MC-202 — Aula 22
Tabela de Espalhamento

Lehilton Pedrosa

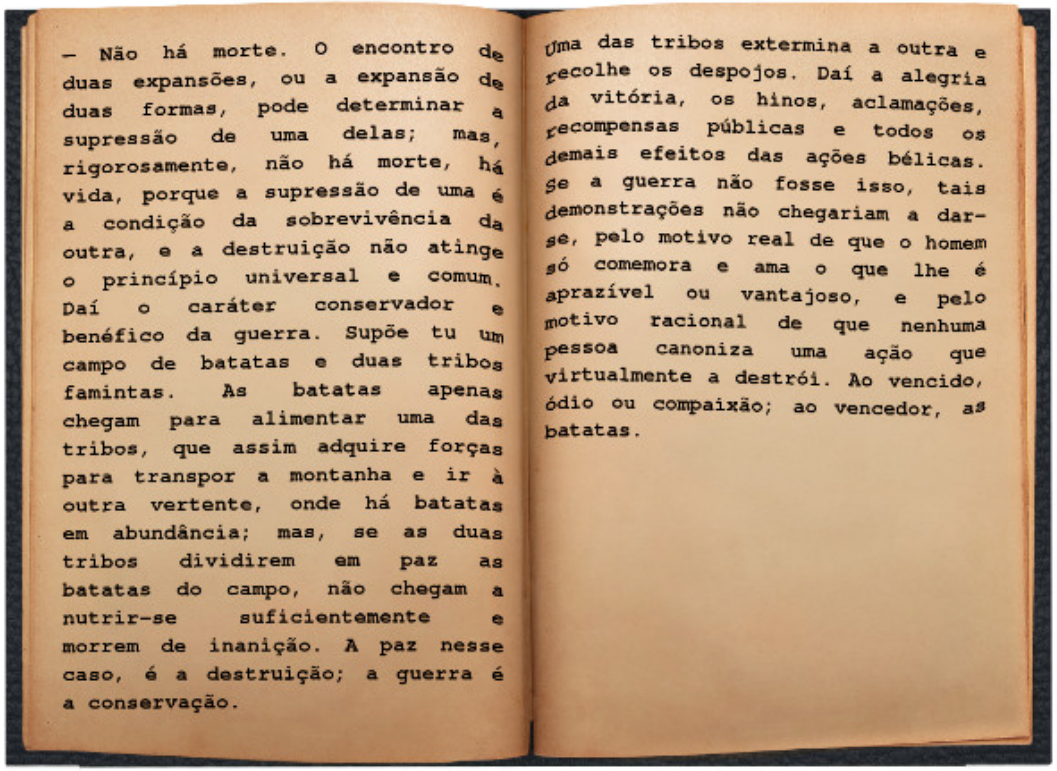
Instituto de Computação – Unicamp

Segundo Semestre de 2015

Roteiro

Introdução

Queremos contar o número de ocorrências de cada palavra da biblioteca.

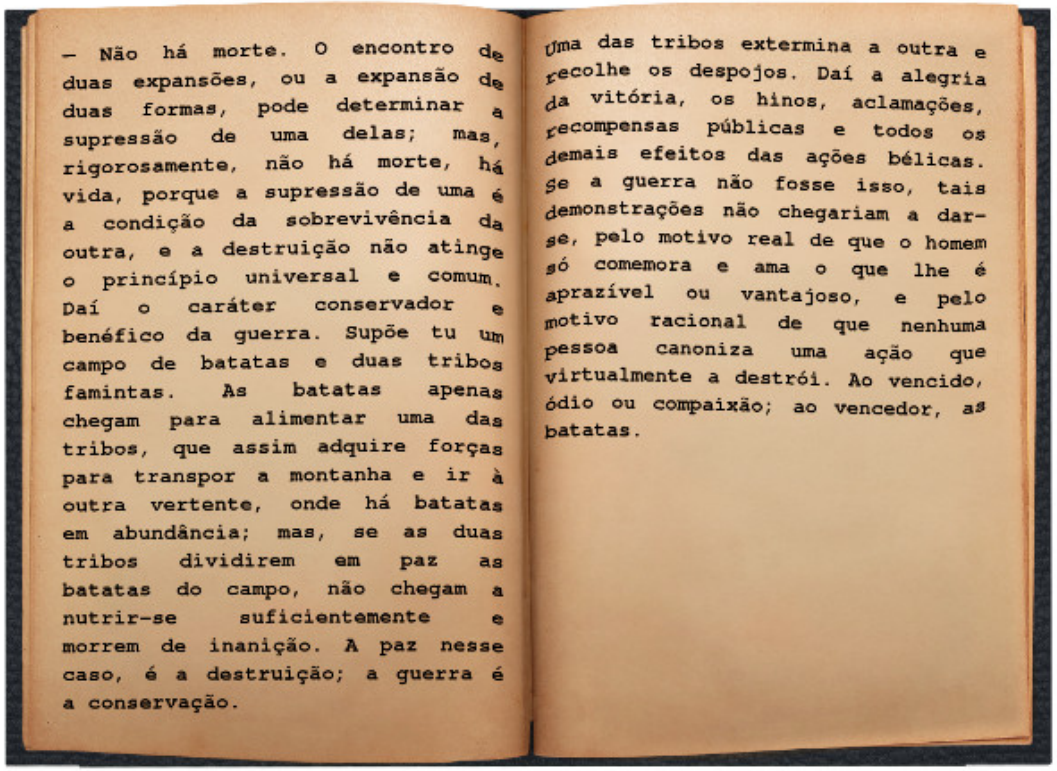


— Não há morte. O encontro de duas expansões, ou a expansão de duas formas, pode determinar a supressão de uma delas; mas, rigorosamente, não há morte, há vida, porque a supressão de uma é a condição da sobrevivência da outra, e a destruição não atinge o princípio universal e comum. Daí o caráter conservador e benéfico da guerra. Supõe tu um campo de batatas e duas tribos famintas. As batatas apenas chegam para alimentar uma das tribos, que assim adquire forças para transpor a montanha e ir à outra vertente, onde há batatas em abundância; mas, se as duas tribos dividirem em paz as batatas do campo, não chegam a nutrir-se suficientemente e morrem de inanição. A paz nesse caso, é a destruição; a guerra é a conservação.

Uma das tribos extermina a outra e recolhe os despojos. Daí a alegria da vitória, os hinos, aclamações, recompensas públicas e todos os demais efeitos das ações bélicas. Se a guerra não fosse isso, tais demonstrações não chegariam a dar-se, pelo motivo real de que o homem só comemora e ama o que lhe é aprazível ou vantajoso, e pelo motivo racional de que nenhuma pessoa canoniza uma ação que virtualmente a destrói. Ao vencido, ódio ou compaixão; ao vencedor, as batatas.

Introdução

Queremos contar o número de ocorrências de cada palavra da biblioteca.



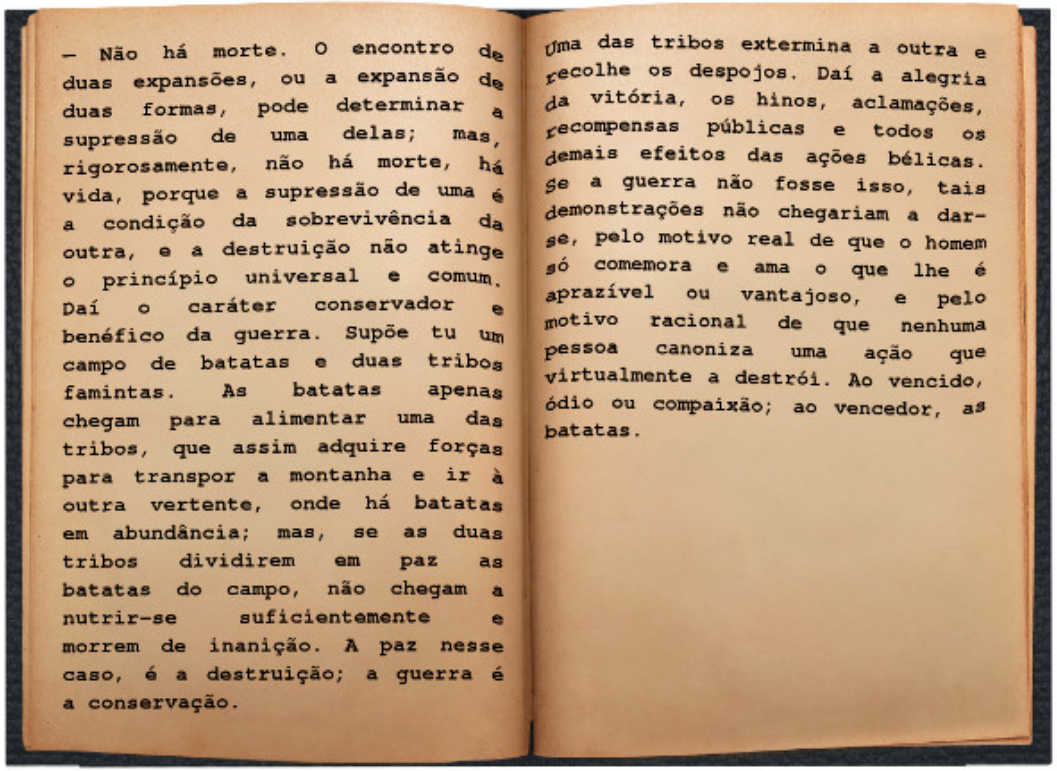
— Não há morte. O encontro de duas expansões, ou a expansão de duas formas, pode determinar a supressão de uma delas; mas, rigorosamente, não há morte, há vida, porque a supressão de uma é a condição da sobrevivência da outra, e a destruição não atinge o princípio universal e comum. Daí o caráter conservador e benéfico da guerra. Supõe tu um campo de batatas e duas tribos famintas. As batatas apenas chegam para alimentar uma das tribos, que assim adquire forças para transpor a montanha e ir à outra vertente, onde há batatas em abundância; mas, se as duas tribos dividirem em paz as batatas do campo, não chegam a nutrir-se suficientemente e morrem de inanição. A paz nesse caso, é a destruição; a guerra é a conservação.

uma das tribos extermina a outra e recolhe os despojos. Daí a alegria da vitória, os hinos, aclamações, recompensas públicas e todos os demais efeitos das ações bélicas. Se a guerra não fosse isso, tais demonstrações não chegariam a dar-se, pelo motivo real de que o homem só comemora e ama o que lhe é aprazível ou vantajoso, e pelo motivo racional de que nenhuma pessoa canoniza uma ação que virtualmente a destrói. Ao vencido, ódio ou compaixão; ao vencedor, as batatas.

- no idioma, há cerca de milhares de palavras

Introdução

Queremos contar o número de ocorrências de cada palavra da biblioteca.



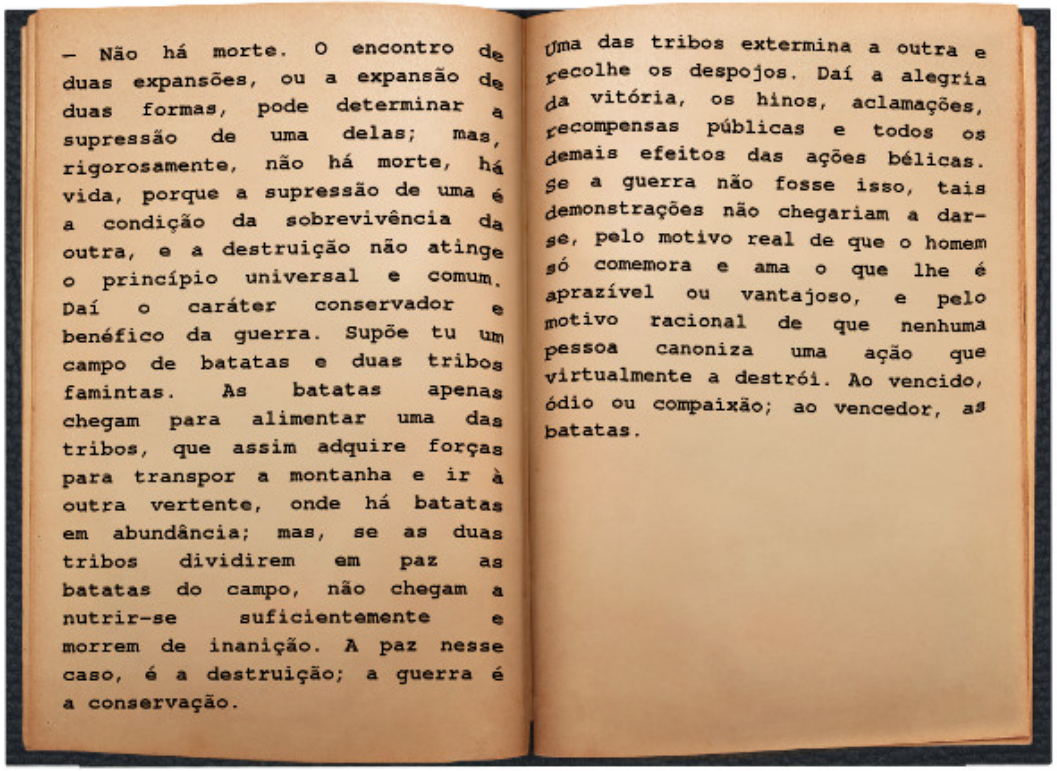
– Não há morte. O encontro de duas expansões, ou a expansão de duas formas, pode determinar a supressão de uma delas; mas, rigorosamente, não há morte, há vida, porque a supressão de uma é a condição da sobrevivência da outra, e a destruição não atinge o princípio universal e comum. Daí o caráter conservador e benéfico da guerra. Supõe tu um campo de batatas e duas tribos famintas. As batatas apenas chegam para alimentar uma das tribos, que assim adquire forças para transpor a montanha e ir à outra vertente, onde há batatas em abundância; mas, se as duas tribos dividirem em paz as batatas do campo, não chegam a nutrir-se suficientemente e morrem de inanição. A paz nesse caso, é a destruição; a guerra é a conservação.

Uma das tribos extermina a outra e recolhe os despojos. Daí a alegria da vitória, os hinos, aclamações, recompensas públicas e todos os demais efeitos das ações bélicas. Se a guerra não fosse isso, tais demonstrações não chegariam a dar-se, pelo motivo real de que o homem só comemora e ama o que lhe é aprazível ou vantajoso, e pelo motivo racional de que nenhuma pessoa canoniza uma ação que virtualmente a destrói. Ao vencido, ódio ou compaixão; ao vencedor, as batatas.

- no idioma, há cerca de milhares de palavras
- mas no total, há milhões de ocorrências!

Introdução

Queremos contar o número de ocorrências de cada palavra da biblioteca.



— Não há morte. O encontro de duas expansões, ou a expansão de duas formas, pode determinar a supressão de uma delas; mas, rigorosamente, não há morte, há vida, porque a supressão de uma é a condição da sobrevivência da outra, e a destruição não atinge o princípio universal e comum. Daí o caráter conservador e benéfico da guerra. Supõe tu um campo de batatas e duas tribos famintas. As batatas apenas chegam para alimentar uma das tribos, que assim adquire forças para transpor a montanha e ir à outra vertente, onde há batatas em abundância; mas, se as duas tribos dividirem em paz as batatas do campo, não chegam a nutrir-se suficientemente e morrem de inanição. A paz nesse caso, é a destruição; a guerra é a conservação.

Uma das tribos extermina a outra e recolhe os despojos. Daí a alegria da vitória, os hinos, aclamações, recompensas públicas e todos os demais efeitos das ações bélicas. Se a guerra não fosse isso, tais demonstrações não chegariam a dar-se, pelo motivo real de que o homem só comemora e ama o que lhe é aprazível ou vantajoso, e pelo motivo racional de que nenhuma pessoa canoniza uma ação que virtualmente a destrói. Ao vencido, ódio ou compaixão; ao vencedor, as batatas.

- no idioma, há cerca de milhares de palavras
- mas no total, há milhões de ocorrências!

