Finish

Bloom Filters

Anna Karlin
Most slides by Shreya Jayaraman, Luxi Wang, Alex Tsun

Bloom Filters: Motivation

- Large universe of possible data items.
- Hash table is stored on disk or in network, so any lookup is expensive.
- Many (if not most) of the lookups return “Not found”.

Altogether, this is bad. You’re wasting **a lot of time and space** doing lookups for items that aren’t even present.

Examples:

- Google Chrome: wants to warn you if you’re trying to access a malicious URL. Keep hash table of malicious URLs.
- Network routers: want to track source IP addresses of certain packets, .e.g., blocked IP addresses.

Bloom Filters: Motivation

- Probabilistic data structure.
- Close cousins of hash tables.
- Ridiculously space efficient
- To get that, make occasional errors, specifically false positives.

Typical implementation: only 8 bits per element!

Bloom Filters

- Stores information about a set of elements.
- Supports two operations:
  1. **add(x)** - adds x to bloom filter
  2. **contains(x)** - returns true if x in bloom filter, otherwise returns false
     a. If return false, **definitely** not in bloom filter.
     b. If return true, **possibly** in the structure (some false positives).

Bloom Filters: Example

bloom filter t with m = 5 that uses k = 3 hash functions

function INITIALIZE(k,m)
    for $i = 1, \ldots, k$: **do**
        t_i = new bit vector of m 0's

Index →	0	1	2	3	4
t_1	0	0	0	0	0
t_2	0	0	0	0	0
t_3	0	0	0	0	0

Bloom Filters: Example

bloom filter t of length m = 5 that uses k = 3 hash functions

function ADD(x)
  for $i = 1, \ldots, k$: **do**
    $t_i[h_i(x)] = 1$

add("thisisavirus.com")

h_1("thisisavirus.com") → 2
h_2("thisisavirus.com") → 1
h_3("thisisavirus.com") → 4

Index →	**0**	**1**	**2**	**3**	**4**
t_1	0	0	**1**	0	0
t_2	0	**1**	0	0	0
t_3	0	0	0	0	**1**

Bloom Filters: Example

bloom filter t of length m = 5 that uses k = 3 hash functions

contains("thisisavirus.com")

h_1("thisisavirus.com") → 2

h_2("thisisavirus.com") → 1

h_3("thisisavirus.com") → 4

function CONTAINS(x)
    return $t_1[h_1(x)] == 1 \wedge t_2[h_2(x)] == 1 \wedge \cdots \wedge t_k[h_k(x)] == 1$

True   True   True

Index →	0	1	2	3	4
t_1	0	0	1	0	0
t_2	0	1	0	0	0
t_3	0	0	0	0	1

Bloom Filters: Example

bloom filter t of length m = 5 that uses k = 3 hash functions

contains("thisisavirus.com")

h_1("thisisavirus.com") → 2

h_2("thisisavirus.com") → 1

h_3("thisisavirus.com") → 4

function CONTAINS(x)
  return $t_1[h_1(x)] == 1 \wedge t_2[h_2(x)] == 1 \wedge \cdots \wedge t_k[h_k(x)] == 1$

True True True

Since all conditions satisfied, returns True (correctly)

Index →	0	1	2	3	4
t_1	0	0	**1**	0	0
t_2	0	**1**	0	0	0
t_3	0	0	0	0	**1**

Bloom Filters: Example

bloom filter t of length m = 5 that uses k = 3 hash functions

contains("verynormalsite.com")

h_1("verynormalsite.com") → 2

h_2("verynormalsite.com") → 0

h_3("verynormalsite.com") → 4

function CONTAINS(x)
  return $t_1[h_1(x)] == 1 \wedge t_2[h_2(x)] == 1 \wedge \cdots \wedge t_k[h_k(x)] == 1$

True **True** **True**

Since all conditions satisfied, returns True (incorrectly)

Index →	0	1	2	3	4
t_1	0	1	**1**	0	0
t_2	**1**	1	0	0	0
t_3	0	0	0	0	**1**

Bloom Filters: Summary

- An empty bloom filter is an empty k x m bit array with all values initialized to zeros
  - k = number of hash functions
  - m = size of each array in the bloom filter
- add(x) runs in O(k) time
- contains(x) runs in O(k) time
- requires O(km) space (in bits!)
- Probability of false positives from collisions can be reduced by increasing the size of the bloom filter

