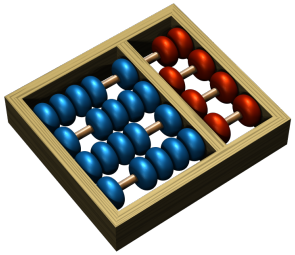
MC202 - Estruturas de Dados

Bloom Filters

Emilio Francesquini
francesquini@ic.unicamp.br

Instituto de Computação - UNICAMP

Aula 26 - 9 de junho de 2017



UNICAMP

Disclaimer

- Esses slides foram preparados para um curso de Estrutura de Dados ministrado na UNICAMP
- Este material pode ser usado livremente desde que sejam mantidos os créditos dos autores e da instituição.

Bloom Filter

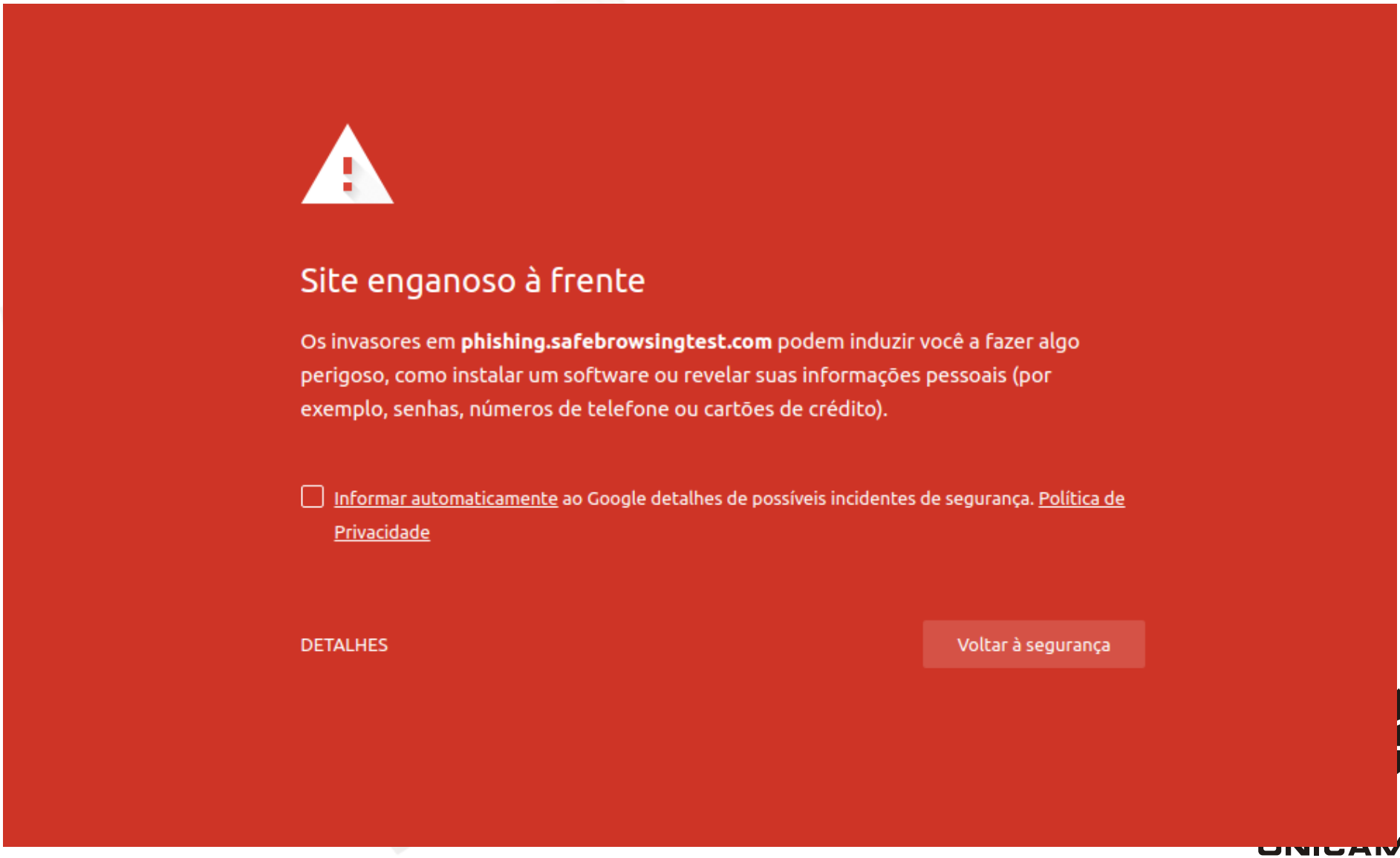
- Um filtro de Bloom (=Bloom Filter) é uma estrutura de dados probabilística que é muito eficiente em termos desempenho e de consumo de memória.
- Foi inventada em 1970 por Burton H. Bloom.
- É utilizada para verificar, de maneira rápida, se um elemento pertence ou não a um conjunto.
  - Neste sentido é parecida com outras estruturas de dados que já conhecemos como: vetores, listas ligadas, árvores de busca e tabelas de hash
  - Apesar de um filtro de Bloom utilizar **consideravelmente menos memória**, ele também pode dar respostas incorretas

UNICAMP

Bloom Filter - Operações

- Um filtro de Bloom tem as seguintes operações:
  - Inserção - Insere o elemento no filtro
  - Consulta - Verifica se o elemento está presente no filtro
    - Contudo, ele pode dar falsos positivos
    - Se a consulta indicar que o elemento não está presente, então com certeza não está presente
    - Se a consulta indicar que o elemento está presente então **provavelmente** ele está presente
- Então, para que serve tal coisa?

Web Browsers



Web Browsers

- O seu browser verifica todos os endereços que você acessa para determinar se é um endereço seguro e poder te dar um aviso.
- Há bilhões de endereços na Internet.
- Podemos supor com uma boa margem de segurança que dentre eles há alguns milhões de sites fraudulentos.
- Seria ineficiente fazer com que todos os computadores do mundo fizessem download de uma lista completa de sites fraudulentos.
- Também não seria muito eficiente checar um banco de dados central a cada endereço que fosse acessado.

Web Browsers

- O browser faz download de um filtro de Bloom contendo todos os sites reconhecidos como inseguros
- A cada endereço acessado, o browser verifica se ele está contido no filtro
- Se a consulta ao filtro indicar que o endereço não está contido, então com certeza o endereço não está na lista de sites inseguros e o acesso continua normalmente
- Se o filtro indicar que está contido, então **pode ser** que site esteja na lista de endereços inseguros. Neste caso o seu browser faz uma consulta adicional a um banco de dados na Internet (esta sim com a lista completa)
- Note que caso seja um falso positivo, a única consequência é uma demora um pouco maior para acessar o site desejado

UNICAMP

Outras Aplicações

- Verificação preliminar de direitos de acesso
- Evitar recomendar artigos/páginas que o usuário não gostou ou já leu
- Evitar leituras no disco em bancos de dados
- Verificar senhas fracas
- Corretores ortográficos
- ...

Implementação

- Filtros de Bloom são baseados em
  - Funções de hash
  - Bit fields
- Para um bom funcionamento são essenciais
  - Diversas funções de hash
  - Espalhamento uniforme
- Felizmente, basta possuirmos 2 funções de hash boas para criar quantas funções de hash quisermos[1]
  - $\mathscr{H}_i = (a + b * i) \ mod \ m$

[1] Adam Kirsch, Michael Mitzenmacher. "Less Hashing, Same Performance: Building a Better Bloom Filter"

UNICAMP

Exemplo

- No nosso exemplo, o filtro de Bloom tem tamanho $m = 10$ e vamos utilizar $k = 3$ funções de hash $\mathscr{H}_1$, $\mathscr{H}_2$ e $\mathscr{H}_3$
- Vamos também indicar por $\mathscr{H}(x) = \{\mathscr{H}_1(x), \mathscr{H}_2(x), \mathscr{H}_3(x)\}$
- Nós começamos com um vetor de bits zeros de tamanho $m = 10$

Índice	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Valor	0	0	0	0	0	0	0	0	0	0

Exemplo

Inserção de x_0: $\mathscr{H}(x_0) = \{1, 4, 9\}$

Índice	0	1	2	3	4	5	6	7	8	9
Valor	0	1	0	0	1	0	0	0	0	1

Inserção de x_1: $\mathscr{H}(x_1) = \{4, 5, 8\}$

Índice	0	1	2	3	4	5	6	7	8	9
Valor	0	1	0	0	1	1	0	0	1	1

UNICAMP

Exemplo

$\mathscr{H}(x_0) = \{1, 4, 9\}$
$\mathscr{H}(x_1) = \{4, 5, 8\}$

Índice	0	1	2	3	4	5	6	7	8	9
Valor	0	1	0	0	1	1	0	0	1	1

Consulta a presença de x_2, onde $\mathscr{H}(x_2) = \{0, 4, 8\}$

Não está presente

Exemplo

$\mathscr{H}(x_0) = \{1, 4, 9\}$
$\mathscr{H}(x_1) = \{4, 5, 8\}$

Índice	0	1	2	3	4	5	6	7	8	9
Valor	0	1	0	0	1	1	0	0	1	1

Consulta a presença de x_3, onde $\mathscr{H}(x_3) = \{1, 5, 8\}$

Falso positivo

Falsos Positivos

- Suponha que:
  - m é o número de entradas no filtro
  - k é o número de funções de hash sendo utilizadas
  - n é o número de elementos já inseridos no filtro
- Neste caso, a probabilidade p de falsos positivos é:

$$p = (1 - e^{-\frac{k*n}{m}})^k$$

- Como escolher apropriadamente k e m?

Para ver todo o raciocínio (e provas formais) por trás do cálculo dessas probabilidades veja: Andrei Broder, Michael Mitzenmacher. "Network Applications of Bloom Filters: A Survey", 2005.
`http://www.internetmathematicsjournal.com/article/1393`

Escolhendo k e m

- Sabe-se que k é ótimo quando $k = \frac{m \log 2}{n}$
- Note que $\frac{m}{n}$ indica o número de bits por elementos no filtro. Logo, o número ótimo de funções de hash k cresce linearmente com o número de bits por elemento b: $k = b \log 2$
- Assumindo que escolhamos o valor ótimo para k, obtemos:

$$p = 2^{-\frac{m \log 2}{n}} \rightarrow m = \frac{n \log \frac{1}{p}}{\log^2 2}$$

Desempenho e Consumo de Memória

- Um filtro de Bloom de m entradas pode ser armazenado usando-se m bits, ou $\frac{m}{8}$ bytes
- Tanto a inserção quanto a consulta em um filtro que utiliza k funções de hash levam tempo $O(k)$
- O espaço m utilizado por um filtro de Bloom varia em função de k, n e p
- Caso não se tenha nenhuma ideia sobre o valor de n, filtros de Bloom escaláveis podem ser utilizados[2]

[2] Paulo S. Almeida, Carlos Baquero, Nuno Preguiça. "Scalable Bloom Filters". `http://gsd.di.uminho.pt/members/cbm/ps/dbloom.pdf`

UNICAMP

Recursos adicionais

- Breve explicação e simulação online:
  `https://www.jasondavies.com/bloomfilter/`
- Uma explicação em guardanapos de papel:
  `https://blog.medium.com/what-are-bloom-filters-1ec2a50c68ff`
- Outra simulação online:
  `https://llimllib.github.io/bloomfilter-tutorial/`
- Página da Wikipédia Sobre Filtros de Bloom:
  `https://en.wikipedia.org/wiki/Bloom_filter`

UNICAMP