Problema: usar uma árvore balanceada já não parece tão rápido

Objetivo: acesso direto

Objetivo: acesso direto

Exemplo, para algumas palavras:

Objetivo: acesso direto

Exemplo, para algumas palavras:

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

Objetivo: acesso direto

Exemplo, para algumas palavras:

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

Acesso direto

Queremos acessar em tempo $O(1)$, como se fosse um vetor:

Objetivo: acesso direto

Exemplo, para algumas palavras:

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

Acesso direto

Queremos acessar em tempo $O(1)$, como se fosse um vetor:

```
ocorrencias['igreja'] = 8
```

Uma tentativa

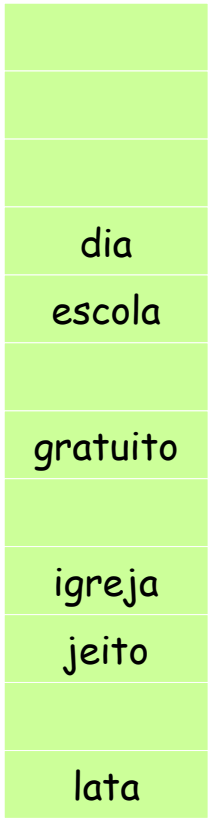
Ideia

Uma tentativa

Ideia

- guardamos em um vetor

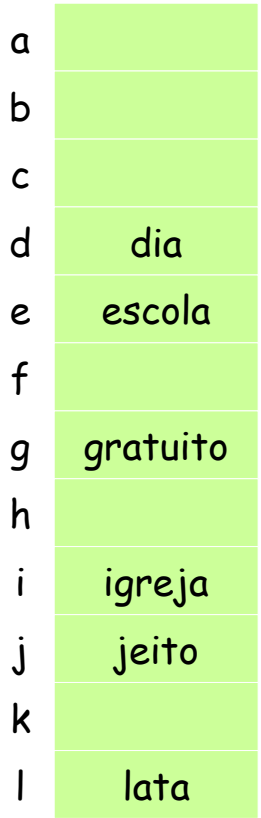
Uma tentativa



Ideia

- guardamos em um vetor

Uma tentativa



Ideia

- guardamos em um vetor
- escolhemos a posição de acordo com a "primeira letra"

Uma tentativa

0	a	
1	b	
2	c	
3	d	dia
4	e	escola
5	f	
6	g	gratuito
7	h	
8	i	igreja
9	j	jeito
10	k	
11	l	lata

Ideia

- guardamos em um vetor
- escolhemos a posição de acordo com a "primeira letra"

Uma tentativa

Ideia

- guardamos em um vetor
- escolhemos a posição de acordo com a "primeira letra"

Uma tentativa

Ideia

- guardamos em um vetor
- escolhemos a posição de acordo com a "primeira letra"

Testando

Testando

Inserir: bola

Inserindo bola

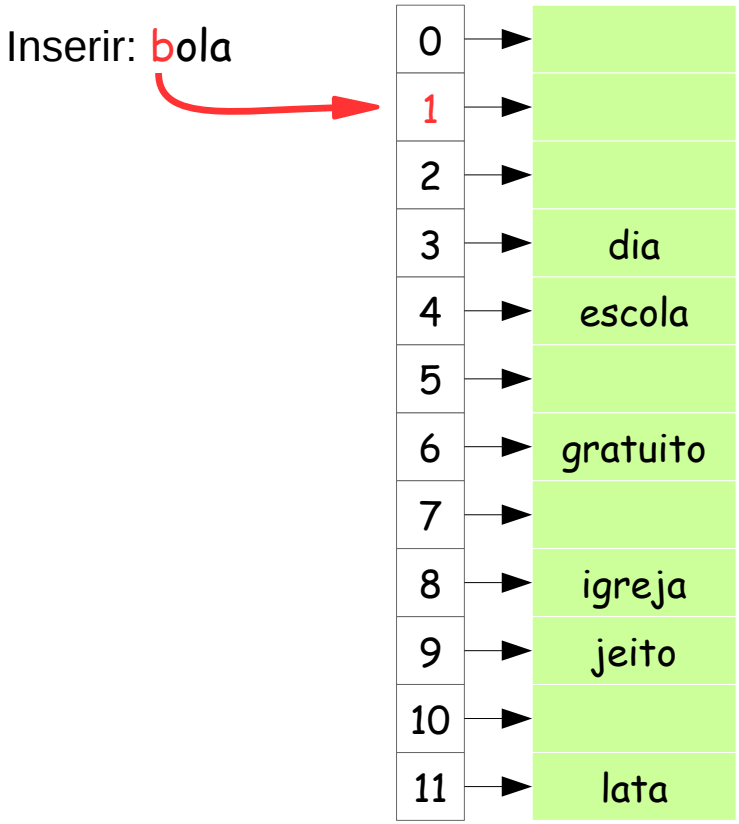
Testando

Inserir: bola

Inserindo bola

- descobrimos a posição pela letra da chave

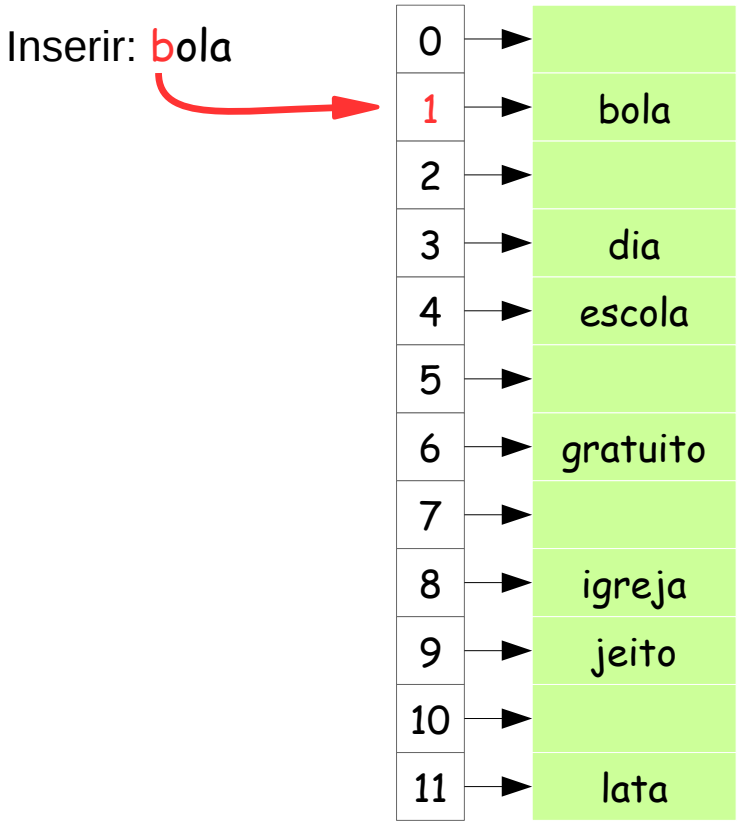
Testando



Inserindo bola

- descobrimos a posição pela letra da chave

Testando



Inserindo bola

- descobrimos a posição pela letra da chave
- guardamos o ponteiro do nó correspondente

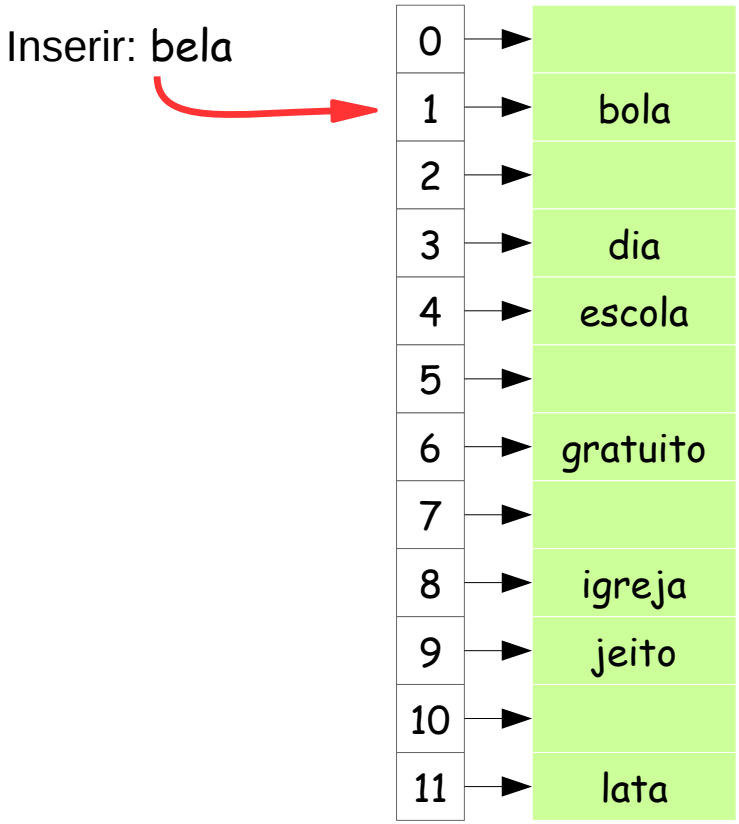
Lidando com conflitos

Lidando com conflitos

Inserir: bela

Inserindo bela

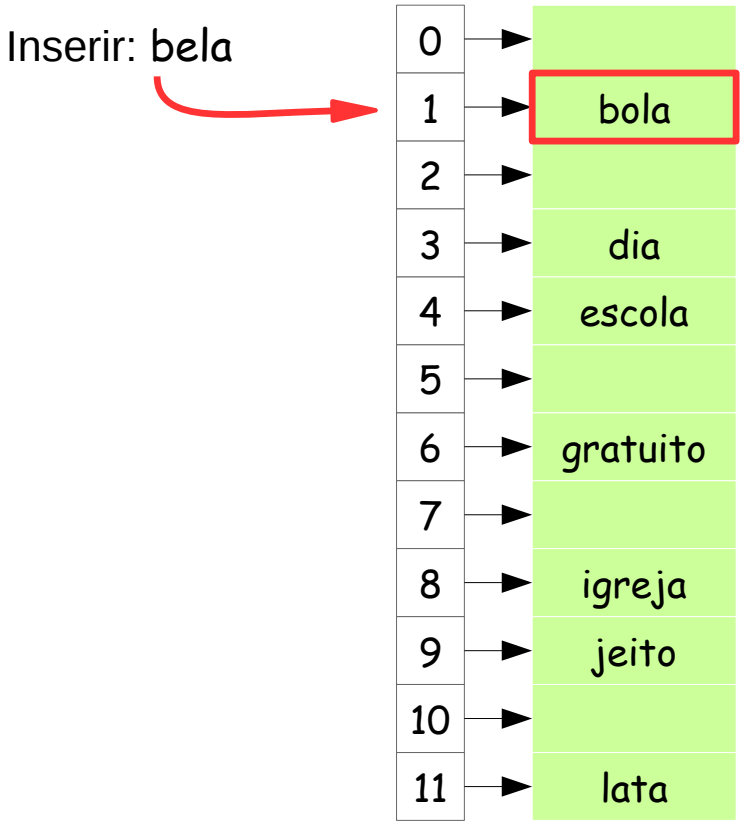
Lidando com conflitos



Inserindo bela

- descobrimos a posição pela letra da chave

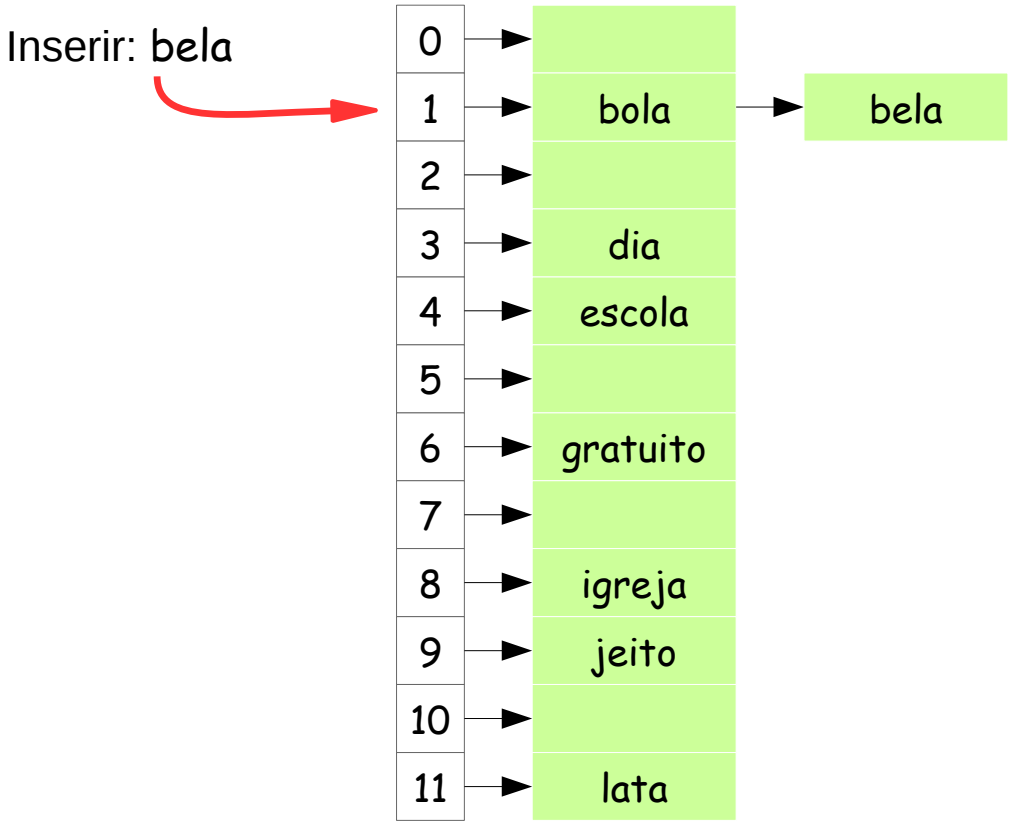
Lidando com conflitos



Inserindo bela

- descobrimos a posição pela letra da chave
- se já houver elemento

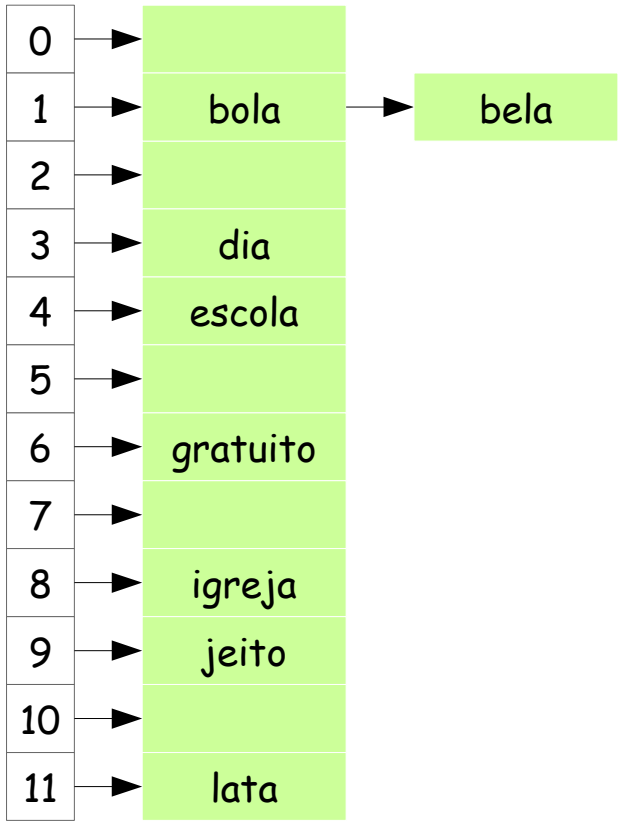
Lidando com conflitos



Inserindo bela

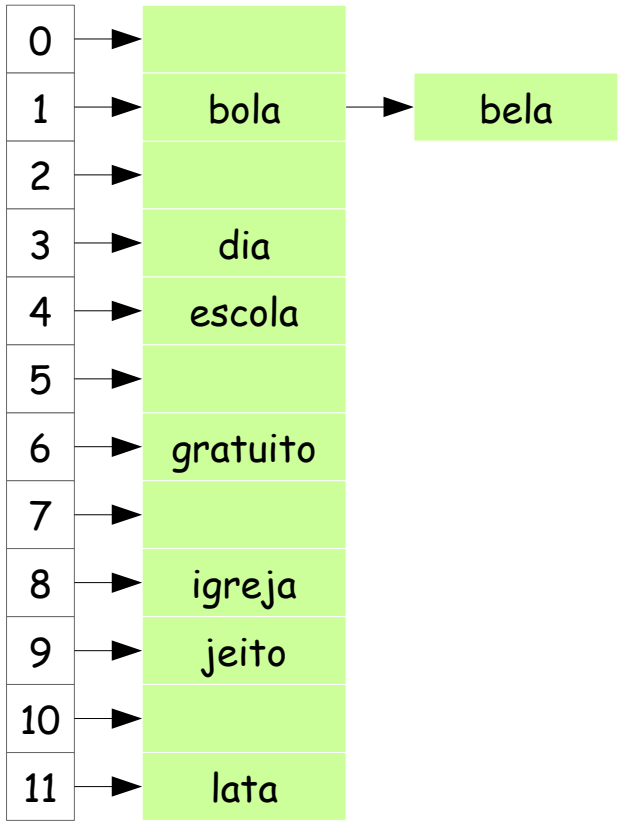
- descobrimos a posição pela letra da chave
- se já houver elemento, adicionamos no fim da lista

