

CSE 312

Foundations of Computing II

**Lecture 11: Bloom Filters (continued)
& Zoo of Random Variables I**

Review Variance – Properties

Definition. The **variance** of a (discrete) RV X is

$$\text{Var}(X) = \mathbb{E}[(X - \mathbb{E}[X])^2] = \sum_x p_X(x) \cdot (x - \mathbb{E}[X])^2$$

Theorem. For any $a, b \in \mathbb{R}$, $\text{Var}(a \cdot X + b) = a^2 \cdot \text{Var}(X)$

Theorem. $\text{Var}(X) = \mathbb{E}[X^2] - \mathbb{E}[X]^2$

Review Important Facts about Independent Random Variables

Theorem. If X, Y independent, $\mathbb{E}[X \cdot Y] = \mathbb{E}[X] \cdot \mathbb{E}[Y]$

Theorem. If X, Y independent, $\text{Var}(X + Y) = \text{Var}(X) + \text{Var}(Y)$

Corollary. If X_1, X_2, \dots, X_n mutually independent,

$$\text{Var}\left(\sum_{i=1}^n X_i\right) = \sum_i \text{Var}(X_i)$$

Agenda

- Bloom Filters Example & Analysis ◀
- Zoo of Discrete RVs
 - Uniform Random Variables
 - Bernoulli Random Variables
 - Binomial Random Variables
 - Applications

Basic Problem

Problem: Store a subset S of a large set U .

Example. U = set of 128 bit strings
 S = subset of strings of interest

$$|U| \approx 2^{128}$$

$$|S| \approx 1000$$

Two goals:

1. **Very fast** (ideally constant time) answers to queries “Is $x \in S$?” for any $x \in U$.
2. **Minimal storage** requirements.

Naïve Solution I – Constant Time

Idea: Represent S as an array A with 2^{128} entries.

$$A[x] = \begin{cases} 1 & \text{if } x \in S \\ 0 & \text{if } x \notin S \end{cases}$$

$S = \{0, 2, \dots, K\}$

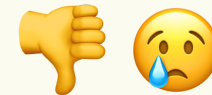


0	1	2	...	K	...		
1	0	1	0	1	...	0	0

Membership test: To check $x \in S$ just check whether $A[x] = 1$.

→ constant time! 👍 😊

Storage: Require storing 2^{128} bits, even for small S .



Naïve Solution II – Small Storage

Idea: Represent S as a list with $|S|$ entries.



Storage: Grows with $|S|$ only 👍 😊

Membership test: Check $x \in S$ requires time linear in $|S|$

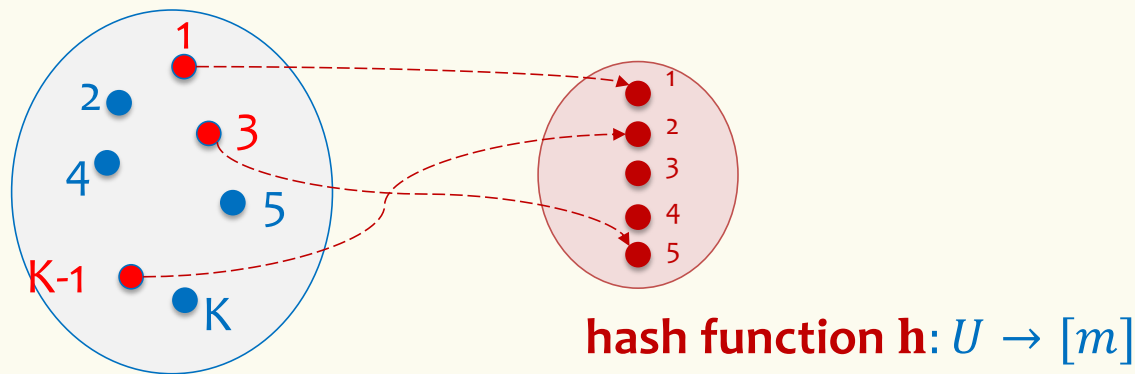
(Can be made logarithmic by using a tree) 👎 😓

Hash Table

Idea: Map elements in S into an array A of size m using a hash function h

Membership test: To check $x \in S$ just check whether $A[h(x)] = x$

Storage: m elements (size of array)

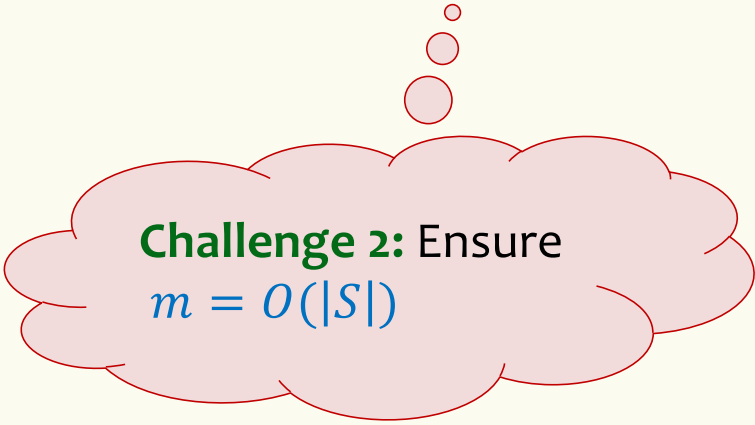


Hash Table

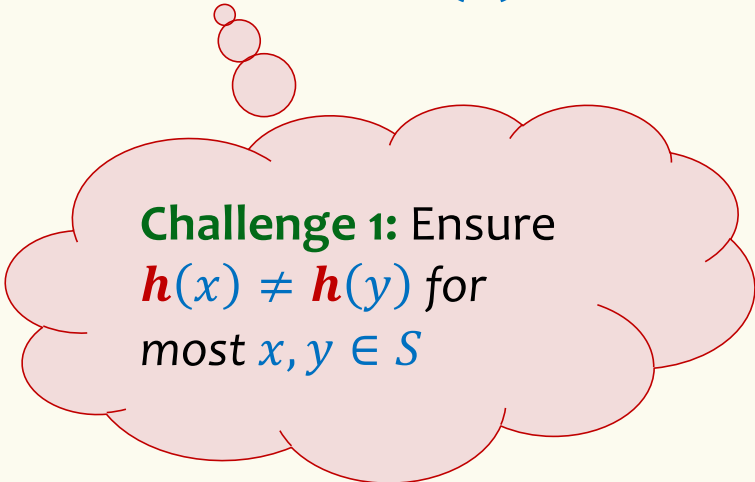
Idea: Map elements in S into an array A of size m using a hash function h

Membership test: To check $x \in S$ just check whether $A[h(x)] = x$

Storage: m elements (size of array)



Challenge 2: Ensure
 $m = O(|S|)$

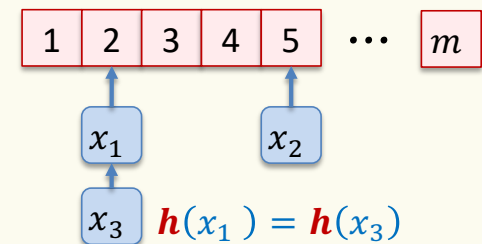


Challenge 1: Ensure
 $h(x) \neq h(y)$ for
most $x, y \in S$

Hashing: collisions

Collisions occur when $h(x) = h(y)$ for some distinct $x, y \in S$, i.e., two elements of set map to the same location

- Common solution: chaining – at each location (bucket) in the table, keep linked list of all elements that hash there.



Good hash functions to keep collisions low

- The hash function h is good iff it
 - distributes elements uniformly across the m array locations so that
 - pairs of elements are mapped independently

“Universal Hash Functions” – see CSE 332

Hashing: summary

Hash Tables

- They store the data itself
- With a good hash function, the data is well distributed in the table and lookup times are small.
- However, they need at least as much space as all the data being stored, i.e., $m = \Omega(|S|)$

In some cases, $|S|$ is huge, or not known a-priori ...

Can we do better!?

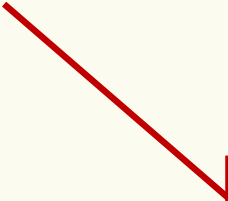


**Bloom Filters
to the rescue**

(Named after Burton Howard Bloom)

Bloom Filters

- Stores information about a set of elements $S \subseteq U$.
- Supports two operations:
 1. **add**(x) - adds $x \in U$ to the set S
 2. **contains**(x) – ideally: true if $x \in S$, false otherwise



Possible *false positives*

Combine with fallback mechanism – can distinguish false positives from true positives with extra cost

Bloom Filters – Ingredients

Basic data structure is a $k \times m$ binary array
“the Bloom filter”

- k rows t_1, \dots, t_k , each of size m
- Think of each row as an m -bit vector

k different hash functions $\mathbf{h}_1, \dots, \mathbf{h}_k: U \rightarrow [m]$

t_1	1	0	1	0	0
t_2	0	1	0	0	1
t_3	1	0	0	1	0

Bloom Filters – Three operations

- Set up Bloom filter for $S = \emptyset$

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

- Update Bloom filter for $S \leftarrow S \cup \{x\}$

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

- Check if $x \in S$

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


Bloom Filters - Initialization

Number of
hash
functions

Size of array
associated to
each hash
function.