Bloom Filters: Application

- Google Chrome has a database of malicious URLs, but it takes a long time to query.
- Want an in-browser structure, so needs to be efficient and be space-efficient
- Want it so that can check if a URL is in structure:
  - If return False, then definitely not in the structure (don’t need to do expensive database lookup, website is safe)
  - If return True, the URL may or may not be in the structure. Have to perform expensive lookup in this rare case.

Added $\{y_1, \ldots, y_n\}$ to Bloom filter

FALSE POSITIVE PROBABILITY

new item x shows up.

Assumptions

$\{h_i(y_j), h_i(x)\}$ $j = 1, \ldots, n$, $i = 1 \ldots k$

all mutually independent & uniformly dist'd over corresponding array

$h_i(y_j) \sim \text{Unif}\{0, 1, \ldots, m-1\}$

$$\Pr(\text{false pos on } x)$$

$$= \Pr\left(t_1[\underbrace{h_1(x)}_{a_1}] = 1,\ t_2[\underbrace{h_2(x)}_{a_2}] = 1,\ \ldots,\ t_k[\underbrace{h_k(x)}_{a_k}] = 1\right)$$

$$\underset{\text{indep}}{=} \prod_{i=1}^{k} \underbrace{\Pr(t_i[a_i] = 1)}_{\Pr(\exists\ y_j \in S \text{ s.t. } h_i(y_j) = a_i)}$$

$$= 1 - \underbrace{\Pr(\forall j \ \ h_i(y_j) \neq a_i)}$$

$$\underset{\text{indep}}{=} \prod_{j=1}^{n} \underbrace{\Pr(h_i(y_j) \neq a_i)}_{1 - \frac{1}{m}}$$

$$\underbrace{\qquad\qquad}_{\left(1 - \frac{1}{m}\right)^n}$$

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

$$= \left[1 - \left(1 - \frac{1}{m}\right)^n\right]^k \leq 0.05$$

Comparison with Hash tables - Space

- Google storing 5 million URLs, each URL 40 bytes.
- Bloom filter with k=8 and m = 10,000,000.

Hash Table

$5{,}000{,}000 \times 40$

$= 200$ MB

Bloom Filter

$\frac{10{,}000{,}000 \cdot 8}{8} = 10$ MB

Comparison with Hash tables - Time

- Say avg user visits 100,000 URLs in a year, of which 2,000 are malicious.
- 0.5 seconds to do lookup in the database, 1ms for lookup in Bloom filter.
- Suppose the false positive rate is 2%

Hash Table

$100{,}000 \times 0.5\,\text{sec}$

$= 50{,}000\,\text{sec}$

Bloom Filter

$100{,}000 \times 1\,\text{ms}$

$+ 2000 \times 0.5\,\text{sec}$

$+ 100{,}000 \cdot 0.02 \cdot 0.5\,\text{sec}$

$= 2100\,\text{sec.}$

Bloom Filters: Many Applications

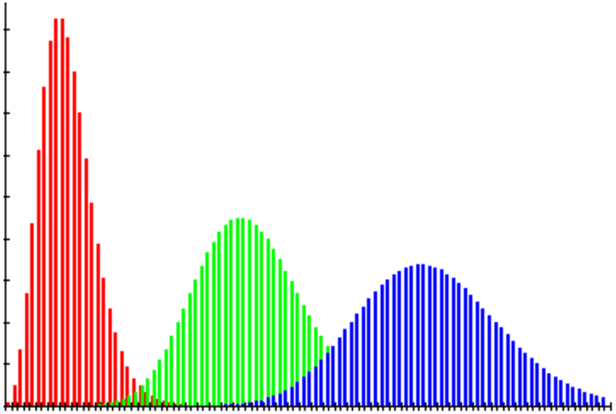
- Any scenario where space and efficiency are important.
- Used a lot in networking
- In distributed systems when want to check consistency of data across different locations, might send a Bloom filter rather than the full set of data being stored.
- Google BigTable uses Bloom filters to reduce the disk lookups for non-existent rows and columns
- Internet routers often use Bloom filters to track blocked IP addresses.
- And on and on…

