The Power of Random Bits

Randomized Algorithms: Applications & Principles

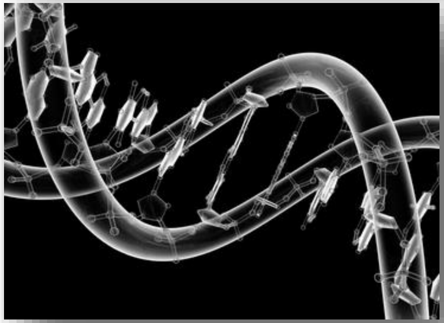
Part I: Hashing and Its Many Applications

Sid Chi-Kin Chau
sidckchau@gmail.com

Why Randomized Algorithms?

- Randomized Algorithms are algorithms that make "random choices" during the execution
- We also make lots of random choices everyday, because
  - Lack of information
  - Convenience and simplicity
  - To diversify risk and try luck!
- These reasons apply to algorithmic design
- But unscrupulous random choices may end with useless results
- Question: ***How do we make <u>smart</u> random choices?***
- In practice:
  Simple random choices often work amazingly well
- In theory:
  Simple maths can justify these simple random choices

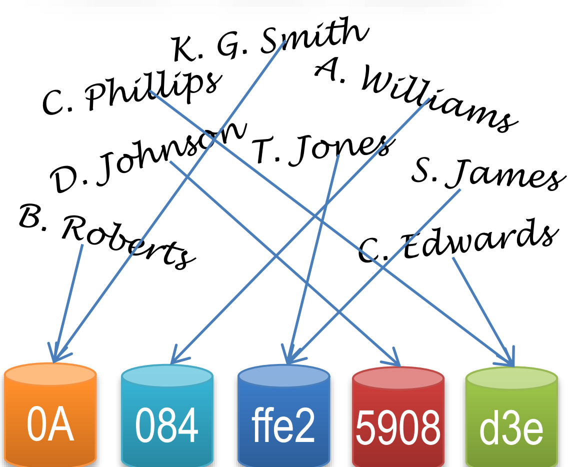
Applications of Randomized Algorithms



- Randomized algorithms are especially useful for applications with
  - Large data set and insufficient memory
  - Limited computational power
  - Uninformed knowledge
  - Minor fault tolerability
- A long list of applications include
  - Information retrieval, databases, bioinformatics (e.g. Google, DNA matching)
  - Networking, telecommunications (e.g. AT&T)
  - Optimization, data prediction, financial trading
  - Artificial intelligence, machine learning
  - Graphics, multi-media, computer games
  - Information security, and a lot more …

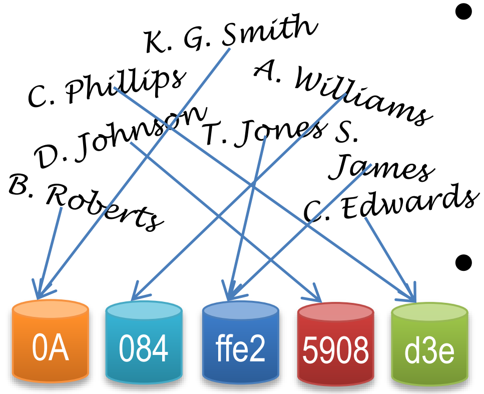
A Key Example: Hashing

- Hashing enables large-scale, fast data processing
  - Expedite the performance of large data/file systems in search engines (Google, Bing)
  - Enable fast response time in small low-power devices (iPhone, iPad)
- Hashing is a *random* sampling/projection of some more complicated data objects (e.g. strings, graphs, functions, sets, data structures)
  - E.g. String-based hashing maps a input string to a shorter hash (string) by a hash function
  - Assuming that a hash function is selected randomly (without a priori knowledge) from a large class of hash functions
  - Hence, when we do not specify the detailed implementation of a particular hash function, the behaviour of hashing appears probabilistic

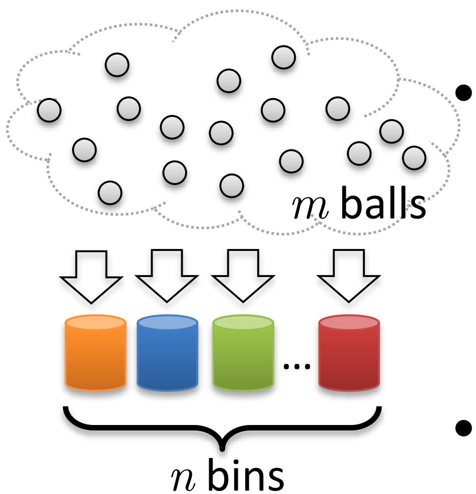


string-based hashing for address book

Balls and Bins Model



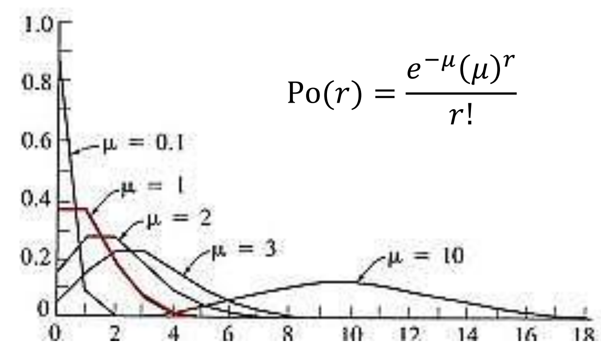
- A generic model for hashing is balls-and-bins model
  - Throw m balls into n bins, such that each ball is uniformly randomly distributed among the bins
- Interpretations of the model
  - Balls = data objects, Bins = hashes
  - (Coupon Collector Problem) Balls = coupons, Bins = types of coupons
  - (Birthday Attack Prob.) Balls = people, Bins = birthdates
- Key questions
  - Efficiency: How many non-empty bins?
  - Performance: What is the maximum number of balls in all the bins?
- Balls-and-bins model is a random model
  - Its behaviour is naturally analysed by probability theory



Poisson Approximation

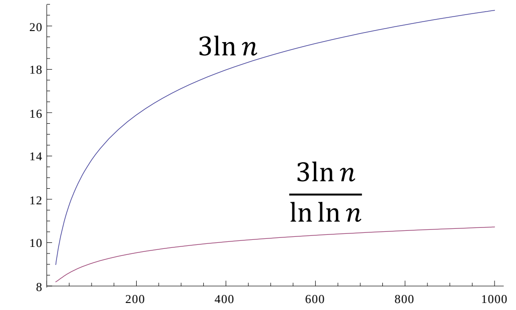
- The probability that bin i has r balls follows binominal distribution
  - $\mathbb{P}\{X_i = r\} = \binom{m}{r}\left(\frac{1}{n}\right)^r\left(1-\frac{1}{n}\right)^{m-r} = \frac{1}{r!}\frac{m(m-1)\ldots(m-r+1)}{n^r}\left(1-\frac{1}{n}\right)^{m-r}$
  - But the expression can be too unwieldy
- When m and n are very large, we can approximate by
  - $\frac{m(m-1)\ldots(m-r+1)}{n^r} \approx \left(\frac{m}{n}\right)^r$ and $\left(1-\frac{1}{n}\right)^{m-r} \approx e^{\frac{-m}{n}}$
  - Hence, $\mathbb{P}\{X_i = r\} \approx \frac{e^{\frac{-m}{n}}\left(\frac{m}{n}\right)^r}{r!}$
- This is known as Poisson distribution $\text{Po}(r) = \frac{e^{-\mu}(\mu)^r}{r!}$
  - The mean of Poisson distribution is $\mu = \frac{m}{n}$
  - The probability of a non-empty bin is

    $\mathbb{P}\{X_i \neq 0\} \approx 1 - \text{Po}(0) = 1 - e^{-\mu}$

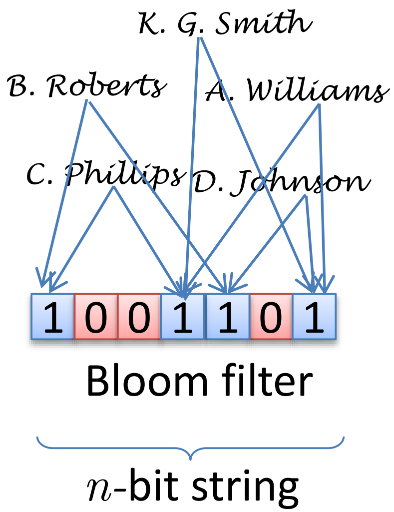


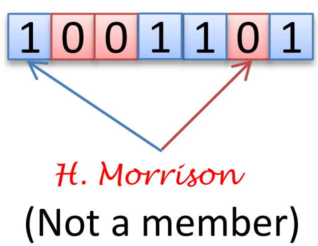
Maximum Load

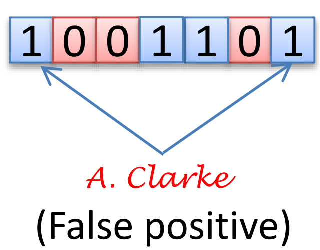
- Recall a well-known technique called Union Bound
  - $\mathbb{P}\{X_1 \geq r \text{ or} \ldots \text{or } X_n \geq r\} \leq \mathbb{P}\{X_1 \geq r\} + \cdots + \mathbb{P}\{X_n \geq r\}$
- The probability that one bin has more than M balls is
  - $\mathbb{P}\{\max_{i=1,\ldots,n} X_i \geq M\} \leq n\, \mathbb{P}\{X_i \geq M\}$
  - If $M > \mu = \frac{m}{n}$, then $\mathbb{P}\{X_i \geq M\} \leq \frac{e^{-\mu}(e\,\mu)^M}{M^M}$
    - (Shown by Chernoff Bound)
  - If $m = n$ (hence $\mu = 1$) and $M = \frac{3\ln n}{\ln\ln n}$, then
    - $\mathbb{P}\left\{X_i \geq \frac{3\ln n}{\ln\ln n}\right\} \leq \frac{e^{-1}e^M}{M^M} = \frac{\left(\frac{e\ln\ln n}{3\ln n}\right)^{\frac{3\ln n}{\ln\ln n}}}{e} \leq \frac{\left(\frac{\ln\ln n}{\ln n}\right)^{\frac{3\ln n}{\ln\ln n}}}{e} = \frac{e^{(\ln\ln\ln n\; - \ln\ln n)\frac{3\ln n}{\ln\ln n}}}{e}$
    - $\mathbb{P}\left\{\max_{i=1,\ldots,n} X_i \geq \frac{3\ln n}{\ln\ln n}\right\} \leq n\, \frac{e^{(\ln\ln\ln n\; - \ln\ln n)\frac{3\ln n}{\ln\ln n}}}{e} \leq \frac{n \cdot n^{-3\left(1 - \frac{\ln\ln\ln n}{\ln\ln n}\right)}}{e} \leq \frac{1}{e\,n}$
    - Therefore , the maximum load is larger than $\frac{3\ln n}{\ln\ln n}$ has a vanishing probability (i.e. , $\mathbb{P}\left\{\max_{i=1,\ldots,n} X_i \geq \frac{3\ln n}{\ln\ln n}\right\} \longrightarrow 0$, as $n \rightarrow \infty$)
    - Or we say that the maximum load is less than $\frac{3\ln n}{\ln\ln n}$ ***with high probability***



