

# MC-202

## Tabela de Espalhamento

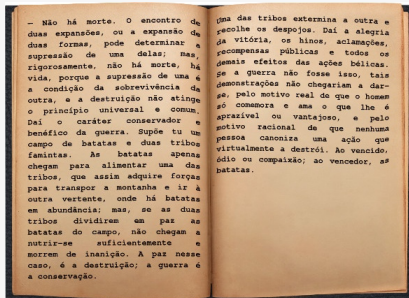
Lehilton Pedrosa  
lehilton@ic.unicamp.br

Universidade Estadual de Campinas

Segundo semestre de 2024

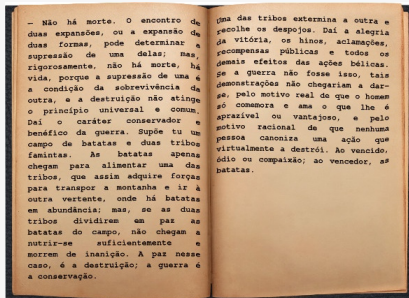
# Introdução

Queremos contar o número de ocorrências de cada palavra da biblioteca



# Introdução

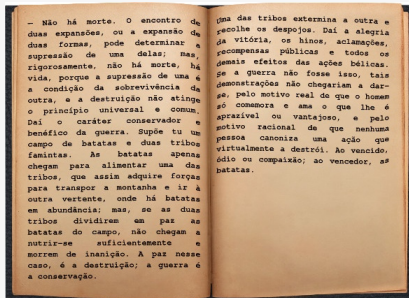
Queremos contar o número de ocorrências de cada palavra da biblioteca



- no idioma, há cerca de milhares de palavras ( $\approx 435.000$ )

# Introdução

Queremos contar o número de ocorrências de cada palavra da biblioteca



- no idioma, há cerca de milhares de palavras ( $\approx 435.000$ )
- mas no total, há milhões de ocorrências!

## Exemplo

dia:	6 ocorrências
escola:	13 ocorrências
gratuito:	1 ocorrência
ilha:	8 ocorrências
jeito:	5 ocorrências
lata:	2 ocorrências

## Exemplo

dia:	6 ocorrências
escola:	13 ocorrências
gratuito:	1 ocorrência
ilha:	8 ocorrências
jeito:	5 ocorrências
lata:	2 ocorrências

Queremos acessar uma palavra como se fosse um vetor:

```
ocorrencias["ilha"] = 8
```

## Exemplo

dia:	6 ocorrências
escola:	13 ocorrências
gratuito:	1 ocorrência
ilha:	8 ocorrências
jeito:	5 ocorrências
lata:	2 ocorrências

Queremos acessar uma palavra como se fosse um vetor:

```
ocorrencias["ilha"] = 8
```

Primeiras opções:

## Exemplo

dia:	6 ocorrências
escola:	13 ocorrências
gratuito:	1 ocorrência
ilha:	8 ocorrências
jeito:	5 ocorrências
lata:	2 ocorrências

Queremos acessar uma palavra como se fosse um vetor:

```
ocorrencias["ilha"] = 8
```

Primeiras opções:

- Vetor - acesso/escrita em  $O(n)$



## Exemplo

dia:	6 ocorrências
escola:	13 ocorrências
gratuito:	1 ocorrência
ilha:	8 ocorrências
jeito:	5 ocorrências
lata:	2 ocorrências

Queremos acessar uma palavra como se fosse um vetor:

```
ocorrencias["ilha"] = 8
```

Primeiras opções:

- Vetor - acesso/escrita em  $O(n)$ 
  - inserir uma nova palavra leva  $O(1)$

## Exemplo

dia:	6 ocorrências
escola:	13 ocorrências
gratuito:	1 ocorrência
ilha:	8 ocorrências
jeito:	5 ocorrências
lata:	2 ocorrências

Queremos acessar uma palavra como se fosse um vetor:

```
ocorrencias["ilha"] = 8
```

Primeiras opções:

- Vetor - acesso/escrita em  $O(n)$ 
  - inserir uma nova palavra leva  $O(1)$
- Vetor ordenado - acesso/escrita em  $O(\lg n)$

## Exemplo

dia:	6 ocorrências
escola:	13 ocorrências
gratuito:	1 ocorrência
ilha:	8 ocorrências
jeito:	5 ocorrências
lata:	2 ocorrências

Queremos acessar uma palavra como se fosse um vetor:

```
ocorrencias["ilha"] = 8
```

Primeiras opções:

- Vetor - acesso/escrita em  $O(n)$ 
  - inserir uma nova palavra leva  $O(1)$
- Vetor ordenado - acesso/escrita em  $O(\lg n)$ 
  - inserir uma nova palavra leva  $O(n)$

## Exemplo

dia:	6 ocorrências
escola:	13 ocorrências
gratuito:	1 ocorrência
ilha:	8 ocorrências
jeito:	5 ocorrências
lata:	2 ocorrências

Queremos acessar uma palavra como se fosse um vetor:

`ocorrencias["ilha"] = 8`

Primeiras opções:

- Vetor - acesso/escrita em  $O(n)$ 
  - inserir uma nova palavra leva  $O(1)$
- Vetor ordenado - acesso/escrita em  $O(\lg n)$ 
  - inserir uma nova palavra leva  $O(n)$
- ABB balanceada - acesso/escrita/inserção em  $O(\lg n)$

## Exemplo

dia:	6 ocorrências
escola:	13 ocorrências
gratuito:	1 ocorrência
ilha:	8 ocorrências
jeito:	5 ocorrências
lata:	2 ocorrências

Queremos acessar uma palavra como se fosse um vetor:

`ocorrencias["ilha"] = 8`

Primeiras opções:

- Vetor - acesso/escrita em  $O(n)$ 
  - inserir uma nova palavra leva  $O(1)$
- Vetor ordenado - acesso/escrita em  $O(\lg n)$ 
  - inserir uma nova palavra leva  $O(n)$
- ABB balanceada - acesso/escrita/inserção em  $O(\lg n)$

Conseguimos fazer em  $O(1)$ ?

# Caso fácil

Se tivéssemos apenas uma palavra começando com cada letra era fácil

a	0	
b	1	
c	2	
d	3	dia
e	4	escola
f	5	
g	6	gratuito
h	7	
i	8	ilha
⋮	⋮	⋮

## Caso fácil

Se tivéssemos apenas uma palavra começando com cada letra era fácil

- bastaria ter um vetor de **26** posições

a	0	
b	1	
c	2	
d	3	dia
e	4	escola
f	5	
g	6	gratuito
h	7	
i	8	ilha
⋮	⋮	⋮

## Palavras que começam com a mesma letra

0	→	NULL
1	→	NULL
2	→	NULL
3	→	dia
4	→	escola
5	→	NULL
6	→	gratuito
7	→	NULL
8	→	ilha
⋮	⋮	⋮



# Palavras que começam com a mesma letra



Ideia:

# Palavras que começam com a mesma letra



Ideia:

- uma lista ligada para cada letra

# Palavras que começam com a mesma letra



Ideia:

- uma lista ligada para cada letra
- guardamos os ponteiros para as listas em um vetor

## Palavras que começam com a mesma letra

0	→	NULL
1	→	NULL
2	→	NULL
3	→	dia
4	→	escola
5	→	NULL
6	→	gratuito
7	→	NULL
8	→	ilha
⋮	⋮	⋮

# Palavras que começam com a mesma letra



Inserindo “bala”:

## Palavras que começam com a mesma letra

0	→	NULL
1	→	NULL
2	→	NULL
3	→	dia
4	→	escola
5	→	NULL
6	→	gratuito
7	→	NULL
8	→	ilha
⋮	⋮	⋮

Inserindo “bala”:

- descobrimos a posição pela primeira letra

## Palavras que começam com a mesma letra

0	→	NULL
<b>1</b>	→	NULL
2	→	NULL
3	→	dia
4	→	escola
5	→	NULL
6	→	gratuito
7	→	NULL
8	→	ilha
⋮	⋮	⋮

Inserindo “bala”:

- descobrimos a posição pela primeira letra
- atualizamos o vetor para apontar para o nó de “bala”

## Palavras que começam com a mesma letra

0	→	NULL
1	→	bala
2	→	NULL
3	→	dia
4	→	escola
5	→	NULL
6	→	gratuito
7	→	NULL
8	→	ilha
⋮	⋮	⋮

Inserindo “bala”:

- descobrimos a posição pela primeira letra
- atualizamos o vetor para apontar para o nó de “bala”



# Palavras que começam com a mesma letra

0	→	NULL
1	→	bala
2	→	NULL
3	→	dia
4	→	escola
5	→	NULL
6	→	gratuito
7	→	NULL
8	→	ilha
⋮	⋮	⋮

# Palavras que começam com a mesma letra



Inserindo “bela”:

## Palavras que começam com a mesma letra

0	→	NULL
1	→	bala
2	→	NULL
3	→	dia
4	→	escola
5	→	NULL
6	→	gratuito
7	→	NULL
8	→	ilha
⋮	⋮	⋮

Inserindo “bela”:

- descobrimos a posição pela primeira letra

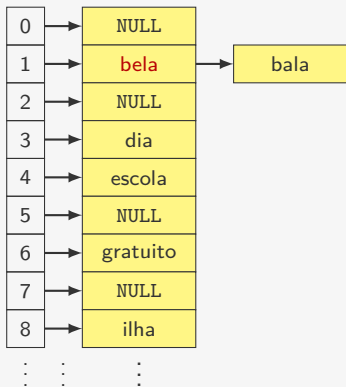
## Palavras que começam com a mesma letra

0	→	NULL
1	→	bala
2	→	NULL
3	→	dia
4	→	escola
5	→	NULL
6	→	gratuito
7	→	NULL
8	→	ilha
⋮	⋮	⋮

Inserindo “bela”:

- descobrimos a posição pela primeira letra
- temos uma **colisão** com “bala”

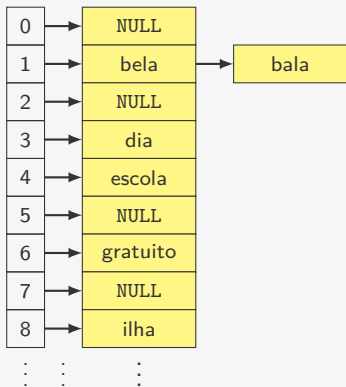
## Palavras que começam com a mesma letra



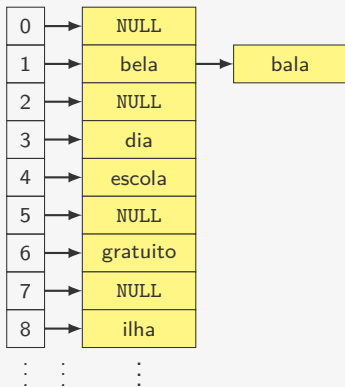
Inserindo “bela”:

- descobrimos a posição pela primeira letra
- temos uma **colisão** com “bala”
- inserimos no começo da lista da letra **b**

# Palavras que começam com a mesma letra

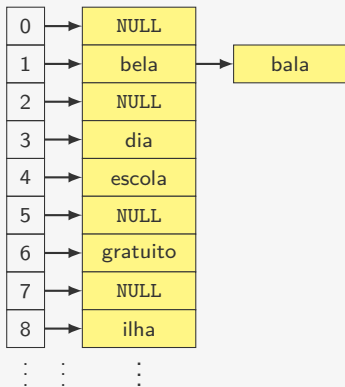


## Palavras que começam com a mesma letra



Após a inserção de várias palavras começando com **b**:

## Palavras que começam com a mesma letra

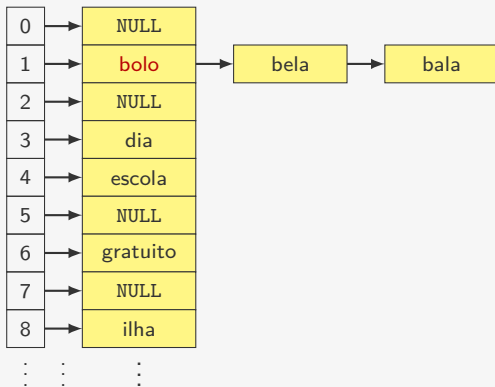


Após a inserção de várias palavras começando com **b**:

- inserimos “bolo”,



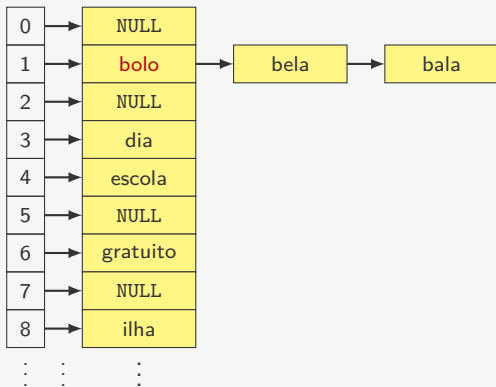
## Palavras que começam com a mesma letra



Após a inserção de várias palavras começando com **b**:

- inserimos “bolo”,

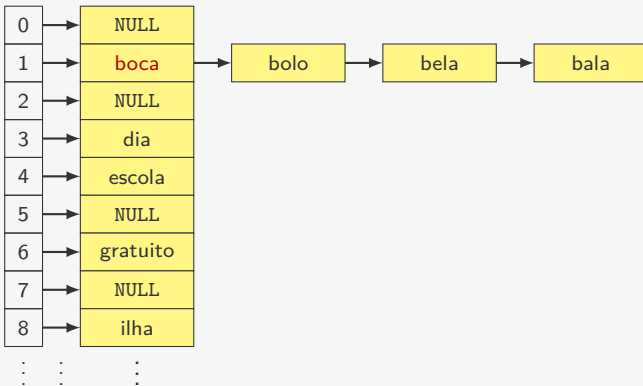
# Palavras que começam com a mesma letra



Após a inserção de várias palavras começando com **b**:

- inserimos “bolo”, “boca”,

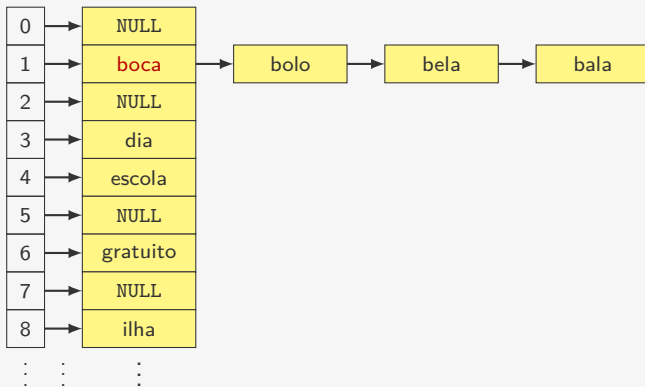
## Palavras que começam com a mesma letra



Após a inserção de várias palavras começando com **b**:

- inserimos “bolo”, “boca”,

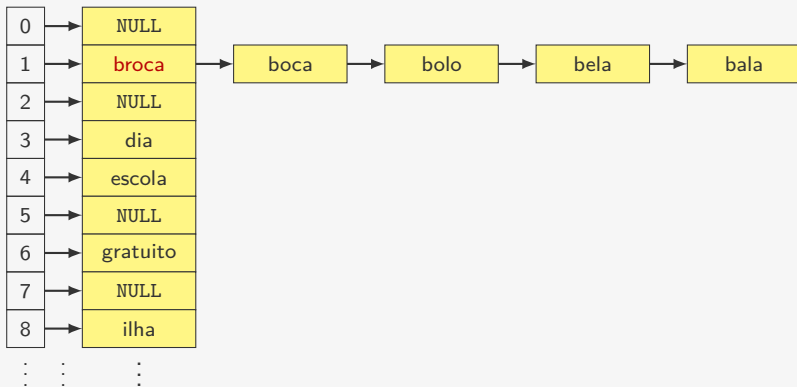
## Palavras que começam com a mesma letra



Após a inserção de várias palavras começando com **b**:

- inserimos “bolo”, “boca”, “broca”

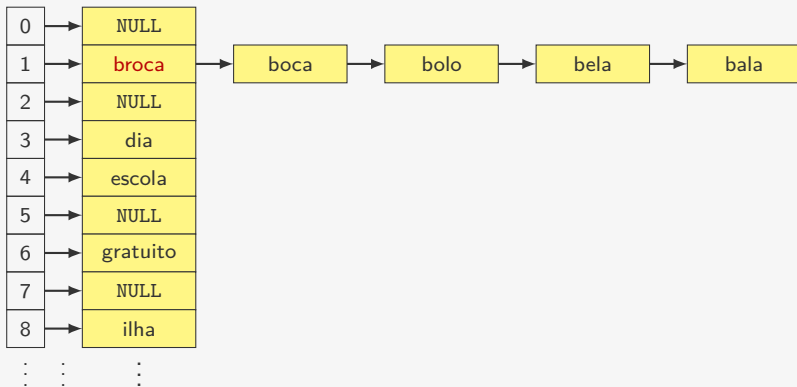
## Palavras que começam com a mesma letra



Após a inserção de várias palavras começando com **b**:

- inserimos “bolo”, “boca”, “broca”

## Palavras que começam com a mesma letra



Após a inserção de várias palavras começando com **b**:

- inserimos “bolo”, “boca”, “broca”
- a tabela ficou **degenerada em lista**

# Espalhamento com Encadeamento Separado

broca

boca

bolo

bela

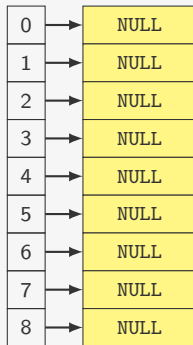
bala

dia

escola

gratuito

ilha



Corrigindo:

# Espalhamento com Encadeamento Separado

broca

boca

bolo

bela

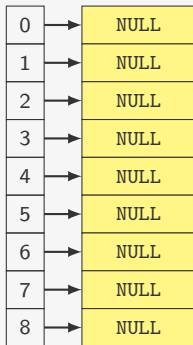
bala

dia

escola

gratuito

ilha



Corrigindo:

- vamos tentar **espalhar** melhor



# Espalhamento com Encadeamento Separado

broca

boca

bolo

bela

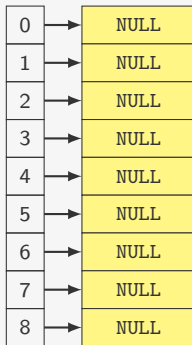
bala

dia

escola

gratuito

ilha



Corrigindo:

- vamos tentar **espalhar** melhor
- usamos um **hash** da chave (palavra)

# Espalhamento com Encadeamento Separado

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca

bolo

bela

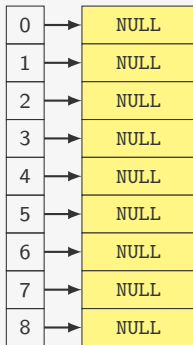
bala

dia

escola

gratuito

ilha



Corrigindo:

- vamos tentar **espalhar** melhor
- usamos um **hash** da chave (palavra)
- vamos associar a chave a um número inteiro (entre 0 e 8)

# Espalhamento com Encadeamento Separado

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca

bolo

bela

bala

dia

escola

gratuito

ilha

0	→	NULL
1	→	NULL
2	→	NULL
3	→	broca
4	→	NULL
5	→	NULL
6	→	NULL
7	→	NULL
8	→	NULL

Corrigindo:

- vamos tentar **espalhar** melhor
- usamos um **hash** da chave (palavra)
- vamos associar a chave a um número inteiro (entre 0 e 8)

# Espalhamento com Encadeamento Separado

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo

bela

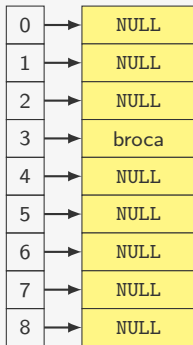
bala

dia

escola

gratuito

ilha



Corrigindo:

- vamos tentar **espalhar** melhor
- usamos um **hash** da chave (palavra)
- vamos associar a chave a um número inteiro (entre 0 e 8)

# Espalhamento com Encadeamento Separado

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo

bela

bala

dia

escola

gratuito

ilha

0	→	boca
1	→	NULL
2	→	NULL
3	→	broca
4	→	NULL
5	→	NULL
6	→	NULL
7	→	NULL
8	→	NULL

Corrigindo:

- vamos tentar **espalhar** melhor
- usamos um **hash** da chave (palavra)
- vamos associar a chave a um número inteiro (entre 0 e 8)

# Espalhamento com Encadeamento Separado

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela

bala

dia

escola

gratuito

ilha

0	→	boca
1	→	NULL
2	→	NULL
3	→	broca
4	→	NULL
5	→	NULL
6	→	NULL
7	→	NULL
8	→	NULL

Corrigindo:

- vamos tentar **espalhar** melhor
- usamos um **hash** da chave (palavra)
- vamos associar a chave a um número inteiro (entre 0 e 8)

# Espalhamento com Encadeamento Separado

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela

bala

dia

escola

gratuito

ilha

0	→	boca
1	→	NULL
2	→	NULL
3	→	broca
4	→	NULL
5	→	bolo
6	→	NULL
7	→	NULL
8	→	NULL

Corrigindo:

- vamos tentar **espalhar** melhor
- usamos um **hash** da chave (palavra)
- vamos associar a chave a um número inteiro (entre 0 e 8)

# Espalhamento com Encadeamento Separado

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela  $\rightsquigarrow$   $h(\text{"bela"}) = 2$

bala

dia

escola

gratuito

ilha

0	→	boca
1	→	NULL
2	→	NULL
3	→	broca
4	→	NULL
5	→	bolo
6	→	NULL
7	→	NULL
8	→	NULL

Corrigindo:

- vamos tentar **espalhar** melhor
- usamos um **hash** da chave (palavra)
- vamos associar a chave a um número inteiro (entre 0 e 8)



# Espalhamento com Encadeamento Separado

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela  $\rightsquigarrow$   $h(\text{"bela"}) = 2$

bala

dia

escola

gratuito

ilha

0	→	boca
1	→	NULL
2	→	bela
3	→	broca
4	→	NULL
5	→	bolo
6	→	NULL
7	→	NULL
8	→	NULL

Corrigindo:

- vamos tentar **espalhar** melhor
- usamos um **hash** da chave (palavra)
- vamos associar a chave a um número inteiro (entre 0 e 8)

# Espalhamento com Encadeamento Separado

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela  $\rightsquigarrow$   $h(\text{"bela"}) = 2$

bala  $\rightsquigarrow$   $h(\text{"bala"}) = 0$

dia

escola

gratuito

ilha

0	→	boca
1	→	NULL
2	→	bela
3	→	broca
4	→	NULL
5	→	bolo
6	→	NULL
7	→	NULL
8	→	NULL

Corrigindo:

- vamos tentar **espalhar** melhor
- usamos um **hash** da chave (palavra)
- vamos associar a chave a um número inteiro (entre 0 e 8)

# Espalhamento com Encadeamento Separado

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela  $\rightsquigarrow$   $h(\text{"bela"}) = 2$

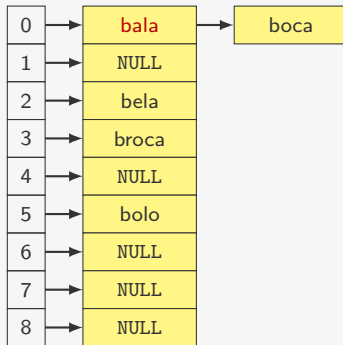
bala  $\rightsquigarrow$   $h(\text{"bala"}) = 0$

dia

escola

gratuito

ilha



Corrigindo:

- vamos tentar **espalhar** melhor
- usamos um **hash** da chave (palavra)
- vamos associar a chave a um número inteiro (entre 0 e 8)

# Espalhamento com Encadeamento Separado

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela  $\rightsquigarrow$   $h(\text{"bela"}) = 2$

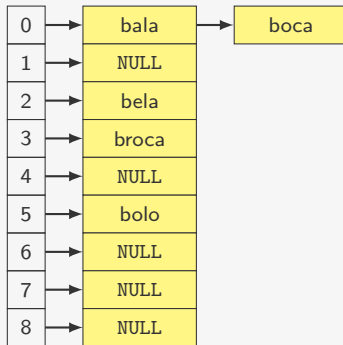
bala  $\rightsquigarrow$   $h(\text{"bala"}) = 0$

dia  $\rightsquigarrow$   $h(\text{"dia"}) = 2$

escola

gratuito

ilha



Corrigindo:

- vamos tentar **espalhar** melhor
- usamos um **hash** da chave (palavra)
- vamos associar a chave a um número inteiro (entre 0 e 8)

# Espalhamento com Encadeamento Separado

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela  $\rightsquigarrow$   $h(\text{"bela"}) = 2$

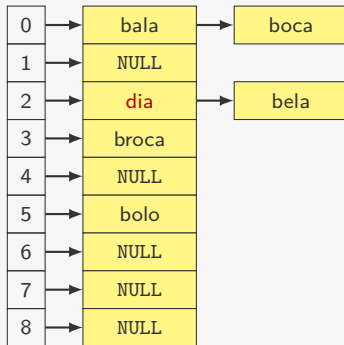
bala  $\rightsquigarrow$   $h(\text{"bala"}) = 0$

dia  $\rightsquigarrow$   $h(\text{"dia"}) = 2$

escola

gratuito

ilha



Corrigindo:

- vamos tentar **espalhar** melhor
- usamos um **hash** da chave (palavra)
- vamos associar a chave a um número inteiro (entre 0 e 8)

# Espalhamento com Encadeamento Separado

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela  $\rightsquigarrow$   $h(\text{"bela"}) = 2$

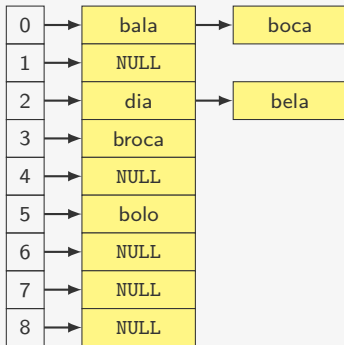
bala  $\rightsquigarrow$   $h(\text{"bala"}) = 0$

dia  $\rightsquigarrow$   $h(\text{"dia"}) = 2$

escola  $\rightsquigarrow$   $h(\text{"escola"}) = 7$

gratuito

ilha



Corrigindo:

- vamos tentar **espalhar** melhor
- usamos um **hash** da chave (palavra)
- vamos associar a chave a um número inteiro (entre 0 e 8)

# Espalhamento com Encadeamento Separado

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela  $\rightsquigarrow$   $h(\text{"bela"}) = 2$

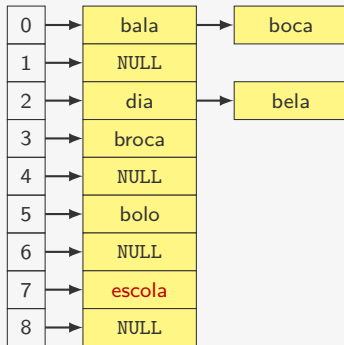
bala  $\rightsquigarrow$   $h(\text{"bala"}) = 0$

dia  $\rightsquigarrow$   $h(\text{"dia"}) = 2$

escola  $\rightsquigarrow$   $h(\text{"escola"}) = 7$

gratuito

ilha



Corrigindo:

- vamos tentar **espalhar** melhor
- usamos um **hash** da chave (palavra)
- vamos associar a chave a um número inteiro (entre 0 e 8)

# Espalhamento com Encadeamento Separado

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela  $\rightsquigarrow$   $h(\text{"bela"}) = 2$

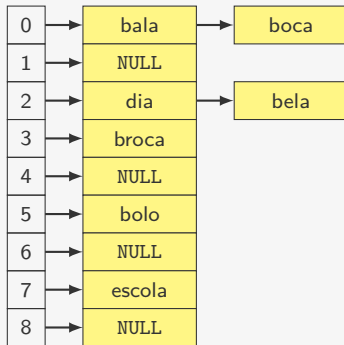
bala  $\rightsquigarrow$   $h(\text{"bala"}) = 0$

dia  $\rightsquigarrow$   $h(\text{"dia"}) = 2$

escola  $\rightsquigarrow$   $h(\text{"escola"}) = 7$

gratuito  $\rightsquigarrow$   $h(\text{"gratuito"}) = 0$

ilha



Corrigindo:

- vamos tentar **espalhar** melhor
- usamos um **hash** da chave (palavra)
- vamos associar a chave a um número inteiro (entre 0 e 8)



# Espalhamento com Encadeamento Separado

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela  $\rightsquigarrow$   $h(\text{"bela"}) = 2$

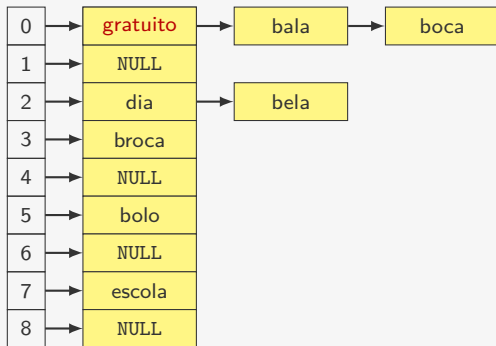
bala  $\rightsquigarrow$   $h(\text{"bala"}) = 0$

dia  $\rightsquigarrow$   $h(\text{"dia"}) = 2$

escola  $\rightsquigarrow$   $h(\text{"escola"}) = 7$

gratuito  $\rightsquigarrow$   $h(\text{"gratuito"}) = 0$

ilha



Corrigindo:

- vamos tentar **espalhar** melhor
- usamos um **hash** da chave (palavra)
- vamos associar a chave a um número inteiro (entre 0 e 8)

# Espalhamento com Encadeamento Separado

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela  $\rightsquigarrow$   $h(\text{"bela"}) = 2$

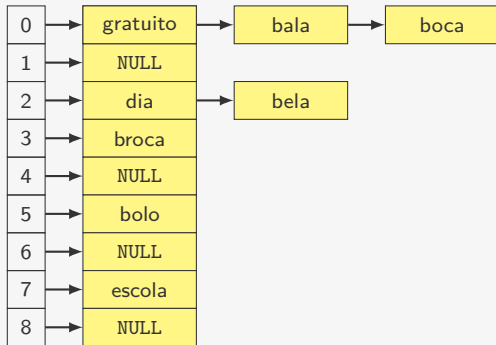
bala  $\rightsquigarrow$   $h(\text{"bala"}) = 0$

dia  $\rightsquigarrow$   $h(\text{"dia"}) = 2$

escola  $\rightsquigarrow$   $h(\text{"escola"}) = 7$

gratuito  $\rightsquigarrow$   $h(\text{"gratuito"}) = 0$

ilha  $\rightsquigarrow$   $h(\text{"ilha"}) = 6$



Corrigindo:

- vamos tentar **espalhar** melhor
- usamos um **hash** da chave (palavra)
- vamos associar a chave a um número inteiro (entre 0 e 8)

# Espalhamento com Encadeamento Separado

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela  $\rightsquigarrow$   $h(\text{"bela"}) = 2$

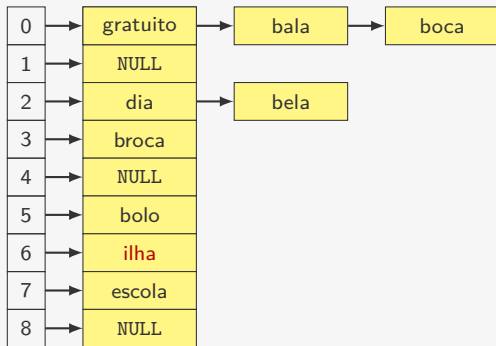
bala  $\rightsquigarrow$   $h(\text{"bala"}) = 0$

dia  $\rightsquigarrow$   $h(\text{"dia"}) = 2$

escola  $\rightsquigarrow$   $h(\text{"escola"}) = 7$

gratuito  $\rightsquigarrow$   $h(\text{"gratuito"}) = 0$

ilha  $\rightsquigarrow$   $h(\text{"ilha"}) = 6$



Corrigindo:

- vamos tentar **espalhar** melhor
- usamos um **hash** da chave (palavra)
- vamos associar a chave a um número inteiro (entre 0 e 8)

# Tabela de Espalhamento

Uma função de hashing associa

# Tabela de Espalhamento

Uma **função de hashing** associa

- um elemento de um conjunto (strings, números, etc.)

# Tabela de Espalhamento

Uma **função de hashing** associa

- um elemento de um conjunto (strings, números, etc.)
- a um **número inteiro** de tamanho conhecido

# Tabela de Espalhamento

Uma **função de hashing** associa

- um elemento de um conjunto (strings, números, etc.)
- a um **número inteiro** de tamanho conhecido

Uma **tabela de espalhamento** é um TAD para conjuntos dinâmicos com certas propriedades:

# Tabela de Espalhamento

Uma **função de hashing** associa

- um elemento de um conjunto (strings, números, etc.)
- a um **número inteiro** de tamanho conhecido

Uma **tabela de espalhamento** é um TAD para conjuntos dinâmicos com certas propriedades:

- os dados são acessado por meio de um vetor de tamanho conhecido



# Tabela de Espalhamento

Uma **função de hashing** associa

- um elemento de um conjunto (strings, números, etc.)
- a um **número inteiro** de tamanho conhecido

Uma **tabela de espalhamento** é um TAD para conjuntos dinâmicos com certas propriedades:

- os dados são acessado por meio de um vetor de tamanho conhecido
- a posição do vetor é calculada por uma função de hashing

# Características das tabelas de espalhamento

Restrições

# Características das tabelas de espalhamento

## Restrições

- estimativa do tamanho do conjunto de dados deve ser conhecida

# Características das tabelas de espalhamento

## Restrições

- estimativa do tamanho do conjunto de dados deve ser conhecida
- bits da chave devem estar disponíveis

# Características das tabelas de espalhamento

## Restrições

- estimativa do tamanho do conjunto de dados deve ser conhecida
- bits da chave devem estar disponíveis
  - em uma ABB, basta uma função de comparação

# Características das tabelas de espalhamento

## Restrições

- estimativa do tamanho do conjunto de dados deve ser conhecida
- bits da chave devem estar disponíveis
  - em uma ABB, basta uma função de comparação

## Tempo das operações

# Características das tabelas de espalhamento

## Restrições

- estimativa do tamanho do conjunto de dados deve ser conhecida
- bits da chave devem estar disponíveis
  - em uma ABB, basta uma função de comparação

## Tempo das operações

- depende principalmente da função de hashing escolhida

# Características das tabelas de espalhamento

## Restrições

- estimativa do tamanho do conjunto de dados deve ser conhecida
- bits da chave devem estar disponíveis
  - em uma ABB, basta uma função de comparação

## Tempo das operações

- depende principalmente da função de hashing escolhida
- chaves bem espalhadas: tempo “quase”  $O(1)$



# Características das tabelas de espalhamento

## Restrições

- estimativa do tamanho do conjunto de dados deve ser conhecida
- bits da chave devem estar disponíveis
  - em uma ABB, basta uma função de comparação

## Tempo das operações

- depende principalmente da função de hashing escolhida
- chaves bem espalhadas: tempo “quase”  $O(1)$ 
  - se temos  $n$  itens

# Características das tabelas de espalhamento

## Restrições

- estimativa do tamanho do conjunto de dados deve ser conhecida
- bits da chave devem estar disponíveis
  - em uma ABB, basta uma função de comparação

## Tempo das operações

- depende principalmente da função de hashing escolhida
- chaves bem espalhadas: tempo “quase”  $O(1)$ 
  - se temos  $n$  itens
  - uma tabela de tamanho  $M$

# Características das tabelas de espalhamento

## Restrições

- estimativa do tamanho do conjunto de dados deve ser conhecida
- bits da chave devem estar disponíveis
  - em uma ABB, basta uma função de comparação

## Tempo das operações

- depende principalmente da função de hashing escolhida
- chaves bem espalhadas: tempo “quase”  $O(1)$ 
  - se temos  $n$  itens
  - uma tabela de tamanho  $M$
  - tempo de acesso é o tempo de calcular a função de hashing mais  $O(n/M)$

# Características das tabelas de espalhamento

## Restrições

- estimativa do tamanho do conjunto de dados deve ser conhecida
- bits da chave devem estar disponíveis
  - em uma ABB, basta uma função de comparação

## Tempo das operações

- depende principalmente da função de hashing escolhida
- chaves bem espalhadas: tempo “quase”  $O(1)$ 
  - se temos  $n$  itens
  - uma tabela de tamanho  $M$
  - tempo de acesso é o tempo de calcular a função de hashing mais  $O(n/M)$
- chaves muito agrupadas: pior caso de tempo  $O(n)$

# Características das tabelas de espalhamento

## Restrições

- estimativa do tamanho do conjunto de dados deve ser conhecida
- bits da chave devem estar disponíveis
  - em uma ABB, basta uma função de comparação

## Tempo das operações

- depende principalmente da função de hashing escolhida
- chaves bem espalhadas: tempo “quase”  $O(1)$ 
  - se temos  $n$  itens
  - uma tabela de tamanho  $M$
  - tempo de acesso é o tempo de calcular a função de hashing mais  $O(n/M)$
- chaves muito agrupadas: pior caso de tempo  $O(n)$ 
  - vira uma lista ligada com todos os elementos

## Obtendo funções de hashing

Uma boa função de hashing deve espalhar bem:

## Obtendo funções de hashing

Uma boa função de hashing deve espalhar bem:

- A probabilidade de uma chave ter um hash específico é (aproximadamente)  $1/M$

## Obtendo funções de hashing

Uma boa função de hashing deve espalhar bem:

- A probabilidade de uma chave ter um hash específico é (aproximadamente)  $1/M$
- Ou seja, esperamos que cada lista tenha  $n/M$  elementos



## Obtendo funções de hashing

Uma boa função de hashing deve espalhar bem:

- A probabilidade de uma chave ter um hash específico é (aproximadamente)  $1/M$
- Ou seja, esperamos que cada lista tenha  $n/M$  elementos

Métodos genéricos (que funcionam bem na prática):

# Obtendo funções de hashing

Uma boa função de hashing deve espalhar bem:

- A probabilidade de uma chave ter um hash específico é (aproximadamente)  $1/M$
- Ou seja, esperamos que cada lista tenha  $n/M$  elementos

Métodos genéricos (que funcionam bem na prática):

1. Método da divisão

## Obtendo funções de hashing

Uma boa função de hashing deve espalhar bem:

- A probabilidade de uma chave ter um hash específico é (aproximadamente)  $1/M$
- Ou seja, esperamos que cada lista tenha  $n/M$  elementos

Métodos genéricos (que funcionam bem na prática):

1. Método da divisão
2. Método da multiplicação

# Obtendo funções de hashing

Uma boa função de hashing deve espalhar bem:

- A probabilidade de uma chave ter um hash específico é (aproximadamente)  $1/M$
- Ou seja, esperamos que cada lista tenha  $n/M$  elementos

Métodos genéricos (que funcionam bem na prática):

1. Método da divisão
2. Método da multiplicação

**Hashing perfeito:** Se conhecermos todas as chaves a priori, é possível encontrar uma função de hashing injetora

# Obtendo funções de hashing

Uma boa função de hashing deve espalhar bem:

- A probabilidade de uma chave ter um hash específico é (aproximadamente)  $1/M$
- Ou seja, esperamos que cada lista tenha  $n/M$  elementos

Métodos genéricos (que funcionam bem na prática):

1. Método da divisão
2. Método da multiplicação

**Hashing perfeito:** Se conhecermos todas as chaves a priori, é possível encontrar uma função de hashing injetora

- isto é, não temos colisões

# Obtendo funções de hashing

Uma boa função de hashing deve espalhar bem:

- A probabilidade de uma chave ter um hash específico é (aproximadamente)  $1/M$
- Ou seja, esperamos que cada lista tenha  $n/M$  elementos

Métodos genéricos (que funcionam bem na prática):

1. Método da divisão
2. Método da multiplicação

**Hashing perfeito:** Se conhecermos todas as chaves a priori, é possível encontrar uma função de hashing injetora

- isto é, não temos colisões
- tais funções podem ser difíceis de encontrar

## Interpretando chaves

Pressupomos que as chaves são **números inteiros**

## Interpretando chaves

Pressupomos que as chaves são **números inteiros**

E se não forem?



## Interpretando chaves

Pressupomos que as chaves são **números inteiros**

E se não forem?

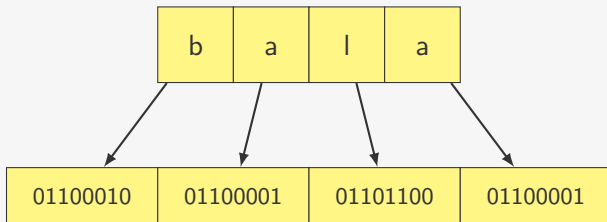
- Reinterpretamos a chave como uma sequência de bits

## Interpretando chaves

Pressupomos que as chaves são **números inteiros**

E se não forem?

- Reinterpretamos a chave como uma sequência de bits

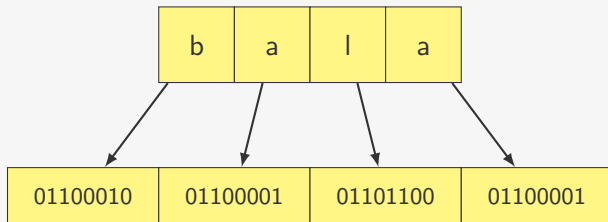


## Interpretando chaves

Pressupomos que as chaves são **números inteiros**

E se não forem?

- Reinterpretamos a chave como uma sequência de bits



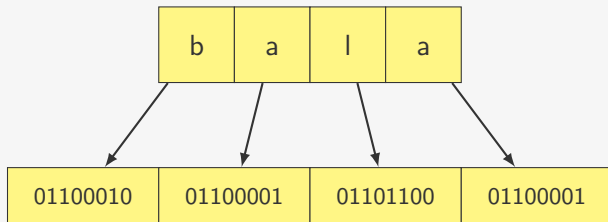
Assim, **"bala"** se torna o número **1.650.551.905**

## Interpretando chaves

Pressupomos que as chaves são **números inteiros**

E se não forem?

- Reinterpretamos a chave como uma sequência de bits



Assim, **"bala"** se torna o número **1.650.551.905**

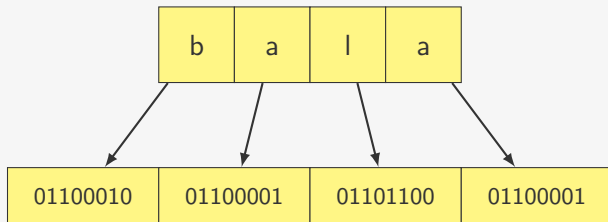
- Esse número pode explodir rapidamente

## Interpretando chaves

Pressupomos que as chaves são **números inteiros**

E se não forem?

- Reinterpretamos a chave como uma sequência de bits



Assim, **"bala"** se torna o número **1.650.551.905**

- Esse número pode explodir rapidamente
- Veremos como contornar isso para strings...

## Método da divisão

- obtemos o resto da divisão pelo tamanho  $M$  do hashing

## Método da divisão

- obtemos o resto da divisão pelo **tamanho**  $M$  do hashing

$$h(x) = x \bmod M$$

## Método da divisão

- obtemos o resto da divisão pelo **tamanho**  $M$  do hashing

$$h(x) = x \bmod M$$

Exemplo:

$$h(\text{"bala"}) = 1.650.551.905 \bmod 1783 = 277$$



## Método da divisão

- obtemos o resto da divisão pelo **tamanho**  $M$  do hashing

$$h(x) = x \bmod M$$

Exemplo:

$$h(\text{"bala"}) = 1.650.551.905 \bmod 1783 = 277$$

Escolhendo  $M$ :

## Método da divisão

- obtemos o resto da divisão pelo **tamanho**  $M$  do hashing

$$h(x) = x \bmod M$$

Exemplo:

$$h(\text{"bala"}) = 1.650.551.905 \bmod 1783 = 277$$

Escolhendo  $M$ :

- escolher  $M$  como uma potência de **2** não é uma boa ideia

## Método da divisão

- obtemos o resto da divisão pelo **tamanho**  $M$  do hashing

$$h(x) = x \bmod M$$

Exemplo:

$$h(\text{"bala"}) = 1.650.551.905 \bmod 1783 = 277$$

Escolhendo  $M$ :

- escolher  $M$  como uma potência de **2** não é uma boa ideia
  - considera apenas os bits menos significativos

## Método da divisão

- obtemos o resto da divisão pelo **tamanho**  $M$  do hashing

$$h(x) = x \bmod M$$

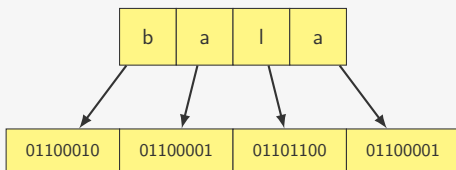
Exemplo:

$$h(\text{"bala"}) = 1.650.551.905 \bmod 1783 = 277$$

Escolhendo  $M$ :

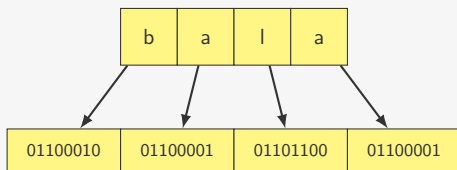
- escolher  $M$  como uma potência de **2** não é uma boa ideia
  - considera apenas os bits menos significativos
- normalmente escolheremos  $M$  como um número primo longe de uma potência de **2**

## Método da divisão para Strings



Como podemos calcular o número  $x$  que representa “bala”?

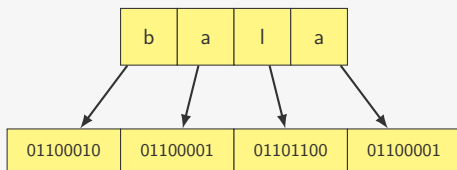
## Método da divisão para Strings



Como podemos calcular o número  $x$  que representa “bala”?

- $x = 'b' \cdot 256^3 + 'a' \cdot 256^2 + 'l' \cdot 256^1 + 'a' \cdot 256^0$

## Método da divisão para Strings

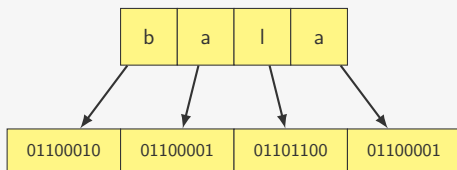


Como podemos calcular o número  $x$  que representa “bala”?

- $x = 'b' \cdot 256^3 + 'a' \cdot 256^2 + 'l' \cdot 256^1 + 'a' \cdot 256^0$

que pode ser rescrito como

## Método da divisão para Strings



Como podemos calcular o número  $x$  que representa “bala”?

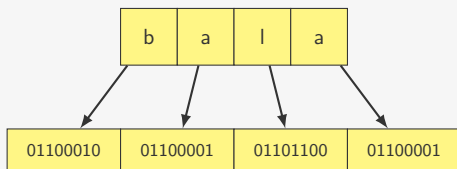
- $x = 'b' \cdot 256^3 + 'a' \cdot 256^2 + 'l' \cdot 256^1 + 'a' \cdot 256^0$

que pode ser reescrito como

- $x = ((('b') \cdot 256 + 'a') \cdot 256 + 'l') \cdot 256 + 'a'$



## Método da divisão para Strings



Como podemos calcular o número  $x$  que representa “bala”?

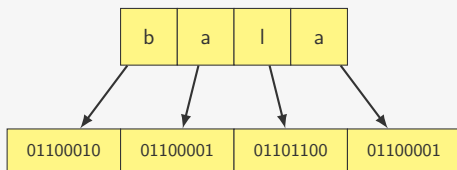
- $x = 'b' \cdot 256^3 + 'a' \cdot 256^2 + 'l' \cdot 256^1 + 'a' \cdot 256^0$

que pode ser reescrito como

- $x = (((('b') \cdot 256 + 'a') \cdot 256 + 'l') \cdot 256 + 'a')$

Mas  $x$  poderia ser muito grande e estourar um `int`...

## Método da divisão para Strings



Como podemos calcular o número  $x$  que representa “bala”?

- $x = 'b' \cdot 256^3 + 'a' \cdot 256^2 + 'l' \cdot 256^1 + 'a' \cdot 256^0$

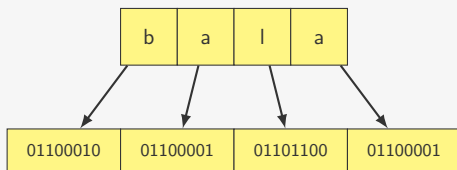
que pode ser reescrito como

- $x = (((('b') \cdot 256 + 'a') \cdot 256 + 'l')) \cdot 256 + 'a'$

Mas  $x$  poderia ser muito grande e estourar um `int`...

Ao invés de calcular  $x \bmod M$ , calculamos

## Método da divisão para Strings



Como podemos calcular o número  $x$  que representa “bala”?

- $x = 'b' \cdot 256^3 + 'a' \cdot 256^2 + 'l' \cdot 256^1 + 'a' \cdot 256^0$

que pode ser reescrito como

- $x = ((('b') \cdot 256 + 'a') \cdot 256 + 'l') \cdot 256 + 'a'$

Mas  $x$  poderia ser muito grande e estourar um `int`...

Ao invés de calcular  $x \bmod M$ , calculamos

$$(((('b' \bmod M) \cdot 256 + 'a' \bmod M) \cdot 256 + 'l' \bmod M) \cdot 256 + 'a' \bmod M)$$

## Método da multiplicação

- multiplicamos por um certo valor real  $A$  e obtemos a parte fracionária

## Método da multiplicação

- multiplicamos por um certo valor real  $A$  e obtemos a parte fracionária
- escolhemos  $A$  conveniente, por exemplo  $A = (\sqrt{5} - 1)/2$

## Método da multiplicação

- multiplicamos por um certo valor real  $A$  e obtemos a parte fracionária
- escolhemos  $A$  conveniente, por exemplo  $A = (\sqrt{5} - 1)/2$
- posição relativa no vetor **não** depende de  $M$  (pode ser  $M = 1024$ )

## Método da multiplicação

- multiplicamos por um certo valor real  $A$  e obtemos a parte fracionária
- escolhemos  $A$  conveniente, por exemplo  $A = (\sqrt{5} - 1)/2$
- posição relativa no vetor **não** depende de  $M$  (pode ser  $M = 1024$ )

$$h(x) = \lfloor M (A \cdot x \bmod 1) \rfloor$$

## Método da multiplicação

- multiplicamos por um certo valor real  $A$  e obtemos a parte fracionária
- escolhemos  $A$  conveniente, por exemplo  $A = (\sqrt{5} - 1)/2$
- posição relativa no vetor **não** depende de  $M$  (pode ser  $M = 1024$ )

$$h(x) = \lfloor M (A \cdot x \bmod 1) \rfloor$$

Exemplo:



## Método da multiplicação

- multiplicamos por um certo valor real  $A$  e obtemos a parte fracionária
- escolhemos  $A$  conveniente, por exemplo  $A = (\sqrt{5} - 1)/2$
- posição relativa no vetor **não** depende de  $M$  (pode ser  $M = 1024$ )

$$h(x) = \lfloor M (A \cdot x \bmod 1) \rfloor$$

Exemplo:

$h(\text{"bala"})$

## Método da multiplicação

- multiplicamos por um certo valor real  $A$  e obtemos a parte fracionária
- escolhemos  $A$  conveniente, por exemplo  $A = (\sqrt{5} - 1)/2$
- posição relativa no vetor **não** depende de  $M$  (pode ser  $M = 1024$ )

$$h(x) = \lfloor M (A \cdot x \bmod 1) \rfloor$$

Exemplo:

$$h(\text{"bala"}) = \lfloor 1024 \cdot [((\sqrt{5} - 1)/2 \cdot 1.650.551.905) \bmod 1] \rfloor$$

## Método da multiplicação

- multiplicamos por um certo valor real  $A$  e obtemos a parte fracionária
- escolhemos  $A$  conveniente, por exemplo  $A = (\sqrt{5} - 1)/2$
- posição relativa no vetor **não** depende de  $M$  (pode ser  $M = 1024$ )

$$h(x) = \lfloor M (A \cdot x \bmod 1) \rfloor$$

Exemplo:

$$\begin{aligned} h(\text{"bala"}) &= \lfloor 1024 \cdot [((\sqrt{5} - 1)/2 \cdot 1.650.551.905) \bmod 1] \rfloor \\ &= \lfloor 1024 \cdot [1020097177,4858876 \bmod 1] \rfloor \end{aligned}$$

## Método da multiplicação

- multiplicamos por um certo valor real  $A$  e obtemos a parte fracionária
- escolhemos  $A$  conveniente, por exemplo  $A = (\sqrt{5} - 1)/2$
- posição relativa no vetor **não** depende de  $M$  (pode ser  $M = 1024$ )

$$h(x) = \lfloor M (A \cdot x \bmod 1) \rfloor$$

Exemplo:

$$\begin{aligned}h(\text{"bala"}) &= \lfloor 1024 \cdot [((\sqrt{5} - 1)/2 \cdot 1.650.551.905) \bmod 1] \rfloor \\ &= \lfloor 1024 \cdot [1020097177,4858876 \bmod 1] \rfloor \\ &= \lfloor 1024 \cdot 0,4858876 \rfloor\end{aligned}$$

## Método da multiplicação

- multiplicamos por um certo valor real  $A$  e obtemos a parte fracionária
- escolhemos  $A$  conveniente, por exemplo  $A = (\sqrt{5} - 1)/2$
- posição relativa no vetor **não** depende de  $M$  (pode ser  $M = 1024$ )

$$h(x) = \lfloor M (A \cdot x \bmod 1) \rfloor$$

Exemplo:

$$\begin{aligned}h(\text{"bala"}) &= \lfloor 1024 \cdot [((\sqrt{5} - 1)/2 \cdot 1.650.551.905) \bmod 1] \rfloor \\ &= \lfloor 1024 \cdot [1020097177,4858876 \bmod 1] \rfloor \\ &= \lfloor 1024 \cdot 0,4858876 \rfloor \\ &= \lfloor 497,5489024 \rfloor\end{aligned}$$

## Método da multiplicação

- multiplicamos por um certo valor real  $A$  e obtemos a parte fracionária
- escolhemos  $A$  conveniente, por exemplo  $A = (\sqrt{5} - 1)/2$
- posição relativa no vetor **não** depende de  $M$  (pode ser  $M = 1024$ )

$$h(x) = \lfloor M (A \cdot x \bmod 1) \rfloor$$

Exemplo:

$$\begin{aligned}h(\text{"bala"}) &= \lfloor 1024 \cdot [((\sqrt{5} - 1)/2 \cdot 1.650.551.905) \bmod 1] \rfloor \\ &= \lfloor 1024 \cdot [1020097177,4858876 \bmod 1] \rfloor \\ &= \lfloor 1024 \cdot 0,4858876 \rfloor \\ &= \lfloor 497,5489024 \rfloor = 497\end{aligned}$$

## Método da multiplicação

- multiplicamos por um certo valor real  $A$  e obtemos a parte fracionária
- escolhemos  $A$  conveniente, por exemplo  $A = (\sqrt{5} - 1)/2$
- posição relativa no vetor **não** depende de  $M$  (pode ser  $M = 1024$ )

$$h(x) = \lfloor M (A \cdot x \bmod 1) \rfloor$$

Exemplo:

$$\begin{aligned}h(\text{"bala"}) &= \lfloor 1024 \cdot [((\sqrt{5} - 1)/2 \cdot 1.650.551.905) \bmod 1] \rfloor \\ &= \lfloor 1024 \cdot [1020097177,4858876 \bmod 1] \rfloor \\ &= \lfloor 1024 \cdot 0,4858876 \rfloor \\ &= \lfloor 497,5489024 \rfloor = 497\end{aligned}$$

O uso da razão áurea como valor de  $A$  é sugestão de Knuth

# Interface do TAD

```
1 #define MAX 1783
2
3 typedef struct no *p_no;
4
5 struct no {
6     char chave[10];
7     int dado;
8     p_no prox;
9 };
```



# Interface do TAD

```
1 #define MAX 1783
2
3 typedef struct no *p_no;
4
5 struct no {
6     char chave[10];
7     int dado;
8     p_no prox;
9 };
10
11 typedef struct hash *p_hash;
12
13 struct hash {
14     p_no vetor[MAX];
15 };
```

# Interface do TAD

```
1 #define MAX 1783
2
3 typedef struct no *p_no;
4
5 struct no {
6     char chave[10];
7     int dado;
8     p_no prox;
9 };
10
11 typedef struct hash *p_hash;
12
13 struct hash {
14     p_no vetor[MAX];
15 };
16
17 p_hash criar_hash();
18
19 void destruir_hash(p_hash t);
20
21 void inserir(p_hash t, char *chave, int dado);
22
23 void remover(p_hash t, char *chave);
24
25 p_no buscar(p_hash t, char *chave);
```

# Exemplo de implementação

```
1 int hash(char *chave) {
```

# Exemplo de implementação

```
1 int hash(char *chave) {  
2     int i, n = 0;
```

## Exemplo de implementação

```
1 int hash(char *chave) {  
2     int i, n = 0;  
3     for (i = 0; i < strlen(chave); i++)  
4         n = (256 * n + chave[i]) % MAX;
```

# Exemplo de implementação

```
1 int hash(char *chave) {  
2     int i, n = 0;  
3     for (i = 0; i < strlen(chave); i++)  
4         n = (256 * n + chave[i]) % MAX;  
5     return n;  
6 }
```

## Exemplo de implementação

```
1 int hash(char *chave) {
2     int i, n = 0;
3     for (i = 0; i < strlen(chave); i++)
4         n = (256 * n + chave[i]) % MAX;
5     return n;
6 }
7
8 void inserir(p_hash t, char *chave, int dado) {
```

## Exemplo de implementação

```
1 int hash(char *chave) {
2     int i, n = 0;
3     for (i = 0; i < strlen(chave); i++)
4         n = (256 * n + chave[i]) % MAX;
5     return n;
6 }
7
8 void inserir(p_hash t, char *chave, int dado) {
9     int n = hash(chave);
10    t->vetor[n] = inserir_lista(t->vetor[n], chave, dado);
11 }
```



## Exemplo de implementação

```
1 int hash(char *chave) {
2     int i, n = 0;
3     for (i = 0; i < strlen(chave); i++)
4         n = (256 * n + chave[i]) % MAX;
5     return n;
6 }
7
8 void inserir(p_hash t, char *chave, int dado) {
9     int n = hash(chave);
10    t->vetor[n] = inserir_lista(t->vetor[n], chave, dado);
11 }
12
13 void remover(p_hash t, char *chave) {
```

## Exemplo de implementação

```
1 int hash(char *chave) {
2     int i, n = 0;
3     for (i = 0; i < strlen(chave); i++)
4         n = (256 * n + chave[i]) % MAX;
5     return n;
6 }
7
8 void inserir(p_hash t, char *chave, int dado) {
9     int n = hash(chave);
10    t->vetor[n] = inserir_lista(t->vetor[n], chave, dado);
11 }
12
13 void remover(p_hash t, char *chave) {
14     int n = hash(chave);
15     t->vetor[n] = remover_lista(t->vetor[n], chave);
16 }
```

## Quebrando um programa que usa hashing

Sabendo a função de hashing, podemos prejudicar o programa:

## Quebrando um programa que usa hashing

Sabendo a função de hashing, podemos prejudicar o programa:

- insira muitos elementos com o mesmo hash

## Quebrando um programa que usa hashing

Sabendo a função de hashing, podemos prejudicar o programa:

- insira muitos elementos com o mesmo hash

Como nos proteger de um adversário malicioso?

## Quebrando um programa que usa hashing

Sabendo a função de hashing, podemos prejudicar o programa:

- insira muitos elementos com o mesmo hash

Como nos proteger de um adversário malicioso?

- Podemos escolher a função de hashing aleatoriamente

## Quebrando um programa que usa hashing

Sabendo a função de hashing, podemos prejudicar o programa:

- insira muitos elementos com o mesmo hash

Como nos proteger de um adversário malicioso?

- Podemos escolher a função de hashing aleatoriamente

Uma boa função de hashing aleatória:

## Quebrando um programa que usa hashing

Sabendo a função de hashing, podemos prejudicar o programa:

- insira muitos elementos com o mesmo hash

Como nos proteger de um adversário malicioso?

- Podemos escolher a função de hashing aleatoriamente

Uma boa função de hashing aleatória:

- fixe  $p$  um primo maior do que  $M$



## Quebrando um programa que usa hashing

Sabendo a função de hashing, podemos prejudicar o programa:

- insira muitos elementos com o mesmo hash

Como nos proteger de um adversário malicioso?

- Podemos escolher a função de hashing aleatoriamente

Uma boa função de hashing aleatória:

- fixe  $p$  um primo maior do que  $M$
- escolha  $a \in \{1, \dots, p\}$  e  $b \in \{0, \dots, p\}$  uniform. ao acaso

## Quebrando um programa que usa hashing

Sabendo a função de hashing, podemos prejudicar o programa:

- insira muitos elementos com o mesmo hash

Como nos proteger de um adversário malicioso?

- Podemos escolher a função de hashing aleatoriamente

Uma boa função de hashing aleatória:

- fixe  $p$  um primo maior do que  $M$
- escolha  $a \in \{1, \dots, p\}$  e  $b \in \{0, \dots, p\}$  uniform. ao acaso
- defina  $h_{a,b}(k) = ((ak + b) \bmod p) \bmod M$

## Quebrando um programa que usa hashing

Sabendo a função de hashing, podemos prejudicar o programa:

- insira muitos elementos com o mesmo hash

Como nos proteger de um adversário malicioso?

- Podemos escolher a função de hashing aleatoriamente

Uma boa função de hashing aleatória:

- fixe  $p$  um primo maior do que  $M$
- escolha  $a \in \{1, \dots, p\}$  e  $b \in \{0, \dots, p\}$  uniform. ao acaso
- defina  $h_{a,b}(k) = ((ak + b) \bmod p) \bmod M$
- sabemos que essa função espalha bem

## Quebrando um programa que usa hashing

Sabendo a função de hashing, podemos prejudicar o programa:

- insira muitos elementos com o mesmo hash

Como nos proteger de um adversário malicioso?

- Podemos escolher a função de hashing aleatoriamente

Uma boa função de hashing aleatória:

- fixe  $p$  um primo maior do que  $M$
- escolha  $a \in \{1, \dots, p\}$  e  $b \in \{0, \dots, p\}$  uniform. ao acaso
- defina  $h_{a,b}(k) = ((ak + b) \bmod p) \bmod M$
- sabemos que essa função espalha bem
  - a probabilidade de colisão é no máximo  $1/M$

## Quebrando um programa que usa hashing

Sabendo a função de hashing, podemos prejudicar o programa:

- insira muitos elementos com o mesmo hash

Como nos proteger de um adversário malicioso?

- Podemos escolher a função de hashing aleatoriamente

Uma boa função de hashing aleatória:

- fixe  $p$  um primo maior do que  $M$
- escolha  $a \in \{1, \dots, p\}$  e  $b \in \{0, \dots, p\}$  uniform. ao acaso
- defina  $h_{a,b}(k) = ((ak + b) \bmod p) \bmod M$
- sabemos que essa função espalha bem
  - a probabilidade de colisão é no máximo  $1/M$
  - é um **hashing universal**

## Endereçamento aberto

Existe uma alternativa para a implementação de tabela de espalhamento

# Endereçamento aberto

Existe uma alternativa para a implementação de tabela de espalhamento

Endereçamento aberto:

# Endereçamento aberto

Existe uma alternativa para a implementação de tabela de espalhamento

Endereçamento aberto:

- os dados são guardados no próprio vetor



# Endereçamento aberto

Existe uma alternativa para a implementação de tabela de espalhamento

Endereçamento aberto:

- os dados são guardados no próprio vetor
- colisões são colocadas em posições livres da tabela

# Endereçamento aberto

Existe uma alternativa para a implementação de tabela de espalhamento

Endereçamento aberto:

- os dados são guardados no próprio vetor
- colisões são colocadas em posições livres da tabela

Características:

# Endereçamento aberto

Existe uma alternativa para a implementação de tabela de espalhamento

Endereçamento aberto:

- os dados são guardados no próprio vetor
- colisões são colocadas em posições livres da tabela

Características:

- evita percorrer usando ponteiros e alocação e deslocação de memória (**malloc** e **free**)

# Endereçamento aberto

Existe uma alternativa para a implementação de tabela de espalhamento

Endereçamento aberto:

- os dados são guardados no próprio vetor
- colisões são colocadas em posições livres da tabela

Características:

- evita percorrer usando ponteiros e alocação e deslocação de memória (**malloc** e **free**)
- se a tabela encher, deve recriar uma tabela maior

# Endereçamento aberto

Existe uma alternativa para a implementação de tabela de espalhamento

Endereçamento aberto:

- os dados são guardados no próprio vetor
- colisões são colocadas em posições livres da tabela

Características:

- evita percorrer usando ponteiros e alocação e deslocação de memória (**malloc** e **free**)
- se a tabela encher, deve recriar uma tabela maior
  - e mudar a função de hashing

# Endereçamento aberto

Existe uma alternativa para a implementação de tabela de espalhamento

Endereçamento aberto:

- os dados são guardados no próprio vetor
- colisões são colocadas em posições livres da tabela

Características:

- evita percorrer usando ponteiros e alocação e deslocação de memória (**malloc** e **free**)
- se a tabela encher, deve recriar uma tabela maior
  - e mudar a função de hashing
- remoção é mais complicada

# Endereçamento aberto com sondagem linear

broca

boca

bolo

bela

bala

dia

escola

gratuito

ilha

0	
1	
2	
3	
4	
5	
6	
7	
8	

Inserindo:

# Endereçamento aberto com sondagem linear

broca  $\rightsquigarrow h(\text{"broca"}) = 3$

boca

bolo

bela

bala

dia

escola

gratuito

ilha

0	
1	
2	
3	
4	
5	
6	
7	
8	

Inserindo:

- procuramos posição



# Endereçamento aberto com sondagem linear

broca  $\rightsquigarrow h(\text{"broca"}) = 3$

boca

bolo

bela

bala

dia

escola

gratuito

ilha

0	
1	
2	
3	
4	
5	
6	
7	
8	



Inserindo:

- procuramos posição
- se houver espaço, guardamos

# Endereçamento aberto com sondagem linear

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca

bolo

bela

bala

dia

escola

gratuito

ilha

0	
1	
2	
3	broca
4	
5	
6	
7	
8	

Inserindo:

- procuramos posição
- se houver espaço, guardamos

# Endereçamento aberto com sondagem linear

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo

bela

bala

dia

escola

gratuito

ilha

0	
1	
2	
3	broca
4	
5	
6	
7	
8	

Inserindo:

- procuramos posição
- se houver espaço, guardamos

# Endereçamento aberto com sondagem linear

broca  $\rightsquigarrow h(\text{"broca"}) = 3$

boca  $\rightsquigarrow h(\text{"boca"}) = 0$

bolo

bela


bala

dia

escola

gratuito

ilha



0	
1	
2	
3	broca
4	
5	
6	
7	
8	

Inserindo:

- procuramos posição
- se houver espaço, guardamos

# Endereçamento aberto com sondagem linear

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo

bela

bala

dia

escola

gratuito

ilha

0	boca
1	
2	
3	broca
4	
5	
6	
7	
8	

Inserindo:

- procuramos posição
- se houver espaço, guardamos

# Endereçamento aberto com sondagem linear

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela

bala

dia

escola

gratuito

ilha

0	boca
1	
2	
3	broca
4	
5	
6	
7	
8	

Inserindo:

- procuramos posição
- se houver espaço, guardamos

# Endereçamento aberto com sondagem linear

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela

bala

dia

escola

gratuito

ilha

0	boca
1	
2	
3	broca
4	
5	
6	
7	
8	



Inserindo:

- procuramos posição
- se houver espaço, guardamos

# Endereçamento aberto com sondagem linear

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela

bala

dia

escola

gratuito

ilha

0	boca
1	
2	
3	broca
4	
5	bolo
6	
7	
8	

Inserindo:

- procuramos posição
- se houver espaço, guardamos



# Endereçamento aberto com sondagem linear

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela  $\rightsquigarrow$   $h(\text{"bela"}) = 2$

bala

dia

escola

gratuito

ilha

0	boca
1	
2	
3	broca
4	
5	bolo
6	
7	
8	

Inserindo:

- procuramos posição
- se houver espaço, guardamos

# Endereçamento aberto com sondagem linear

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela  $\rightsquigarrow$   $h(\text{"bela"}) = 2$

bala

dia

escola

gratuito

ilha

0	boca
1	
2	
3	broca
4	
5	bolo
6	
7	
8	

Inserindo:

- procuramos posição
- se houver espaço, guardamos

# Endereçamento aberto com sondagem linear

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela  $\rightsquigarrow$   $h(\text{"bela"}) = 2$

bala

dia

escola

gratuito

ilha

0	boca
1	
2	bela
3	broca
4	
5	bolo
6	
7	
8	

Inserindo:

- procuramos posição
- se houver espaço, guardamos

# Endereçamento aberto com sondagem linear

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela  $\rightsquigarrow$   $h(\text{"bela"}) = 2$


bala  $\rightsquigarrow$   $h(\text{"bala"}) = 0$

dia

escola

gratuito

ilha



0	boca
1	
2	bela
3	broca
4	
5	bolo
6	
7	
8	

Inserindo:

- procuramos posição
- se houver espaço, guardamos
- se não houver espaço, procuramos a próxima posição livre (módulo  $M$ )

# Endereçamento aberto com sondagem linear

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela  $\rightsquigarrow$   $h(\text{"bela"}) = 2$

bala  $\rightsquigarrow$   $h(\text{"bala"}) = 0$

dia

escola

gratuito

ilha

0	boca
1	
2	bela
3	broca
4	
5	bolo
6	
7	
8	

Inserindo:

- procuramos posição
- se houver espaço, guardamos
- se não houver espaço, procuramos a próxima posição livre (módulo  $M$ )

# Endereçamento aberto com sondagem linear

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela  $\rightsquigarrow$   $h(\text{"bela"}) = 2$

bala  $\rightsquigarrow$   $h(\text{"bala"}) = 0$

dia

escola

gratuito

ilha

0	boca
1	bala
2	bela
3	broca
4	
5	bolo
6	
7	
8	

Inserindo:

- procuramos posição
- se houver espaço, guardamos
- se não houver espaço, procuramos a próxima posição livre (módulo  $M$ )

# Endereçamento aberto com sondagem linear

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela  $\rightsquigarrow$   $h(\text{"bela"}) = 2$

bala  $\rightsquigarrow$   $h(\text{"bala"}) = 0$

dia

escola

gratuito

ilha

0	boca
1	bala
2	bela
3	broca
4	
5	bolo
6	
7	
8	

Inserindo:

- procuramos posição
- se houver espaço, guardamos
- se não houver espaço, procuramos a próxima posição livre (módulo  $M$ )

# Endereçamento aberto com sondagem linear

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela  $\rightsquigarrow$   $h(\text{"bela"}) = 2$

bala  $\rightsquigarrow$   $h(\text{"bala"}) = 0$

dia  $\rightsquigarrow$   $h(\text{"dia"}) = 2$

escola

gratuito

ilha

0	boca
1	bala
2	bela
3	broca
4	
5	bolo
6	
7	
8	

Inserindo:

- procuramos posição
- se houver espaço, guardamos
- se não houver espaço, procuramos a próxima posição livre (módulo  $M$ )



# Endereçamento aberto com sondagem linear

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela  $\rightsquigarrow$   $h(\text{"bela"}) = 2$

bala  $\rightsquigarrow$   $h(\text{"bala"}) = 0$

dia  $\rightsquigarrow$   $h(\text{"dia"}) = 2$

escola

gratuito

ilha

0	boca
1	bala
2	bela
3	broca
4	
5	bolo
6	
7	
8	

Inserindo:

- procuramos posição
- se houver espaço, guardamos
- se não houver espaço, procuramos a próxima posição livre (módulo  $M$ )

# Endereçamento aberto com sondagem linear

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela  $\rightsquigarrow$   $h(\text{"bela"}) = 2$

bala  $\rightsquigarrow$   $h(\text{"bala"}) = 0$

dia  $\rightsquigarrow$   $h(\text{"dia"}) = 2$

escola

gratuito

ilha

0	boca
1	bala
2	bela
3	broca
4	
5	bolo
6	
7	
8	

Inserindo:

- procuramos posição
- se houver espaço, guardamos
- se não houver espaço, procuramos a próxima posição livre (módulo  $M$ )

# Endereçamento aberto com sondagem linear

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela  $\rightsquigarrow$   $h(\text{"bela"}) = 2$

bala  $\rightsquigarrow$   $h(\text{"bala"}) = 0$

dia  $\rightsquigarrow$   $h(\text{"dia"}) = 2$

escola

gratuito

ilha

0	boca
1	bala
2	bela
3	broca
4	
5	bolo
6	
7	
8	

Inserindo:

- procuramos posição
- se houver espaço, guardamos
- se não houver espaço, procuramos a próxima posição livre (módulo  $M$ )

# Endereçamento aberto com sondagem linear

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela  $\rightsquigarrow$   $h(\text{"bela"}) = 2$

bala  $\rightsquigarrow$   $h(\text{"bala"}) = 0$

dia  $\rightsquigarrow$   $h(\text{"dia"}) = 2$

escola

gratuito

ilha

0	boca
1	bala
2	bela
3	broca
4	
5	bolo
6	
7	
8	

Inserindo:

- procuramos posição
- se houver espaço, guardamos
- se não houver espaço, procuramos a próxima posição livre (módulo  $M$ )

# Endereçamento aberto com sondagem linear

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela  $\rightsquigarrow$   $h(\text{"bela"}) = 2$

bala  $\rightsquigarrow$   $h(\text{"bala"}) = 0$

dia  $\rightsquigarrow$   $h(\text{"dia"}) = 2$

escola

gratuito

ilha

0	boca
1	bala
2	bela
3	broca
4	dia
5	bolo
6	
7	
8	

Inserindo:

- procuramos posição
- se houver espaço, guardamos
- se não houver espaço, procuramos a próxima posição livre (módulo  $M$ )

# Endereçamento aberto com sondagem linear

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela  $\rightsquigarrow$   $h(\text{"bela"}) = 2$

bala  $\rightsquigarrow$   $h(\text{"bala"}) = 0$

dia  $\rightsquigarrow$   $h(\text{"dia"}) = 2$

escola  $\rightsquigarrow$   $h(\text{"escola"}) = 7$

gratuito

ilha

0	boca
1	bala
2	bela
3	broca
4	dia
5	bolo
6	
7	
8	

Inserindo:

- procuramos posição
- se houver espaço, guardamos
- se não houver espaço, procuramos a próxima posição livre (módulo  $M$ )

# Endereçamento aberto com sondagem linear

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela  $\rightsquigarrow$   $h(\text{"bela"}) = 2$

bala  $\rightsquigarrow$   $h(\text{"bala"}) = 0$

dia  $\rightsquigarrow$   $h(\text{"dia"}) = 2$

escola  $\rightsquigarrow$   $h(\text{"escola"}) = 7$

gratuito

ilha

0	boca
1	bala
2	bela
3	broca
4	dia
5	bolo
6	
7	
8	



Inserindo:

- procuramos posição
- se houver espaço, guardamos
- se não houver espaço, procuramos a próxima posição livre (módulo  $M$ )

# Endereçamento aberto com sondagem linear

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela  $\rightsquigarrow$   $h(\text{"bela"}) = 2$

bala  $\rightsquigarrow$   $h(\text{"bala"}) = 0$

dia  $\rightsquigarrow$   $h(\text{"dia"}) = 2$

escola  $\rightsquigarrow$   $h(\text{"escola"}) = 7$

gratuito

ilha

0	boca
1	bala
2	bela
3	broca
4	dia
5	bolo
6	
7	escola
8	

Inserindo:

- procuramos posição
- se houver espaço, guardamos
- se não houver espaço, procuramos a próxima posição livre (módulo  $M$ )



# Endereçamento aberto com sondagem linear

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela  $\rightsquigarrow$   $h(\text{"bela"}) = 2$

bala  $\rightsquigarrow$   $h(\text{"bala"}) = 0$

dia  $\rightsquigarrow$   $h(\text{"dia"}) = 2$

escola  $\rightsquigarrow$   $h(\text{"escola"}) = 7$

gratuito  $\rightsquigarrow$   $h(\text{"gratuito"}) = 0$

ilha

0	boca
1	bala
2	bela
3	broca
4	dia
5	bolo
6	
7	escola
8	

Inserindo:

- procuramos posição
- se houver espaço, guardamos
- se não houver espaço, procuramos a próxima posição livre (módulo  $M$ )

# Endereçamento aberto com sondagem linear

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela  $\rightsquigarrow$   $h(\text{"bela"}) = 2$


bala  $\rightsquigarrow$   $h(\text{"bala"}) = 0$

dia  $\rightsquigarrow$   $h(\text{"dia"}) = 2$

escola  $\rightsquigarrow$   $h(\text{"escola"}) = 7$

gratuito  $\rightsquigarrow$   $h(\text{"gratuito"}) = 0$

ilha



0	boca
1	bala
2	bela
3	broca
4	dia
5	bolo
6	
7	escola
8	

Inserindo:

- procuramos posição
- se houver espaço, guardamos
- se não houver espaço, procuramos a próxima posição livre (módulo  $M$ )

# Endereçamento aberto com sondagem linear

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela  $\rightsquigarrow$   $h(\text{"bela"}) = 2$


bala  $\rightsquigarrow$   $h(\text{"bala"}) = 0$

dia  $\rightsquigarrow$   $h(\text{"dia"}) = 2$

escola  $\rightsquigarrow$   $h(\text{"escola"}) = 7$

gratuito  $\rightsquigarrow$   $h(\text{"gratuito"}) = 0$

ilha



0	boca
1	bala
2	bela
3	broca
4	dia
5	bolo
6	
7	escola
8	

Inserindo:

- procuramos posição
- se houver espaço, guardamos
- se não houver espaço, procuramos a próxima posição livre (módulo  $M$ )

# Endereçamento aberto com sondagem linear

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela  $\rightsquigarrow$   $h(\text{"bela"}) = 2$


bala  $\rightsquigarrow$   $h(\text{"bala"}) = 0$

dia  $\rightsquigarrow$   $h(\text{"dia"}) = 2$

escola  $\rightsquigarrow$   $h(\text{"escola"}) = 7$

gratuito  $\rightsquigarrow$   $h(\text{"gratuito"}) = 0$

ilha



0	boca
1	bala
2	bela
3	broca
4	dia
5	bolo
6	
7	escola
8	

Inserindo:

- procuramos posição
- se houver espaço, guardamos
- se não houver espaço, procuramos a próxima posição livre (módulo  $M$ )

# Endereçamento aberto com sondagem linear

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela  $\rightsquigarrow$   $h(\text{"bela"}) = 2$

bala  $\rightsquigarrow$   $h(\text{"bala"}) = 0$

dia  $\rightsquigarrow$   $h(\text{"dia"}) = 2$

escola  $\rightsquigarrow$   $h(\text{"escola"}) = 7$

gratuito  $\rightsquigarrow$   $h(\text{"gratuito"}) = 0$

ilha

0	boca
1	bala
2	bela
3	broca
4	dia
5	bolo
6	
7	escola
8	

Inserindo:

- procuramos posição
- se houver espaço, guardamos
- se não houver espaço, procuramos a próxima posição livre (módulo  $M$ )

# Endereçamento aberto com sondagem linear

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela  $\rightsquigarrow$   $h(\text{"bela"}) = 2$

bala  $\rightsquigarrow$   $h(\text{"bala"}) = 0$

dia  $\rightsquigarrow$   $h(\text{"dia"}) = 2$

escola  $\rightsquigarrow$   $h(\text{"escola"}) = 7$

gratuito  $\rightsquigarrow$   $h(\text{"gratuito"}) = 0$

ilha

0	boca
1	bala
2	bela
3	broca
4	dia
5	bolo
6	
7	escola
8	

Inserindo:

- procuramos posição
- se houver espaço, guardamos
- se não houver espaço, procuramos a próxima posição livre (módulo  $M$ )

# Endereçamento aberto com sondagem linear

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela  $\rightsquigarrow$   $h(\text{"bela"}) = 2$

bala  $\rightsquigarrow$   $h(\text{"bala"}) = 0$

dia  $\rightsquigarrow$   $h(\text{"dia"}) = 2$

escola  $\rightsquigarrow$   $h(\text{"escola"}) = 7$

gratuito  $\rightsquigarrow$   $h(\text{"gratuito"}) = 0$

ilha

0	boca
1	bala
2	bela
3	broca
4	dia
5	bolo
6	
7	escola
8	



Inserindo:

- procuramos posição
- se houver espaço, guardamos
- se não houver espaço, procuramos a próxima posição livre (módulo  $M$ )

# Endereçamento aberto com sondagem linear

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela  $\rightsquigarrow$   $h(\text{"bela"}) = 2$

bala  $\rightsquigarrow$   $h(\text{"bala"}) = 0$

dia  $\rightsquigarrow$   $h(\text{"dia"}) = 2$

escola  $\rightsquigarrow$   $h(\text{"escola"}) = 7$

gratuito  $\rightsquigarrow$   $h(\text{"gratuito"}) = 0$

ilha

0	boca
1	bala
2	bela
3	broca
4	dia
5	bolo
6	
7	escola
8	



Inserindo:

- procuramos posição
- se houver espaço, guardamos
- se não houver espaço, procuramos a próxima posição livre (módulo  $M$ )



# Endereçamento aberto com sondagem linear

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela  $\rightsquigarrow$   $h(\text{"bela"}) = 2$

bala  $\rightsquigarrow$   $h(\text{"bala"}) = 0$

dia  $\rightsquigarrow$   $h(\text{"dia"}) = 2$

escola  $\rightsquigarrow$   $h(\text{"escola"}) = 7$

gratuito  $\rightsquigarrow$   $h(\text{"gratuito"}) = 0$

ilha

0	boca
1	bala
2	bela
3	broca
4	dia
5	bolo
6	gratuito
7	escola
8	

Inserindo:

- procuramos posição
- se houver espaço, guardamos
- se não houver espaço, procuramos a próxima posição livre (módulo  $M$ )

# Endereçamento aberto com sondagem linear

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela  $\rightsquigarrow$   $h(\text{"bela"}) = 2$

bala  $\rightsquigarrow$   $h(\text{"bala"}) = 0$

dia  $\rightsquigarrow$   $h(\text{"dia"}) = 2$

escola  $\rightsquigarrow$   $h(\text{"escola"}) = 7$

gratuito  $\rightsquigarrow$   $h(\text{"gratuito"}) = 0$

ilha  $\rightsquigarrow$   $h(\text{"ilha"}) = 6$

0	boca
1	bala
2	bela
3	broca
4	dia
5	bolo
6	gratuito
7	escola
8	

Inserindo:

- procuramos posição
- se houver espaço, guardamos
- se não houver espaço, procuramos a próxima posição livre (módulo  $M$ )

# Endereçamento aberto com sondagem linear

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela  $\rightsquigarrow$   $h(\text{"bela"}) = 2$

bala  $\rightsquigarrow$   $h(\text{"bala"}) = 0$

dia  $\rightsquigarrow$   $h(\text{"dia"}) = 2$

escola  $\rightsquigarrow$   $h(\text{"escola"}) = 7$

gratuito  $\rightsquigarrow$   $h(\text{"gratuito"}) = 0$

ilha  $\rightsquigarrow$   $h(\text{"ilha"}) = 6$

0	boca
1	bala
2	bela
3	broca
4	dia
5	bolo
6	gratuito
7	escola
8	



Inserindo:

- procuramos posição
- se houver espaço, guardamos
- se não houver espaço, procuramos a próxima posição livre (módulo  $M$ )

# Endereçamento aberto com sondagem linear

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela  $\rightsquigarrow$   $h(\text{"bela"}) = 2$

bala  $\rightsquigarrow$   $h(\text{"bala"}) = 0$

dia  $\rightsquigarrow$   $h(\text{"dia"}) = 2$

escola  $\rightsquigarrow$   $h(\text{"escola"}) = 7$

gratuito  $\rightsquigarrow$   $h(\text{"gratuito"}) = 0$

ilha  $\rightsquigarrow$   $h(\text{"ilha"}) = 6$

0	boca
1	bala
2	bela
3	broca
4	dia
5	bolo
6	gratuito
7	escola
8	



Inserindo:

- procuramos posição
- se houver espaço, guardamos
- se não houver espaço, procuramos a próxima posição livre (módulo  $M$ )

# Endereçamento aberto com sondagem linear

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela  $\rightsquigarrow$   $h(\text{"bela"}) = 2$

bala  $\rightsquigarrow$   $h(\text{"bala"}) = 0$

dia  $\rightsquigarrow$   $h(\text{"dia"}) = 2$

escola  $\rightsquigarrow$   $h(\text{"escola"}) = 7$

gratuito  $\rightsquigarrow$   $h(\text{"gratuito"}) = 0$

ilha  $\rightsquigarrow$   $h(\text{"ilha"}) = 6$

0	boca
1	bala
2	bela
3	broca
4	dia
5	bolo
6	gratuito
7	escola
8	



Inserindo:

- procuramos posição
- se houver espaço, guardamos
- se não houver espaço, procuramos a próxima posição livre (módulo  $M$ )

# Endereçamento aberto com sondagem linear

broca  $\rightsquigarrow$   $h(\text{"broca"}) = 3$

boca  $\rightsquigarrow$   $h(\text{"boca"}) = 0$

bolo  $\rightsquigarrow$   $h(\text{"bolo"}) = 5$

bela  $\rightsquigarrow$   $h(\text{"bela"}) = 2$

bala  $\rightsquigarrow$   $h(\text{"bala"}) = 0$

dia  $\rightsquigarrow$   $h(\text{"dia"}) = 2$

escola  $\rightsquigarrow$   $h(\text{"escola"}) = 7$

gratuito  $\rightsquigarrow$   $h(\text{"gratuito"}) = 0$

ilha  $\rightsquigarrow$   $h(\text{"ilha"}) = 6$

0	boca
1	bala
2	bela
3	broca
4	dia
5	bolo
6	gratuito
7	escola
8	ilha

Inserindo:

- procuramos posição
- se houver espaço, guardamos
- se não houver espaço, procuramos a próxima posição livre (módulo  $M$ )

## Busca em endereçamento aberto

Como fazer uma busca com endereçamento aberto?

## Busca em endereçamento aberto

Como fazer uma busca com endereçamento aberto?

- Basta simular a inserção:



# Busca em endereçamento aberto

Como fazer uma busca com endereçamento aberto?

- Basta simular a inserção:
  - Calcule a função de hashing

# Busca em endereçamento aberto

Como fazer uma busca com endereçamento aberto?

- Basta simular a inserção:
  - Calcule a função de hashing
  - Percorra a tabela em sequência procurando pela chave

# Busca em endereçamento aberto

Como fazer uma busca com endereçamento aberto?

- Basta simular a inserção:
  - Calcule a função de hashing
  - Percorra a tabela em sequência procurando pela chave
  - Se encontrar a chave, devolva o item correspondente

# Busca em endereçamento aberto

Como fazer uma busca com endereçamento aberto?