Bloom Filters typical example…

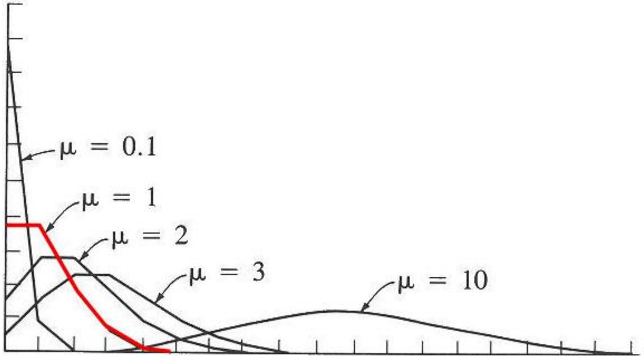
of randomized algorithms and randomized data structures.

- Simple
- Fast
- Efficient
- Elegant
- Useful!

- You’ll be implementing Bloom filters on pset 4. Enjoy!



a zoo of (discrete) random variables



A discrete random variable *X* equally likely to take any (integer) value between integers *a* and *b*, inclusive, is *uniform*.

$\text{Range}(X) = \{a, a+1, \dots, b\}$

Notation: $\text{Unif}(a, b)$

Probability mass function: $p_X(k) = \begin{cases} \frac{1}{b-a+1} & k \in \text{Range}(X) \\ 0 & \text{o.w.} \end{cases}$

Mean: $\frac{a+b}{2}$

Variance:

A discrete random variable X equally likely to take any (integer) value between integers a and b, inclusive, is *uniform*.

Notation: $X \sim \text{Unif}(a,b)$

Probability: $$P(X=i) = \frac{1}{b-a+1}$$

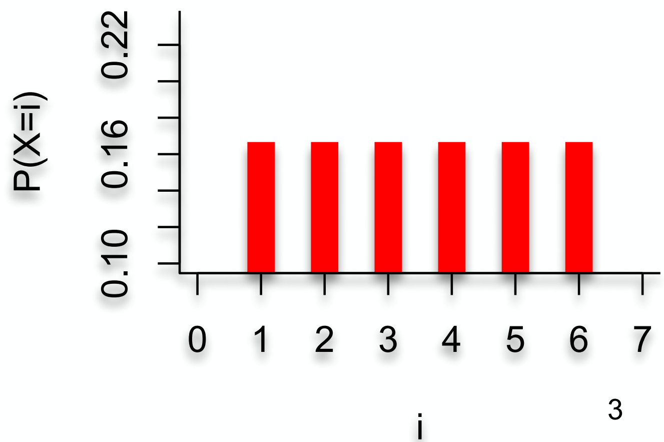
Mean, Variance: $$E[X] = \frac{a+b}{2}, \quad \text{Var}[X] = \frac{(b-a)(b-a+2)}{12}$$

Example: value shown on one roll of a fair die is Unif(1,6):

$P(X=i) = 1/6$

$E[X] = 7/2$

$\text{Var}[X] = 35/12$



indicator

An experiment results in "Success" or "Failure"

X is an *indicator random variable* (1 = success, 0 = failure)

$P(X=1) = p$ and $P(X=0) = 1-p$

X is called a *Bernoulli* random variable: $X \sim \text{Ber}(p)$

Mean:

Variance:

	mean	variance
a)	p	p
b)	p	$1-p$
c)	p	$p(1-p)$
d)	$1-p$	$p(1-p)$

$\text{Var}(X) = \underline{E(X^2)} - [\underline{E(X)}]^2 = p - p^2$

$E(X^2) = p$

$E(X) = 1 \cdot p + 0 \cdot (1-p)$

An experiment results in "Success" or "Failure"

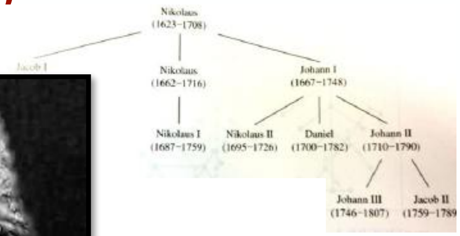
X is an *indicator random variable* (1 = success, 0 = failure)

$P(X=1) = p$ and $P(X=0) = 1-p$

X is called a *Bernoulli* random variable: $X \sim Ber(p)$

$E[X] = E[X^2] = p$

$Var(X) = E[X^2] - (E[X])^2 = p - p^2 = p(1-p)$



Examples:

- coin flip
- random binary digit
- whether a disk drive crashed

Jacob (aka James, Jacques) Bernoulli, 1654 – 1705

Consider n independent random variables $Y_i \sim \mathrm{Ber}(p)$

indep.

$X = \Sigma_i Y_i$ is the number of successes in n trials

X is a *Binomial* random variable: $X \sim \mathrm{Bin}(n,p)$

p.m.f.?

$p_X(k) = \Pr(X=k) =$

$p_X(k)$

a) $p^k (1-p)^{n-k}$

b) np

c) $\binom{n}{k} p^k (1-p)^{n-k}$

d) $\binom{n}{n-k} p^k (1-p)^{n-k}$

Examples

\# of heads in n coin flips

\# of 1's in a randomly generated length n bit string

\# of disk drive crashes in a 1000 computer cluster

\# bit errors in file written to disk

\# of typos in a book

\# of elements in particular bucket of large hash table

\# of server crashes per day in giant data center

Consider n independent random variables $Y_i \sim \text{Ber}(p)$

$X = \Sigma_i Y_i$ is the number of successes in n trials

X is a *Binomial* random variable: $X \sim \text{Bin}(n,p)$

$X = Y_1 + Y_2 + \dots + Y_n$

$E(Y_i) = p$

$\text{Var}(Y_i) = p(1-p)$

Probability mass function:

$$p_X(k) = \binom{n}{k} p^k (1-p)^{n-k}$$

$k = 0, 1, \dots, n$

Mean:

	mean	variance
a)	p	$p(1-p)$
b)	np	$np(1-p)$
c)	np	np^2
d)	np	n^2p

Variance:

mean, variance of the binomial (II)

If $Y_1, Y_2, \ldots, Y_n \sim \text{Ber}(p)$ and independent,

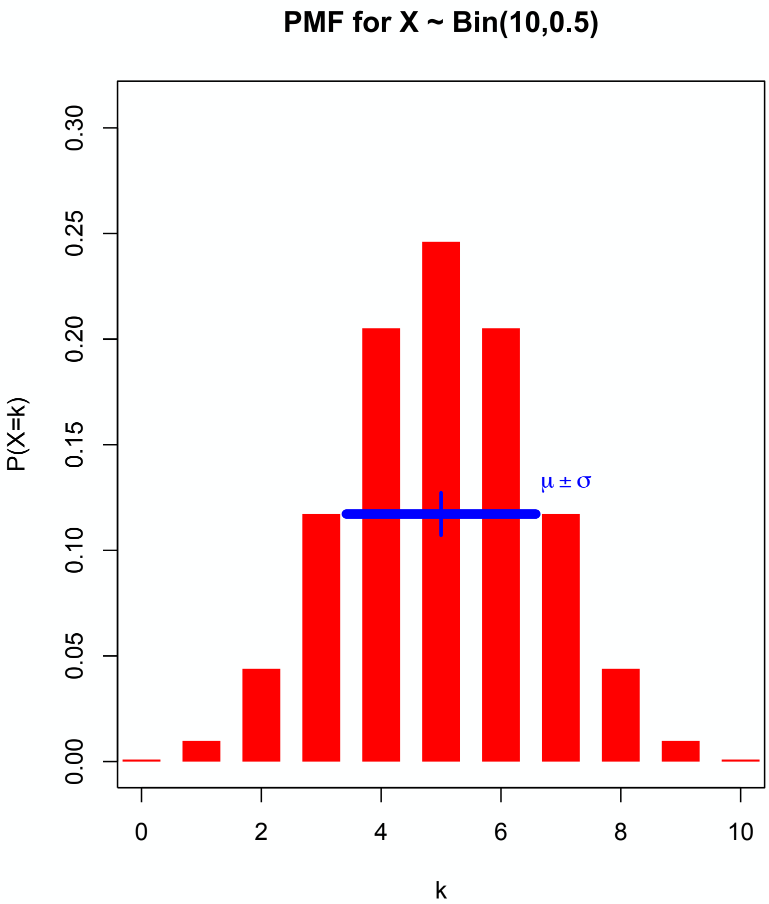
then $X = \sum_{i=1}^{n} Y_i \sim \text{Bin}(n, p)$.