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

for each hash
function, initialize
an empty bit
vector of size m

Bloom Filters: Example

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

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

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: Add

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

for each hash
function \mathbf{h}_i

Index into i -th bit-vector, at index produced
by hash function and set to 1

$\mathbf{h}_i(x) \rightarrow$ result of hash
function \mathbf{h}_i on x

Bloom Filters: Example

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

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

add("thisisavirus.com")

$h_1(\text{"thisisavirus.com"}) \rightarrow 2$

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  $i = 1, \dots, k$ : do  
     $t_i[h_i(x)] = 1$ 
```

add("thisisavirus.com")

h_1 ("thisisavirus.com") \rightarrow 2

h_2 ("thisisavirus.com") \rightarrow 1

Index \rightarrow	0	1	2	3	4
t_1	0	0	1	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  $i = 1, \dots, k$ : do  
     $t_i[h_i(x)] = 1$ 
```

add("thisisavirus.com")

h_1 ("thisisavirus.com") \rightarrow 2

h_2 ("thisisavirus.com") \rightarrow 1

h_3 ("thisisavirus.com") \rightarrow 4

Index \rightarrow	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	0

Bloom Filters: Example

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

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

add("thisisavirus.com")

h_1 ("thisisavirus.com") \rightarrow 2

h_2 ("thisisavirus.com") \rightarrow 1

h_3 ("thisisavirus.com") \rightarrow 4

Index \rightarrow	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: Contains

```
function CONTAINS( $x$ )
```

```
return  $t_1[h_1(x)] == 1 \wedge t_2[h_2(x)] == 1 \wedge \dots \wedge t_k[h_k(x)] == 1$ 
```

Returns True if the bit vector t_i for each hash function has bit 1 at index determined by $h_i(x)$,

Returns False otherwise

Bloom Filters: Example

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

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

contains("thisisavirus.com")

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

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

True

contains("thisisavirus.com")

$h_1(\text{"thisisavirus.com"}) \rightarrow 2$

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

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

True

True

contains("thisisavirus.com")

$h_1(\text{"thisisavirus.com"}) \rightarrow 2$

$h_2(\text{"thisisavirus.com"}) \rightarrow 1$

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

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

True

True

True

contains("thisisavirus.com")

h_1 ("thisisavirus.com") \rightarrow 2

h_2 ("thisisavirus.com") \rightarrow 1

h_3 ("thisisavirus.com") \rightarrow 4

Index \rightarrow	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

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

True

True

True

contains("thisisavirus.com")

h_1 ("thisisavirus.com") \rightarrow 2

h_2 ("thisisavirus.com") \rightarrow 1

h_3 ("thisisavirus.com") \rightarrow 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

Since all conditions satisfied, returns **True** (correctly)

Bloom Filters: False Positives

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

add("totallynotsuspicious.com")

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

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: False Positives

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

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

add("totallynotsuspicious.com")

$h_1(\text{"totallynotsuspicious.com"}) \rightarrow 1$

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: False Positives

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

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

add("totallynotsuspicious.com")

h_1 ("totallynotsuspicious.com") \rightarrow 1

h_2 ("totallynotsuspicious.com") \rightarrow 0

Index \rightarrow	0	1	2	3	4
t_1	0	1	1	0	0
t_2	0	1	0	0	0
t_3	0	0	0	0	1

Bloom Filters: False Positives

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

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

add("totallynotsuspicious.com")

h_1 ("totallynotsuspicious.com") \rightarrow 1

h_2 ("totallynotsuspicious.com") \rightarrow 0

h_3 ("totallynotsuspicious.com") \rightarrow 4

Index \rightarrow	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: False Positives

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

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

add("totallynotsuspicious.com")

h_1 ("totallynotsuspicious.com") \rightarrow 1

h_2 ("totallynotsuspicious.com") \rightarrow 0

h_3 ("totallynotsuspicious.com") \rightarrow 4

Index \rightarrow	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: False Positives

