# TABELAS HASH COM ENDEREÇAMENTO ABERTO DCE792 - AEDs II (Prática)

Atualizado em: 25 de novembro de 2024

Iago Carvalho

Departamento de Ciência da Computação



#### TABELA HASH

Tabelas hash também são conhecidas como tabelas de espalhamento, tabelas de indexação ou mapas

Uma tabela hash é uma estrutura de dados que possibilita o acesso a seus registros de forma immediata

O(1)

Além disso, a inserção e remoção de registros também de forma imediata - O(1)

Claro, tudo isto na teoria

O Na prática, pode haver diferenças

2

#### TABELA HASH

Na aula passada nós vimos uma maneira de resolução de conflitos em tabelas hash

Uso de listas lineares

A ideia é interessante, mas ocasiona o uso de memória adicional

 Além disso, também pode ocorrer a degeneração da tabela para uma lista linear

Para superar estes pontos fracos, podemos desenvolver uma tabela hash que tem resolução de conflitos por **endereçamento aberto** 

3

A ideia é procurar um outro índice livre da tabela para armazenar o elemento quando houver colisão

Existem diversas estratégias para procurar esse índice

- O mais simples deles é o hashing linear
- Busca a próxima posição vazia
- O Considerar a tabela hash como uma lista circular

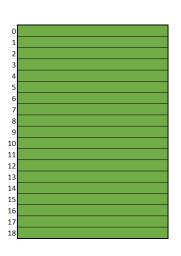
Este método preenche a tabela hash com 3 diferentes tipos de entrada

- Vazio
- Removido
- Preenchido

Considere a tabela hash ao lado

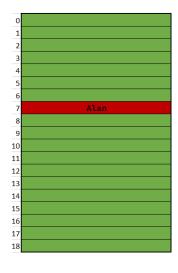
- 19 posições
- Todas as posições vazias

Vamos começar a inserir elementos em nossa tabela



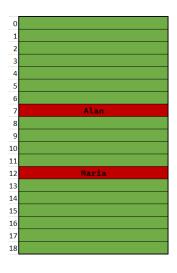
Vamos inserir a chave Alan

f(Alan) = 7



Vamos inserir a chave Maria

 $\bigcirc$  k(Maria) = 12



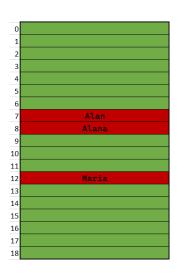
Vamos inserir a chave Alana

 $\bigcirc$  k(Alana) = 7

A posição 7 já está ocupada

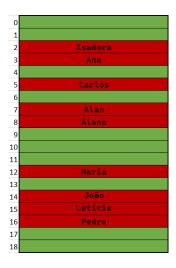
Vamos tentar inserir *Alana* na próxima posição

Sucesso! Inserida na posição 8



Vamos inserir mais algumas chaves em nossa tabela hash

- Isadora
- Ana
- Carlos
- João
- Letícia
- Pedro



Vamos inserir a chave Paulo

- $\bigcirc$  k(Paulo) = 15
- Posição 15 já está ocupada por Letícia

Vamos tentar inserir *Paulo* na próxima posição

 Falha... Posição 16 já está ocupada por *Pedro*

Vamos tentar inserir *Paulo* na próxima posição

O Sucesso! Inserido na posição 17



Vamos remover a chave *Pedro* 

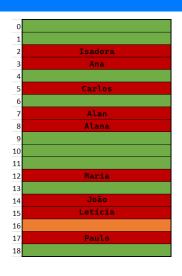
- $\bigcirc$  k(Pedro) = 14
- O Posição 14 é ocupada por *João*

Vamos procurar *Pedro* na próxima posição

 Falha... Posição 15 está ocupada por Letícia

Vamos procurar *Pedro* na próxima posição

- Sucesso! Pedro foi encontrado na posição 16
- Posição 16 é marcada como removida



Vamos remover a chave Júlia

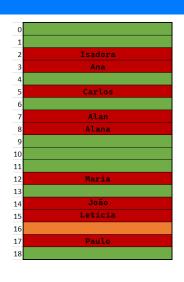
- $\bigcirc$  k(Julia) = 16
- Posição 16 está marcada como removida

Vamos procurar *Júlia* na próxima posição

Falha... Posição 17 está ocupada

Vamos procurar *Júlia* na próxima posição

- Falha... Posição 18 está marcada como vazia
- A chave Júlia nunca fez parte de nossa tabela hash
- Busca retorna inexistente

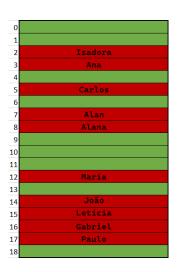


Vamos inserir a chave Gabriel

- $\bigcirc$  k(Gabriel) = 15
- Posição 15 está ocupada por Letícia

Vamos tentar inserir *Gabriel* na próxima posição

 Sucesso! Gabriel pode ser inserido na posição 16



# INSERÇÕES E REMOÇÕES

Como visto no exemplo, a remoção (ou busca) procura linearmente todas as posições até encontrar

- Uma entrada vazia
  - Retorna que a chave é inexistente
- A chave desejada
  - Marca a posição como removida

Já a inserção procura linearmente pela próxima posição vazia ou removida

- Insere na posição alvo
- Marca a posição como preenchida

# DETALHES DE IMPLEMENTAÇÃO

Necessário criar um contador de posições preenchidas

○ Contador vai de zero até *M* (tamanho da hash)

Caso este contador seja igual a *M*, não deve-se permitir inserções

Já na remoção e na busca, cria-se outro contador que indica o número de casas testadas

- Imagine uma tabela hash cheia e que não possui a chave desejada
- Caso este novo contador chegue a M, então a chave não está presente na tabela hash



#### **ATIVIDADE**

Implementar uma heap com este método de resolução de colisão

O código base está disponível no Github

- Mesmo código da aula anterior
- $\bigcirc$  Ele implements uma função hash f(k) = k%M
- O Possui funções de inserção, pesquisa e remoção
- Não possui tratamento de colisão

# VAMOS CONTINUAR NOSSOS

ESTUDOS EM AEDS III

PRÓXIMO SEMESTRE: