M. R. C. van Dongen

Introduction

Bitmaps

Bloom Filters

Question Time

For Monday

Summary

About this Document

Introduction to Java (cs2514)

Lecture 17: Bloom Filters

M. R. C. van Dongen

March 17, 2017

Introduction

Bitmaps

Bloom Filters

Question Time

For Monday

Summary

- □ This lecture studies bitmaps and Bloom Filters.
- Bitmaps are space-optimal data structures for representing sets.
- □ Candidate set members come from an index set $\{0, ..., n-1\}$.
- ☐ They can answer simple questions about the set:
 - Iis a candidate member in the set?
- Bloom Filters are also used to represent sets.
- \blacksquare Here the members come from a key set $\{k_0, \ldots, k_{n-1}\}$.
- □ The answers from Bloom Filters are *probabilistic*:
 - Not all answers are correct with 100% probability.
- \blacksquare When n gets large, we cannot represent such sets as bitmaps.
- However, Bloom Filters for such sets are small.
- □ They require only a few bits per element.

Question Time

For Monday

Summary

- A bitmap represents the members of a given index set.
- Here an index set is a set of the form $\{0, ..., n-1\}$.
- A bitmap may be represented using a boolean array.
- ☐ Java does not prescribe how represent a boolean.
- If a boolean is represented as an int then this wastes memory.
- Especially if the index size is large.
- □ In practice bitmaps are often represented as int arrays.

Question Time

For Monday

Summary

```
Bitmap( int capacity ): Create a bitmap.

add( int index ): Add a given index.

remove( int index ): Remove a given index.

size( ): What is the cardinality of the bitmap?

contains( int index ): Does the bitmap contains an index?
```

Implementation

```
public class Bitmap {
    private final boolean[] bits;
    private int size;
    public Bitmap( int capacity ) {
        bits = new boolean[ capacity ];
        size = 0:
    public void add( int index ) {
        size += (bits[ index ]) ? 0 : 1;
        bits[ index ] = true;
    public void remove( int index ) {
        size -= (bits[ index ]) ? 1 : 0;
        bits[ index ] = false:
    public int size( ) {
        return size;
    public boolean contains( int index ) {
        return bits[ index ];
```

Introduction

Bitmaps

Bloom Filters

Question Time

For Monday Summary

```
public class Bitmap {
    private final boolean[] bits;
    private int size;
    public Bitmap( int capacity ) {
        bits = new boolean[ capacity ];
        size = 0:
    public void add( int index ) {
        size += (bits[ index ]) ? 0 : 1;
        bits[ index ] = true;
    public void remove( int index ) {
        size -= (bits[ index ]) ? 1 : 0;
        bits[ index ] = false:
    public int size( ) {
        return size;
    public boolean contains( int index ) {
        return bits[ index ];
```

Introduction

Bitmaps

Bloom Filters

Question Time

For Monday

Summary

Implementation

```
public class Bitmap {
    private final boolean[] bits;
    private int size;
    public Bitmap( int capacity ) {
        bits = new boolean[ capacity ];
        size = 0:
    public void add( int index ) {
        size += (bits[ index ]) ? 0 : 1;
        bits[ index ] = true;
    public void remove( int index ) {
        size -= (bits[ index ]) ? 1 : 0;
        bits[ index ] = false:
    public int size( ) {
        return size;
    public boolean contains( int index ) {
        return bits[ index ];
```

Introduction

Bitmaps

Bloom Filters

Question Time

For Monday Summary

Java

```
public class Bitmap {
    private final boolean[] bits;
    private int size;
    public Bitmap( int capacity ) {
        bits = new boolean[ capacity ];
        size = 0:
    public void add( int index ) {
        size += (bits[ index ]) ? 0 : 1;
        bits[ index ] = true;
    public void remove( int index ) {
        size -= (bits[ index ]) ? 1 : 0;
        bits[ index ] = false:
    public int size( ) {
        return size;
    public boolean contains( int index ) {
        return bits[ index ];
```