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

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

contains("verynormalsite.com")

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: False Positives

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

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

True

contains("verynormalsite.com")

$h_1(\text{"verynormalsite.com"}) \rightarrow 2$

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: False Positives

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

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

True

True

contains("verynormalsite.com")

$h_1(\text{"verynormalsite.com"}) \rightarrow 2$

$h_2(\text{"verynormalsite.com"}) \rightarrow 0$

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: False Positives

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

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

True

True

True

contains("verynormalsite.com")

h_1 ("verynormalsite.com") \rightarrow 2

h_2 ("verynormalsite.com") \rightarrow 0

h_3 ("verynormalsite.com") \rightarrow 4

Index \rightarrow	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: False Positives

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

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

True

True

True

contains("verynormalsite.com")

h_1 ("verynormalsite.com") \rightarrow 2

h_2 ("verynormalsite.com") \rightarrow 0

h_3 ("verynormalsite.com") \rightarrow 4

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

Since all conditions satisfied, returns **True** (incorrectly)

Analysis: False positive probability

Question: For an element $x \in U$, what is the probability that **contains**(x) returns true if **add**(x) was never executed before?

Probability over what?! Over the choice of the h_1, \dots, h_k

Assumptions for the analysis (somewhat stronger than for ordinary hashing):

- Each $h_i(x)$ is uniformly distributed in $[m]$ for all x and i
- Hash function outputs for each h_i are mutually independent (not just in pairs)
- Different hash functions are independent of each other

False positive probability – Events

Assume we perform **add**(x_1), ..., **add**(x_n)
+ **contains**(x) for $x \notin \{x_1, \dots, x_n\}$

Event E_i holds iff $\mathbf{h}_i(x) \in \{\mathbf{h}_i(x_1), \dots, \mathbf{h}_i(x_n)\}$

$$P(\text{false positive}) = P(E_1 \cap E_2 \cap \dots \cap E_k) = \prod_{i=1}^k P(E_i)$$

$\mathbf{h}_1, \dots, \mathbf{h}_k$ independent



False positive probability – Events

Event E_i holds iff $\mathbf{h}_i(x) \in \{\mathbf{h}_i(x_1), \dots, \mathbf{h}_i(x_n)\}$

Event E_i^c holds iff $\mathbf{h}_i(x) \neq \mathbf{h}_i(x_1)$ and ... and $\mathbf{h}_i(x) \neq \mathbf{h}_i(x_n)$

$$P(E_i^c) = \sum_{z=1}^m P(\mathbf{h}_i(x) = z) \cdot P(E_i^c \mid \mathbf{h}_i(x) = z)$$

LTP



False positive probability – Events

Event E_i^c holds iff $\mathbf{h}_i(x) \neq \mathbf{h}_i(x_1)$ and ...
and $\mathbf{h}_i(x) \neq \mathbf{h}_i(x_n)$

$$P(E_i^c | \mathbf{h}_i(x) = z) = P(\mathbf{h}_i(x_1) \neq z, \dots, \mathbf{h}_i(x_n) \neq z | \mathbf{h}_i(x) = z)$$

Independence of values
of \mathbf{h}_i on different inputs

$$= P(\mathbf{h}_i(x_1) \neq z, \dots, \mathbf{h}_i(x_n) \neq z)$$

$$= \prod_{j=1}^n P(\mathbf{h}_i(x_j) \neq z)$$

Outputs of \mathbf{h}_i uniformly spread

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


$$\longrightarrow P(E_i^c) = \sum_{z=1}^m P(\mathbf{h}_i(x) = z) \cdot P(E_i^c | \mathbf{h}_i(x) = z) = \left(1 - \frac{1}{m}\right)^n$$

False positive probability – Events

Event E_i holds iff $\mathbf{h}_i(x) \in \{\mathbf{h}_i(x_1), \dots, \mathbf{h}_i(x_n)\}$

Event E_i^c holds iff $\mathbf{h}_i(x) \neq \mathbf{h}_i(x_1)$ and ... and $\mathbf{h}_i(x) \neq \mathbf{h}_i(x_n)$

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


$$\text{FPR} = \prod_{i=1}^k (1 - P(E_i^c)) = \left(1 - \left(1 - \frac{1}{m}\right)^n\right)^k$$

False Positivity Rate – Example

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

e.g., $n = 5,000,000$

$k = 30$

$m = 2,500,000$



FPR = 1.28%

Comparison with Hash Tables - Space

- Google storing 5 million URLs, each URL 40 bytes.
- Bloom filter with $k = 30$ and $m = 2,500,000$