Bloom Filter





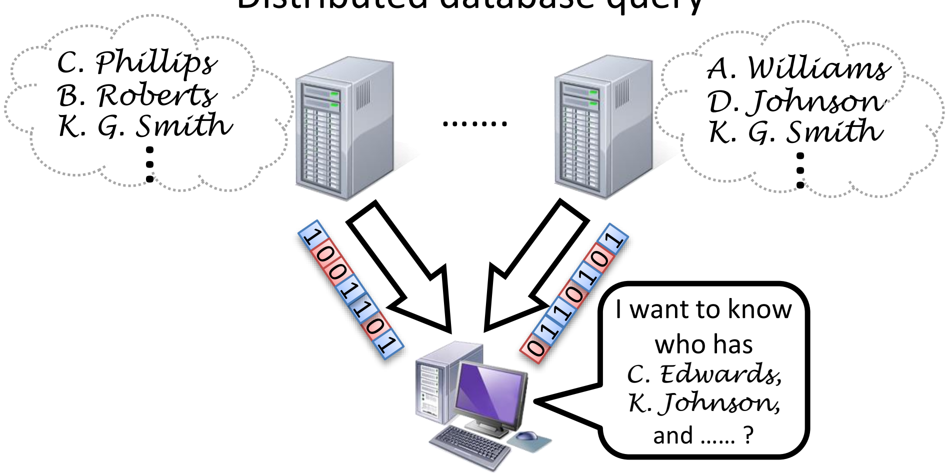


- Instead of hashing from a string, we also consider more complicated objects
- A Bloom filter maps a “set” of strings to a n-bit string
- There are k hash functions, each hash function h_i maps a string to a value in $\{1,..,n\}$
- We initially set the Bloom filter to be an n-bit zero string
- If we include a string s in the Bloom filter, we set the $h_i(s)$-th bit in the Bloom filter to be one for every $i \le k$
- To validate whether a string s is a member of a Bloom filter, we check if the $h_i(s)$-th bit in the Bloom filter is one for every i
- A string belonging to a Bloom filter will be confirmed as a member by the validation (i.e., there is no *false negative*)
- However, it is possible that a string not belonging to a Bloom filter will also be confirmed as a member by the validation (i.e., there can be *false positive*)

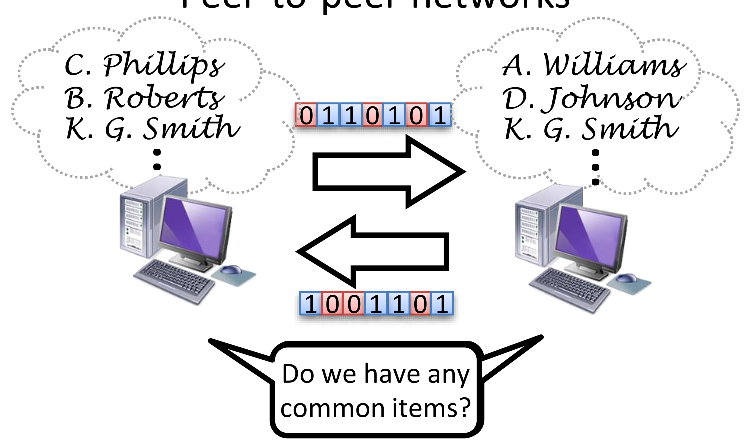
Applications of Bloom Filter

- Bloom filter is a compact representation of a set of strings
- Useful to applications with minor fault tolerance to false positives:
  1) Spell and password checkers with a set unsuitable words
  2) Distributed database query
  3) Content distribution and web cache
  4) Peer-to-peer networks
  5) Packet filtering and measurement of pre-defined flows
  6) Information security, computer graphics, etc.