Introduction

Bitmaps

Bloom Filters

Question Time

For Monday Summary

Java

```
public class Bitmap {
    private final boolean[] bits;
    private int size;
    public Bitmap( int capacity ) {
        bits = new boolean[ capacity ];
        size = 0:
    public void add( int index ) {
        size += (bits[ index ]) ? 0 : 1;
        bits[ index ] = true;
    public void remove( int index ) {
        size -= (bits[ index ]) ? 1 : 0;
        bits[ index ] = false:
    public int size( ) {
        return size;
    public boolean contains( int index ) {
        return bits[ index ];
```

Introduction Bitmaps

Bloom Filters

Question Time

For Monday

Summary

Bitmaps

Bloom Filters

Question Time

For Monday

Summary

```
Java

Bitmap set = new Bitmap( 10 );
set.add( 4 );
set.add( 8 );
set.add( 1 );
set.remove( 8 );
set.add( 4 );
```

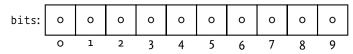
Question Time

For Monday Summary

About this Document

```
Java

Bitmap set = new Bitmap( 10 );
set.add( 4 );
set.add( 8 );
set.add( 1 );
set.remove( 8 );
set.add( 4 );
```



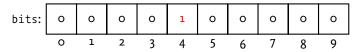
Question Time

For Monday Summary

About this Document

```
Java

Bitmap set = new Bitmap( 10 );
set.add( 4 );
set.add( 8 );
set.add( 1 );
set.remove( 8 );
set.add( 4 );
```



Introduction

Bitmaps

Bloom Filters

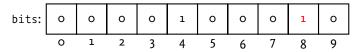
Question Time For Monday

Summary

About this Document

```
Java

Bitmap set = new Bitmap( 10 );
set.add( 4 );
set.add( 8 );
set.add( 1 );
set.remove( 8 );
set.add( 4 );
```



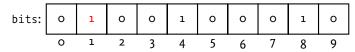
Question Time

For Monday Summary

About this Document

```
Java

Bitmap set = new Bitmap( 10 );
set.add( 4 );
set.add( 8 );
set.add( 1 );
set.remove( 8 );
set.add( 4 );
```



Question Time

For Monday Summary

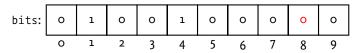
About this Document

```
Java
```

Bitmap set = new Bitmap(10);
set.add(4);
set.add(8);
set.add(1);
set.remove(8);

size:2

set.add(4);



Introduction

Bitmaps

Bloom Filters

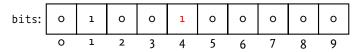
Question Time

For Monday Summary

About this Document

```
Java
```

```
Bitmap set = new Bitmap( 10 );
set.add( 4 );
set.add( 8 );
set.add( 1 );
set.remove( 8 );
set.add( 4 );
```



Question Time

For Monday

Summary

About this Document

```
Remember a Java int is implemented with 32 bits.
```

- We may represent a bitmap with capacity of n using $\lceil n/32 \rceil$ ints.
- □ The Integer class defines the number of bits in an int:
 - □ Integer.SIZE.

```
private final int[] bits;
private int size;
public Bitmap( int capacity ) {
    bits = new int[ (capacity + Integer.SIZE - 1) / Integer.SIZE ];
    size = 0;
}
```

 \square We use bit n to represent index n.

Question Time

For Monday

Summary

```
Remember a Java int is implemented with 32 bits.
```

- \blacksquare We may represent a bitmap with capacity of n using $\lceil n/32 \rceil$ ints.
- The Integer class defines the number of bits in an int:
 - Integer.SIZE.

```
private final int[] bits;
private int size;
public Bitmap( int capacity ) {
    bits = new int[ (capacity + Integer.SIZE - 1) / Integer.SIZE ];
    size = 0;
}
```

- \square We use bit n to represent index n.
- But how do we turn the bits on and off?

Question Time

For Monday

Summary

- Implementing bit operations may be done with arithmetic.
- For example, let bits be an int.
- □ Then (bits / 1) % 2 is the zeroth (least significant) bit.
- □ Then (bits / 2) % 2 is the first bit.
- □ Then (bits / 4) % 2 is the second bit.
- Then (bits $/ 2^n$) % 2 is the nth bit.

Question Time

For Monday

Summary

- Implementing bit operations may be done with arithmetic.
- For example, let bits be an int.
- ☐ Then (bits / 1) % 2 is the zeroth (least significant) bit.
- □ Then (bits / 2) % 2 is the first bit.
- □ Then (bits / 4) % 2 is the second bit.
- □ Then (bits $/ 2^n$) % 2 is the *n*th bit.
- \square bits (((bits / 2^n) % 2) * 2^n) turns the *n*th bit off.

Question Time

For Monday

Summary

- Implementing bit operations may be done with arithmetic.
- For example, let bits be an int.
- ☐ Then (bits / 1) % 2 is the zeroth (least significant) bit.
- □ Then (bits / 2) % 2 is the first bit.
- □ Then (bits / 4) % 2 is the second bit.
- Then (bits $/ 2^n$) % 2 is the *n*th bit.
- \square bits (((bits / 2^n) % 2) * 2^n) turns the *n*th bit off.
- \square bits + ((1 (bits / 2^n) % 2) * 2^n) turns the bit on.

Question Time

For Monday

Summary

```
Java
```

```
final static int SHIFT = Integer.numberOfTrailingZeros( Integer.SIZE );
final static int MASK = Integer.SIZE - 1;
public boolean contains( int index ) {
    return (bits[ index >> SHIFT ] & (1 << (index & MASK))) != 0;
}</pre>
```

Bitmaps

Bloom Filters

Question Time

For Monday Summary

Operator	Example	Result	Description
a >> b	32 >> 4	2	Shift a right by b bits. (Keep sign.)
a >>> b	-1 >>> 30	3	Shift a right by b bits. (Reset sign.)
a << b	3 << 3	24	Shift a left by b bits.
a & b	5 & 3	1	Bitwise and of a and b.
a b	5 3	7	Bitwise or of a and b.
a ^ b	5 ^ 3	6	Bitwise exclusive or of a and b.
~a	~0	-1	Bitwise complement of a.

Bitmaps

Bloom Filters

Question Time

For Monday

Summary

- \square An *n*-bit bitmap may be represented with (n + 7)/8 bytes.
- \blacksquare We can represent all possible 2ⁿ subset configurations.
- All operations take constant time.
- All operations are correct: there are no errors.

Question Time

For Monday

Summary

- Bitmaps are nice if your sets are of the form $\{0, ..., n-1\}$.
- Many applications rely on sets U, with n = |U|, but $n \ll \max_{u \in U}(u)$.
- We could still represent them as bitmaps.
- However, this would require $\max_{u \in U}(u)$ bits.
- ☐ Should this happen, a lot of time is wasted on swopping.
- A datastructure that fits in to memory would be *much* faster:
 - Even if the resulting "set" operations are sometimes inaccurate.

Implementation

Applications Properties

Question Time

For Monday

Summary

About this Document

- A Bloom Filter is a probabilistic version of a "set."
- It requires much less memory than a bitmap-based set.
- It cannot answer questions about size.
- You can add to the set, but you cannot remove from the set.
- □ It can be used to answer set membership queries:
 - □ contains(int key)?
 - The accuracy of the answer depends on the answer:

false: Set definitely doesn't contain key.

true: Set contains key with a certain degree of confidence.

Implementation

Applications Properties

Question Time

For Monday

Summary

- \blacksquare Let u be a random member from the set member universe U.
- \blacksquare Furthermore, let B be a Bloom Filter.
- There are four scenarios.
 - \blacksquare B returns false and $u \notin B$: u is definitely not in B.
 - \blacksquare B returns false and $u \in B$: this situation cannot occur.
 - \blacksquare B returns true and $u \in B$: $u \in B$.
 - \square B returns true and $u \notin B$: False positive. (May happen).

Implementation

Applications

Properties

Question Time

For Monday Summary

- \blacksquare Let u be a random member from the set member universe U.
- \blacksquare Furthermore, let B be a Bloom Filter.
- □ There are four scenarios.
 - \blacksquare B returns false and $u \notin B$: u is definitely not in B.
 - \blacksquare B returns false and $u \in B$: this situation cannot occur.
 - \blacksquare B returns true and $u \in B$: $u \in B$.
 - \square B returns true and $u \notin B$: False positive. (May happen).

Implementation

Applications

Properties

Question Time For Monday

Summary

- \blacksquare Let u be a random member from the set member universe U.
- \blacksquare Furthermore, let B be a Bloom Filter.
- There are four scenarios.
 - \blacksquare B returns false and $u \notin B$: u is definitely not in B.
 - \blacksquare B returns false and $u \in B$: this situation cannot occur.
 - B returns true and $u \in B$: $u \in B$.
 - \square B returns true and $u \notin B$: False positive. (May happen).

Implementation

Applications Properties

Question Time

For Monday

Summary

- \blacksquare Let u be a random member from the set member universe U.
- \blacksquare Furthermore, let B be a Bloom Filter.
- □ There are four scenarios.
 - \blacksquare B returns false and $u \notin B$: u is definitely not in B.
 - \blacksquare B returns false and $u \in B$: this situation cannot occur.
 - \blacksquare B returns true and $u \in B$: $u \in B$.
 - \blacksquare B returns true and $u \notin B$: False positive. (May happen).

Implementation

Applications Properties

Question Time

For Monday

Summary

- \blacksquare Let u be a random member from the set member universe U.
- \blacksquare Furthermore, let B be a Bloom Filter.
- There are four scenarios.
 - \blacksquare B returns false and $u \notin B$: u is definitely not in B.
 - \blacksquare B returns false and $u \in B$: this situation cannot occur.
 - B returns true and $u \in B$: $u \in B$.
 - \square B returns true and $u \notin B$: False positive. (May happen).
- □ In general we don't know whether $u \in B$ or $u \notin B$.
 - \square (That's why we're using B.)

Implementation

Applications Properties

Question Time

For Monday

Summary

- $lue{}$ Let u be a random member from the set member universe U.
- \blacksquare Furthermore, let B be a Bloom Filter.
- There are four scenarios.
 - \blacksquare B returns false and $u \notin B$: u is definitely not in B.
 - \blacksquare B returns false and $u \in B$: this situation cannot occur.
 - \blacksquare B returns true and $u \in B$: $u \in B$.
 - \square B returns true and $u \notin B$: False positive. (May happen).
- In general we don't know whether $u \in B$ or $u \notin B$.
 - \square (That's why we're using B.)
- \square We never have to worry when B returns false.
- We only have to be careful that false positives may occur.
- \square So when B returns true, the result may be incorrect.

Implementation
Applications

Properties

Question Time For Monday

Summary

- Let's revisit our bitmaps.
- We may view a bitmap as a key-value system.
- The keys and values are equal.
- With perfect hashing we could decide set membership of *n* members using *n* bits:
 - We hash each set members to its hash code.
 - The hash codes should be in the range $\{0, ..., n-1\}$.
 - Use the hash code as an index in the array.
- Unfortunately, collisions are very common.

Applications Properties

Question Time

For Monday Summary

- \blacksquare Let *U* be the universe of candidate set members.
- Let n = |U| and let $I = \{0, \ldots, m-1\}$, with $m \ge n$.
- \square With one *perfect* hash function, $h_0: U \to I$ we can decide set membership with a bitmap.
- lacksquare Bloom Filters decide set membership with several hash functions, $h_i:U o I$.
- \blacksquare An empty filter is represented with m bits: each bit is o.
- \blacksquare To add $u \in U$, we set Bit $h_i(u)$ to 1 for each hash function $h_i(\cdot)$.
- Then u is not contained if Bit $h_i(u)$ is o for some hash function.
- \square Otherwise, u is in the set, but false positives may occur.

Implementation

Applications Properties

Question Time For Monday

Summary

- For example, let's assume $U = \{0, 1, 7\}$.
- Furthermore, let's use 4 bits and 2 hash functions:
 - \square $h_0(k) = k + 1 \mod 3$ and $h_1(k) = k \mod 4$.

Implementation

Applications Properties

Question Time

For Monday

Summary

About this Document

■ For example, let's assume $U = \{0, 1, 7\}$.

■ Furthermore, let's use 4 bits and 2 hash functions:

 \square $h_0(k) = k + 1 \mod 3$ and $h_1(k) = k \mod 4$.

Implementation

Applications Properties

Question Time

For Monday

Summary

- For example, let's assume $U = \{0, 1, 7\}$.
- □ Furthermore, let's use 4 bits and 2 hash functions:

$$h_0(k)$$
 $h_1(k)$ Union of Bits

Bitmaps Bloom Filters

Implementation

Applications Properties

Question Time

For Monday

Summary

- For example, let's assume $U = \{0, 1, 7\}$.
- Furthermore, let's use 4 bits and 2 hash functions:
 - $h_0(k) = k + 1 \mod 3$ and $h_1(k) = k \mod 4$.

k	$h_{\circ}(k)$	$h_1(k)$	Union of Bit
)	1	0	0011
	2	1	0110

Bitmaps Bloom Filters

Implementation

Applications Properties

Question Time

For Monday

Summary

- For example, let's assume $U = \{0, 1, 7\}$.
- Furthermore, let's use 4 bits and 2 hash functions:
 - $h_0(k) = k + 1 \mod 3$ and $h_1(k) = k \mod 4$.

k	$h_{\circ}(k)$	$h_1(k)$	Union of Bits
0	1	0	0011
1	2	1	0110
7	2	3	1100

Applications

Properties

Question Time

For Monday

Summary

About this Document

- For example, let's assume $U = \{0, 1, 7\}$.
- □ Furthermore, let's use 4 bits and 2 hash functions:
 - $\square \ h_0(k) = k + 1 \mod 3 \text{ and } h_1(k) = k \mod 4.$

k	$h_{\circ}(k)$	$h_1(k)$	Union of Bits
0	1	0	0011
1	2	1	0110
7	2	2	1100

Keys Added Bits Set $o \in B$? $1 \in B$? $7 \in B$?

Applications

Properties

Question Time

For Monday

Summary

- For example, let's assume $U = \{0, 1, 7\}$.
- □ Furthermore, let's use 4 bits and 2 hash functions:
 - $h_0(k) = k + 1 \mod 3 \text{ and } h_1(k) = k \mod 4.$

Keys Added Bits Set
$$0 \in B$$
? $1 \in B$? $7 \in B$?

Applications Properties

Question Time

For Monday

Summary

- For example, let's assume $U = \{0, 1, 7\}$.
- Furthermore, let's use 4 bits and 2 hash functions:
 - $h_0(k) = k + 1 \mod 3 \text{ and } h_1(k) = k \mod 4.$

k	$h_{\circ}(k)$	$h_1(k)$	Union of Bits
0	1	0	0011
1	2	1	0110
7	2	3	1100

Keys Added	Bits Set	o ∈ B?	$\mathtt{1} \in \mathit{B}$?	7 ∈ B?
{}	0000	_	-	_
{o}	0011	+	-	_

Implementation

Applications Properties

Question Time

For Monday

Summary

- For example, let's assume $U = \{0, 1, 7\}$.
- □ Furthermore, let's use 4 bits and 2 hash functions:
 - $h_0(k) = k + 1 \mod 3$ and $h_1(k) = k \mod 4$.

k	$h_o(k)$	$h_1(k)$	Union of
0	1	0	0011
1	2	1	0110
7	2	3	1100

Keys Added	Bits Set	$o \in B$?	$\mathtt{1} \in B$?	7 ∈ B
{}	0000	_	-	_
{o}	0011	+	-	_
{ı}	0110	-	+	_

Applications

Properties

Question Time

For Monday Summary

- For example, let's assume $U = \{0, 1, 7\}$.
- Furthermore, let's use 4 bits and 2 hash functions:
 - □ $h_0(k) = k + 1 \mod 3$ and $h_1(k) = k \mod 4$.

k	$h_{\circ}(k)$	$h_1(k)$	Union of Bits
0	1	0	0011
1	2	1	0110
7	2	3	1100

Keys Added	Bits Set	$o \in B$?	$1 \in B$?	7 ∈ B
{}	0000	-	-	_
{o}	0011	+	-	_
{1}	0110	-	+	_
{7}	1100	_	_	+

Applications

Properties

Question Time

For Monday Summary

- □ For example, let's assume $U = \{0, 1, 7\}$.
- Furthermore, let's use 4 bits and 2 hash functions:

ĸ	$n_o(\kappa)$	$n_1(\kappa)$	Union of
0	1	0	0011
1	2	1	0110
7	2	3	1100

Keys Added	Bits Set	o ∈ B ?	$\mathtt{1} \in \mathit{B}$?	7 ∈ B
{}	0000	-	-	-
{o}	0011	+	-	_
{ 1 }	0110	-	+	-
{7}	1100	_	_	+
{0,1}	0111	+	+	_

Applications Properties

Question Time

For Monday

Summary

- For example, let's assume $U = \{0, 1, 7\}$.
- Furthermore, let's use 4 bits and 2 hash functions:

ĸ	$h_o(k)$	$h_1(k)$	Union of I
0	1	0	0011
1	2	1	0110
7	2	3	1100

Keys Added	Bits Set	$o \in B$?	$\mathtt{1} \in B$?	7 ∈ B ?
{}	0000	-	-	_
{o}	0011	+	-	_
{1}	0110	-	+	_
{7}	1100	_	_	+
{0,1}	0111	+	+	_
{0,7}	1111	+	+	+

Implementation

Applications Properties

Question Time

For Monday

Summary

- □ For example, let's assume $U = \{0, 1, 7\}$.
- □ Furthermore, let's use 4 bits and 2 hash functions:
 - \square $h_0(k) = k + 1 \mod 3$ and $h_1(k) = k \mod 4$.

$n_{o}(k)$	$h_1(k)$	Union of B
1	0	0011
2	1	0110
2	3	1100
	1 2	1 0 2 1

Keys Added	Bits Set	$o \in B$?	$\mathtt{1} \in B$?	7 ∈ B?
{}	0000	-	-	-
{o}	0011	+	-	_
{ı}	0110	_	+	_
{7}	1100	-	-	+
{0,1}	0111	+	+	_
{0,7}	1111	+	+	+
{1,7}	1110	_	+	+

Applications Properties

Question Time

For Monday

Summary

About this Document

- For example, let's assume $U = \{0, 1, 7\}$.
- Furthermore, let's use 4 bits and 2 hash functions:
 - □ $h_{\circ}(k) = k + 1 \mod 3$ and $h_{1}(k) = k \mod 4$. $k \quad h_{\circ}(k) \quad h_{1}(k) \quad \text{Union of Bits}$ 0 1 0 0011

1 2 1 0110 7 2 3 1100

Keys Added	Bits Set	$o \in B$?	$\mathtt{1} \in \mathit{BP}$	7 ∈ B ?
{}	0000	_	_	_
{o}	0011	+	-	_
{1 }	0110	_	+	_
{7}	1100	_	_	+
{0,1}	0111	+	+	_
{0,7}	1111	+	+	+
{1,7}	1110	-	+	+
{0,1,7}	1111	+	+	+

Implementation

Applications

Key-Value Storage Systems

Detecting Malicious Websites

Probabilistic Spell Checkers

Database Joins

Properties

Question Time

For Monday

Summary

- One application of Bloom Filters are key-value storage systems.
- These systems use *slow* secondary media to store key values.
- Not all candidate keys correspond to values.
- For simplicity, let's assume there's only one slow disk.
- \square We want to know if some value (and if so which) has key k.
- \blacksquare The query B.contains (k) may result in three cases:
 - B returns false:
 - Avoids disk access.
 - $oxed{2}$ B returns true and $k \in B$.
 - Looking up the value using a slow method, returns the value.
 - \blacksquare B returns true and $k \notin B$.
 - \blacksquare Looking up the value fails, we return \bot .
- □ If there are enough queries for bogus keys then this saves time.

Implementation

Applications

Key-Value Storage Systems

Detecting Malicious Websites

Probabilistic Spell Checkers

Database Joins Properties

Question Time

For Monday

Summary

- Let's assume we want to know whether a given URL is malicious.
- We hash the URL into hash codes $h_0, ..., h_{n-1}$.
- If Bit h_i = 0 for some i s.t. 0 $\leq i < n$, then the URL isn't know.
- □ Otherwise the URL is "known" (but this may be a false positive).
- There are two possibilities:
 - We trust the answer.
 - We use a slow database operation to see if the URL is known.

Probabilistic Spell Checkers

- We start with an empty Bloom Filter, B.
- For each allowed word, we add the word's hash code to B.
- The user enters a word.
- We compute the hash codes of the word.
- □ If some hash code bits aren't in the filter, the word is invalid.
- Otherwise, we assume the word is valid.

Introduction to Java

M. R. C. van Dongen

Introduction

Bitmaps

Bloom Filters

Implementation

Applications

Key-Value Storage Systems

Detecting Malicious Websites

Probabilistic Spell Checkers

Database Joins

Properties

Ouestion Time

For Monday

Summary

Julillialy

Implementation

Applications

Key-Value Storage Systems Detecting Malicious Websites

Probabilistic Spell Checkers

Database Joins

Properties

Ouestion Time

For Monday

Summary

- \blacksquare Let's assume we have two database tables T_1 and T_2 .
- \blacksquare Let's compute the join $T_1 \bowtie T_2$.
- This is expensive and takes $\mathcal{O}(|T_1| \times |T_2|)$ worst-case time.
- \blacksquare Removing redundant rows from T_1 and T_2 takes much time.
- This also takes $\mathcal{O}(|T_1| \times |T_2|)$ worst-case time.
- Using Bloom Filters we do this in time $\mathcal{O}(|T_1| + |T_2|)$.

Pre-Processing Database Joins (Continued)

- \square Let's remove the redundant rows of T_1 .
- Let *S* be the intersection of the scopes of T_1 and T_1 .
- \square We start with an empty filter B.
- There are two phases:
 - \square Add hash values of the rows in T_2 to B.
 - \square Remove rows from T_1 whose hash values aren't in B.
- In practice, this pre-processing speeds up the join.

Introduction to Java

M. R. C. van Dongen

Introduction

Bitmaps

Bloom Filters

Implementation

Applications

Key-Value Storage Systems
Detecting Malicious Websites

Probabilistic Spell Checkers

Probabilistic Spell

Database Joins

Properties

Ouestion Time

For Monday

Summary

Bloom Filters Implementation

Applications

Properties

Question Time

For Monday

Summary

- Time to decide membership is independent of current "size."
 - lacksquare With k constant-time hash functions we need $\mathcal{O}(k)$ time.
- Reliability may be improved by increasing the number of bits.
- \blacksquare Let R be the false positive rate.
- \blacksquare For \pm 4.8 extra bits per member, R reduces by a factor of \pm 10.

Introduction

Bitmaps Bloom Filters

Question Time For Monday

Summary

About this Document

Questions Anybody?

Bitmaps

Bloom Filters

Question Time

For Monday

Summary

- Study the presentation,
- Implement the bitmap class from scratch.

Question Time For Monday

Summary

- \square We've studied *n*-bit bitmaps and *m*-bit Bloom Filters.
- Bitmap decides which numbers in $\{0, ..., n-1\}$ are in the set.
- \Box The *i*-th member is in the set if and only if the *i*th bit is set.
- They are 100% accurate.
- With perfect hashing you can use them for other kinds of sets.
- Bloom Filters are probabilistic data structures.
- They use several hash functions, $h_i(\cdot)$.
- Candidate, c, isn't in the filter if Bit $h_i(c)$ is o, for some i.
- Otherwise, Bit $h_i(c)$ is 1 for each i.
- \blacksquare In this case u is in the set, but false positives may occur.

M. R. C. van Dongen

Introduction

Bitmaps

Bloom Filters

Question Time For Monday

Summary

- ☐ This document was created with pdflatex.
- The LATEX document class is beamer.