Hash Table

(optimistic)

$$5,000,000 \times 40B = 200MB$$

Bloom Filter

$$2,500,000 \times 30 = 75,000,000 \text{ bits}$$

$$< 10 \text{ MB}$$

Time

- Say avg user visits 102,000 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 3%

$$1\text{ms} + \frac{100000 \times 0.03 \times 500\text{ms} + 2000 \times 500\text{ms}}{102000} \approx 25.51\text{ms}$$

Bloom filter lookup (points to 1ms)

false positives (points to $100000 \times 0.03 \times 500\text{ms}$)

total URLs (points to 102000)

malicious URLs (points to $2000 \times 500\text{ms}$)

0.5 seconds DB lookup (points to 500ms in both terms of the fraction)

Bloom Filters typical of...

... randomized algorithms and randomized data structures.

- **Simple**
- **Fast**
- **Efficient**
- **Elegant**
- **Useful!**

Brain Break



Motivation for “Named” Random Variables

Random Variables that show up all over the place.

- Easily solve a problem by recognizing it’s a special case of one of these random variables.

Each RV introduced today will show:

- A general situation it models
- Its name and parameters
- Its PMF, Expectation, and Variance
- Example scenarios you can use it

Welcome to the Zoo! (Preview)

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

$$P(X = k) = \frac{1}{b - a + 1}$$
$$\mathbb{E}[X] = \frac{a + b}{2}$$
$$\text{Var}(X) = \frac{(b - a)(b - a + 2)}{12}$$

$X \sim \text{Ber}(p)$

$$P(X = 1) = p, P(X = 0) = 1 - p$$
$$\mathbb{E}[X] = p$$
$$\text{Var}(X) = p(1 - p)$$

$X \sim \text{Bin}(n, p)$

$$P(X = k) = \binom{n}{k} p^k (1 - p)^{n-k}$$
$$\mathbb{E}[X] = np$$
$$\text{Var}(X) = np(1 - p)$$

$X \sim \text{Geo}(p)$

$$P(X = k) = (1 - p)^{k-1} p$$
$$\mathbb{E}[X] = \frac{1}{p}$$
$$\text{Var}(X) = \frac{1 - p}{p^2}$$

$X \sim \text{NegBin}(r, p)$

$$P(X = k) = \binom{k-1}{r-1} p^r (1 - p)^{k-r}$$
$$\mathbb{E}[X] = \frac{r}{p}$$
$$\text{Var}(X) = \frac{r(1 - p)}{p^2}$$

$X \sim \text{HypGeo}(N, K, n)$

$$P(X = k) = \frac{\binom{K}{k} \binom{N-K}{n-k}}{\binom{N}{n}}$$
$$\mathbb{E}[X] = n \frac{K}{N}$$
$$\text{Var}(X) = n \frac{K(N-K)(N-n)}{N^2(N-1)}$$

Agenda

- Bloom Filters Example & Analysis
- Zoo of Discrete RVs, Part I
 - Uniform Random Variables ◀
 - Bernoulli Random Variables
 - Binomial Random Variables
 - Applications

Discrete Uniform Random Variables

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

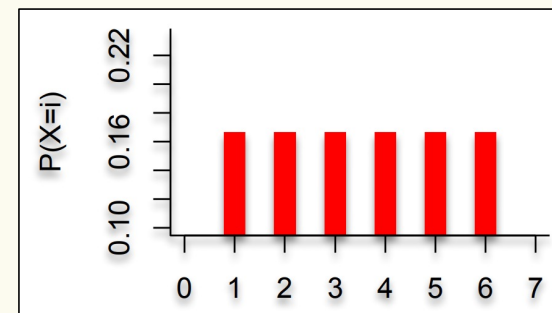
Notation:

PMF:

Expectation:

Variance:

Example: value shown on one roll of a fair die



Discrete Uniform Random Variables

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

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

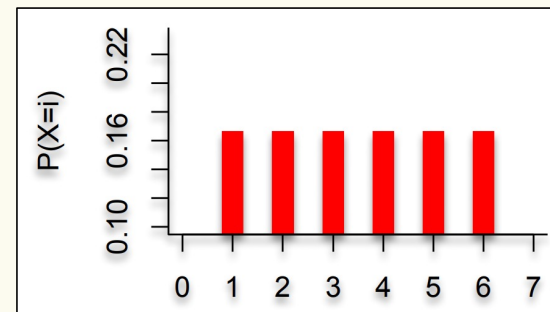
PMF: $P(X = i) = \frac{1}{b - a + 1}$

Expectation: $\mathbb{E}[X] = \frac{a+b}{2}$

Variance: $\text{Var}(X) = \frac{(b-a)(b-a+1)}{12}$

Example: value shown on one roll of a fair die is $\text{Unif}(1,6)$:

- $P(X = i) = 1/6$
- $\mathbb{E}[X] = 7/2$
- $\text{Var}(X) = 35/12$



Agenda

- Bloom Filters Example & Analysis
- Zoo of Discrete RVs, Part I
 - Uniform Random Variables
 - Bernoulli Random Variables ◀
 - Binomial Random Variables
 - Applications

Bernoulli Random Variables

A random variable X that takes value 1 (“Success”) with probability p , and 0 (“Failure”) otherwise. X is called a **Bernoulli random variable**.

Notation: $X \sim \text{Ber}(p)$

PMF: $P(X = 1) = p, P(X = 0) = 1 - p$

Expectation:

Variance:

Poll:

pollev.com/rachel312

Mean Variance

- | | | |
|----|-----|------------|
| A. | p | p |
| B. | p | $1 - p$ |
| C. | p | $p(1 - p)$ |
| D. | p | p^2 |

Bernoulli Random Variables

A random variable X that takes value **1** (“Success”) with probability p , and **0** (“Failure”) otherwise. X is called a **Bernoulli random variable**.

Notation: $X \sim \text{Ber}(p)$

PMF: $P(X = 1) = p, P(X = 0) = 1 - p$

Expectation: $\mathbb{E}[X] = p$ Note: $\mathbb{E}[X^2] = p$

Variance: $\text{Var}(X) = \mathbb{E}[X^2] - \mathbb{E}[X]^2 = p - p^2 = p(1 - p)$

Examples:

- Coin flip
- Randomly guessing on a MC test question
- A server in a cluster fails
- Any indicator RV

Agenda

- Bloom Filters Example & Analysis
- Zoo of Discrete RVs, Part I
 - Uniform Random Variables
 - Bernoulli Random Variables
 - Binomial Random Variables ◀
 - Applications

Binomial Random Variables

A discrete random variable X that is the number of successes in n independent random variables $Y_i \sim \text{Ber}(p)$.

X is a **Binomial random variable** where $X = \sum_{i=1}^n Y_i$

Examples:

- # of heads in n coin flips
- # of 1s in a randomly generated n bit string
- # of servers that fail in a cluster of n computers
- # of bit errors in file written to disk
- # of elements in a bucket of a large hash table

Poll:

pollev.com/rachel312

$P(X = k)$

A. $p^k(1-p)^{n-k}$

B. np

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

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

Binomial Random Variables

A discrete random variable X that is the number of successes in n independent random variables $Y_i \sim \text{Ber}(p)$.

X is a **Binomial random variable** where $X = \sum_{i=1}^n Y_i$

Notation: $X \sim \text{Bin}(n, p)$

PMF: $P(X = k) = \binom{n}{k} p^k (1 - p)^{n-k}$

Expectation:

Variance:

Poll:

pollev.com/Rachel312

	Mean	Variance
--	------	----------

- | | | |
|----|------|-------------|
| A. | p | p |
| B. | np | $np(1 - p)$ |
| C. | np | np^2 |
| D. | np | n^2p |

Binomial Random Variables

A discrete random variable X that is the number of successes in n independent random variables $Y_i \sim \text{Ber}(p)$.

X is a **Binomial random variable** where $X = \sum_{i=1}^n Y_i$

Notation: $X \sim \text{Bin}(n, p)$

PMF: $P(X = k) = \binom{n}{k} p^k (1 - p)^{n-k}$

Expectation: $\mathbb{E}[X] = np$

Variance: $\text{Var}(X) = np(1 - p)$

Mean, Variance of the Binomial

“i.i.d.” is a commonly used phrase.