Continuando nosso teste



Continuando nosso teste

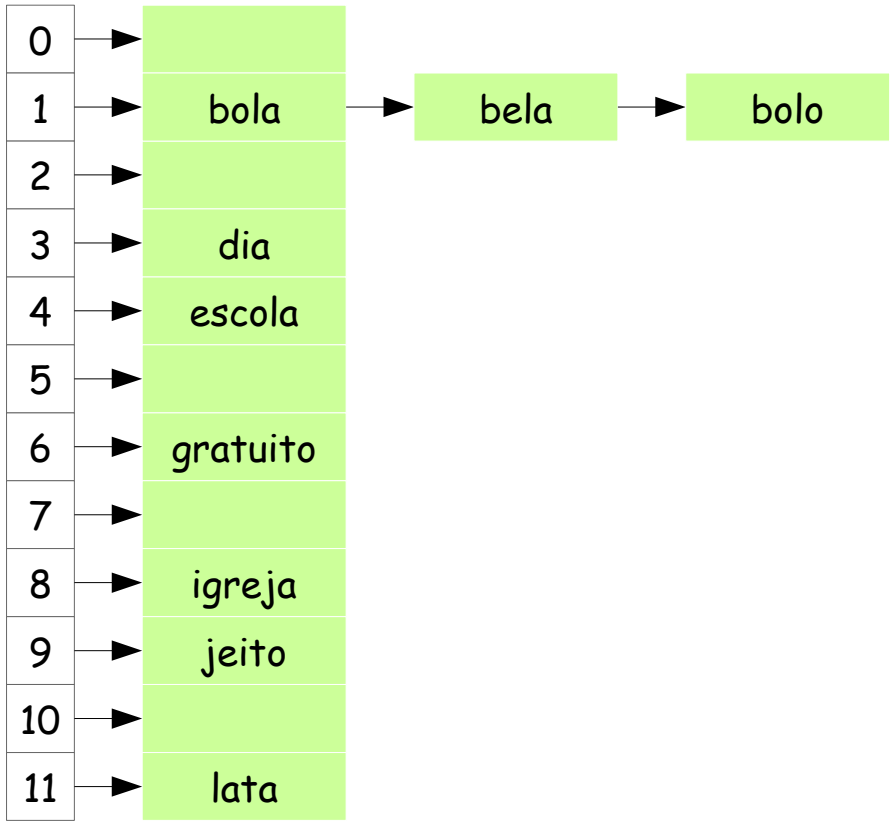
Inserir: bolo



Inserindo vários com letra b

Continuando nosso teste

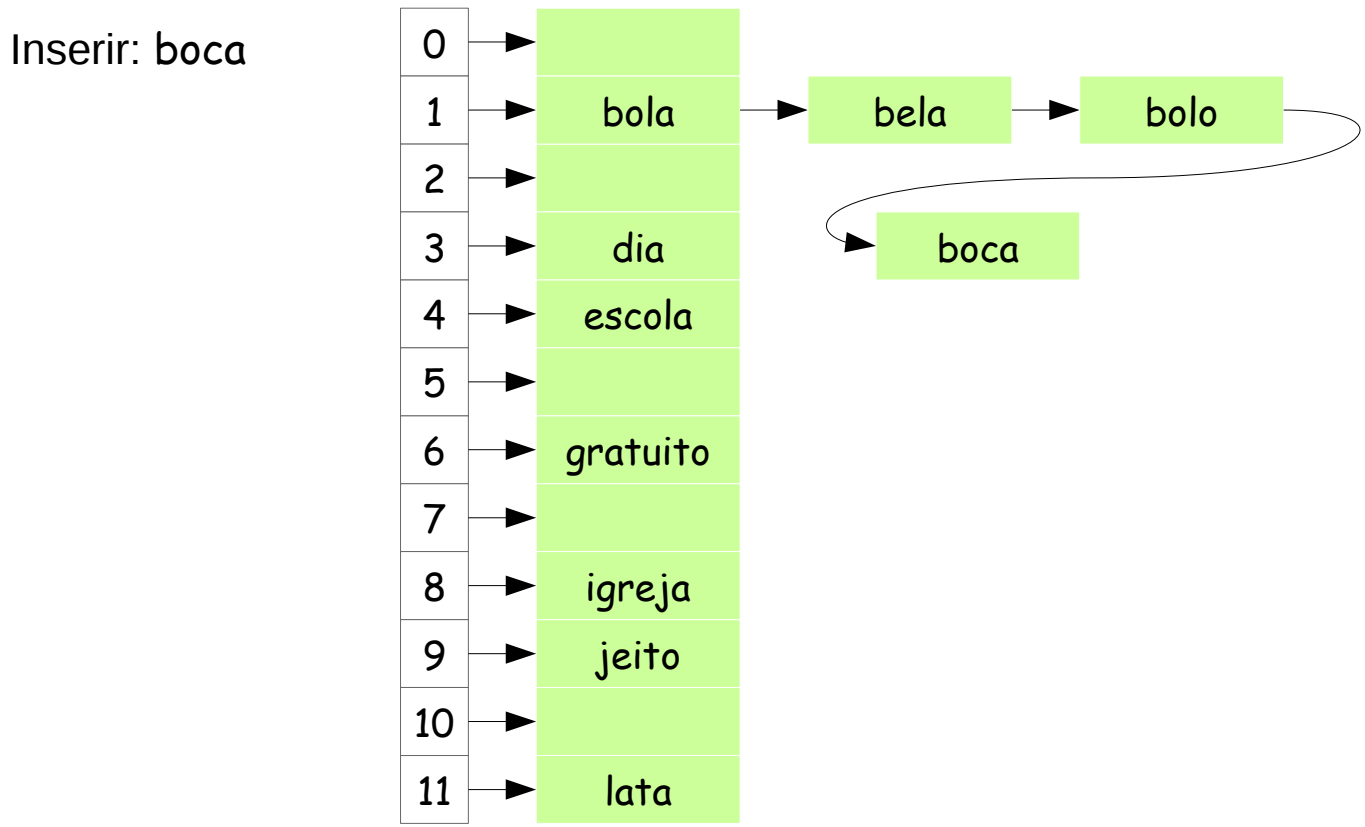
Inserir: bolo



Inserindo vários com letra b

- inserimos bolo,

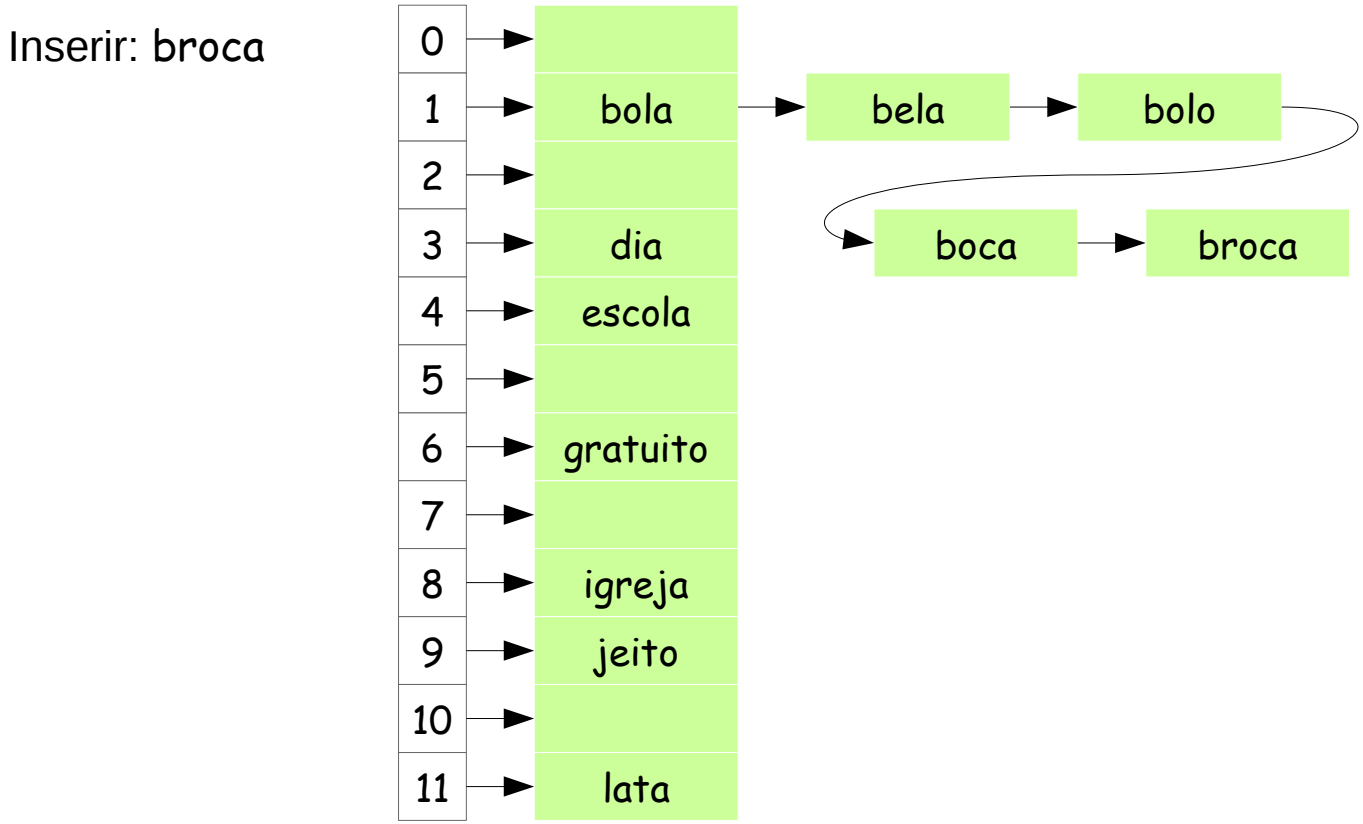
Continuando nosso teste



Inserindo vários com letra b

- inserimos bolo, boca,

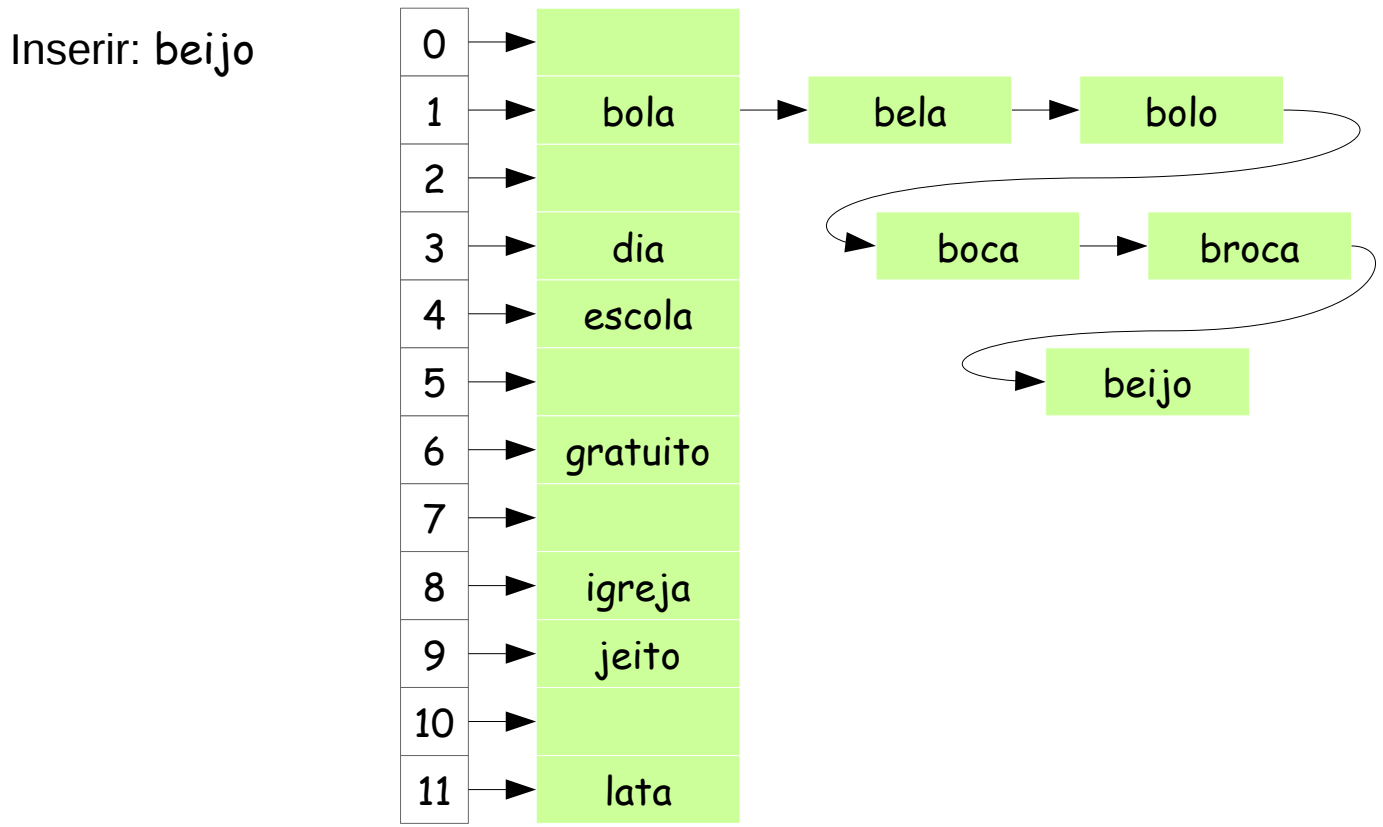
Continuando nosso teste



Inserindo vários com letra b

- inserimos bolo, boca, broca,

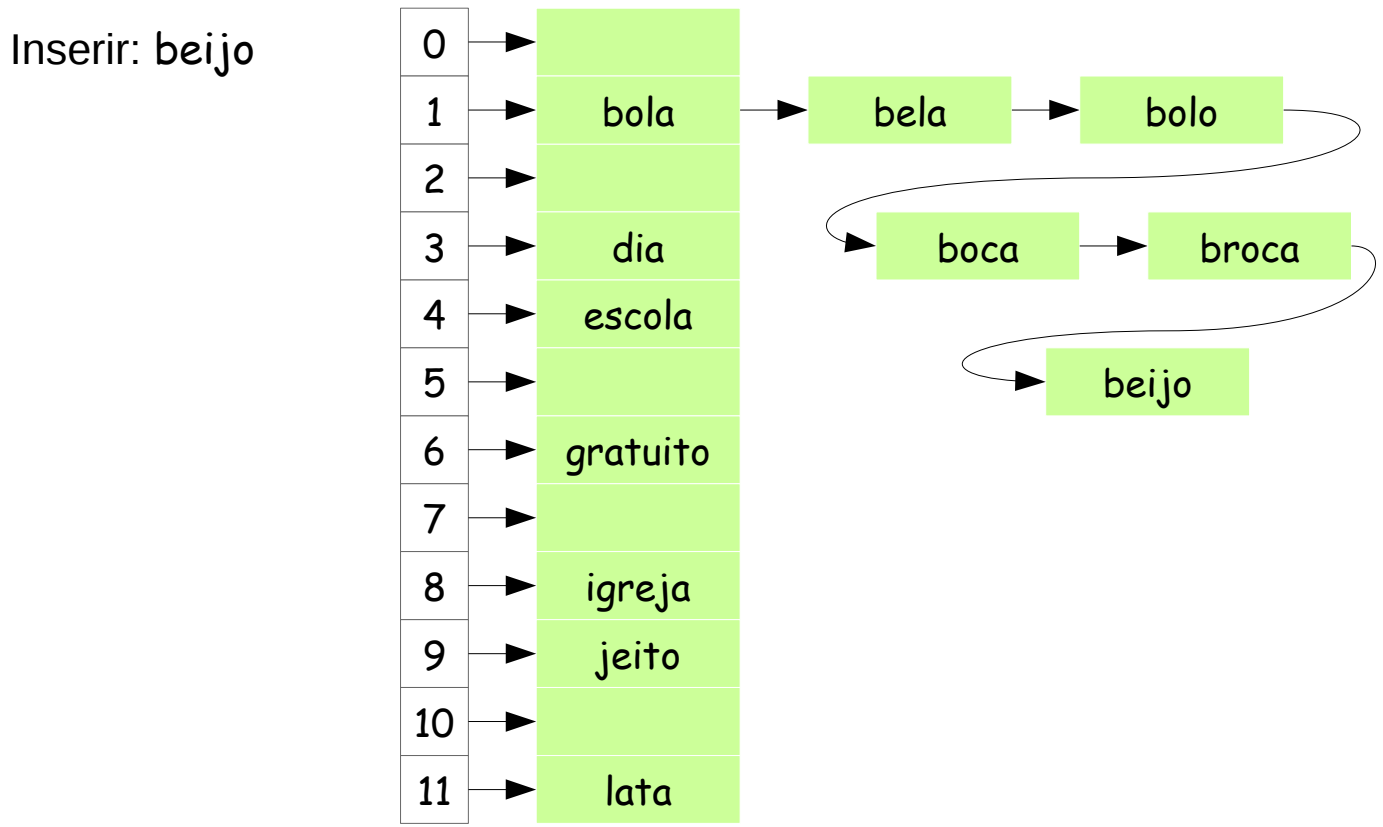
Continuando nosso teste



Inserindo vários com letra b

- inserimos bolo, boca, broca, beijo

Continuando nosso teste



Inserindo vários com letra b

- inserimos bolo, boca, broca, beijo
- a tabela está degenerada em lista

Espalhamento

Corrigimos

Espalhamento

Corrigimos

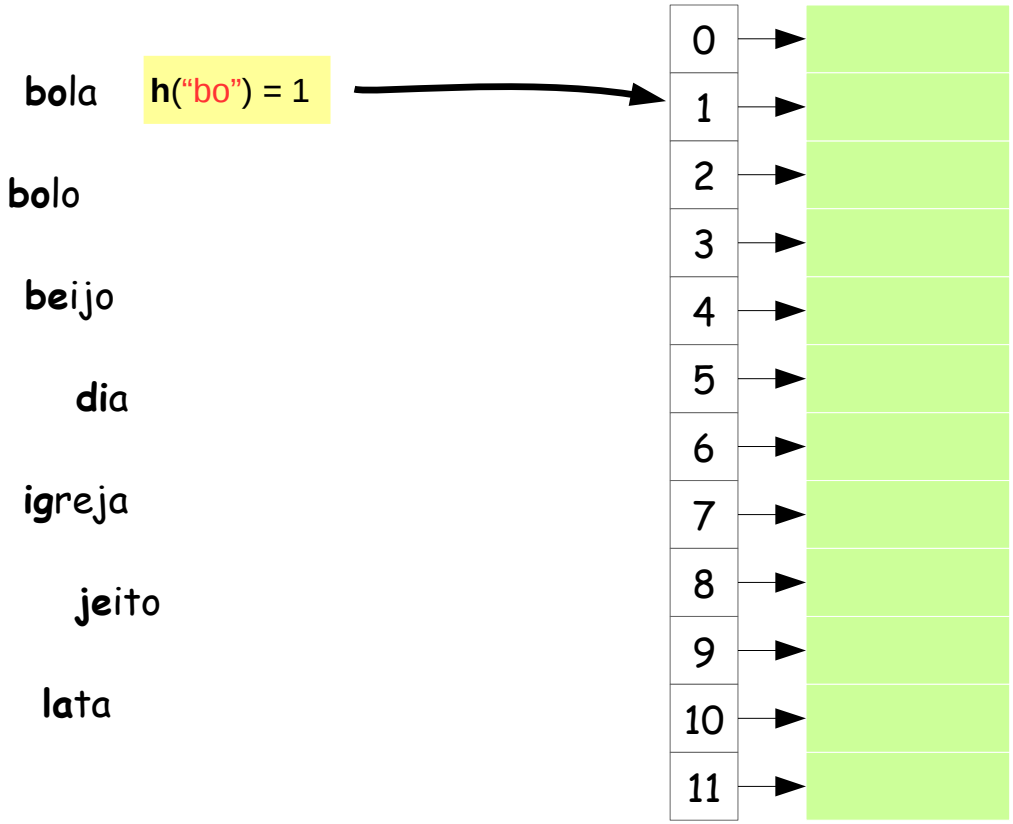
- vamos tentar **espalhar** melhor:

Espalhamento

Corrigimos

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

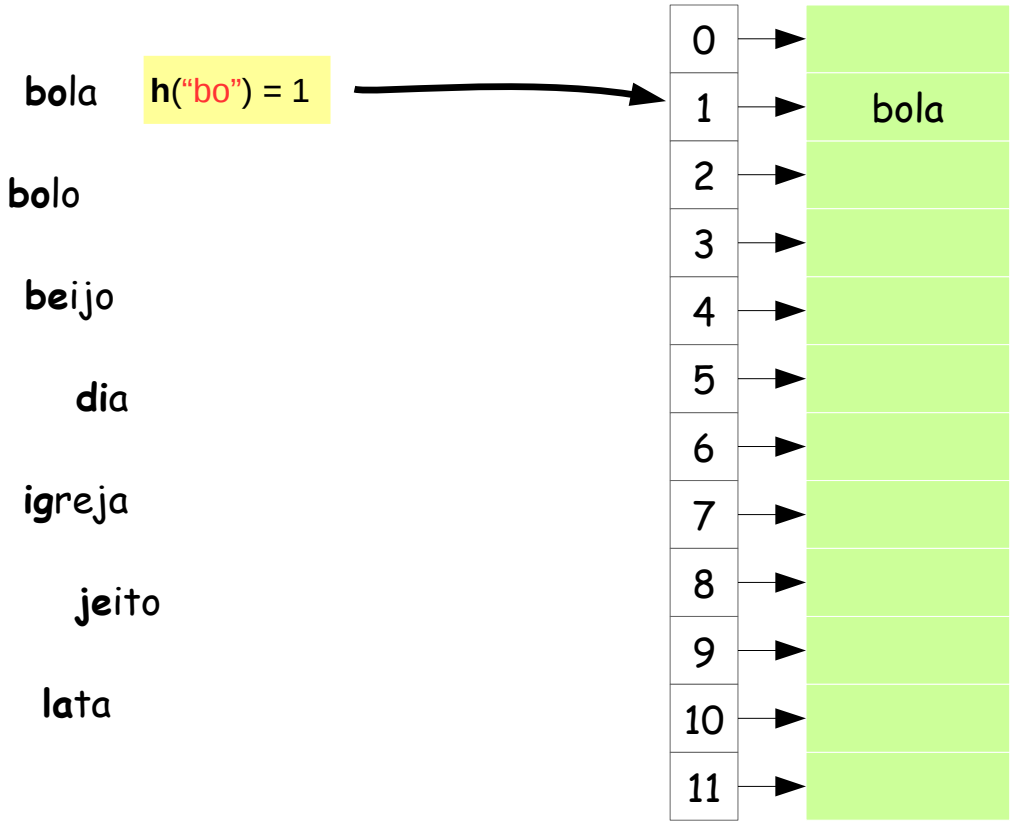
Espalhamento



Corrigimos

- vamos tentar **espalhar** melhor: usamos um picado (*hash*) da chave
- vamos associar a chave a um número inteiro (entre 0 e 11)

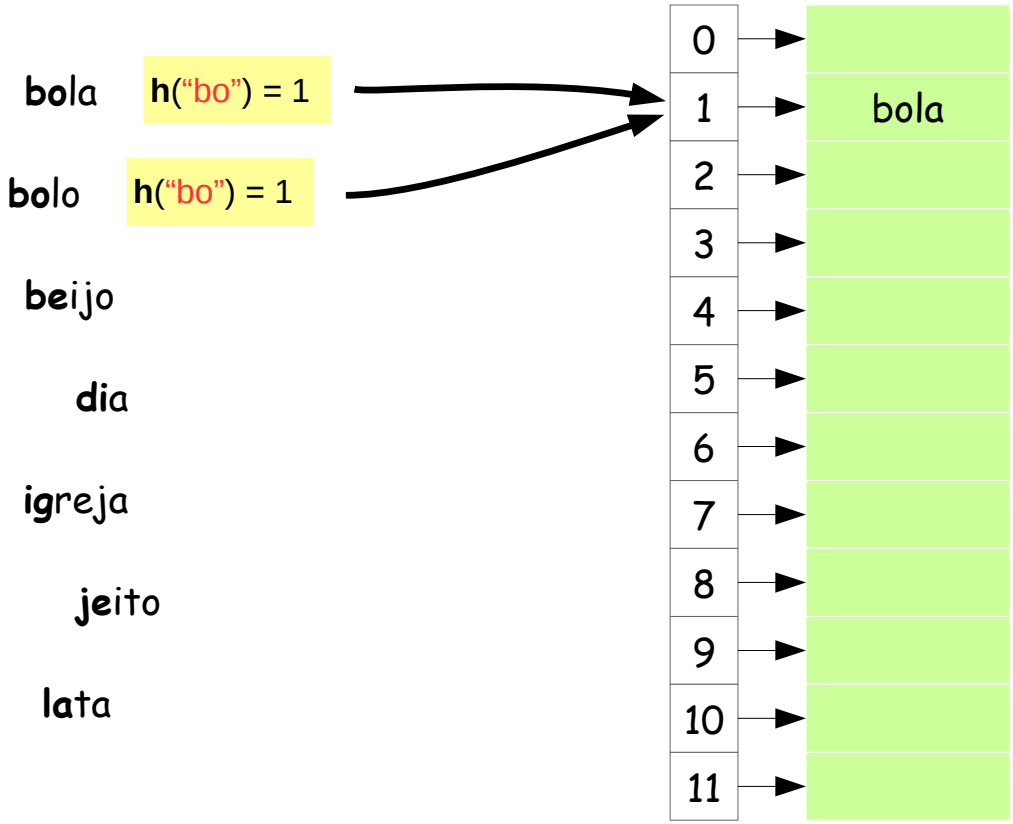
Espalhamento



Corrigimos

- vamos tentar **espalhar** melhor: usamos um picado (*hash*) da chave
- vamos associar a chave a um número inteiro (entre 0 e 11)

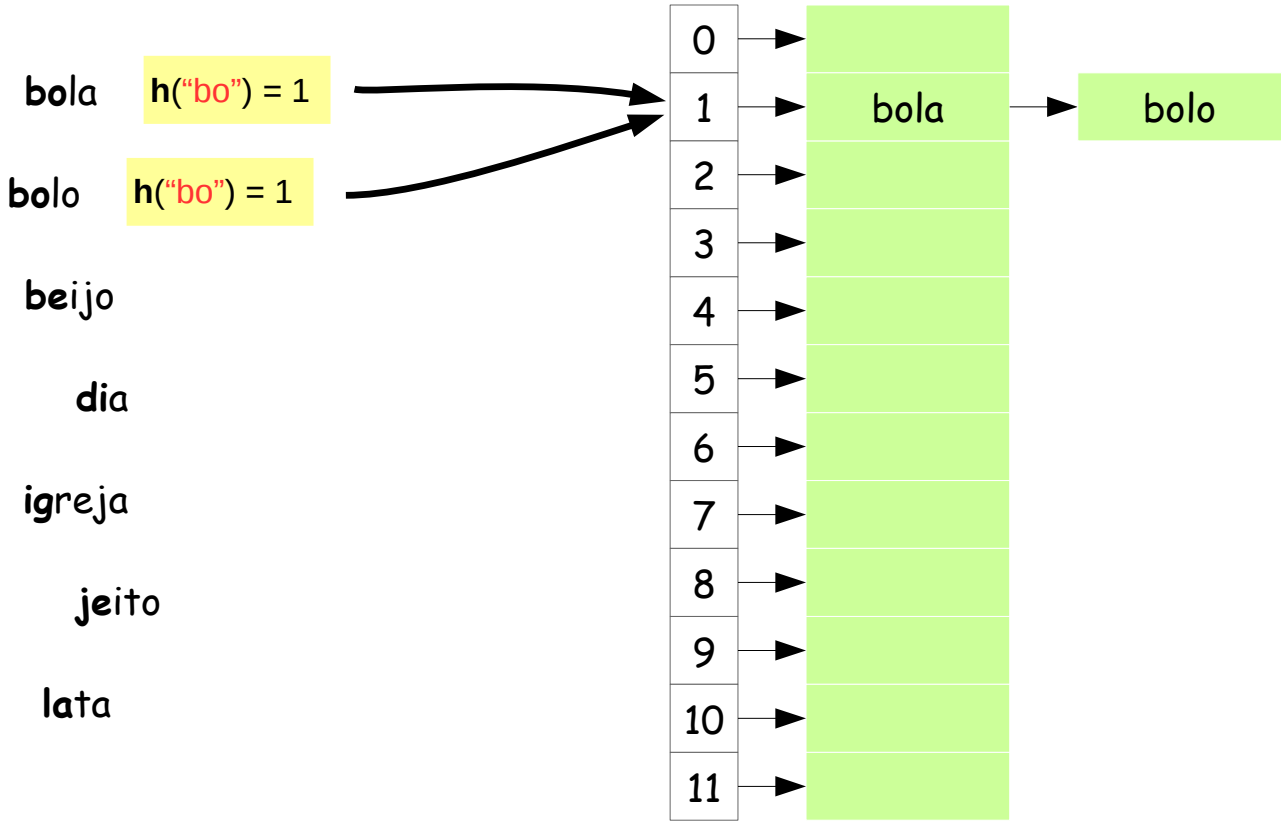
Espalhamento



Corrigimos

- vamos tentar **espalhar** melhor: usamos um picado (*hash*) da chave
- vamos associar a chave a um número inteiro (entre 0 e 11)

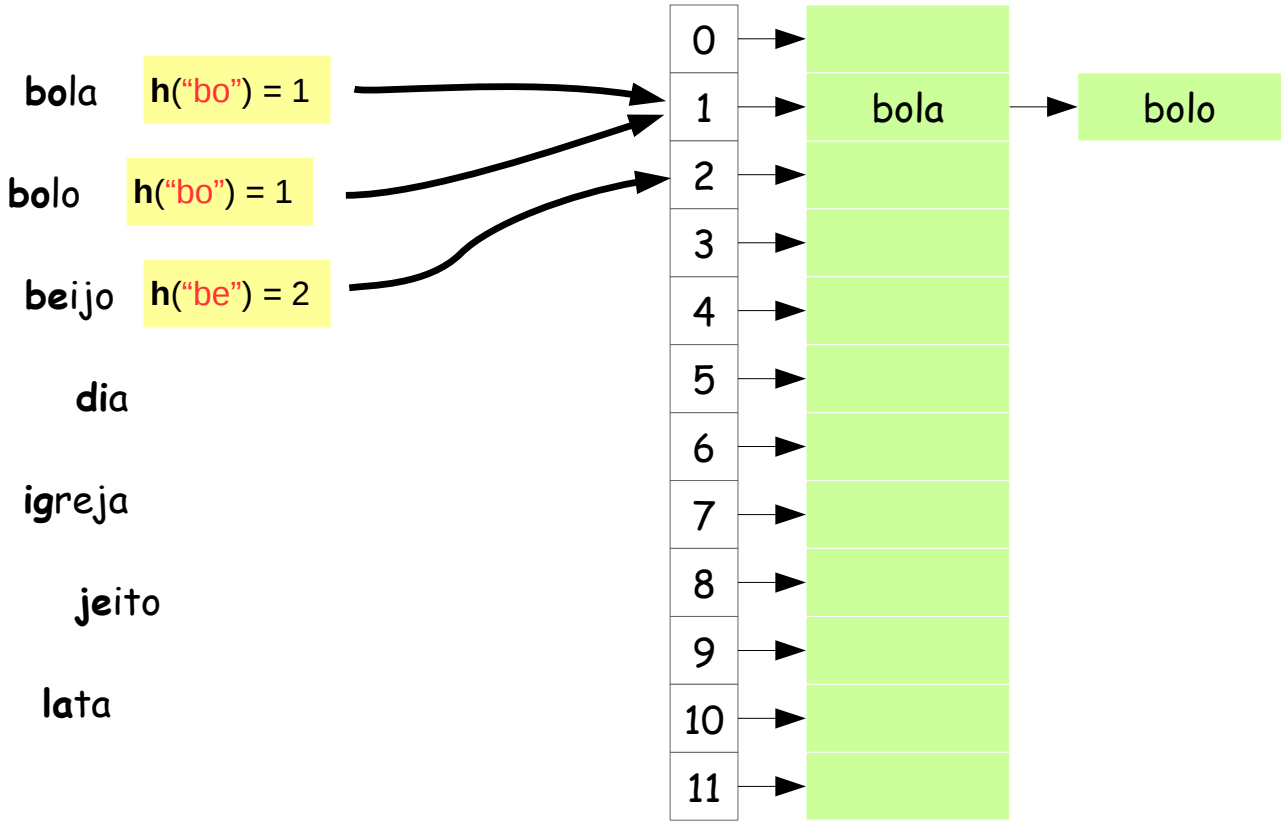
Espalhamento



Corrigimos

- vamos tentar **espalhar** melhor: usamos um picado (*hash*) da chave
- vamos associar a chave a um número inteiro (entre 0 e 11)

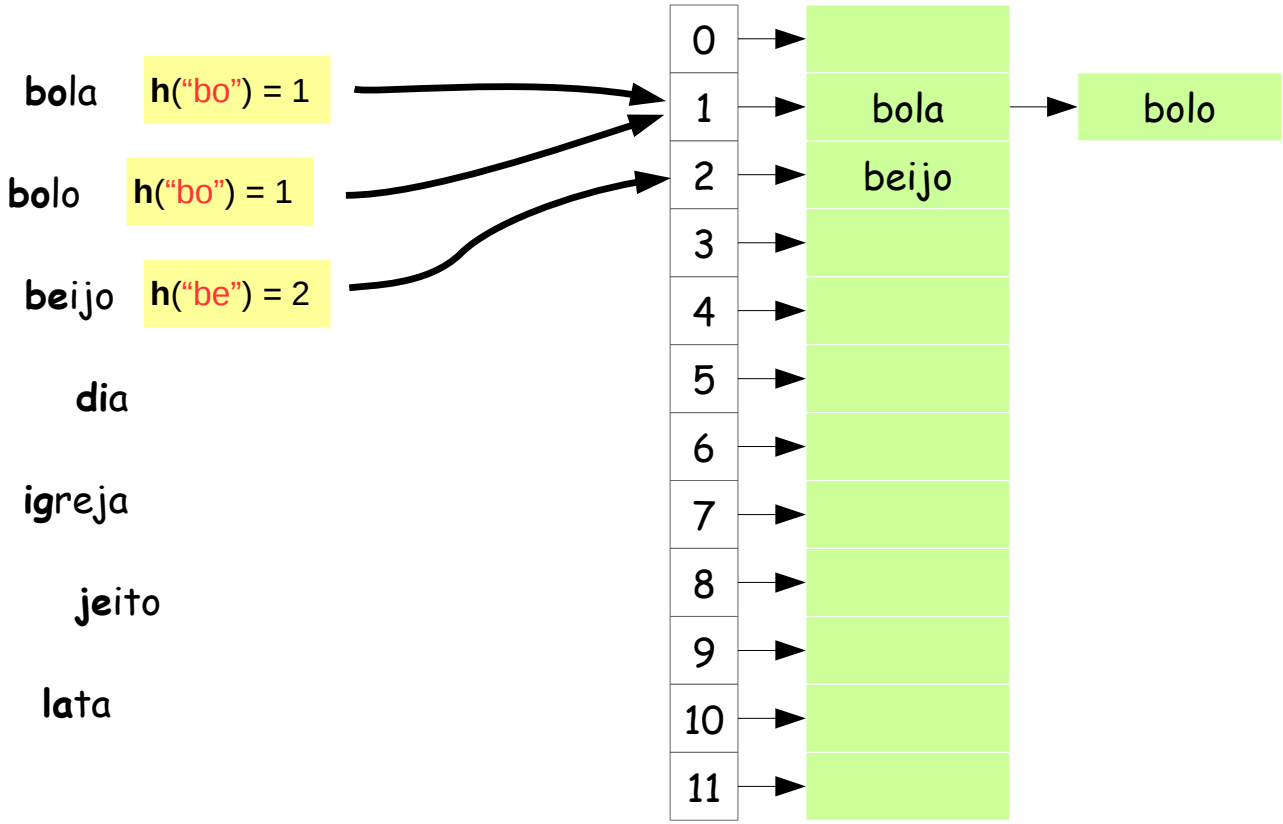
Espalhamento



Corrigimos

- vamos tentar **espalhar** melhor: usamos um picado (*hash*) da chave
- vamos associar a chave a um número inteiro (entre 0 e 11)

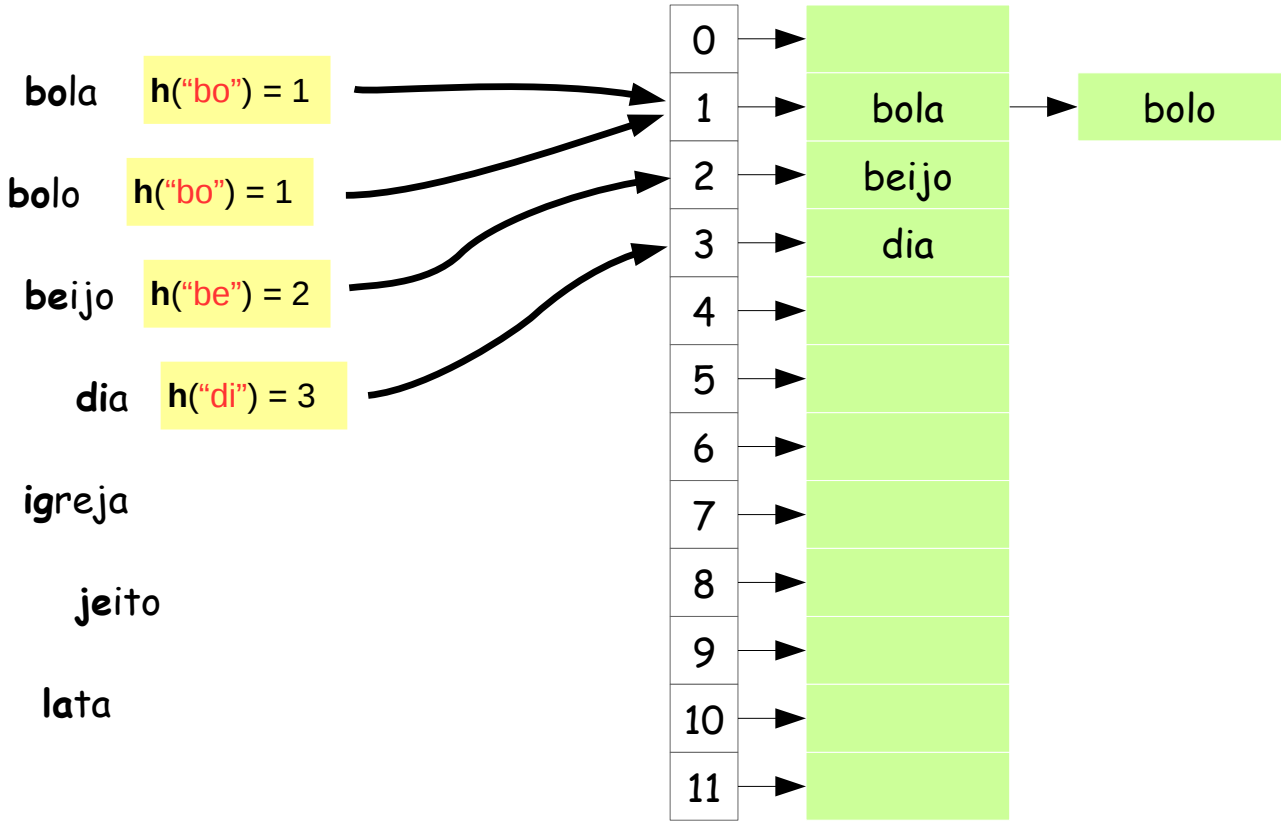
Espalhamento



Corrigimos

- vamos tentar **espalhar** melhor: usamos um picado (*hash*) da chave
- vamos associar a chave a um número inteiro (entre 0 e 11)

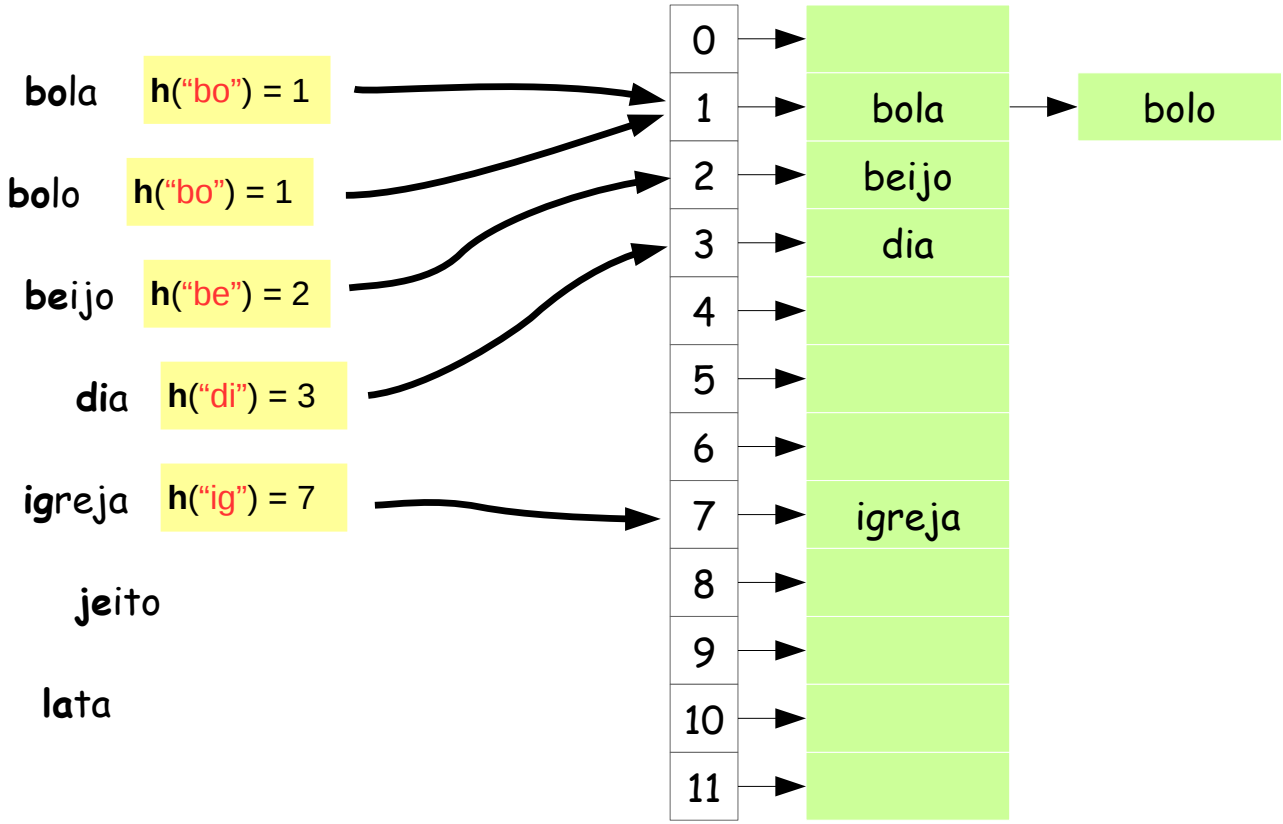
Espalhamento



Corrigimos

- vamos tentar **espalhar** melhor: usamos um picado (*hash*) da chave
- vamos associar a chave a um número inteiro (entre 0 e 11)

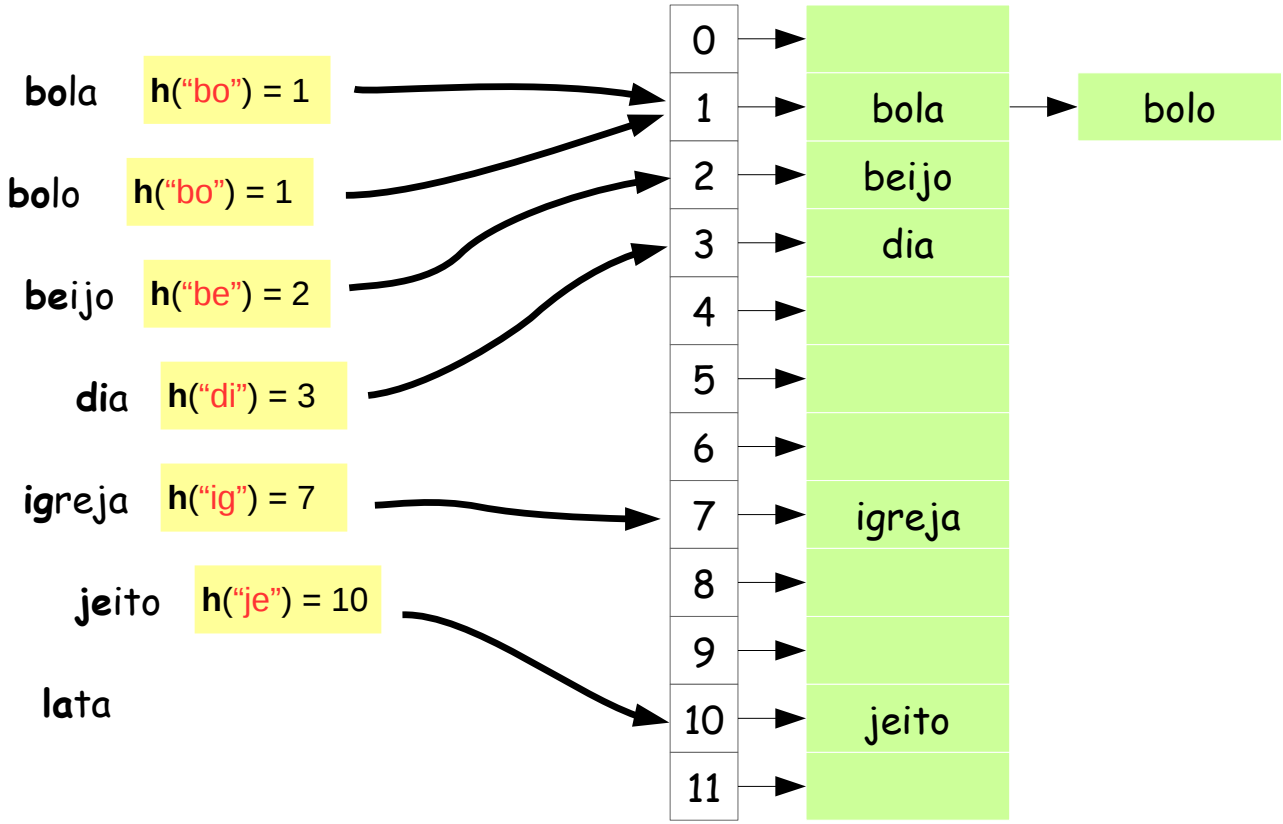
Espalhamento



Corrigimos

- vamos tentar **espalhar** melhor: usamos um picado (*hash*) da chave
- vamos associar a chave a um número inteiro (entre 0 e 11)

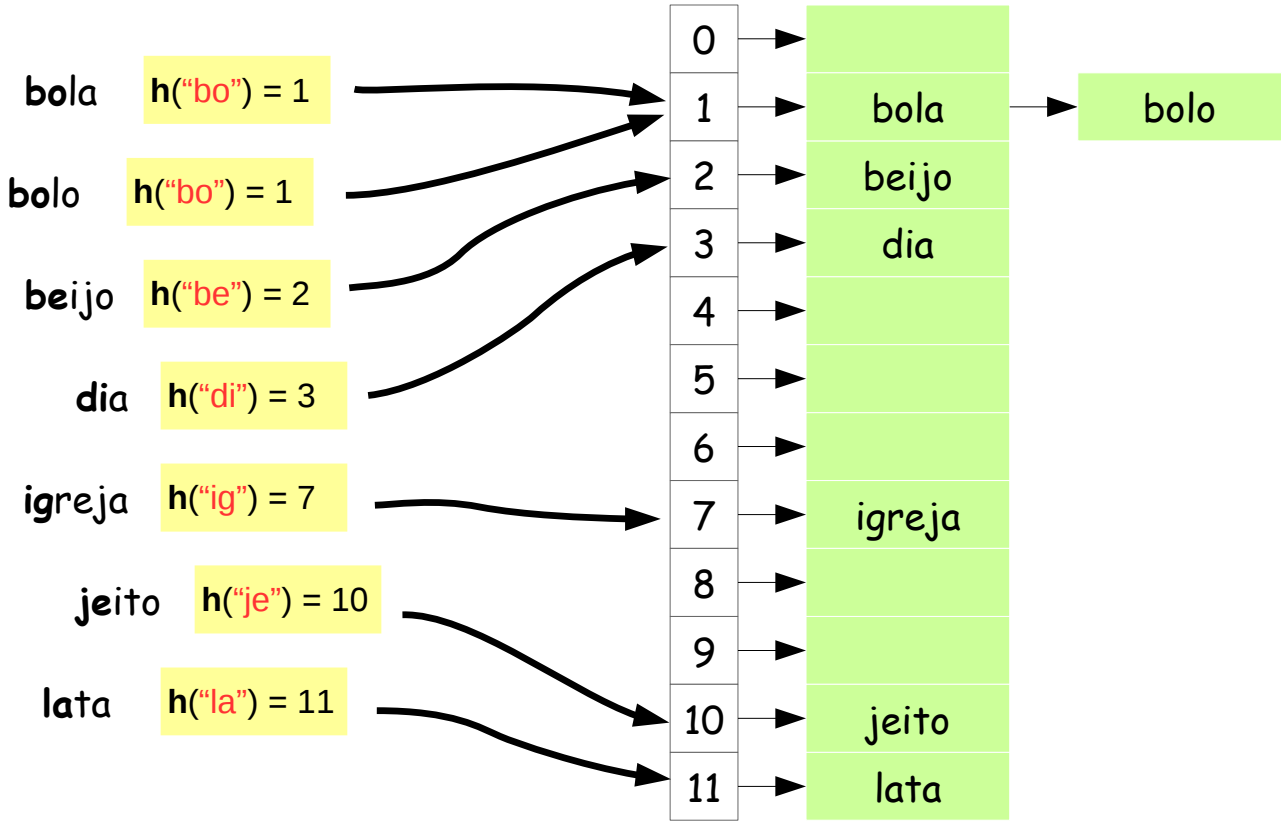
Espalhamento



Corrigimos

- vamos tentar **espalhar** melhor: usamos um picado (*hash*) da chave
- vamos associar a chave a um número inteiro (entre 0 e 11)

Espalhamento



Corrigimos

- vamos tentar **espalhar** melhor: usamos um picado (*hash*) da chave
- vamos associar a chave a um número inteiro (entre 0 e 11)

Tabela de Espalhamento

Função de hashing

É uma função **rápida** que associa um elemento de um certo conjunto (strings, números, arquivos, etc.) a um número inteiro de tamanho conhecido.

Tabela de Espalhamento

Função de hashing

É uma função **rápida** que associa um elemento de um certo conjunto (strings, números, arquivos, etc.) a um número inteiro de tamanho conhecido.

Tabela de Espalhamento

É um tipo abstrato de dados para busca em conjuntos dinâmicos cuja implementação tem 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

Propriedades

- estimativa do tamanho do conjunto de dados deve ser conhecida
- tempo das operações depende da função de hashing escolhida:
  - chaves bem espalhadas: tempo "quase" $O(1)$
  - chaves agrupadas: pior caso de tempo $O(n)$

Características das tabelas de Espalhamento

Propriedades

- estimativa do tamanho do conjunto de dados deve ser conhecida
- tempo das operações depende da função de hashing escolhida:
  - chaves bem espalhadas: tempo "quase" $O(1)$
  - chaves agrupadas: pior caso de tempo $O(n)$

Obtendo funções de hashing

Uma boa função de hashing depende do conjunto de chaves específico.

Características das tabelas de Espalhamento

Propriedades

- estimativa do tamanho do conjunto de dados deve ser conhecida
- tempo das operações depende da função de hashing escolhida:
  - chaves bem espalhadas: tempo "quase" $O(1)$
  - chaves agrupadas: pior caso de tempo $O(n)$

Obtendo funções de hashing

Uma boa função de hashing depende do conjunto de chaves específico.
Métodos genéricos:

1. Método da divisão
2. Método da multiplicação
3. Método do dobramento

Características das tabelas de Espalhamento

Propriedades

- estimativa do tamanho do conjunto de dados deve ser conhecida
- tempo das operações depende da função de hashing escolhida:
  - chaves bem espalhadas: tempo "quase" $O(1)$
  - chaves agrupadas: pior caso de tempo $O(n)$

Obtendo funções de hashing

Uma boa função de hashing depende do conjunto de chaves específico.
Métodos genéricos:

1. Método da divisão
2. Método da multiplicação
3. Método do dobramento

Hashing perfeito: Se conhecermos todos as chaves a priori, é possível encontrar uma função de hashing injetora, mas tais funções podem ser difíceis de encontrar.

Interpretando chaves

Normalmente pressupomos que as chaves são números inteiros.

Interpretando chaves

Normalmente pressupomos que as chaves são números inteiros.

E se não for?

Interpretando chaves

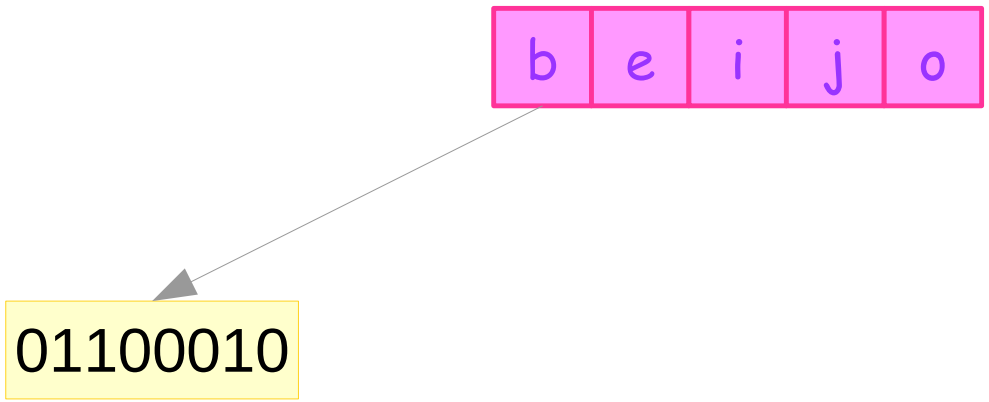
Normalmente pressupomos que as chaves são números inteiros.

