*Universidade Federal do Estado do Rio de Janeiro*
*Centro de Ciências Exatas e Tecnologia*
*Escola de Informática Aplicada*
*Bacharelado em Sistemas de Informação*
*Estruturas de Dados II - 2022.2*
*Priscilla Aparecida Matias de Souza*

---

# Tabelas Hash

## O que é uma Tabela Hash

Tabelas Hash são um tipo de estrutura de dados que armazenam dados de forma associativa. Ela mapeia chaves a valores utilizando funções hash. Nelas, os dados são armazenados no formato de uma array e o valor de cada um possui o seu próprio índice (chave). Nessas tabelas, a inserção de dados é uma operação bastante rápida independente do tamanho dos dados que estão sendo adicionados. A busca de dados pode ser rápida se o índice do valor a ser procurado já é conhecido.

## Papel da Função Hash nos algoritmos de Tabelas Hash

Hashing é uma técnica utilizada para mapear chaves e valores em uma Tabela Hash através do uso de uma função hash com o intuito de promover um acesso mais rápido aos elementos da tabela. Assim, uma função hash gera um mapeamento entre uma chave e um valor. Entende-se que a eficiência do mapeamento depende da eficiência da Função Hash que está sendo utilizada.
Com Hashing, é possível armazenar, remover e recuperar valores em tempo constante, ou seja, com uma complexidade O(1).

## Propriedades de uma boa função Hash

Uma boa função hash tem que ser capaz de ser computada com eficiência, ou seja, sua complexidade deve ser a melhor possível. Além disso, precisa distribuir as chaves inseridas de maneira uniforme - cada posição da tabela deve armazenar apenas um valor, fazendo assim com que cada índice seja único, original.
Uma função hash também deve evitar colisões. Ou seja, como o processo de hashing gera um número pequeno para uma chave grande, existe a possibilidade de

que duas chaves venham a gerar o mesmo valor. Assim, deve-se evitar esses cenários em que o valor a ser mapeado é inserido em um índice que já está ocupado.

Por fim, uma função hash deve possuir um fator de carga (load factor) baixo. O fator de carga de uma tabela hash é definido como o número de itens que existem na tabela dividido pelo tamanho dela. Esse é o fator decisivo quando se quer fazer um “rehash” da função hash anterior ou inserir mais elementos dentro de uma tabela hash já existente. O fator de carga é o que determina se a função hash que está sendo utilizada está distribuindo chave de forma uniforme ou não na tabela, determinando assim a eficiência dessa função.

## Diferença entre resolução de colisão por encadeamento e por probing (sondagem) - Quais as vantagens das duas estratégias com relação uma à outra?

Entende-se que a possibilidade de haver uma colisão em uma tabela existe mesmo que ela seja grande. Assim, existem dois métodos de tratar colisões. O primeiro é por encadeamento, sendo um dos meios mais populares para resolver colisões. O conceito de encadeamento é implementado através de uma lista encadeada que é denominada corrente. Quando existem vários elementos que foram mapeados a um mesmo índice, os inserimos nessa lista encadeada. Então, utilizamos uma chave K para percorrer a lista encadeada de forma linear. Se em algum momento a chave de algum dos valores da lista for igual a K, então entende-se que a entrada foi encontrada. Se caso a lista for percorrida por completo e ainda assim o elemento equivalente a K não foi encontrado, então entende-se que essa entrada não existe naquele índice. Então, através desse método de resolução de colisão, guardamos todos os elementos diferentes com o mesmo valor hash em uma lista encadeada.

A principal vantagem desse método é a sua fácil implementação. Além disso, a tabela hash nunca vai ficar cheia, já que pode-se continuar adicionando elementos na lista encadeada. Sendo uma resolução menor sensível a função hash ou fatores de carga, ela acaba sendo mais utilizada quando não se conhece a quantidade e a frequência de chaves que serão adicionadas ou removidas da tabela hash.

O segundo método para tratamento de colisões é o endereçamento aberto através da sondagem (probing), em que todos os elementos são inseridos na própria tabela hash sem nenhuma lista encadeada para auxiliar. Dessa forma, em algum momento,

o tamanho da tabela vai acabar por ser maior ou igual ao número total de chaves. Existem vários tipos de probing. Por exemplo, na inserção, deve-se sondar a tabela até que um índice vazio seja encontrado para que possa inserir o novo elemento.
Um dos meios de tratar as colisões é através da sondagem linear (linear probing) em que a tabela hash é percorrida de forma sequencial, começando da localização original do hash. Se a localização que for retornada estiver ocupada, checa-se a próxima localização. Isso pode se tornar um problema se existirem vários elementos consecutivos que acabam formando grupos, aumentando o tempo para encontrar um índice vazio ou também de busca de um elemento. Ainda assim, essa forma de tratamento de colisões possui uma melhor performance de cache, já que todos os elementos são armazenados na mesma tabela apenas, sem auxílio de listas encadeadas auxiliares - não há uso de ponteiros. É importante ressaltar que a sondagem também pode ser quadrática (quadratic probing), fazendo assim com que exista um maior número de estratégias para resolver as colisões, variando de acordo com as especificações de cada caso.

## Por que não se utiliza complexidade de pior caso ao se estudar o desempenho de tabelas hash?

O pior caso de complexidade de uma tabela hash seria O(n) para várias operações: busca, inserção e remoção. No pior caso de busca, estaríamos lidando com uma tabela hash completamente cheia. Se estivermos tratando as possíveis colisões por encadeamento, uma lista encadeada vai armazenar todos os elementos com o mesmo índice nela. Essa lista pode ter n elementos dentro dela e isso vai fazer com que o algoritmo de busca itere por todo o seu comprimento, já que todos os elementos precisam ser verificados. Lembrando que, caso seja aplicada uma busca binária, o pior caso é O(log n). Se a sondagem estiver sendo usada como método de tratamento de colisões, a tabela hash terá que ser percorrida por completo para que possamos checar um elemento por vez.
Os piores casos de inserção e remoção também sofrem do mesmo problema, já que, no caso da inserção, é preciso iterar por toda a tabela até encontrar um índice vazio em que o novo elemento possa ser inserido. O pior caso da remoção também seria percorrer toda a tabela e encontrar o elemento que se quer remover somente no último índice da mesma.

## Bibliografia


ROBERT SEDGEWICK. **Algorithms,** 2015. Hash Tables. Disponível em: https://algs4.cs.princeton.edu/34hash/. Acesso em: 31 jan. 2023.

NELSON CRUZ SAMPAIO NETO. Tabelas de Dispersão (ou Hash), 2016. Disponível em: https://www2.unifap.br/furtado/files/2016/11/Aula7.pdf. Acesso em: 31 jan. 2023.

GEEKSFORGEEKS. **GeeksForGeeks**, 2023. Hashing Data Structure. Disponível em: https://www.geeksforgeeks.org/hashing-data-structure/. Acesso em: 31 jan. 2023.

GEEKSFORGEEKS. **GeeksForGeeks**, 2023. Hash Functions and list/types of Hash functions. Disponível em: https://www.geeksforgeeks.org/hash-functions-and-list-types-of-hash-functions/. Acesso em: 31 jan. 2023.

GEEKSFORGEEKS. **GeeksForGeeks**, 2023. Introduction to Hashing – Data Structure and Algorithm Tutorials. Disponível em: https://www.geeksforgeeks.org/introduction-to-hashing-data-structure-and-algorithm-tutorials/. Acesso em: 31 jan. 2023.

GEEKSFORGEEKS. **GeeksForGeeks**, 2023. Separate Chaining Collision Handling Technique in Hashing Disponível em: https://www.geeksforgeeks.org/separate-chaining-collision-handling-technique-in-hashing/. Acesso em: 3 fev. 2023.

GEEKSFORGEEKS. **GeeksForGeeks**, 2023. Open Addressing Collision Handling technique in Hashing. Disponível em: https://www.geeksforgeeks.org/open-addressing-collision-handling-technique-in-hashing/. Acesso em: 3 fev. 2023.

SAYLOR.ORG. **Saylor.Org**. Searching and Hashing. Disponível em: https://learn.saylor.org/mod/book/view.php?id=32990&chapterid=12838#. Acesso em: 3 fev. 2023.