It means “independent & identically distributed”

If $Y_1, Y_2, \dots, Y_n \sim \text{Ber}(p)$ and independent (i.i.d.), then

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

Claim $\mathbb{E}[X] = np$

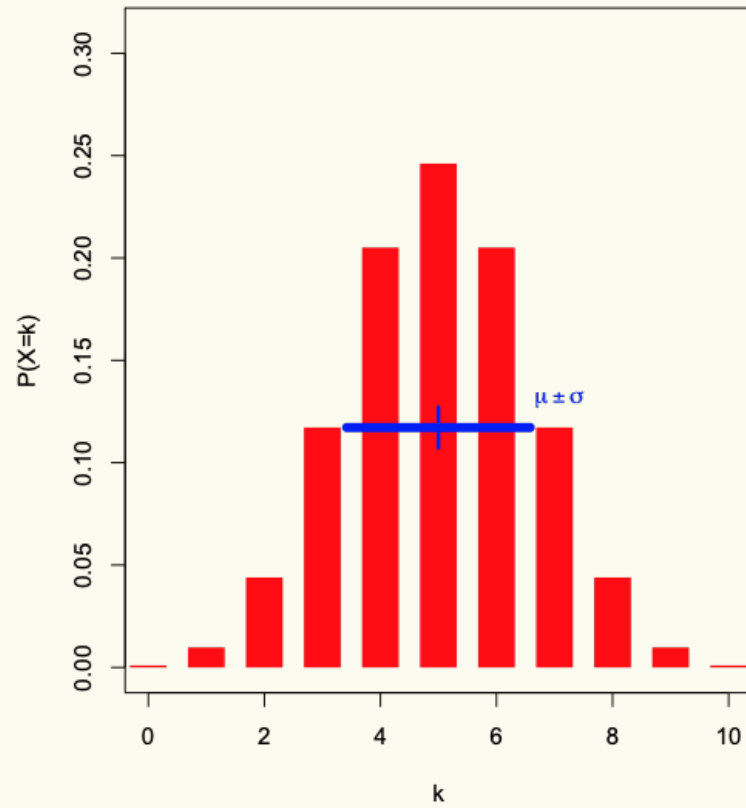
$$\mathbb{E}[X] = \mathbb{E}\left[\sum_{i=1}^n Y_i\right] = \sum_{i=1}^n \mathbb{E}[Y_i] = n\mathbb{E}[Y_1] = np$$

Claim $\text{Var}(X) = np(1 - p)$

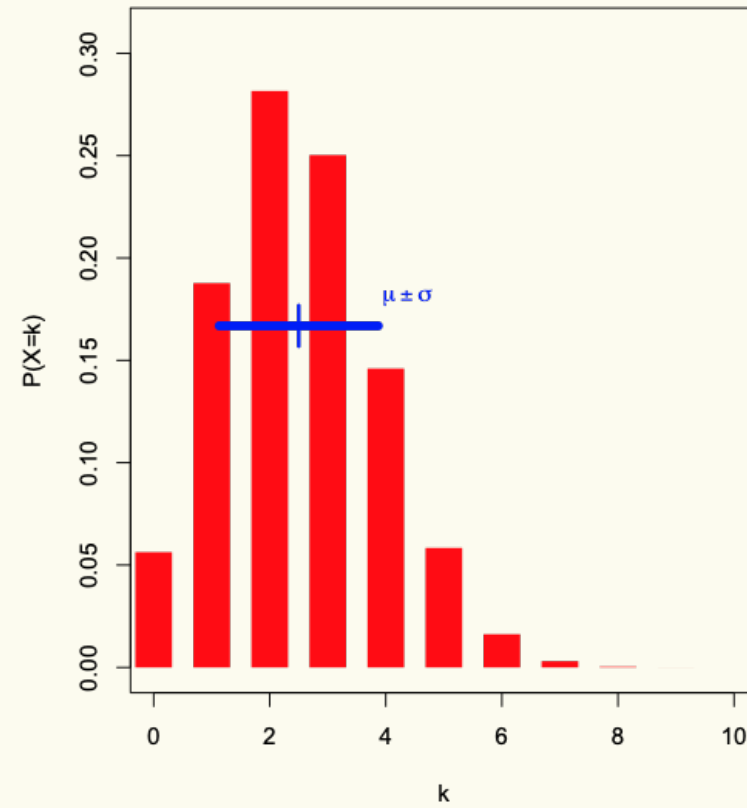
$$\text{Var}(X) = \text{Var}\left(\sum_{i=1}^n Y_i\right) = \sum_{i=1}^n \text{Var}(Y_i) = n\text{Var}(Y_1) = np(1 - p)$$