$$\boxed{E[X] = np}$$

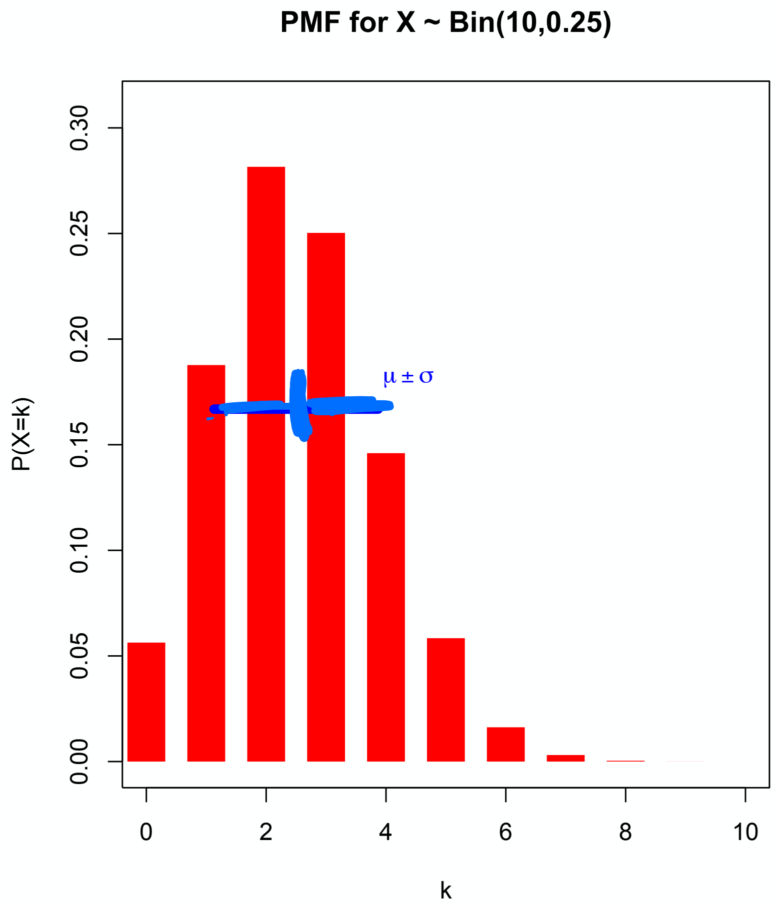
$$E[X] = E\left[\sum_{i=1}^{n} Y_i\right] = \sum_{i=1}^{n} E\left[Y_i\right] = nE[Y_1] = np$$

