UTILIZANDO FILTRO DE BLOOM PARA VERIFICAR VACINAÇÃO EM DUPLICIDADE

USING BLOOM FILTER TO CHECK DUPLICATE VACCINATION

PAULO DIOGO R. LEÃO^a REGINA S. LANZILLOTTI^b

Resumo

Neste artigo foram propostos duas soluções para armazenamento de dados de cidadãos para verificação de aplicação de vacinas contra coronavírus. O país está com problemas de falta de centralização no combate ao vírus e por isso alguns cidadãos tomaram doses a mais em diferentes locais do município ou mesmo em estados diferentes, uma vez que não temos uma base unificada. Os experimentos mostraram que uma estrutura de dados probabilística pode ajudar no desenvolvimento de uma aplicação offline que seja utilizada no país inteiro, uma vez que o espaço necessário para armazenar os dados dos cidadãos seria reduzido, ajudando assim o país a ter uma vacinação eficiente e segura para todos.

Palavras-chave: Filtro de bloom, estruturas de dados probabilísticas, distribuição gama.

Abstract

In this article, two solutions for storing citizen data to verify the application of vaccines against coronavirus were proposed. The country has problems with the lack of centralization in the fight against the virus and that is why some citizens took more doses in different places in the city or even in different states, since we do not have a unified base. The experiments showed that a probabilistic data structure can help in the development of an offline application that is used nationwide, as the space needed to store citizens' data would be reduced, thus helping the country to have an efficient and safe vaccination for all.

Keywords: Bloom filter, probabilistic data structures, gamma distribution

MSC2010: 60E05, 62E99

^aInstituto de Matemática e Estatística – IME, Universidade do Estado do Rio de Janeiro – UERJ, Rio de Janeiro, RJ, Brasil; ORCID: <https://orcid.org/0000-0003-1406-0162> **E-mail:** paulo.leao@pos.ime.uerj.br

^bInstituto de Matemática e Estatística – IME, Universidade do Estado do Rio de Janeiro – UERJ, Rio de Janeiro, RJ, Brasil; ORCID: <https://orcid.org/0000-0001-7789-6843> **E-mail:** reginalazillotti@gmail.com

1 Introdução

A pandemia do coronavírus trouxe vários problemas para a humanidade, dentre eles estão a falta de preparo dos países para a quantidade de enfermos de uma só vez, falta de amparo para as pessoas que não tem um trabalho formal e no caso do Brasil uma total falta de organização central, fazendo com que algumas pessoas tomem várias doses de vacinas em diferentes postos de saúde na mesma cidade ¹² e até casos onde cidadãos foram para outros estados tomar mais algumas doses de vacina ³, aproveitando a falta de um sistema de controle de informação para centralização dos dados de aplicação de vacina.

As vacinas contra o coronavírus foram produzidas em uma velocidade nunca vista antes, as primeiras vacinas estavam sendo testadas ainda em 2021 [6]. Os desafios para realização de uma campanha de vacinação, haja visto que o coronavírus se dissemina muito rapidamente, são enormes [4]. Esses desafios vão muito além do desenvolvimento das vacinas, sobretudo à logística para que a vacina chegue ao destino, para aplicação nos cidadãos, uma medida sem precedentes foi o Covax Facility, que tem o objetivo de acelerar o desenvolvimento de vacinas, capitaneada pela OMS [5].

Apesar dessa falta de organização central, mais de 133 milhões de brasileiros já tomaram a primeira dose de alguma vacina e 59 milhões já estão completamente imunizados [11], a população compareceu em peso. A ausência de centralização tornou a organização da vacinação mais complexa, uma vez que cada município é o responsável por organizá-la de forma independente, porém, causou problemas. Se um cidadão precisasse viajar para outro estado na época que devesse tomar a sua primeira ou segunda dose, o cidadão precisaria contar com a boa vontade do servidor público, pois em algumas localidades ocorreu a exigência da apresentação do comprovante de residência em função das normas (ou resoluções) da sua própria cidade [12, 7].

Um sistema de informação que armazene os dados dos cidadãos imunizados, pode ser proposto para evitar que alguns cidadãos se imunizem mais do que é necessário, evitando futuros problemas com a mistura de diferentes vacinas, uma vez que ainda não foram realizados estudo que comprovem a segurança na utilização de mais de

¹<https://www.uol.com.br/vivabem/noticias/redacao/2021/07/03/mp-investiga-caso-de-medica-que-tomou-3-doses-da-vacina-contracovid-19.htm>

²<https://g1.globo.com/pa/para/noticia/2021/07/15/pesquisadores-apontam-que-quase-700-pessoas-podem-ter-tomado-tres-doses-de-vacina-contracovid-19-no-pa.ghtml>

³<https://g1.globo.com/mg/zona-da-mata/noticia/2021/07/07/morador-de-vicosa-tomou-4-doses-de-vacinas-contracovid-19-em-mg-e-rj-afirma-prefeitura.ghtml>

um imunizante, ou múltiplas doses de um imunizante.

Com o sistema proposto poderia diminuir os casos de imunização múltipla, uma vez que, haveria uma checagem para verificar se o individuo foi imunizado em mais de uma localidade do país de forma simples e eficiente. A solução proposta visa evitar que um individuo tome doses de forma incorreta.

Este artigo está organizado em cinco seções, sendo esta primeira a introdução, seguida das soluções propostas. A terceira seção apresenta os métodos e técnicas que foram utilizados. Uma descrição dos resultados experimentais é descrita na quarta seção e na última seção são apresentadas as considerações finais.

2 Solução

Nesta seção serão apresentadas duas possíveis soluções para o problema proposto, ou seja, conhecidos os CPFs dos cidadãos imunizados, como evitar que um cidadão venha a se vacinar em outro local.

O cenário descreve-se: todos os postos de saúde do país terão um ou mais dispositivos que trabalham offline e o banco de dados é atualizado num servidor central com as informações dos cidadãos vacinados até dado momento. Essa atualização caracteriza-se a partir de um computador com acesso a internet, que vai dar acesso aos dados e os dispositivos são atualizados por meio da rede sem fio do local. Uma vez que o dispositivo esteja atualizado, o funcionário responsável pela checagem da fila faz uma verificação das pessoas, para saber quem realmente não se vacinou. Caso a resposta seja negativa o cidadão permanece na fila, caso contrário, se dirige para uma outra fila para uma verificação complementar de verificação pode ser tanto no posto ou centralizado regionalmente, porque este pode ser um falso positivo.

2.1 Solução 1

Uma primeira solução poderia ser a utilização de um banco de dados com todos os CPFs dos cidadãos imunizados. Neste caso cada caractere precisa 1 byte para ser armazenado, como o CPF tem 11 caracteres, precisaremos de 11 bytes para cada CPF. Sendo assim, precisaremos de 2.2 Gigabytes para o armazenamento dos 200 milhões de cidadãos do país.

Como o país inteiro utilizaria esta solução, isso envolveria baixar um arquivo de 2.2 Gigabytes em todo o país, uma vez que a solução seria implementada de maneira offline em milhares de dispositivos pelo país.

2.2 Solução 2

Uma segunda solução seria utilizar uma estrutura que armazene esses dados de forma probabilística, ou seja, podemos representar esses CPFs de uma forma que não seja necessário utilizar tanto espaço. Essa estrutura sempre retorna uma resposta correta quando um CPF não está contido e poderia ter uma probabilidade baixa para erro ao retornar que um CPF faz parte.

Caso seja utilizada uma estrutura de dados de apenas 10 bits para representar cada CPF, a necessidade de armazenamento seria reduzida para 250 Megabytes referentes aos mesmos 200 milhões de CPFs, ou seja menos de 10% do armazenamento necessário na primeira solução. Neste caso as atualizações poderiam ser feitas aos poucos e frequentemente, uma vez que o arquivo é relativamente pequeno.

3 Técnicas e métodos

Nesta seção será apresentado o método para implementar a estrutura de dados proposta na seção anterior.

3.1 Distribuição gama

Conforme [9], a variável aleatória contínua X , assumindo valores positivos, tem uma distribuição gama com parâmetros $\alpha > 0$ e $\beta > 0$, se sua função densidade de probabilidade for dada por:

$$f(x; \alpha, \beta) = \begin{cases} \frac{1}{\Gamma(\alpha)\beta^\alpha} x^{\alpha-1} e^{-x/\beta}, & x > 0 \\ 0, & x < 0 \end{cases} \quad (3.1)$$

Para encontrar os valores de α e β nesta expressão, foram utilizados a Expectância e a Variância da distribuição de probabilidade Gama,

$$E(X) = \alpha\beta, Var(X) = \alpha\beta^2 \quad (3.2)$$

Para verificar adequabilidade da distribuição Gama à distribuição de frequência observada foi utilizado um teste de aderência Qui-quadrado, tendo hipótese nula (H_0), a distribuição dos dados observados é aderente a distribuição gama segundo a expressão para

$$\chi^2 = \sum_{i=1}^s \frac{(O_i - E_i)^2}{E_i} \quad (3.3)$$

onde O_i é o valor observado e E_i representa o valor esperado.

Aceitamos H_0 se o resultado da estatística teste for menor que a estatística esperada Qui-quadrado obtida na tabela se a distribuição Qui-quadrado para um determinado grau de liberdade e nível de significância adotado [9].

3.2 Filtro de bloom

O filtro de bloom é uma estrutura de dados probabilística que permite verificar a pertinência de itens e que possibilita testar se um item pertence ou definitivamente não pertence a um conjunto de dados [2]. Conforme [3] para se obter uma probabilidade de menos de 1% de falsos positivos é necessário apenas 10 bits por item.

O filtro representa um conjunto S de cardinalidade n utilizando um vetor V com m bits, com $V[i] = 0, \forall i \in [1..m]$, estando associado a k funções hash $h_i : S \rightarrow [1..m], \forall i \in [1..k]$. Cada função h_i mapeia itens de S em $[1..m]$ de maneira aleatória com uma distribuição uniforme.

Dito isto, o algoritmo de inserção no filtro, no qual atribuiremos 1 nas posições mapeadas pelas funções h_i , conforme pseudocódigo (1).

Algorithm 1 Adicionar um item no filtro de bloom

```

1: procedure ADICIONAR( $x$ )
2:   for  $i \leftarrow 1, k$  do
3:      $V[h_i(x)] \leftarrow 1$ 
4:   end for
5: end procedure

```

Para verificar se um item está contido no filtro, podemos executar um algoritmo semelhante, neste caso precisamos verificar se todas as posições mapeadas pelas funções h_i estão com valor igual a 1, conforme pseudocódigo (2).

Algorithm 2 Verificar se um item está contido no filtro de bloom

```

1: function CONTEM( $x$ )
2:    $contido \leftarrow true$ 
3:   for  $i \leftarrow 1, k$  do
4:      $contido \leftarrow contido \wedge V[h_i(x)] = 1$ 
5:   end for
6:   return  $contido$ 
7: end function

```

Sabe-se que nos filtros de Bloom colisões vão existir, desta forma, falsos positivos são possíveis. Entretanto, falsos negativos não são possíveis. Este é um comportamento desejado em aplicações que podem postergar operações custosas, onde pode-se esperar para executar essas operações quando atingir um determinado número de operações [10].

A probabilidade de colisão determina a probabilidade de um falso positivo, desta forma, ao utilizar uma quantidade maior de bits no filtro, menor a probabilidade de falso positivo. Múltiplas funções hash podem ser utilizadas para diminuir a possibilidade de colisão, porém, deve ser analisado com cuidado, uma vez que utilizar múltiplos hashes por elemento pode aumentar a quantidade de colisões.

3.2.1 Exemplo

Um exemplo prático do filtro de bloom, pode ser criado seguindo os seguintes passos:

(1) $m = 20$, utilizando 3 funções hash retornam valores entre $[1..20]$.

Primeiro faremos inserções no filtro, conforme tabela 1.

Tabela 1: Exemplo inserção filtro de bloom

Valor	F[1..m]																			
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
João da Silva (1, 3, 17)	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
Maria Joaquina (8, 12, 19)	1	0	1	0	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0	1
José Borges (4, 7, 14)	1	0	1	1	0	0	1	1	0	0	0	1	0	1	0	0	0	1	0	1
Joana José (9, 13, 18)	1	0	1	1	0	0	1	1	1	0	0	1	1	1	0	0	1	1	1	1

Na primeira coluna da tabela inserção de dados teremos o valor inserido e em seguida os valores gerados pelas funções hash. Esses valores serão os índices onde o valor 1 será inserido no vetor. A tabela está na ordem de inserção dos valores e a primeira linha representa o estado inicial do vetor do filtro.

Após a inserção dos dados, podemos iniciar a consulta dos dados, conforme tabela 2.

Tabela 2: Exemplo inserção filtro de bloom

Valor	F[1..m]																			Encontrou?
João da Silva (1, 3, 17)	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	Sim	
Geraldo Rodrigues (5, 11, 19)	1	0	1	0	0	0	0	1	0	0	0	1	0	0	0	0	1	0	1	Não
Maria Lina (4, 7, 14)	1	0	1	1	0	0	1	1	0	0	0	1	0	1	0	0	1	0	1	Sim

Nas consultas efetuadas no filtro podemos verificar que quando realizamos a consulta por "João da Silva" obtivemos um resultado positivo, uma vez que, os três

valores dos hashes gerados são iguais a 1, indicando que o valor existe no filtro. A segunda consulta por "Geraldo Rodrigues" diz que o valor não existe, pois, dois valores gerados nas funções hash são iguais a zero, este resultado nos dá a certeza que o valor consultado não pertence ao filtro. A última consulta avalia se diz que o valor "Maria Lina" está presente no filtro, entretanto este é um falso positivo, dado que o valor não faz parte do filtro.

A estrutura utilizada neste trabalho é baseada em funções hash e dependem de suas propriedades. O objetivo dessas funções é a capacidade de simular aleatoriedade para um conjunto de dados estruturado, entretanto, conforme [8], não é possível gerar dados aleatórios a partir de dados não aleatórios. Na próxima seção será descrita a função hash que foi utilizada neste trabalho.

3.3 MurmurHash

Uma das principais aplicações de funções hash é o mapeamento de grandes conjuntos de dados em tabelas com tamanho menor do que o conjunto inicial, ou seja, uma tabela hash.

MurmurHash é uma família de hash não-criptográficas de propósito geral criadas por Austin Appleby. Sua vantagem é ter um desempenho superior a outras famílias de sua época, além de passar nos testes de uniformidade de funções hash. Essa função foi utilizada nos experimentos deste trabalho [1].

Na próxima seção será executado um estudo experimental para verificar se a estrutura de dados filtro de bloom é realmente uma boa opção para a solução proposta.

4 Resultados experimentais

Para verificar se o filtro de Bloom realmente tem uma baixa taxa de falsos positivos, foram executados alguns experimentos. Os testes foram executados numa população de 400.000 CPFs, sendo esses gerados de forma aleatória. Para a execução a quantidade de bits para cada elemento do filtro variou entre [1..10] e a quantidade de hashes variou entre [1..7].

Podemos verificar que até 3 bits ainda existe um número grande de colisões, mesmo utilizando-se 7 funções hash. Entretanto, verifica-se que a partir de 4 bits existe uma redução acentuada quando o número de funções hash é maior que 4. Neste mesmo contexto, podemos verificar que utilizar uma função hash é ruim para qualquer quantidade de bits por elemento.

Nos experimentos foram coletadas as proporções de falsos positivos para cada filtro gerado. Por exemplo: para um filtro com 8 bits por elemento que utiliza 5 hashes para cada elemento adicionado a proporção de falsos positivos foi 0.0013. Sendo assim, foram geradas 70 proporções, conforme histograma da figura 1.

Podemos verificar na tabela 3 que quanto mais bits e funções hash menor a proporção de falsos positivos, que o valor mínimo foi 0,00019 quando utilizadas 7 funções hash e 10 bits por elemento. Por outro lado, a maior proporção foi encontrada em 0.80697 quando utilizamos 7 funções hash e apenas 1 bit para representar os elementos. A média encontrada foi de 0.0975 com uma variância de 0.0309 para as 70 proporções.

Ao usar a expressão (3.2) referente a Expectância e Variância da distribuição de probabilidade Gama, os parâmetros assumem valores 0.3074 e 0.317, respectivamente, sendo possível gerar o histograma para os valores esperados conforme figura 2.

Tabela 3: Proporções número de funções hash x número de bits por elemento

	1	2	3	4	5	6	7	8	9	10
1	0,39179	0,22068	0,15244	0,11718	0,09479	0,07975	0,06832	0,06058	0,05318	0,04872
2	0,39942	0,15413	0,08055	0,04872	0,03272	0,02376	0,01797	0,01372	0,01075	0,00881
3	0,46648	0,14711	0,06136	0,03046	0,01716	0,01113	0,00725	0,00513	0,00372	0,00254
4	0,55544	0,15911	0,05606	0,02406	0,01164	0,00636	0,00376	0,00248	0,00157	0,00110
5	0,65089	0,18458	0,05774	0,02176	0,00937	0,00470	0,00239	0,00143	0,00079	0,00055
6	0,73616	0,22025	0,06310	0,02149	0,00815	0,00379	0,00192	0,00094	0,00060	0,00031
7	0,80697	0,26299	0,07258	0,02296	0,00799	0,00339	0,00157	0,00069	0,00032	0,00019

Figura 1: Histograma proporção falsos positivos

Figura 2: **Histograma proporção falsos positivos**

O valor do qui-quadrado observado $\chi^2 = 2.753$. Para um nível de significância igual a 5% e um grau de liberdade igual a 7 o Qui-quadrado esperado observado na tabela da Distribuição Qui-quadrado corresponde a 14.067, como $2.753 < 14.067$, e aceitaremos H_0 , logo podemos dizer que os dados seguem uma distribuição gama de parâmetros alfa igual a 0.3074 e o beta igual a 0.317.

Os experimentos foram executados nas seguintes configurações indicados no quadro a seguir:

AMD Ryzen 5 3600 (Cores 6, Threads 12, 3600 MHz)
Memória: 16GB DDR4 1333MHz
Windows 10 Pro 64bits
Visual Code 1.55.0
Python: 3.9.1

5 Considerações finais

Podemos verificar que o filtro de Bloom é uma maneira eficiente para armazenamento de dados, pois sua utilização no cenário proposto poderia ser uma opção viável, uma vez que a quantidade de falsos positivos é menor que 1%.

Uma vez que os dados observados seguem uma distribuição gama, ou seja, podemos ter certeza que quanto maior a quantidade de bits e funções hash, menor a probabilidade de termos falsos positivos, pois ao variar a quantidade de bits e funções hash diminuimos a quantidade de falsos positivos observados.

A análise estatística dos dados permite a proposição, quanto maior a quantidade de bits e funções hash para representação, menor a quantidade de falsos positivos. A estrutura mostrou-se viável para o cenário.

Conclui-se que a quantidade bits afeta diretamente a proporção de falsos positivos e ao utilizar 1 bit apenas a quantidade de falsos positivos torna-se alta para todos os casos, quanto que quando a quantidade de bits é maior que 4 a proporção de falsos positivos fica quase sempre abaixo de 5%. Observou-se que dependendo da aplicação uma outra quantidade de bits e funções pode ser utilizada, diminuindo ainda mais o espaço de armazenamento necessário.

Em trabalhos futuros, a utilização de outras funções hash poderiam ser comparadas em relação ao MurmurHash, para verificar se é possível criar uma melhor distribuição dos itens no vetor e assim diminuir a proporção de falsos positivos.

Referências

- [1] APPLEBY, A. **Murmurhash3**. URL: <https://github.com/aappleby/smhasher/wiki/MurmurHash3>, 2016.
- [2] BLOOM, B. **Space/time trade-offs in hash coding with allowable errors**. *Communications Of The ACM*, 422-426, 1970.
- [3] BONOMI, F.; Mitzenmacher, M.; Panigrahy, R.; Singh, S.; Varghese, G. **An improved construction for counting bloom filters**. *European Symposium On Algorithms*. pp. 684-69, 2006.
- [4] DOMINGUES, C. **Desafios para a realização da campanha de vacinação contra a COVID-19 no Brasil**, SciELO Brasil, 2021.
- [5] ECCLESTON-TURNER, M.; Upton, H. **International Collaboration to Ensure Equitable Access to Vaccines for COVID-19: The ACT-Accelerator and the COVAX Facility**. *The Milbank Quarterly*, 2021.
- [6] LIMA, E.; Almeida, A; Kfour, R. **Vacinas para COVID-19-o estado da arte**. *Revista Brasileira De Saúde Materno Infantil*. 21 pp. 13-19, 2021.
- [7] MACHADO, B.; Hermany, R. **OS IMPACTOS E DESAFIOS DA PANDEMIA DA COVID-19 NO FEDERALISMO TRINO BRASILEIRO: DO CONFLITO POSITIVO DE COMPETÊNCIAS À JUDICIALIZAÇÃO DA CRISE**. *Seminário Internacional Demandas Sociais E Políticas Públicas Na Sociedade Contemporânea*, 2021.
- [8] MCCLELLAN, M.; Minker, J. **The art of computer programming, vol. 3: sorting and searching**. JSTOR, 1974.

- [9] MORETTIN, P.; Bussab, W. **Estatística básica**. Saraiva Educação SA, 2017.
- [10] REGO, R. **Aplicação combinada do filtro de Bloom com redes neurais recorrentes para detecção de ataques WEB**. Universidade Federal de Santa Maria, 2020.
- [11] ROSER, M.; Ritchie, H.; Ortiz-Ospina, E.; Hasell, J. **Coronavirus pandemic (COVID-19)**. *Our World In Data*, 2020.
- [12] COMUNICAÇÃO, P.; **O Brasil precisa do SUS: uma campanha pela vida**, ENSP/Fiocruz, 2021.