Binomial PMFs

PMF for $X \sim \text{Bin}(10, 0.5)$

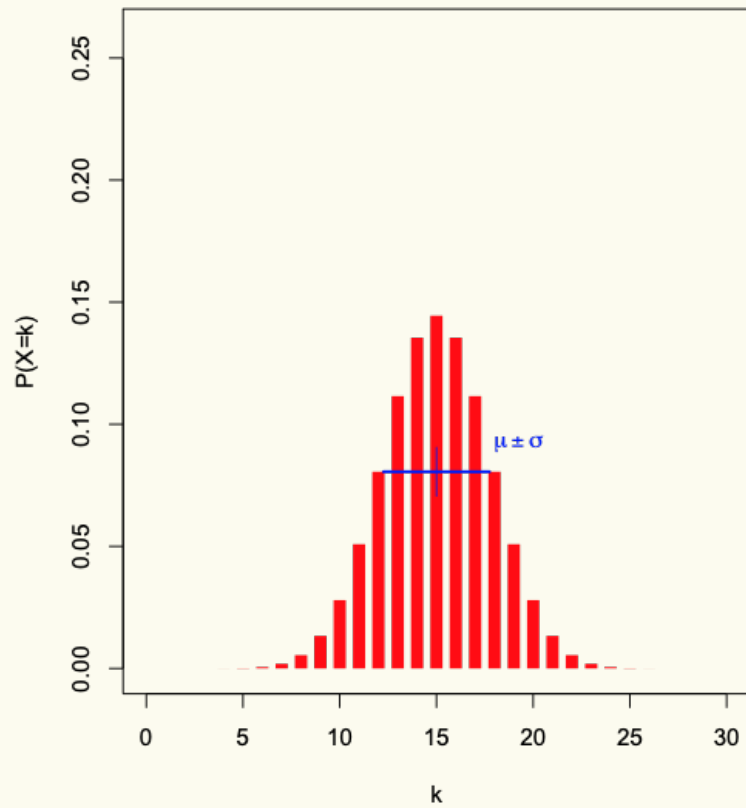


PMF for $X \sim \text{Bin}(10, 0.25)$

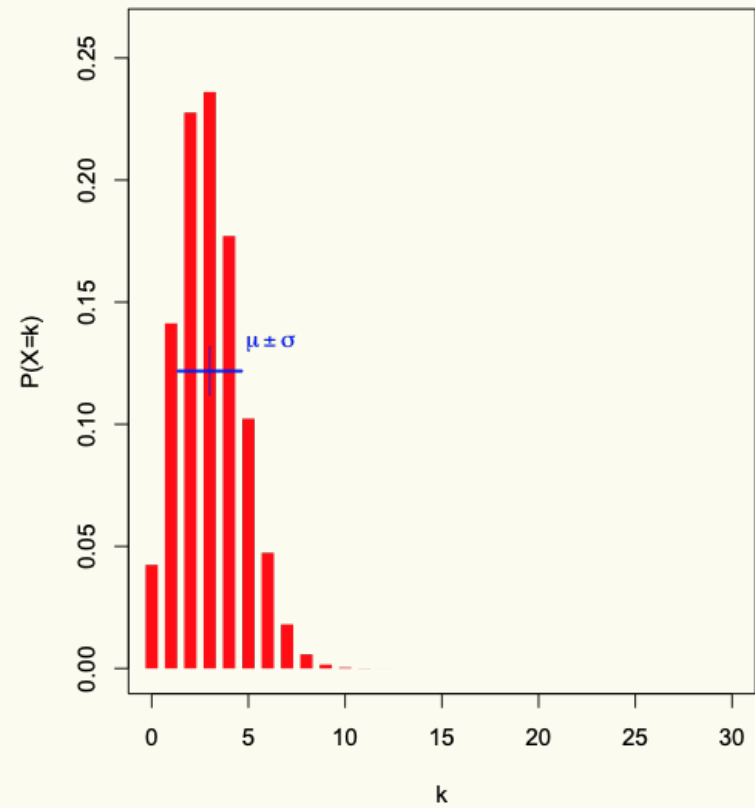


Binomial PMFs

PMF for $X \sim \text{Bin}(30,0.5)$



PMF for $X \sim \text{Bin}(30,0.1)$



Example

Sending a binary message of length **1024** bits over a network with probability **0.999** of correctly sending each bit in the message without corruption (independent of other bits).

Let X be the number of corrupted bits.

What is $\mathbb{E}[X]$?

Poll:

pollev.com/rachel312

- a. 1022.99
- b. 1.024
- c. 1.02298
- d. 1
- e. Not enough information to compute