- Basta simular a inserção:
  - Calcule a função de hashing
  - Percorra a tabela em sequência procurando pela chave
  - Se encontrar a chave, devolva o item correspondente
  - Se encontrar um espaço vazio, devolva **NULL**

# Busca em endereçamento aberto

Como fazer uma busca com endereçamento aberto?

- Basta simular a inserção:
  - Calcule a função de hashing
  - Percorra a tabela em sequência procurando pela chave
  - Se encontrar a chave, devolva o item correspondente
  - Se encontrar um espaço vazio, devolva **NULL**

O que é um espaço vazio em um vetor?

# Busca em endereçamento aberto

Como fazer uma busca com endereçamento aberto?

- Basta simular a inserção:
  - Calcule a função de hashing
  - Percorra a tabela em sequência procurando pela chave
  - Se encontrar a chave, devolva o item correspondente
  - Se encontrar um espaço vazio, devolva **NULL**

O que é um espaço vazio em um vetor?

- Se for um vetor de ponteiros, pode ser **NULL**

# Busca em endereçamento aberto

Como fazer uma busca com endereçamento aberto?

- Basta simular a inserção:
  - Calcule a função de hashing
  - Percorra a tabela em sequência procurando pela chave
  - Se encontrar a chave, devolva o item correspondente
  - Se encontrar um espaço vazio, devolva **NULL**

O que é um espaço vazio em um vetor?

- Se for um vetor de ponteiros, pode ser **NULL**
- Se não for, precisa ser um elemento **dummy**

# Busca em endereçamento aberto

Como fazer uma busca com endereçamento aberto?

- Basta simular a inserção:
  - Calcule a função de hashing
  - Percorra a tabela em sequência procurando pela chave
  - Se encontrar a chave, devolva o item correspondente
  - Se encontrar um espaço vazio, devolva **NULL**

O que é um espaço vazio em um vetor?

- Se for um vetor de ponteiros, pode ser **NULL**
- Se não for, precisa ser um elemento **dummy**
  - ou um valor que nunca será usado



# Busca em endereçamento aberto

Como fazer uma busca com endereçamento aberto?

- Basta simular a inserção:
  - Calcule a função de hashing
  - Percorra a tabela em sequência procurando pela chave
  - Se encontrar a chave, devolva o item correspondente
  - Se encontrar um espaço vazio, devolva **NULL**

O que é um espaço vazio em um vetor?

- Se for um vetor de ponteiros, pode ser **NULL**
- Se não for, precisa ser um elemento **dummy**
  - ou um valor que nunca será usado
  - ou ter um campo indicando que é **dummy**

## Remoção em endereçamento aberto

Como fazer a remoção com endereçamento aberto?

## Remoção em endereçamento aberto

Como fazer a remoção com endereçamento aberto?

- Não podemos apenas remover os elementos da tabela

# Remoção em endereçamento aberto

Como fazer a remoção com endereçamento aberto?

- Não podemos apenas remover os elementos da tabela
  - Por quê?

## Remoção em endereçamento aberto

Como fazer a remoção com endereçamento aberto?

- Não podemos apenas remover os elementos da tabela
  - Por quê? Quebraria a busca...

# Remoção em endereçamento aberto

Como fazer a remoção com endereçamento aberto?

- Não podemos apenas remover os elementos da tabela
  - Por quê? Quebraria a busca...
- Opção 1: rehash dos elementos seguintes do bloco

# Remoção em endereçamento aberto

Como fazer a remoção com endereçamento aberto?

- Não podemos apenas remover os elementos da tabela
  - Por quê? Quebraria a busca...
- Opção 1: rehash dos elementos seguintes do bloco
  - removemos os elementos até a próxima posição vazia

# Remoção em endereçamento aberto

Como fazer a remoção com endereçamento aberto?

- Não podemos apenas remover os elementos da tabela
  - Por quê? Quebraria a busca...
- Opção 1: rehash dos elementos seguintes do bloco
  - removemos os elementos até a próxima posição vazia
  - recalculamos o hash de cada um e reinserimos na tabela



# Remoção em endereçamento aberto

Como fazer a remoção com endereçamento aberto?

- Não podemos apenas remover os elementos da tabela
  - Por quê? Quebraria a busca...
- Opção 1: rehash dos elementos seguintes do bloco
  - removemos os elementos até a próxima posição vazia
  - recalculamos o hash de cada um e reinserimos na tabela
  - pode ser custoso e difícil de implementar

# Remoção em endereçamento aberto

Como fazer a remoção com endereçamento aberto?

- Não podemos apenas remover os elementos da tabela
  - Por quê? Quebraria a busca...
- Opção 1: rehash dos elementos seguintes do bloco
  - removemos os elementos até a próxima posição vazia
  - recalculamos o hash de cada um e reinserimos na tabela
  - pode ser custoso e difícil de implementar
- Opção 2: trocamos por um valor **dummy**

# Remoção em endereçamento aberto

Como fazer a remoção com endereçamento aberto?

- Não podemos apenas remover os elementos da tabela
  - Por quê? Quebraria a busca...
- Opção 1: rehash dos elementos seguintes do bloco
  - removemos os elementos até a próxima posição vazia
  - recalculamos o hash de cada um e reinserimos na tabela
  - pode ser custoso e difícil de implementar
- Opção 2: trocamos por um valor **dummy**
  - valor indica que o item foi removido

# Remoção em endereçamento aberto

Como fazer a remoção com endereçamento aberto?

- Não podemos apenas remover os elementos da tabela
  - Por quê? Quebraria a busca...
- Opção 1: rehash dos elementos seguintes do bloco
  - removemos os elementos até a próxima posição vazia
  - recalculamos o hash de cada um e reinserimos na tabela
  - pode ser custoso e difícil de implementar
- Opção 2: trocamos por um valor **dummy**
  - valor indica que o item foi removido
  - mas não pode ser o mesmo que indica espaço vazio

# Remoção em endereçamento aberto

Como fazer a remoção com endereçamento aberto?

- Não podemos apenas remover os elementos da tabela
  - Por quê? Quebraria a busca...
- Opção 1: rehash dos elementos seguintes do bloco
  - removemos os elementos até a próxima posição vazia
  - recalculamos o hash de cada um e reinserimos na tabela
  - pode ser custoso e difícil de implementar
- Opção 2: trocamos por um valor **dummy**
  - valor indica que o item foi removido
  - mas não pode ser o mesmo que indica espaço vazio
- Opção 3: marcamos o item como removido

# Remoção em endereçamento aberto

Como fazer a remoção com endereçamento aberto?

- Não podemos apenas remover os elementos da tabela
  - Por quê? Quebraria a busca...
- Opção 1: rehash dos elementos seguintes do bloco
  - removemos os elementos até a próxima posição vazia
  - recalculamos o hash de cada um e reinserimos na tabela
  - pode ser custoso e difícil de implementar
- Opção 2: trocamos por um valor **dummy**
  - valor indica que o item foi removido
  - mas não pode ser o mesmo que indica espaço vazio
- Opção 3: marcamos o item como removido
  - usamos um campo adicional

## Inserção e Busca Revisitadas

Se fizermos a remoção marcando o item como removido, precisamos mudar a inserção e a busca

## Inserção e Busca Revisitadas

Se fizermos a remoção marcando o item como removido, precisamos mudar a inserção e a busca

Inserção:



## Inserção e Busca Revisitadas

Se fizermos a remoção marcando o item como removido, precisamos mudar a inserção e a busca

Inserção:

- Calculamos a função hashing e temos um resultado  $h$

## Inserção e Busca Revisitadas

Se fizermos a remoção marcando o item como removido, precisamos mudar a inserção e a busca

Inserção:

- Calculamos a função hashing e temos um resultado  $h$
- Inserimos na primeira posição vazia ou com item removido a partir de  $h$

# Inserção e Busca Revisitadas

Se fizermos a remoção marcando o item como removido, precisamos mudar a inserção e a busca

Inserção:

- Calculamos a função hashing e temos um resultado  $h$
- Inserimos na primeira posição vazia ou com item removido a partir de  $h$

Busca:

# Inserção e Busca Revisitadas

Se fizermos a remoção marcando o item como removido, precisamos mudar a inserção e a busca

Inserção:

- Calculamos a função hashing e temos um resultado  $h$
- Inserimos na primeira posição vazia ou com item removido a partir de  $h$

Busca:

- Calculamos a função hashing e temos um resultado  $h$

# Inserção e Busca Revisitadas

Se fizermos a remoção marcando o item como removido, precisamos mudar a inserção e a busca

Inserção:

- Calculamos a função hashing e temos um resultado  $h$
- Inserimos na primeira posição vazia ou com item removido a partir de  $h$

Busca:

- Calculamos a função hashing e temos um resultado  $h$
- Procuramos cada posição a partir de  $h$  em sequência

# Inserção e Busca Revisitadas

Se fizermos a remoção marcando o item como removido, precisamos mudar a inserção e a busca

Inserção:

- Calculamos a função hashing e temos um resultado  $h$
- Inserimos na primeira posição vazia ou com item removido a partir de  $h$

Busca:

- Calculamos a função hashing e temos um resultado  $h$
- Procuramos cada posição a partir de  $h$  em sequência
  - Se encontrar o item, verifique se ele não foi removido

# Inserção e Busca Revisitadas

Se fizermos a remoção marcando o item como removido, precisamos mudar a inserção e a busca

Inserção:

- Calculamos a função hashing e temos um resultado  $h$
- Inserimos na primeira posição vazia ou com item removido a partir de  $h$

Busca:

- Calculamos a função hashing e temos um resultado  $h$
- Procuramos cada posição a partir de  $h$  em sequência
  - Se encontrar o item, verifique se ele não foi removido
  - Passe por cima de itens removidos

# Inserção e Busca Revisitadas

Se fizermos a remoção marcando o item como removido, precisamos mudar a inserção e a busca

Inserção:

- Calculamos a função hashing e temos um resultado  $h$
- Inserimos na primeira posição vazia ou com item removido a partir de  $h$

Busca:

- Calculamos a função hashing e temos um resultado  $h$
- Procuramos cada posição a partir de  $h$  em sequência
  - Se encontrar o item, verifique se ele não foi removido
  - Passe por cima de itens removidos
  - Pare ao encontrar uma posição vazia



# Inserção e Busca Revisitadas

Se fizermos a remoção marcando o item como removido, precisamos mudar a inserção e a busca

Inserção:

- Calculamos a função hashing e temos um resultado  $h$
- Inserimos na primeira posição vazia ou com item removido a partir de  $h$

Busca:

- Calculamos a função hashing e temos um resultado  $h$
- Procuramos cada posição a partir de  $h$  em sequência
  - Se encontrar o item, verifique se ele não foi removido
  - Passe por cima de itens removidos
  - Pare ao encontrar uma posição vazia
- Cuidado para não ciclar...

## Hashing duplo

É como a sondagem linear:

## Hashing duplo

É como a sondagem linear:

- Estratégia mais geral para lidar com conflitos

## Hashing duplo

É como a sondagem linear:

- Estratégia mais geral para lidar com conflitos
- Ao invés de saltarmos sempre de **1**, saltamos de  $hash_2(k)$

## Hashing duplo

É como a sondagem linear:

- Estratégia mais geral para lidar com conflitos
- Ao invés de saltarmos sempre de **1**, saltamos de  $hash_2(k)$
- Onde  $hash_2$  é uma segunda função de hashing

## Hashing duplo

É como a sondagem linear:

- Estratégia mais geral para lidar com conflitos
- Ao invés de saltarmos sempre de **1**, saltamos de  $hash_2(k)$
- Onde  $hash_2$  é uma segunda função de hashing

Isso é,

$$h(k, i) = (hash_1(k) + i \cdot hash_2(k)) \mod M$$

## Hashing duplo

É como a sondagem linear:

- Estratégia mais geral para lidar com conflitos
- Ao invés de saltarmos sempre de **1**, saltamos de  $hash_2(k)$
- Onde  $hash_2$  é uma segunda função de hashing

Isso é,

$$h(k, i) = (hash_1(k) + i \cdot hash_2(k)) \mod M$$

Cuidados:

## Hashing duplo

É como a sondagem linear:

- Estratégia mais geral para lidar com conflitos
- Ao invés de saltarmos sempre de **1**, saltamos de  $hash_2(k)$
- Onde  $hash_2$  é uma segunda função de hashing

Isso é,

$$h(k, i) = (hash_1(k) + i \cdot hash_2(k)) \mod M$$

Cuidados:

- $hash_2(k)$  nunca pode ser zero



## Hashing duplo

É como a sondagem linear:

- Estratégia mais geral para lidar com conflitos
- Ao invés de saltarmos sempre de **1**, saltamos de  $hash_2(k)$
- Onde  $hash_2$  é uma segunda função de hashing

Isso é,

$$h(k, i) = (hash_1(k) + i \cdot hash_2(k)) \mod M$$

Cuidados:

- $hash_2(k)$  nunca pode ser zero
- $hash_2(k)$  precisa ser coprimo com  $M$

## Hashing duplo

É como a sondagem linear:

- Estratégia mais geral para lidar com conflitos
- Ao invés de saltarmos sempre de **1**, saltamos de  $hash_2(k)$
- Onde  $hash_2$  é uma segunda função de hashing

Isso é,

$$h(k, i) = (hash_1(k) + i \cdot hash_2(k)) \pmod{M}$$

Cuidados:

- $hash_2(k)$  nunca pode ser zero
- $hash_2(k)$  precisa ser coprimo com  $M$ 
  - garante que as sequências são longas

## Hashing duplo

É como a sondagem linear:

- Estratégia mais geral para lidar com conflitos
- Ao invés de saltarmos sempre de **1**, saltamos de  $hash_2(k)$
- Onde  $hash_2$  é uma segunda função de hashing

Isso é,

$$h(k, i) = (hash_1(k) + i \cdot hash_2(k)) \mod M$$

Cuidados:

- $hash_2(k)$  nunca pode ser zero
- $hash_2(k)$  precisa ser coprimo com  $M$ 
  - garante que as sequências são longas

Exemplos:

## Hashing duplo

É como a sondagem linear:

- Estratégia mais geral para lidar com conflitos
- Ao invés de saltarmos sempre de **1**, saltamos de  $hash_2(k)$
- Onde  $hash_2$  é uma segunda função de hashing

Isso é,

$$h(k, i) = (hash_1(k) + i \cdot hash_2(k)) \mod M$$

Cuidados:

- $hash_2(k)$  nunca pode ser zero
- $hash_2(k)$  precisa ser coprimo com  $M$ 
  - garante que as sequências são longas

Exemplos:

- Escolha  $M$  como uma **potência de 2** e faça que  $hash_2(k)$  seja sempre **ímpar**

## Hashing duplo

É como a sondagem linear:

- Estratégia mais geral para lidar com conflitos
- Ao invés de saltarmos sempre de **1**, saltamos de  $hash_2(k)$
- Onde  $hash_2$  é uma segunda função de hashing

Isso é,

$$h(k, i) = (hash_1(k) + i \cdot hash_2(k)) \pmod{M}$$

Cuidados:

- $hash_2(k)$  nunca pode ser zero
- $hash_2(k)$  precisa ser coprimo com  $M$ 
  - garante que as sequências são longas

Exemplos:

- Escolha  $M$  como uma **potência de 2** e faça que  $hash_2(k)$  seja sempre **ímpar**
- Escolha  $M$  como um número **primo** e faça que  $hash_2(k) < M$

## Sondagem linear e Hashing duplo<sup>1</sup>

Sondagem linear - número de acessos médio por busca

$n/M$	1/2	2/3	3/4	9/10
com sucesso	1.5	2.0	3.0	5.5
sem sucesso	2.5	5.0	8.5	55.5

---

<sup>1</sup>Baseado em Sedgewick, R. Algorithms in C, third edition, Addison-Wesley. 1998.