Distributed database query



Peer-to-peer networks

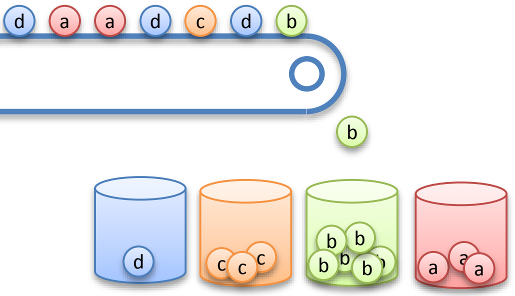


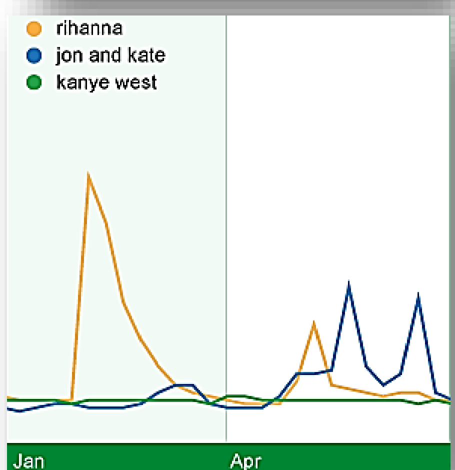
Optimization of Bloom Filter

- We want to minimize the number of false positives
  - There are m strings to be included in an n-bit string Bloom filter
  - There are k hash functions, each hash function h_i maps a string to a value in {1,..,n}
- The probability that a particular bit in the Bloom filter becomes one after including m strings is
  - $1 - \left(1 - \frac{1}{n}\right)^{km} \approx 1 - e^{\frac{-km}{n}}$, assuming that n and m are very large
- Consider validating if a random string is included in the Bloom filter or not
- The probability that the validation succeeds is
  - $\left(1 - \left(1 - \frac{1}{n}\right)^{km}\right)^{k} \approx \left(1 - e^{\frac{-km}{n}}\right)^{k} \triangleq f_{m,n}(k)$
- $f_{m,n}(k)$ is also the probability of a false positive. Hence, we want to minimize $f_{m,n}(k)$ with respect to k
  - $\frac{d \ln f_{m,n}(k)}{dk} = \ln(1 - e^{-km/n}) + \frac{km}{n} \frac{e^{-km/n}}{1-e^{-km/n}}$
  - Hence, $\frac{d \ln f_{m,n}(k)}{dk} = 0 \Rightarrow k = \ln 2 \, \frac{n}{m}$ and $f_{m,n}(k) = \frac{1}{2^k} = (0.612)^{n/m}$
  - For instance, if m =100 and $f_{m,n}(k) = 0.01$, then n = 938 and k = 7

Heavy Hitter Problem

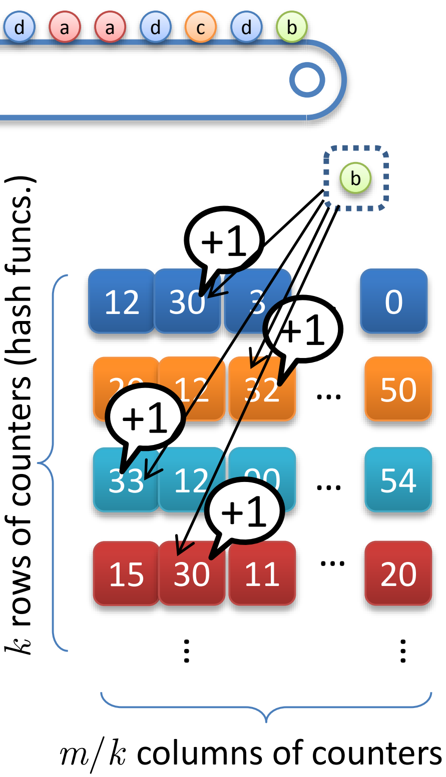
A stream of items with multiple occurrences





- Find the most frequent items in a stream
  - In a network, find the users who consume the most bandwidth by observing a stream of packets
  - In a search engine, find the most queried phrases
  - From the transactions of a supermarket, find the most purchased items
- Heavy hitter problem
  - There is a stream of items with multiple occurrences
  - We want to find the items with the most occurrences, when observing the stream continuously
  - We do not know the number of distinct items in a prior manner
  - We are only allowed to use storage space much less than the number of items in the stream
- Algorithms that process a stream of data with tight space consumption are called *streaming algorithms*

Count-min Sketch



- We use an approach similar to the Bloom filter called count-min sketch
- A sketch is an array of $k \times m/k$ counters, $\{C_{i,j}\}$
- There are k hash functions, each hash function h_i maps an item to a value in $\{1,..,m/k\}$
- Initially set all counters to be zero ($C_{i,j} = 0$)
- When we observe an item s in the stream, increase the $h_i(s)$-th counter ($C_{i,h_i(s)} \leftarrow C_{i,h_i(s)}+1$) for every i
- At the end, we obtain the number of occurrences of an item s by the minimum of all the counters that are mapped by s as $N(s) = \min\{C_{i,h_i(s)}: i = 1, \ldots, k\}$
- $N(s)$ is of course an overestimate of the true number of occurrences, because multiple items can be mapped to the same counter by a hash function
- However, $N(s)$ is not far from the true value

Principle of Count-min Sketch

- Let the true number of occurrences of item s be $T(s)$
- Let the total number of occurrences of all items be T
- The probability that $N(s) \geq T(s) + \varepsilon T$ is at most $\left(\frac{k}{m\varepsilon}\right)^k$, where $\varepsilon \leq 1$
- Let X_t be the random item at time $t = 1, \ldots, T$
- Then the counter $C_{i,h_i(s)} = \sum_{t=1}^{T} \mathbf{1}[h_i(X_t) = h_i(s)]$ and is a random variable, where $\mathbf{1}[\cdot]$ is an indicator function
- We obtain the expected deviation of $C_{i,h_i(s)}$ from $T(s)$ by
  - $\mathbb{E}[C_{i,h_i(s)} - T(s)] = \mathbb{E}[\sum_{t=1:X_t \neq s}^{T} \mathbf{1}[h_i(X_t) = h_i(s)]]$