E se não for? Reinterpretamos a chave como sequência de bits:

Interpretando chaves

Normalmente pressupomos que as chaves são números inteiros.

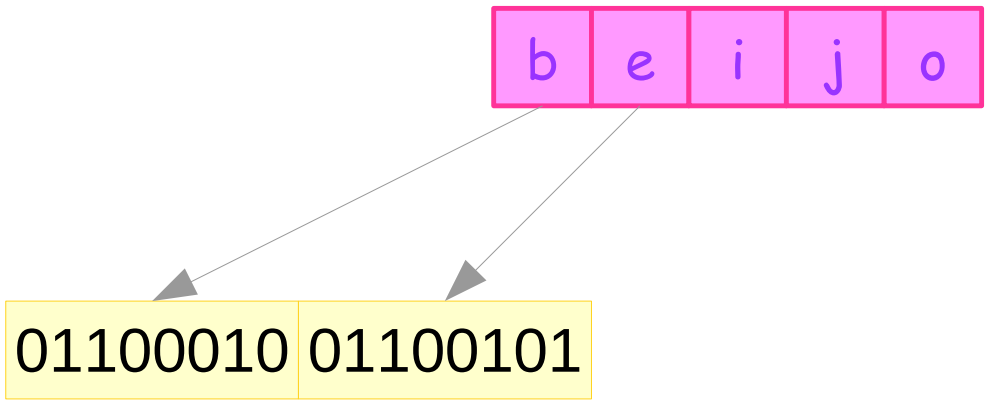
E se não for? Reinterpretamos a chave como sequência de bits:



Interpretando chaves

Normalmente pressupomos que as chaves são números inteiros.

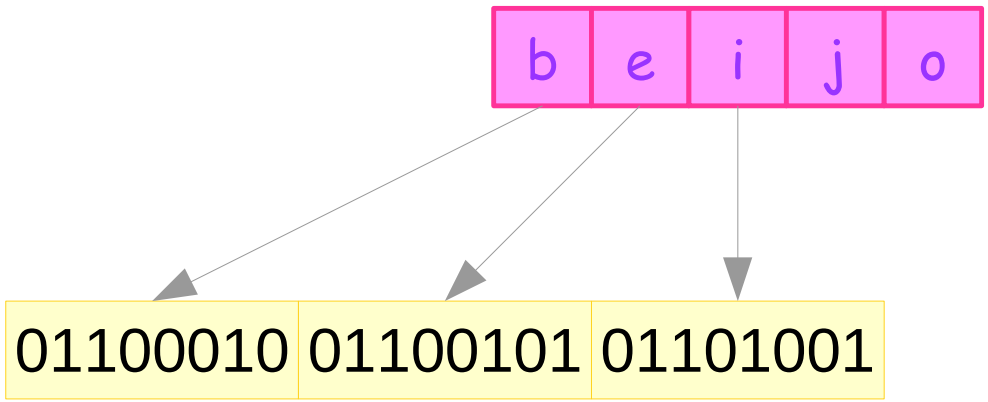
E se não for? Reinterpretamos a chave como sequência de bits:



Interpretando chaves

Normalmente pressupomos que as chaves são números inteiros.

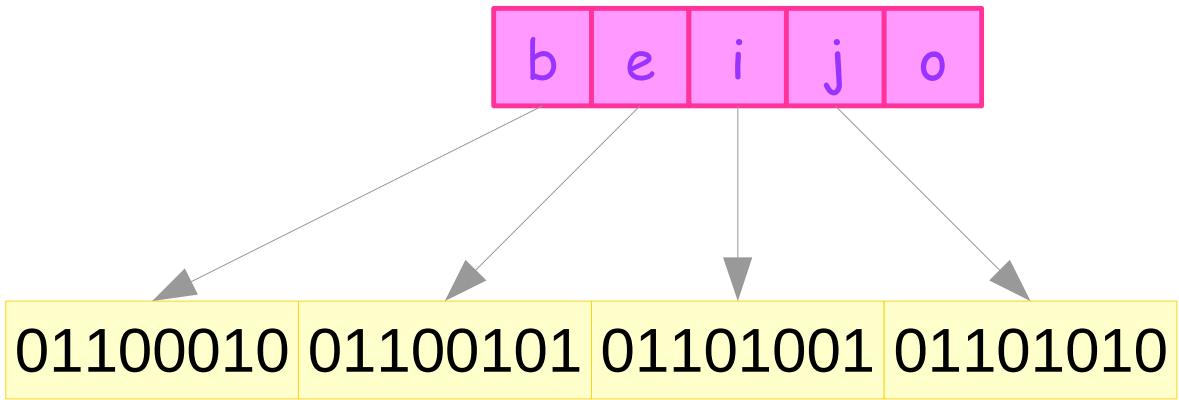
E se não for? Reinterpretamos a chave como sequência de bits:



Interpretando chaves

Normalmente pressupomos que as chaves são números inteiros.

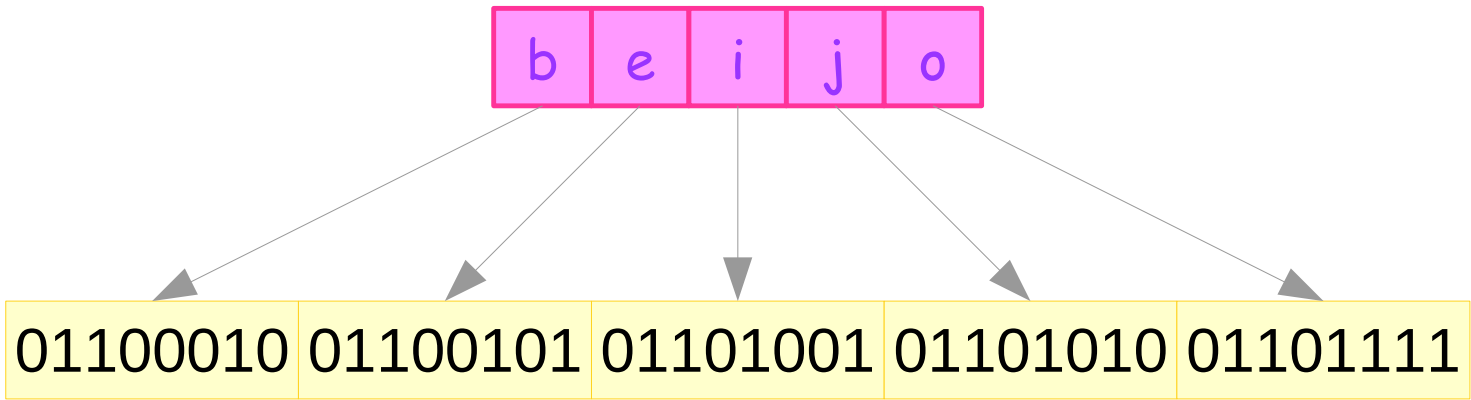
E se não for? Reinterpretamos a chave como sequência de bits:



Interpretando chaves

Normalmente pressupomos que as chaves são números inteiros.

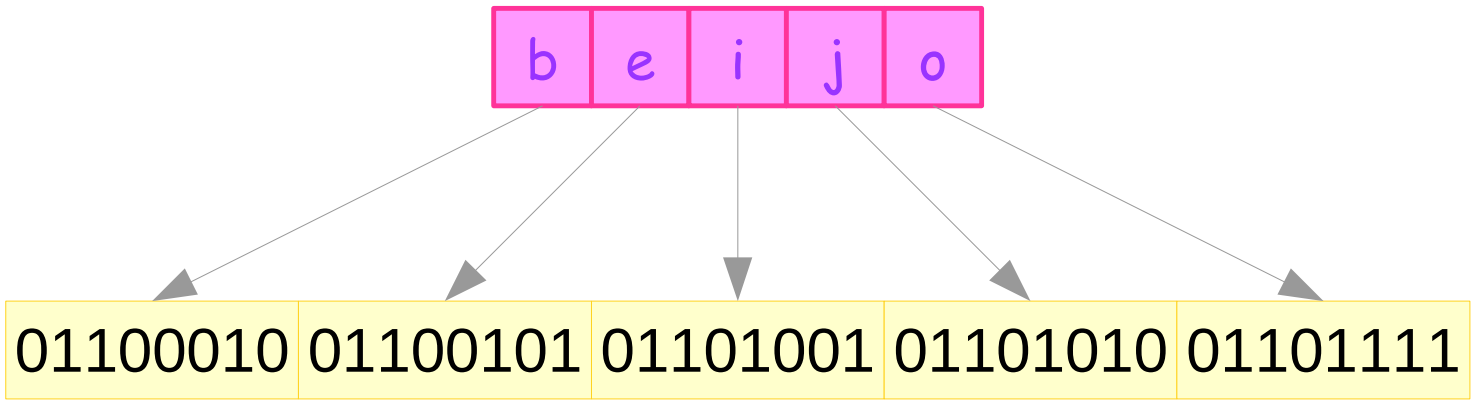
E se não for? Reinterpretamos a chave como sequência de bits:



Interpretando chaves

Normalmente pressupomos que as chaves são números inteiros.

E se não for? Reinterpretamos a chave como sequência de bits:



= 422.608.202.351

Método da divisão

Método da divisão

- obtemos o resto da divisão pelo máximo p do hashing
- normalmente p é um número primo (e.g. $p = 1783$)

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

Método da divisão

Método da divisão

- obtemos o resto da divisão pelo máximo p do hashing
- normalmente p é um número primo (e.g. $p = 1783$)

$$h(x) = x \,\mathbf{mod}\, p$$

Exemplo

$$h(\text{“beijo”}) = 422.608.202.351 \,\mathbf{mod}\, 1783 = 56$$

Método da divisão

Método da divisão

- obtemos o resto da divisão pelo máximo p do hashing
- normalmente p é um número primo (e.g. $p = 1783$)

$$h(x) = x \,\mathbf{mod}\, p$$

Exemplo

$$h(\text{“beijo”}) = 422.608.202.351 \,\mathbf{mod}\, 1783 = 56$$

O número primo evita agrupamento quando várias chaves tem fatores em comum.

Método da multiplicação

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 p (pode ser $p = 1024$)

$$h(x) = \lfloor p\,(A \cdot x \,\mathbf{mod}\, 1)\rfloor$$

Método da multiplicação

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 p (pode ser $p = 1024$)

$$h(x) = \lfloor p\,(A \cdot x \,\mathbf{mod}\, 1)\rfloor$$

Exemplo

$$\begin{aligned} h(\text{“beijo”}) &= \lfloor 1024\,[((\sqrt{5} - 1)/2 \cdot 422.608.202.351)\,\mathbf{mod}\,1]\rfloor \\ &= \lfloor 1024\,[261186232977{,}41125\,\mathbf{mod}\,1]\rfloor \\ &= \lfloor 1024\,[0{,}41125]\rfloor \\ &= \lfloor 421.125\rfloor = 421 \end{aligned}$$

Método da multiplicação

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 p (pode ser $p = 1024$)

$$h(x) = \lfloor p\,(A \cdot x \,\mathbf{mod}\, 1)\rfloor$$

Exemplo

$$\begin{aligned} h(\text{“beijo”}) &= \lfloor 1024\,[((\sqrt{5} - 1)/2 \cdot 422.608.202.351)\,\mathbf{mod}\, 1]\rfloor \\ &= \lfloor 1024\,[261186232977{,}41125\,\mathbf{mod}\, 1]\rfloor \\ &= \lfloor 1024\,[0{,}41125]\rfloor \\ &= \lfloor 421.125\rfloor = 421 \end{aligned}$$

O uso do número áureo com A é sugestão de Donald Knuth.

Endereçamento aberto

Existe uma alternativa pra implementação de tabela de espalhamento,

Endereçamento aberto

- os dados são guardados no próprio vetor (não são ponteiros)
- colisões são colocadas em posições livres da tabela

Endereçamento aberto

Existe uma alternativa pra implementação de tabela de espalhamento,

Endereçamento aberto

- os dados são guardados no próprio vetor (não são ponteiros)
- colisões são colocadas em posições livres da tabela

Características

- evita percorrimento com ponteiros e alocação e deslocação de memória (`malloc` e `free`)
- se a tabela lotar, deve recriar uma tabela maior
- remoção fica mais complicada