## Sondagem linear e Hashing duplo<sup>1</sup>

Sondagem linear - número de acessos médio por busca

$n/M$	1/2	2/3	3/4	9/10
com sucesso	1.5	2.0	3.0	5.5
sem sucesso	2.5	5.0	8.5	55.5

Hashing duplo - número de acessos médio por busca

$n/M$	1/2	2/3	3/4	9/10
com sucesso	1.4	1.6	1.8	2.6
sem sucesso	1.5	2.0	3.0	5.5

---

<sup>1</sup>Baseado em Sedgewick, R. Algorithms in C, third edition, Addison-Wesley. 1998.

## Sondagem linear e Hashing duplo<sup>1</sup>

Sondagem linear - número de acessos médio por busca

$n/M$	1/2	2/3	3/4	9/10
com sucesso	1.5	2.0	3.0	5.5
sem sucesso	2.5	5.0	8.5	55.5

Hashing duplo - número de acessos médio por busca

$n/M$	1/2	2/3	3/4	9/10
com sucesso	1.4	1.6	1.8	2.6
sem sucesso	1.5	2.0	3.0	5.5

De qualquer forma, é muito importante não deixar a tabela encher muito:

---

<sup>1</sup>Baseado em Sedgewick, R. Algorithms in C, third edition, Addison-Wesley. 1998.



## Sondagem linear e Hashing duplo<sup>1</sup>

Sondagem linear - número de acessos médio por busca

$n/M$	1/2	2/3	3/4	9/10
com sucesso	1.5	2.0	3.0	5.5
sem sucesso	2.5	5.0	8.5	55.5

Hashing duplo - número de acessos médio por busca

$n/M$	1/2	2/3	3/4	9/10
com sucesso	1.4	1.6	1.8	2.6
sem sucesso	1.5	2.0	3.0	5.5

De qualquer forma, é muito importante não deixar a tabela encher muito:

- Você pode aumentar o tamanho da tabela dinamicamente

---

<sup>1</sup>Baseado em Sedgewick, R. Algorithms in C, third edition, Addison-Wesley. 1998.

## Sondagem linear e Hashing duplo<sup>1</sup>

Sondagem linear - número de acessos médio por busca

$n/M$	1/2	2/3	3/4	9/10
com sucesso	1.5	2.0	3.0	5.5
sem sucesso	2.5	5.0	8.5	55.5

Hashing duplo - número de acessos médio por busca

$n/M$	1/2	2/3	3/4	9/10
com sucesso	1.4	1.6	1.8	2.6
sem sucesso	1.5	2.0	3.0	5.5

De qualquer forma, é muito importante não deixar a tabela encher muito:

- Você pode aumentar o tamanho da tabela dinamicamente
- Porém, precisa fazer um rehash de cada elemento para a nova tabela

---

<sup>1</sup>Baseado em Sedgewick, R. Algorithms in C, third edition, Addison-Wesley. 1998.

## Conclusão

Hashing é uma boa estrutura de dados para

## Conclusão

Hashing é uma boa estrutura de dados para

- inserir, remover e buscar dados pela sua chave rapidamente

# Conclusão

Hashing é uma boa estrutura de dados para

- inserir, remover e buscar dados pela sua chave rapidamente
- com uma boa função de hashing, essas operações levam tempo  $O(1)$

# Conclusão

Hashing é uma boa estrutura de dados para

- inserir, remover e buscar dados pela sua chave rapidamente
- com uma boa função de hashing, essas operações levam tempo  $O(1)$
- mas não é boa se quisermos fazer operação relacionadas a ordem das chaves

# Conclusão

Hashing é uma boa estrutura de dados para

- inserir, remover e buscar dados pela sua chave rapidamente
- com uma boa função de hashing, essas operações levam tempo  $O(1)$
- mas não é boa se quisermos fazer operação relacionadas a ordem das chaves

Escolhendo a implementação:

# Conclusão

Hashing é uma boa estrutura de dados para

- inserir, remover e buscar dados pela sua chave rapidamente
- com uma boa função de hashing, essas operações levam tempo  $O(1)$
- mas não é boa se quisermos fazer operação relacionadas a ordem das chaves

Escolhendo a implementação:

- Sondagem linear é o mais rápido se a tabela for esparsa



# Conclusão

Hashing é uma boa estrutura de dados para

- inserir, remover e buscar dados pela sua chave rapidamente
- com uma boa função de hashing, essas operações levam tempo  $O(1)$
- mas não é boa se quisermos fazer operação relacionadas a ordem das chaves

Escolhendo a implementação:

- Sondagem linear é o mais rápido se a tabela for esparsa
- Hashing duplo usa melhor a memória

# Conclusão

Hashing é uma boa estrutura de dados para

- inserir, remover e buscar dados pela sua chave rapidamente
- com uma boa função de hashing, essas operações levam tempo  $O(1)$
- mas não é boa se quisermos fazer operação relacionadas a ordem das chaves

Escolhendo a implementação:

- Sondagem linear é o mais rápido se a tabela for esparsa
- Hashing duplo usa melhor a memória
  - mas gasta mais tempo para computar a segunda função de hash

# Conclusão

Hashing é uma boa estrutura de dados para

- inserir, remover e buscar dados pela sua chave rapidamente
- com uma boa função de hashing, essas operações levam tempo  $O(1)$
- mas não é boa se quisermos fazer operação relacionadas a ordem das chaves

Escolhendo a implementação:

- Sondagem linear é o mais rápido se a tabela for esparsa
- Hashing duplo usa melhor a memória
  - mas gasta mais tempo para computar a segunda função de hash
- Encadeamento separado é mais fácil de implementar

# Conclusão

Hashing é uma boa estrutura de dados para

- inserir, remover e buscar dados pela sua chave rapidamente
- com uma boa função de hashing, essas operações levam tempo  $O(1)$
- mas não é boa se quisermos fazer operação relacionadas a ordem das chaves

Escolhendo a implementação:

- Sondagem linear é o mais rápido se a tabela for esparsa
- Hashing duplo usa melhor a memória
  - mas gasta mais tempo para computar a segunda função de hash
- Encadeamento separado é mais fácil de implementar
  - Usa memória a mais para os ponteiros

## Conclusão

Além disso, funções de hashing têm várias outras aplicações, ex:

# Conclusão

Além disso, funções de hashing têm várias outras aplicações, ex:

- Para evitar erros de transmissão, podemos, além de informar uma chave, transmitir o resultado da função de hashing.

# Conclusão

Além disso, funções de hashing têm várias outras aplicações, ex:

- Para evitar erros de transmissão, podemos, além de informar uma chave, transmitir o resultado da função de hashing.
  - dígitos verificadores

# Conclusão

Além disso, funções de hashing têm várias outras aplicações, ex:

- Para evitar erros de transmissão, podemos, além de informar uma chave, transmitir o resultado da função de hashing.
  - dígitos verificadores
  - sequências de verificação para arquivos (MD5 e SHA)



# Conclusão

Além disso, funções de hashing têm várias outras aplicações, ex:

- Para evitar erros de transmissão, podemos, além de informar uma chave, transmitir o resultado da função de hashing.
  - dígitos verificadores
  - sequências de verificação para arquivos (MD5 e SHA)
- Guardamos o hash de uma senha no banco de dados ao invés da senha em si

# Conclusão

Além disso, funções de hashing têm várias outras aplicações, ex:

- Para evitar erros de transmissão, podemos, além de informar uma chave, transmitir o resultado da função de hashing.
  - dígitos verificadores
  - sequências de verificação para arquivos (MD5 e SHA)
- Guardamos o hash de uma senha no banco de dados ao invés da senha em si
  - evitamos vazamento de informação em caso de ataque

# Conclusão

Além disso, funções de hashing têm várias outras aplicações, ex:

- Para evitar erros de transmissão, podemos, além de informar uma chave, transmitir o resultado da função de hashing.
  - dígitos verificadores
  - sequências de verificação para arquivos (MD5 e SHA)
- Guardamos o hash de uma senha no banco de dados ao invés da senha em si
  - evitamos vazamento de informação em caso de ataque
  - mas temos que garantir que a probabilidade de duas senhas terem o mesmo hash seja ínfima...

## Exercício

Você precisa de um TAD que representa um conjunto  $C$ :

## Exercício

Você precisa de um TAD que representa um conjunto  $C$ :

- Permite inserir elementos

## Exercício

Você precisa de um TAD que representa um conjunto  $C$ :

- Permite inserir elementos
- Não permite remover elementos

## Exercício

Você precisa de um TAD que representa um conjunto  $C$ :

- Permite inserir elementos
- Não permite remover elementos
- Permite “testar” se um elemento  $x$  está presente

## Exercício

Você precisa de um TAD que representa um conjunto  $C$ :

- Permite inserir elementos
- Não permite remover elementos
- Permite “testar” se um elemento  $x$  está presente
  - Se  $x \notin C$ , responde, com certeza, que não está



## Exercício

Você precisa de um TAD que representa um conjunto  $C$ :

- Permite inserir elementos
- Não permite remover elementos
- Permite “testar” se um elemento  $x$  está presente
  - Se  $x \notin C$ , responde, com certeza, que não está
  - Se  $x \in C$ , pode responder que está ou que não está

## Exercício

Você precisa de um TAD que representa um conjunto  $C$ :

- Permite inserir elementos
- Não permite remover elementos
- Permite “testar” se um elemento  $x$  está presente
  - Se  $x \notin C$ , responde, com certeza, que não está
  - Se  $x \in C$ , pode responder que está ou que não está
  - Queremos que a probabilidade de errar seja pequena

## Exercício

Você precisa de um TAD que representa um conjunto  $C$ :

- Permite inserir elementos
- Não permite remover elementos
- Permite “testar” se um elemento  $x$  está presente
  - Se  $x \notin C$ , responde, com certeza, que não está
  - Se  $x \in C$ , pode responder que está ou que não está
  - Queremos que a probabilidade de errar seja pequena

Como você implementaria esse TAD sendo econômico com memória?

# Solução

Podemos usar Hash!

# Solução

Podemos usar Hash!

- Faça um vetor binário de tamanho  $M$

# Solução

Podemos usar Hash!

- Faça um vetor binário de tamanho  $M$
- Inicialize todos os bits como  $0$

# Solução

Podemos usar Hash!

- Faça um vetor binário de tamanho  $M$
- Inicialize todos os bits como  $0$
- Para inserir um elemento, use uma função de hashing para transformar o elemento em um número entre  $0$  e  $M - 1$

# Solução

Podemos usar Hash!

- Faça um vetor binário de tamanho  $M$
- Inicialize todos os bits como  $0$
- Para inserir um elemento, use uma função de hashing para transformar o elemento em um número entre  $0$  e  $M - 1$ 
  - Defina o bit correspondente como  $1$



# Solução

Podemos usar Hash!

- Faça um vetor binário de tamanho  $M$
- Inicialize todos os bits como  $0$
- Para inserir um elemento, use uma função de hashing para transformar o elemento em um número entre  $0$  e  $M - 1$ 
  - Defina o bit correspondente como  $1$
- Para testar se um elemento está presente, use a mesma função de hashing

# Solução

Podemos usar Hash!

- Faça um vetor binário de tamanho  $M$
- Inicialize todos os bits como  $0$
- Para inserir um elemento, use uma função de hashing para transformar o elemento em um número entre  $0$  e  $M - 1$ 
  - Defina o bit correspondente como  $1$
- Para testar se um elemento está presente, use a mesma função de hashing
  - Se o bit correspondente for  $1$ , responda que está

# Solução

Podemos usar Hash!

- Faça um vetor binário de tamanho  $M$
- Inicialize todos os bits como  $0$
- Para inserir um elemento, use uma função de hashing para transformar o elemento em um número entre  $0$  e  $M - 1$ 
  - Defina o bit correspondente como  $1$
- Para testar se um elemento está presente, use a mesma função de hashing
  - Se o bit correspondente for  $1$ , responda que está
  - Se o bit correspondente for  $0$ , responda que não está

# Solução

Podemos usar Hash!

- Faça um vetor binário de tamanho  $M$
- Inicialize todos os bits como  $0$
- Para inserir um elemento, use uma função de hashing para transformar o elemento em um número entre  $0$  e  $M - 1$ 
  - Defina o bit correspondente como  $1$
- Para testar se um elemento está presente, use a mesma função de hashing
  - Se o bit correspondente for  $1$ , responda que está
  - Se o bit correspondente for  $0$ , responda que não está
- Se  $M$  for grande o suficiente, a probabilidade de erro é pequena

# Solução

Podemos usar Hash!

- Faça um vetor binário de tamanho  $M$
- Inicialize todos os bits como  $0$
- Para inserir um elemento, use uma função de hashing para transformar o elemento em um número entre  $0$  e  $M - 1$ 
  - Defina o bit correspondente como  $1$
- Para testar se um elemento está presente, use a mesma função de hashing
  - Se o bit correspondente for  $1$ , responda que está
  - Se o bit correspondente for  $0$ , responda que não está
- Se  $M$  for grande o suficiente, a probabilidade de erro é pequena

Essa é a ideia básica de um **Filtro de Bloom**

# Solução

Podemos usar Hash!

- Faça um vetor binário de tamanho  $M$
- Inicialize todos os bits como  $0$
- Para inserir um elemento, use uma função de hashing para transformar o elemento em um número entre  $0$  e  $M - 1$ 
  - Defina o bit correspondente como  $1$
- Para testar se um elemento está presente, use a mesma função de hashing
  - Se o bit correspondente for  $1$ , responda que está
  - Se o bit correspondente for  $0$ , responda que não está
- Se  $M$  for grande o suficiente, a probabilidade de erro é pequena

Essa é a ideia básica de um **Filtro de Bloom**

- Uma estrutura de dados mais avançada baseada em hashing

Dúvidas?