$$\boxed{\text{Var}[X] = np(1-p)}$$

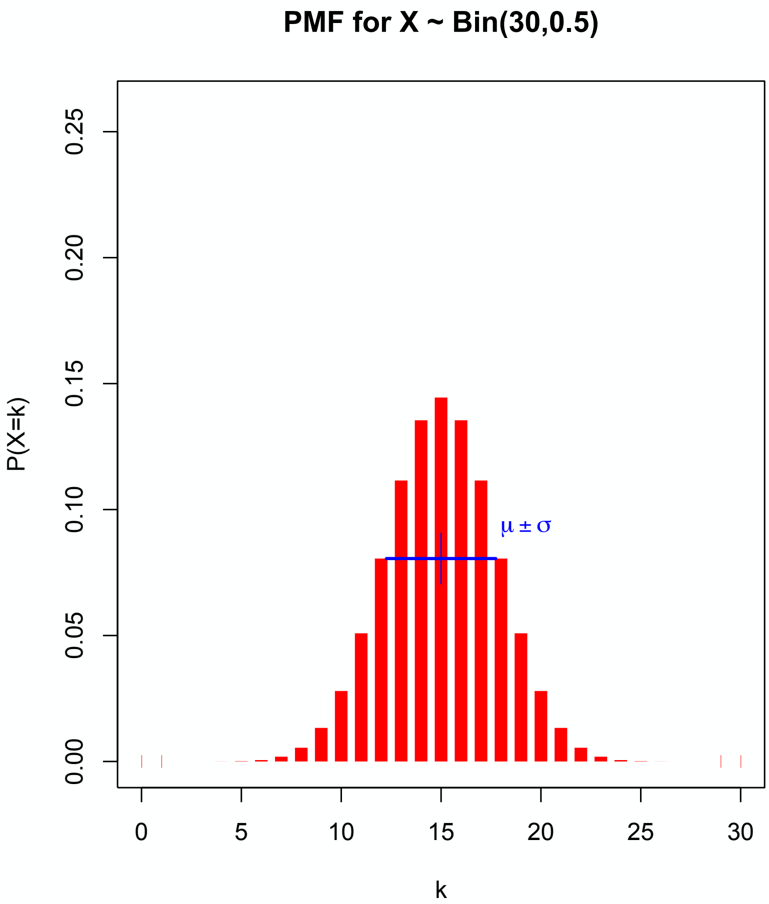
$$\text{Var}[X] = \text{Var}\left[\sum_{i=1}^{n} Y_i\right] = \sum_{i=1}^{n} \text{Var}\left[Y_i\right] = n\text{Var}[Y_1] = np(1-p)$$

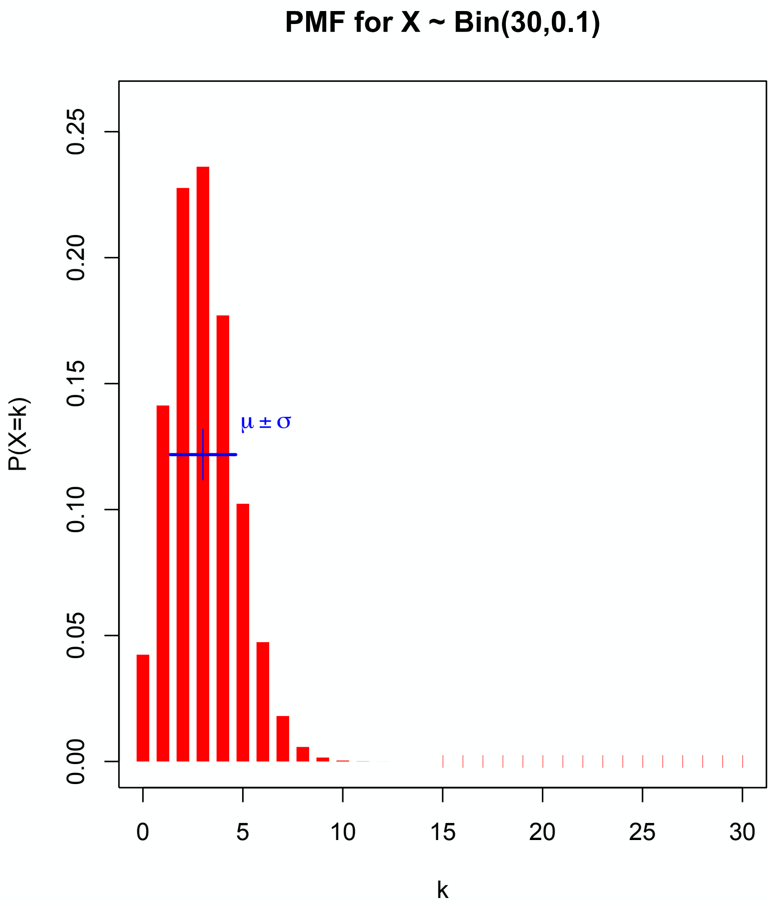


PMF for X ~ Bin(10,0.5)
P(X=k)
k
μ ± σ
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0
2
4
6
8
10



