-Martin algorithm[1]

$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\dots$	$\frac{1}{2^{32}}$

- A $BITMAP$ interpretation:
  - Suppose the probability of mapping item i to table $BITMAP[r]$ is $\frac{1}{2^{r+1}}$
  - If there are n distinct items, then $BITMAP[0]$ is accessed approximately $n/2$ times, $BITMAP[1]$ is accessed approximately $n/4$ times and so on.
  - If $r \gg \log_2 n$, then $BITMAP[r]$ is almost certainly 0, and if $r \ll \log_2 n$, then $BITMAP[r]$ is almost certainly 1.
  - If $r \simeq \log_2 n$, then $BITMAP[r]$ can be expected to be either 1 or 0.

[1] https://en.wikipedia.org/wiki/Flajolet-Martin_algorithm

Analysis of the Flajolet-Martin algorithm

- We will say that the algorithm is correct if

$$\frac{1}{c}n \leq \hat{n} \leq cn$$

Analysis of the Flajolet-Martin algorithm

- We will say that the algorithm is correct if

$$\frac{1}{c}n \le \hat{n} \le cn$$

- The question is what is the probability that the FM algorithm is correct.

Analysis of the Flajolet-Martin algorithm

- We will say that the algorithm is correct if

$$\frac{1}{c}n \leq \hat{n} \leq cn$$

- The question is what is the probability that the FM algorithm is correct.
- It can be shown that this probability is at least $1 - \frac{3}{c}$.

Combining FM estimates

- Taking the average can be not a good solution (overestimation)

Combining FM estimates

- Taking the average can be not a good solution (overestimation)
- Median is almost ok, but it is always a power of 2.

Combining FM estimates

- Taking the average can be not a good solution (overestimation)
- Median is almost ok, but it is always a power of 2.
- Solution: Combination of the two above

Combining FM estimates

- Taking the average can be not a good solution (overestimation)
- Median is almost ok, but it is always a power of 2.
- Solution: Combination of the two above
  - Group hash function into groups, and take their average,

Combining FM estimates

- Taking the average can be not a good solution (overestimation)
- Median is almost ok, but it is always a power of 2.
- Solution: Combination of the two above
  - Group hash function into groups, and take their average,
  - Then, take the median of the averages

Outline

Summary

- Approximate Query Processing
  - Data synopsis,
  - Sampling,
  - Bloom filters,
  - Counting distinct elements.
  - Many other techniques exist:
    - Histograms
    - Compression via wavelet decomposition
    - Sketches based on random projection

Bibliography

- J. Leskovec, A. Rajaraman, and J. D. Ullman. *Mining of Massive Datasets*. Cambridge University Press, 2014
- Graham Cormode, Minos Garofalakis, Peter J. Haas, and Chris Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends in Databases*, 4(1–3):1–294, 2011
- Micheal Jordan. CS 170: Efficient algorithms and intractable problems: Notes 14. Lecture, October 2005
- Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005
- Yufei Tao. Lecture notes: Flajolet-martin sketch, 2012
- Erik Demaine and Srini Devadas. Lecture notes: Introduction to algorithms (MIT 6.006), 2011