  $= \sum_{t=1:X_t \neq s}^{T} \mathbb{E}\big[\mathbf{1}[h_i(X_t) = h_i(s)]\big] = \sum_{t=1:X_t \neq s}^{T} \mathbb{P}\{h_i(X_t) = h_i(s)\}$

  $\leq T\,\mathbb{P}\{h_i(X_t) = h_i(s)\} = \frac{kT}{m}$
- Recall *Markov inequality*: $\mathbb{P}\{X \geq x\} \leq \frac{\mathbb{E}[X]}{x}$, for positive x
  - $0 \cdot \mathbb{P}\{X < x\} + x\,\mathbb{P}\{X \geq x\} \leq \sum_y y\,\mathbb{P}\{X = y\} = \mathbb{E}[X]$
- Hence, $\mathbb{P}\{C_{i,h_i(s)} - T(s) \geq \varepsilon T\} \leq \frac{\mathbb{E}\left[C_{i,h_i(s)} - T(s)\right]}{\varepsilon T} = \frac{k}{\varepsilon m}$

Continue in the next slide

Principle of Count-min Sketch

Follow from the last slide

- Since $\mathbb{P}\{C_{i,h_i(s)} - T(s) \geq \varepsilon T\} \leq \frac{k}{\varepsilon m}$, $\mathbb{P}\{\min_{i=1,..,k}\{C_{i,h_i(s)}\} \geq T(s) + \varepsilon T\} \leq \left(\frac{k}{\varepsilon m}\right)^k$
- If we minimize $\left(\frac{k}{\varepsilon m}\right)^k$ with respect to k, then
  - $k = m\,\varepsilon/e$, $\left(\frac{k}{\varepsilon m}\right)^k = e^{-m\varepsilon/e}$, and $\mathbb{P}\{N(s) \geq T(s) + \varepsilon T\} \leq e^{-m\varepsilon/e}$
- If we let $k = \ln\frac{1}{\delta}$ and $m = \ln\frac{1}{\delta} \cdot \frac{e}{\varepsilon}$, then $\mathbb{P}\{N(s) \geq T(s) + \varepsilon T\} \leq \delta$
- Therefore, ε is a tolerance threshold that bounds the deviation of $N(s)$ from count-min sketch, and δ is an error probability that bounds the probability of $N(s)$ deviating for the at most εT
- For example, if we set $\varepsilon = 0.1$ and $\delta = 0.01$, then the number of counters we need is $m = 125$ and the number of hash functions is $k = 5$ (note that both m and k are independent of the number of items in the stream)
- Streaming algorithms can do much more powerful tasks than finding the most frequent items, such as the distributions, correlations and other statistics in a stream of items in a continuous fashion

Summary

- Randomized algorithms are algorithms that make smart random choices during execution
- Hashing is a key example that enables large-scale and fast data processing
- Simple balls-and-bins model can characterize the probabilistic properties of hashing (e.g. maximum load)
- Bloom filter is an example that generates a hash to determine the membership of a set of strings
- Streaming algorithms use a random compact data structure (sketches) to determine the statistics of a stream of items in continuous fashions
- Hashing can be regarded as a random projection from a high dimensional space of data to a low dimensional space of hashes

References

- Main reference: Mitzenmacher and Upfal book, "*Probability and Computing: Randomized Algorithms and Probabilistic Analysis*"
  - Chapter 5.2-5.3: Balls-and-Bins model, Poisson distribution
  - Chapter 5.5.4: Bloom filter
  - Chapter 13.4: Count-min sketch
- Additional references
  - Broder and Mitzenmacher, "*Network Applications of Bloom Filters: A Survey*", Internet Mathematics 1 (4), pp485–509
  - Cormode and Hadjieleftheriou, "*Finding the frequent items in streams of data*", Communications of the ACM, Oct 2009, pp97-105