Inserindo com endereçamento aberto

0	querer
1	viver
2	
3	
4	momento
5	
6	louvor
7	
8	
9	
10	meu
11	canto

Inserindo espalhar

Inserindo com endereçamento aberto

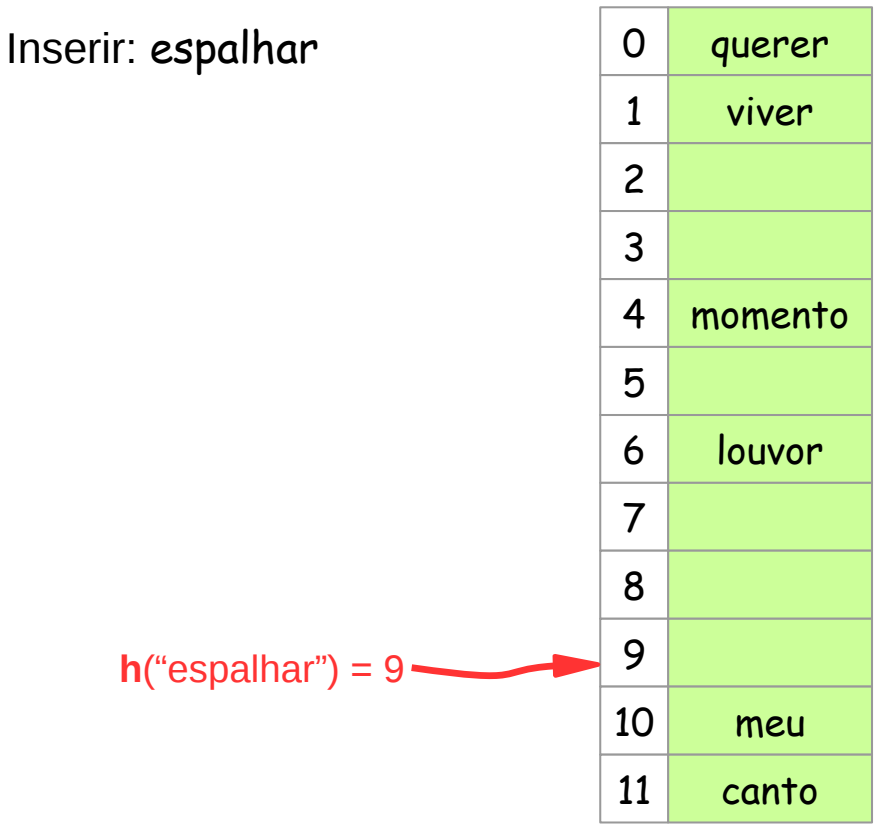
Inserir: **espalhar**

0	querer
1	viver
2	
3	
4	momento
5	
6	louvor
7	
8	
9	
10	meu
11	canto

Inserindo espalhar

- procuramos posição

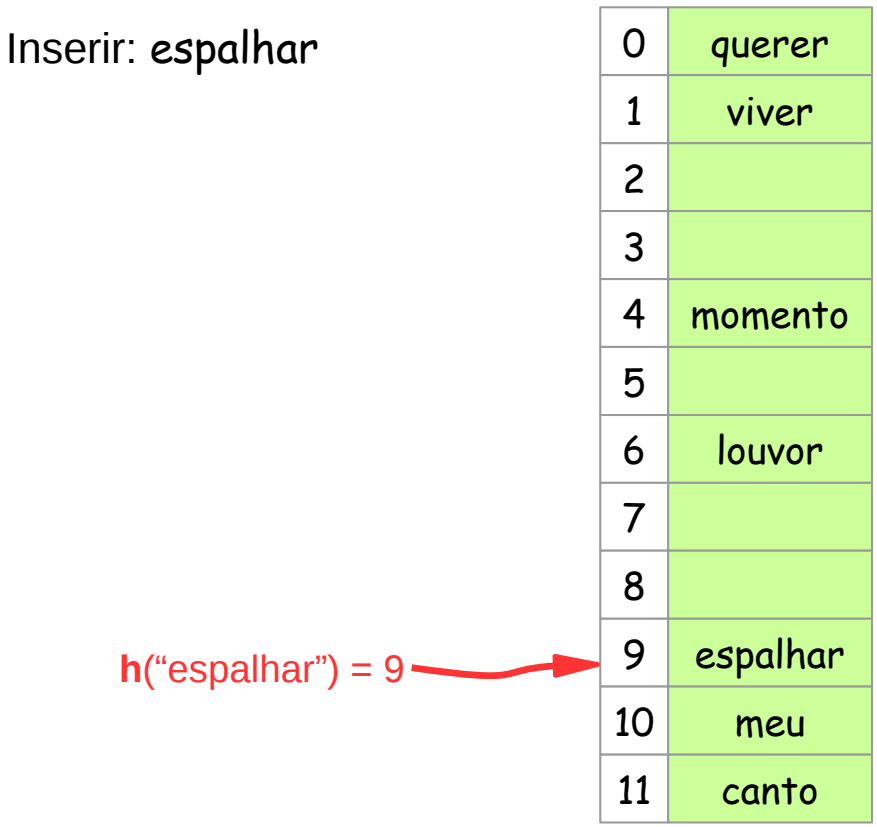
Inserindo com endereçamento aberto



Inserindo espalhar

- procuramos posição
- se houver espaço, guardamos

Inserindo com endereçamento aberto



Inserindo espalhar

- procuramos posição
- se houver espaço, guardamos

Colisão: reespalhamento

0	querer
1	viver
2	
3	
4	momento
5	
6	louvor
7	
8	
9	espalhar
10	meu
11	canto

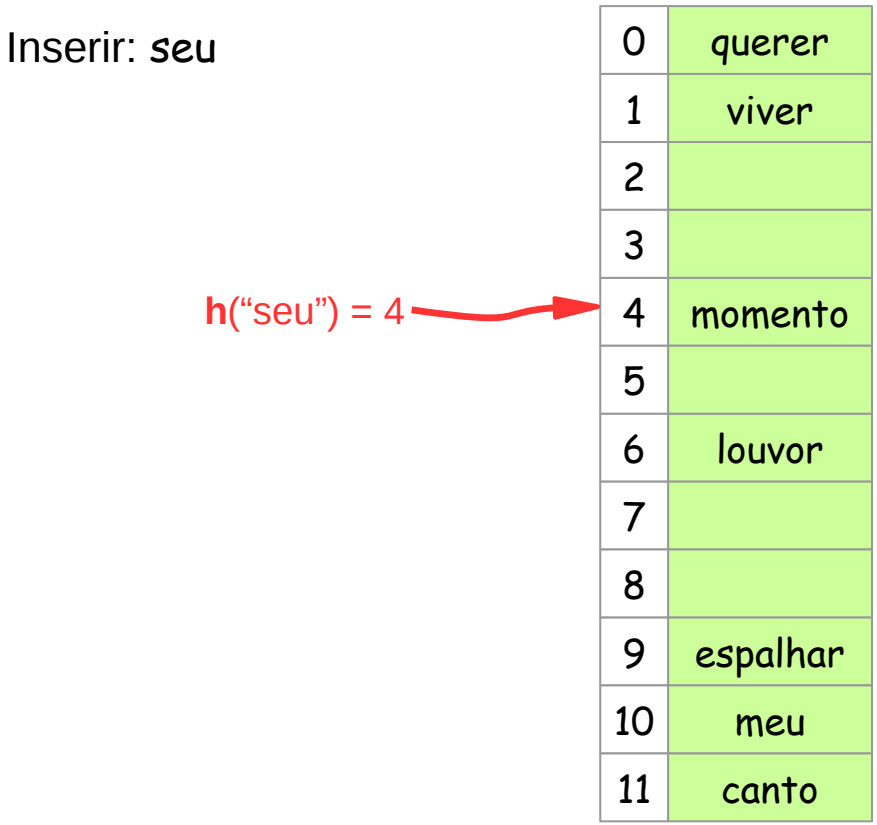
Colisão: reespalhamento

Inserir: **seu**

0	querer
1	viver
2	
3	
4	momento
5	
6	louvor
7	
8	
9	espalhar
10	meu
11	canto

Inserindo com colisão

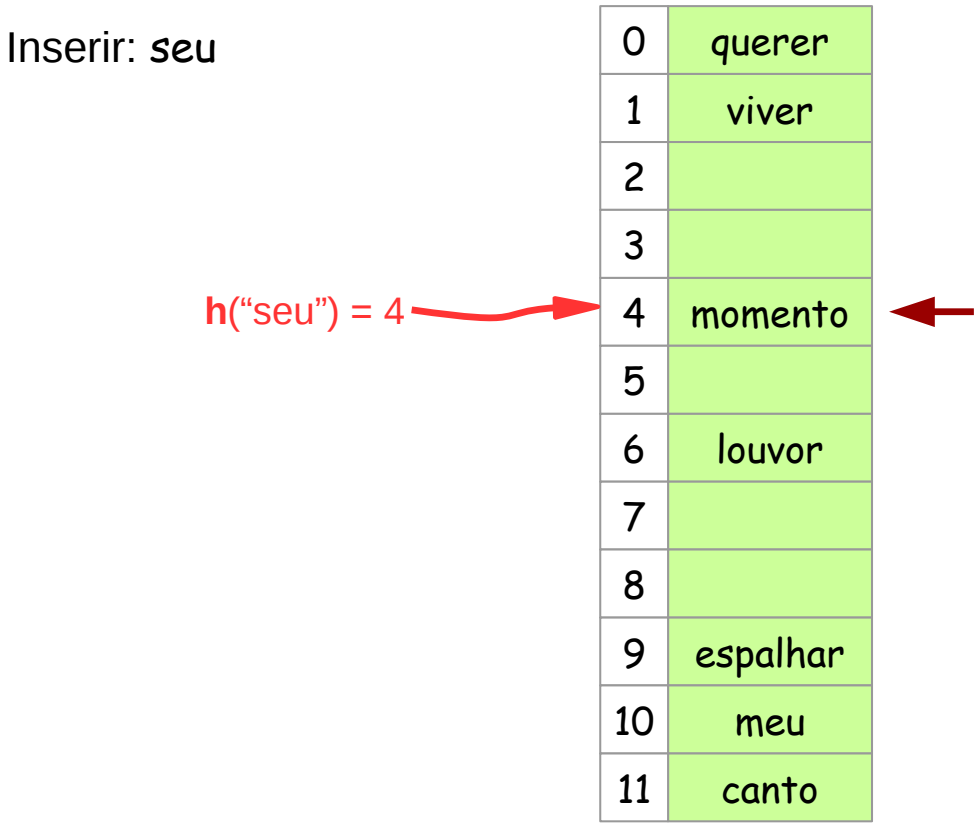
Colisão: reespalhamento



Inserindo com colisão

- procuramos a posição

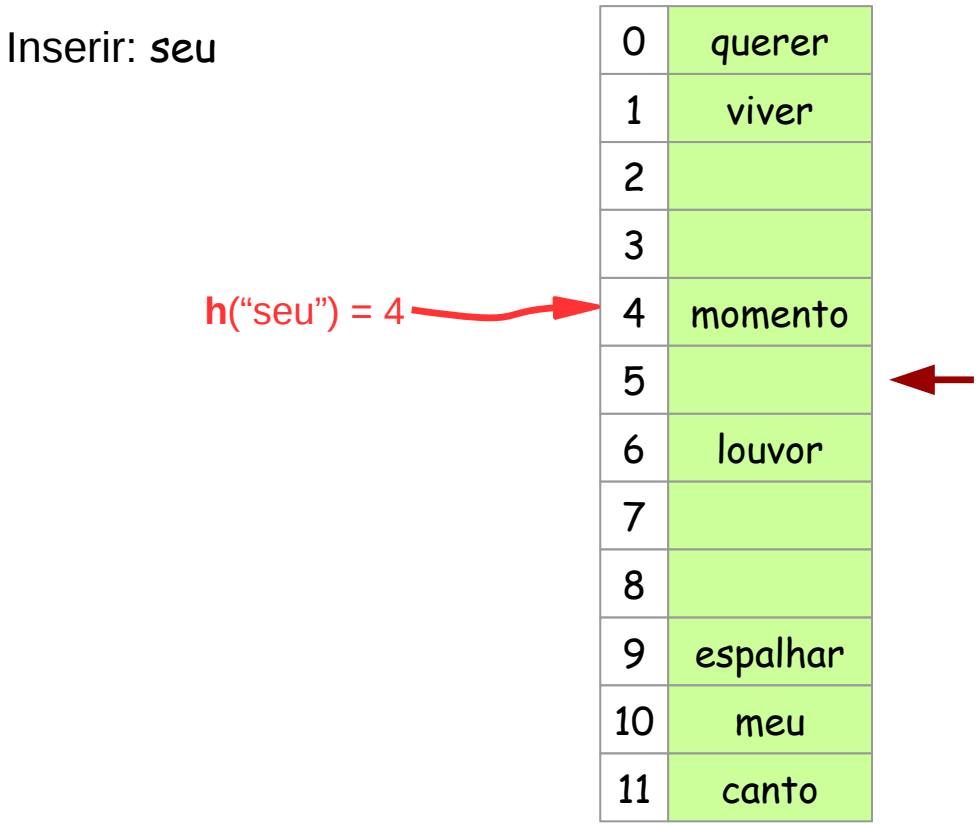
Colisão: reespalhamento



Inserindo com colisão

- procuramos a posição
- se não houver espaço,

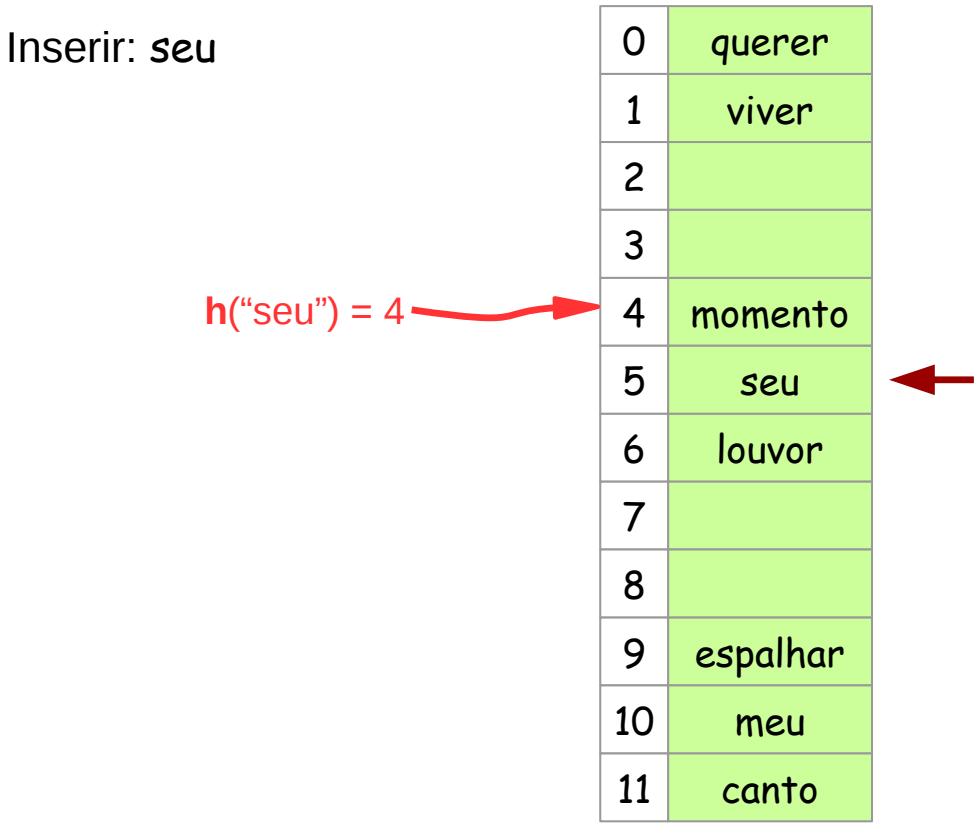
Colisão: reespalhamento



Inserindo com colisão

- procuramos a posição
- se não houver espaço, procuramos posição livre

Colisão: reespalhamento



Inserindo com colisão

- procuramos a posição
- se não houver espaço, procuramos posição livre

Colisão: reespalhamento

0	querer
1	viver
2	
3	
4	momento
5	seu
6	louvor
7	
8	
9	espalhar
10	meu
11	canto

Inserindo com colisão

- procuramos a posição
- se não houver espaço, procuramos posição livre

Colisão: reespalhamento

Inserir: cada

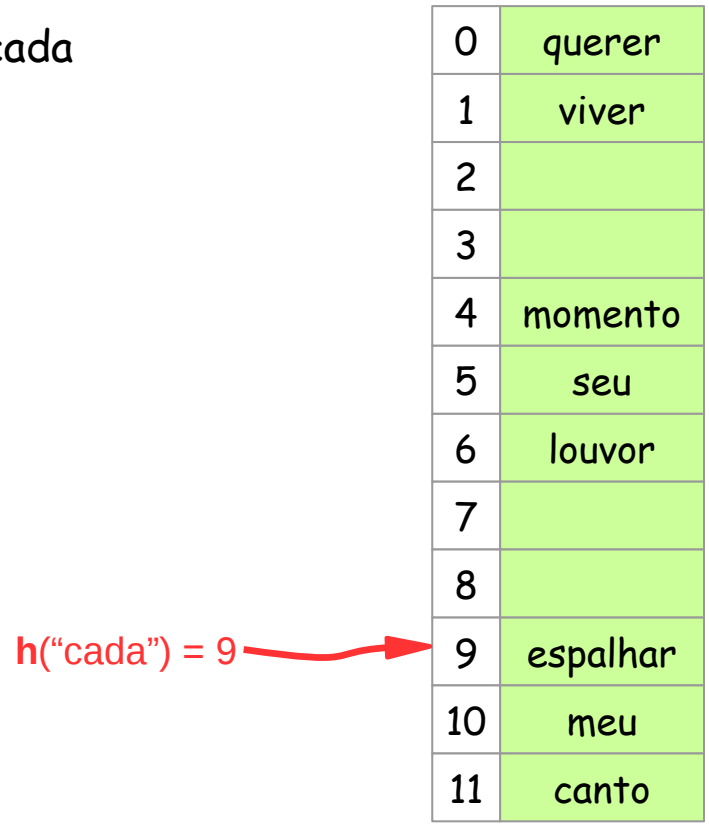
0	querer
1	viver
2	
3	
4	momento
5	seu
6	louvor
7	
8	
9	espalhar
10	meu
11	canto