PMF for X ~ Bin(10,0.25)
P(X=k)
k
μ ± σ
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0
2
4
6
8
10


PMF for X ~ Bin(30,0.5)
P(X=k)
0.00
0.05
0.10
0.15
0.20
0.25
0
5
10
15
20
25
30
k
μ ± σ


PMF for X ~ Bin(30,0.1)
P(X=k)
0.00
0.05
0.10
0.15
0.20
0.25
0
5
10
15
20
25
30
k
μ ± σ

Sending a bit string over the network

- $n = 4$ bits sent, each corrupted with probability 0.1
- X = # of corrupted bits, $X \sim \text{Bin}(4, 0.1)$

In real networks, large bit strings (length $n \approx 10^4$)

Corruption probability is very small: $p \approx 10^{-6}$

$X \sim \text{Bin}(10^4, 10^{-6})$ is unwieldy to compute

n, p

n·p small.

Extreme n and p values arise in many cases

- # bit errors in file written to disk
- # of typos in a book
- # of elements in particular bucket of large hash table
- # of server crashes per day in giant data center

heads up

In a series $X_1, X_2, \ldots$ of Bernoulli trials with success probability p, let Y be the index of the first success, i.e.,

$$X_1 = X_2 = \ldots = X_{Y-1} = 0 \;\&\; X_Y = 1$$

trials up till & including first success

Then Y is a *geometric* random variable with parameter p.

$Y \sim Geo(p)$

$Geom(p)$

Examples:

Number of coin flips until first head

Number of blind guesses on SAT until I get one right

Number of darts thrown until you hit a bullseye

Number of random probes into hash table until empty slot

Number of wild guesses at a password until you hit it

$Pr(Y=k) = Pr(\text{first success is on } k^{th} \text{ trial})$

Probability mass function: $= p_X(k)$

$p_X(k)$

a) $(1-p)^k p$

$\boxed{\text{b) } (1-p)^{k-1} p}$

c) $(1-p)^k$

d) p^k

Mean: $\frac{1}{p}$

Variance:

In a series $X_1, X_2, \ldots$ of Bernoulli trials with success probability p, let Y be the index of the first success, i.e.,

$$X_1 = X_2 = \ldots = X_{Y-1} = 0 \;\&\; X_Y = 1$$

Then Y is a *geometric* random variable with parameter p.

Examples:

Number of coin flips until first head

Number of blind guesses on SAT until I get one right

Number of darts thrown until you hit a bullseye

Number of random probes into hash table until empty slot

Number of wild guesses at a password until you hit it

$P(Y=k) = (1-p)^{k-1}p;$

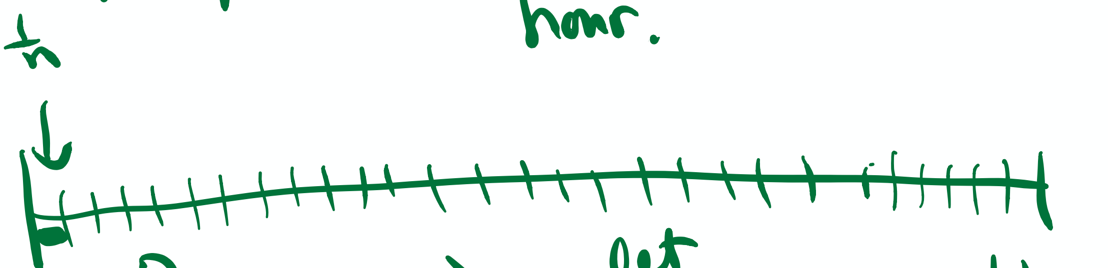
Mean $\boxed{1/p;}$ Variance $\boxed{(1-p)/p^2}$

Poisson motivation

Model: # events that occur in an hour

- rate 3 events per hour.
- exp # events that occur in interval of t hours
  $3t$
- occurence of events in disjoint time intervals is indep.

Y # events that occur in 1 hour.



Bin(n, p)

let $n \to \infty$

$p = ?$

a) $\frac{3}{n}$

b) $3n$

c) 3

d) $\frac{3}{60}$