Inserindo com colisão

- procuramos a posição
- se não houver espaço, procuramos posição livre

Colisão: reespalhamento

Inserir: cada

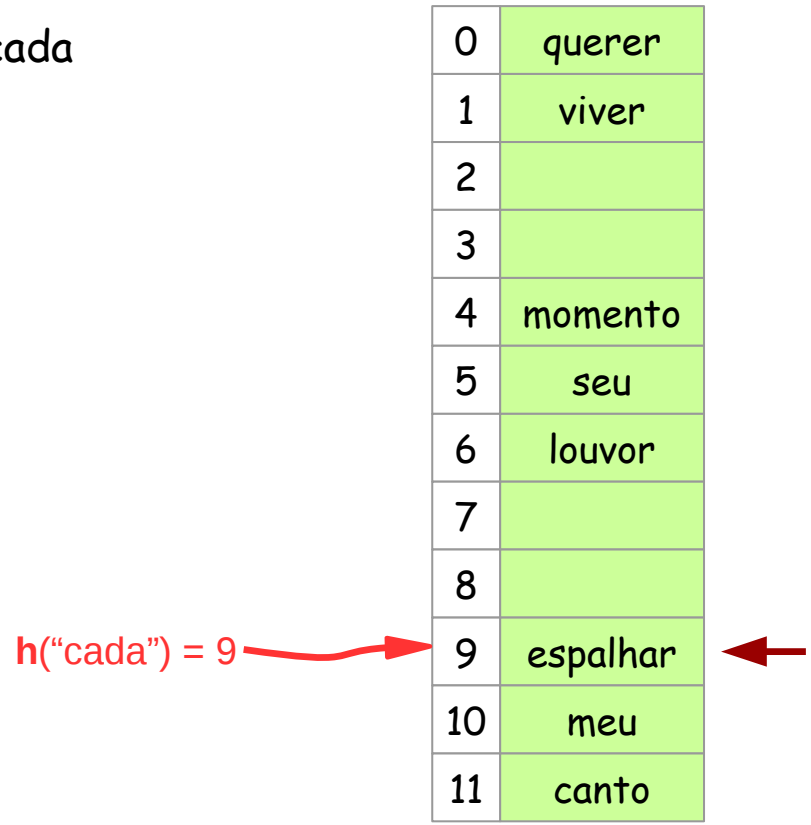


Inserindo com colisão

- procuramos a posição
- se não houver espaço, procuramos posição livre

Colisão: reespalhamento

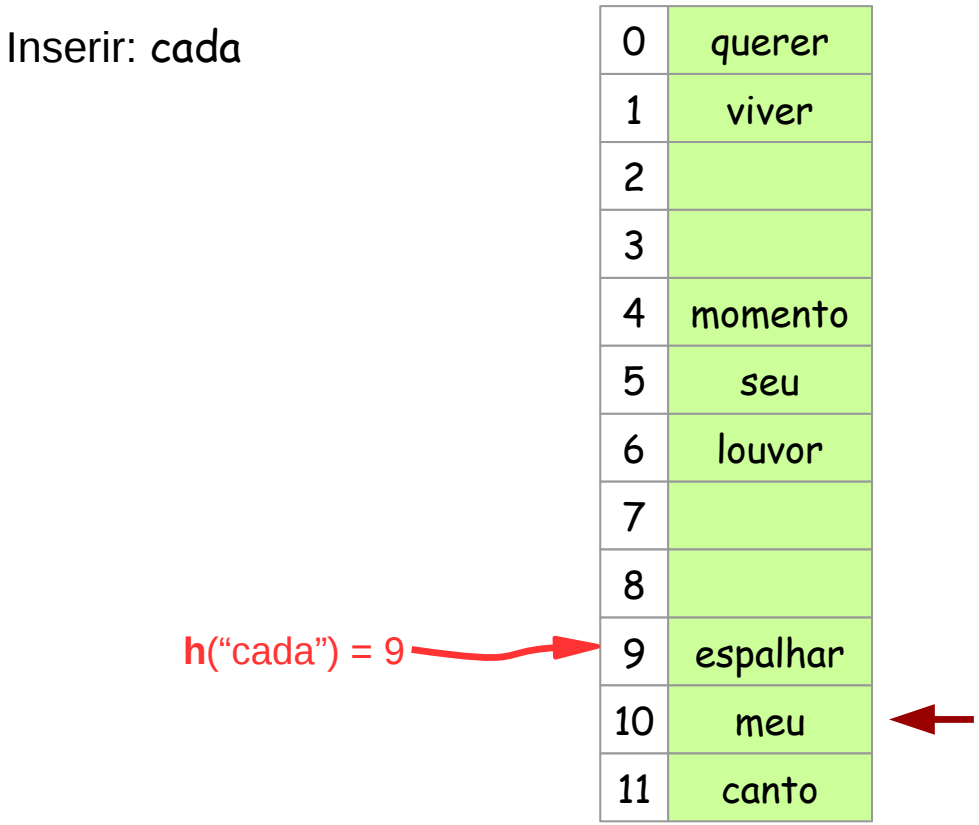
Inserir: cada



Inserindo com colisão

- procuramos a posição
- se não houver espaço, procuramos posição livre

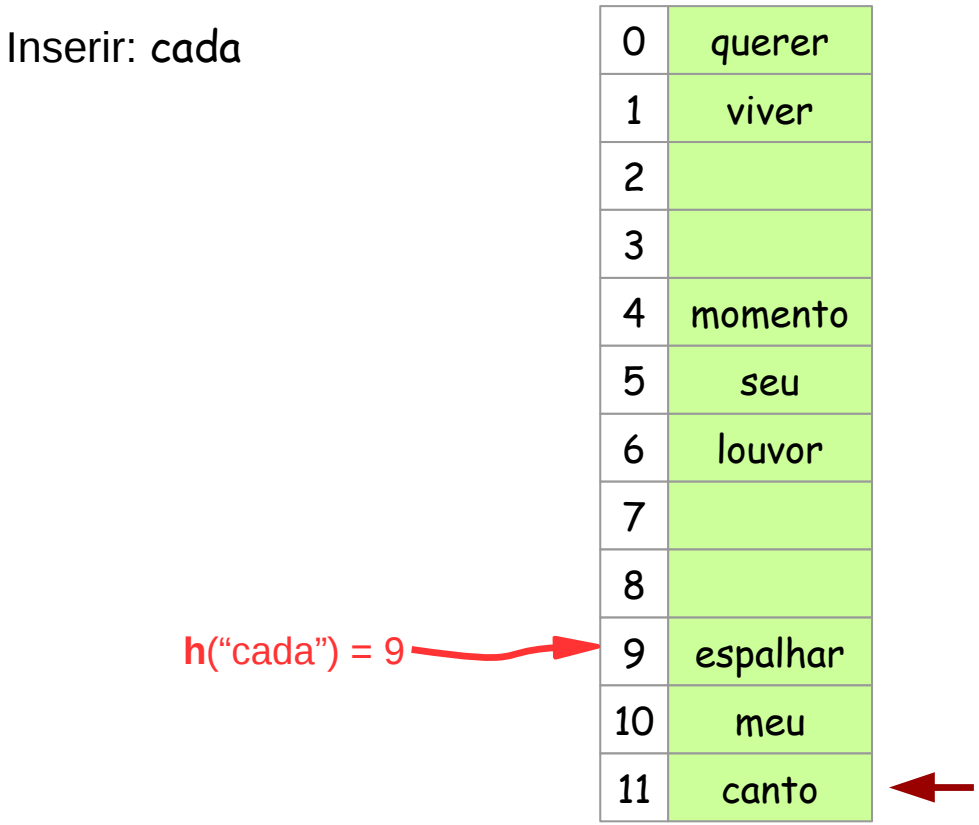
Colisão: reespalhamento



Inserindo com colisão

- procuramos a posição
- se não houver espaço, procuramos posição livre

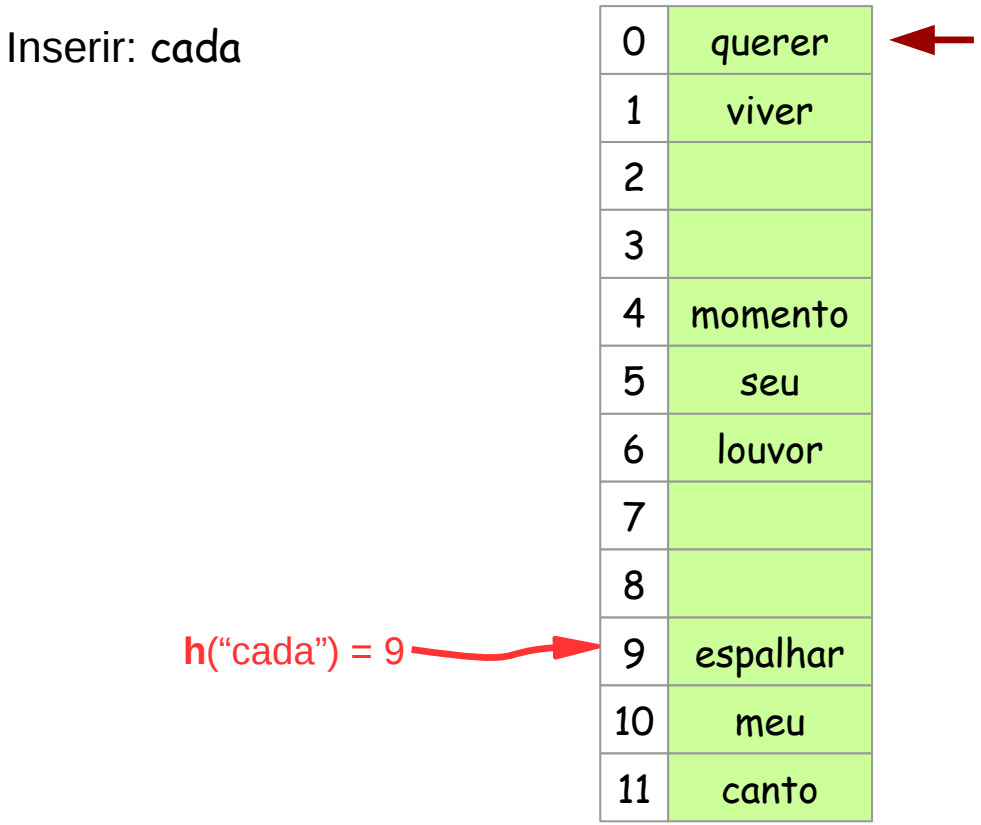
Colisão: reespalhamento



Inserindo com colisão

- procuramos a posição
- se não houver espaço, procuramos posição livre

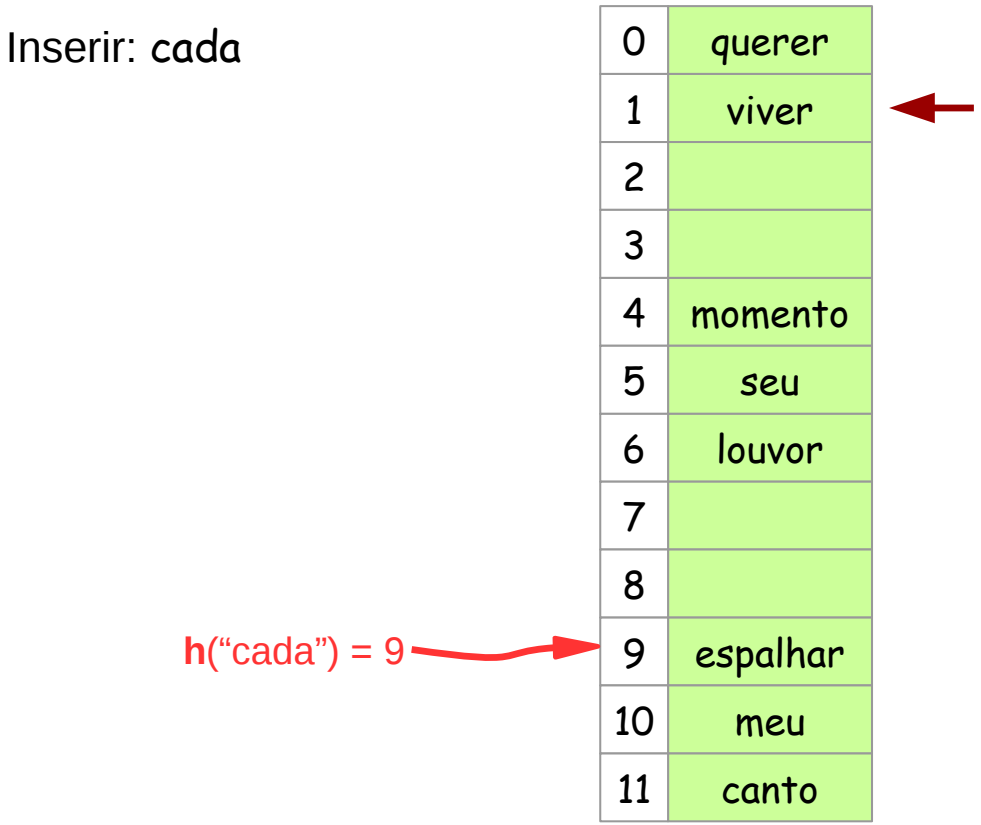
Colisão: reespalhamento



Inserindo com colisão

- procuramos a posição
- se não houver espaço, procuramos posição livre

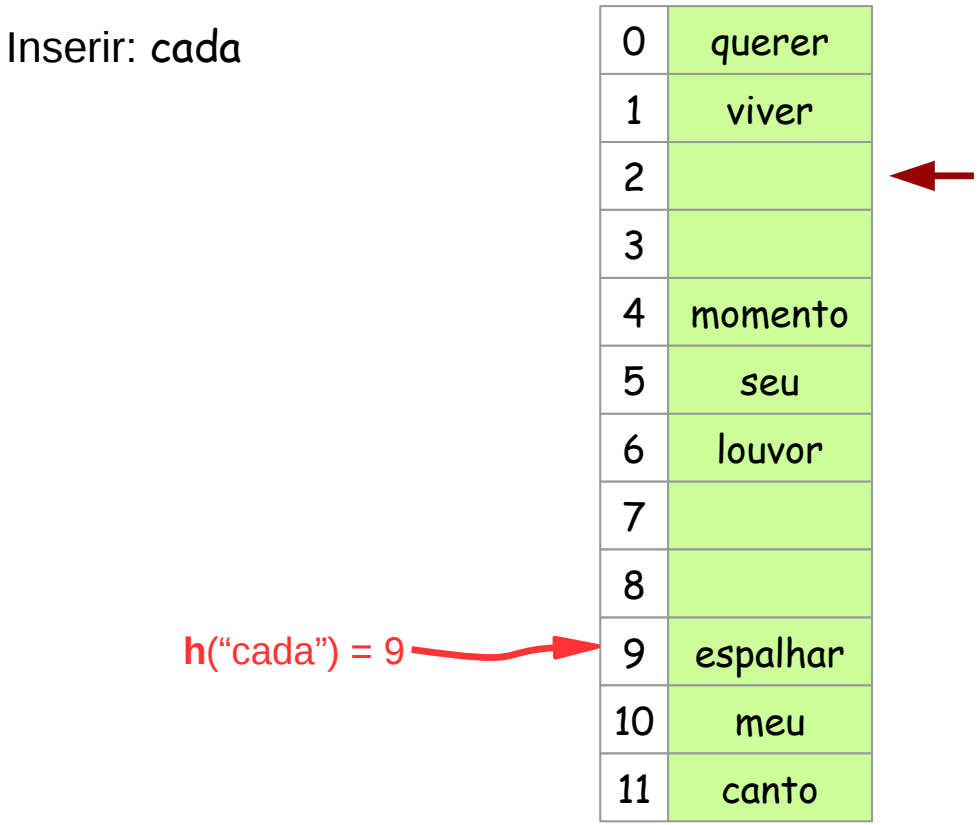
Colisão: reespalhamento



Inserindo com colisão

- procuramos a posição
- se não houver espaço, procuramos posição livre

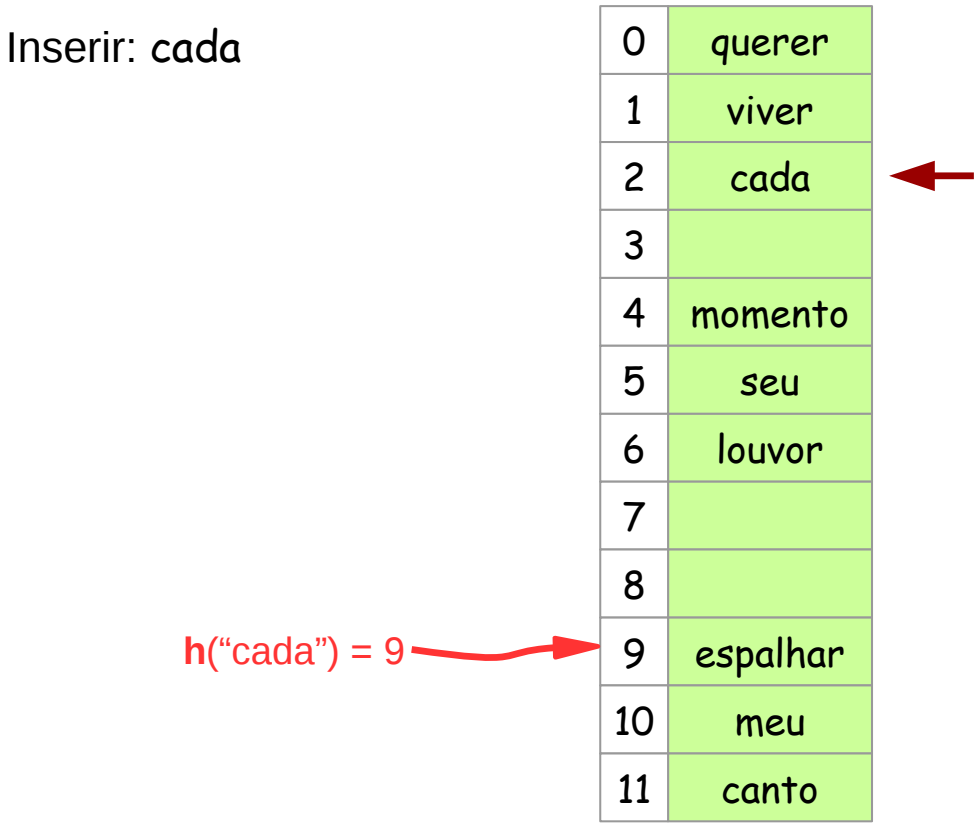
Colisão: reespalhamento



Inserindo com colisão

- procuramos a posição
- se não houver espaço, procuramos posição livre

Colisão: reespalhamento



Inserindo com colisão

- procuramos a posição
- se não houver espaço, procuramos posição livre

Colisão: reespalhamento quadrático

0	querer
1	viver
2	
3	
4	momento
5	
6	louvor
7	
8	
9	
10	meu
11	canto

Inserindo espalhar

Colisão: reespalhamento quadrático

Inserir: **espalhar**

0	querer
1	viver
2	
3	
4	momento
5	
6	louvor
7	
8	
9	
10	meu
11	canto

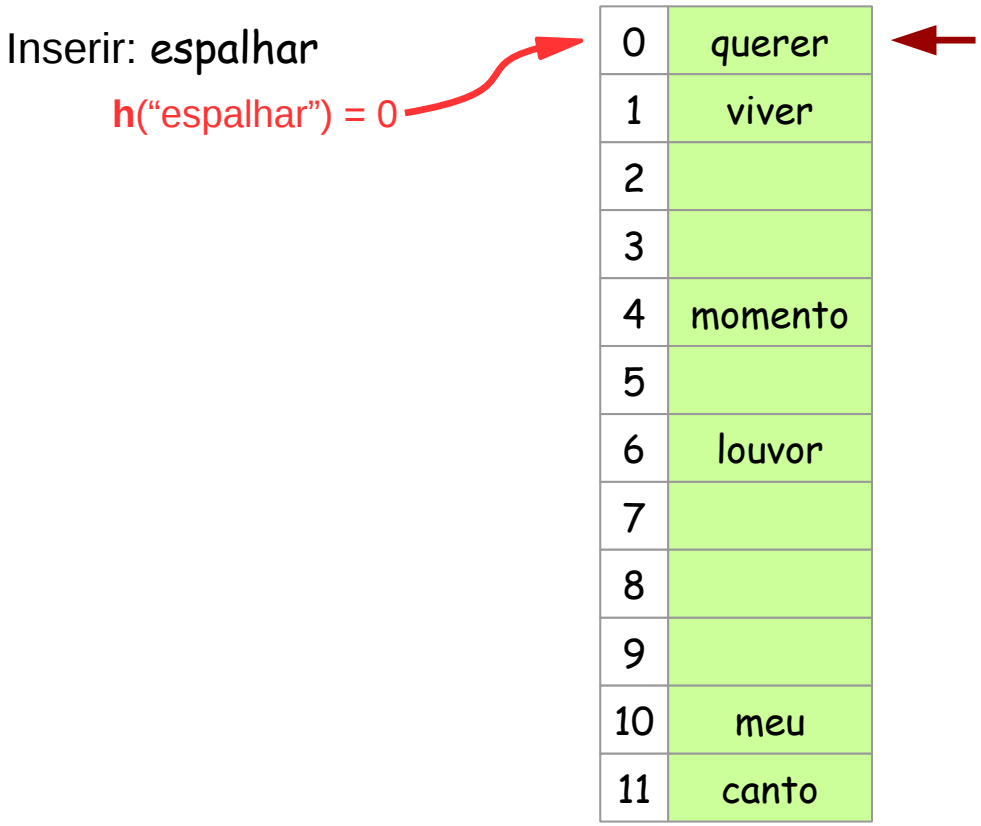
Inserindo espalhar

Colisão: reespalhamento quadrático

Inserindo espalhar

- se não houver espaço,

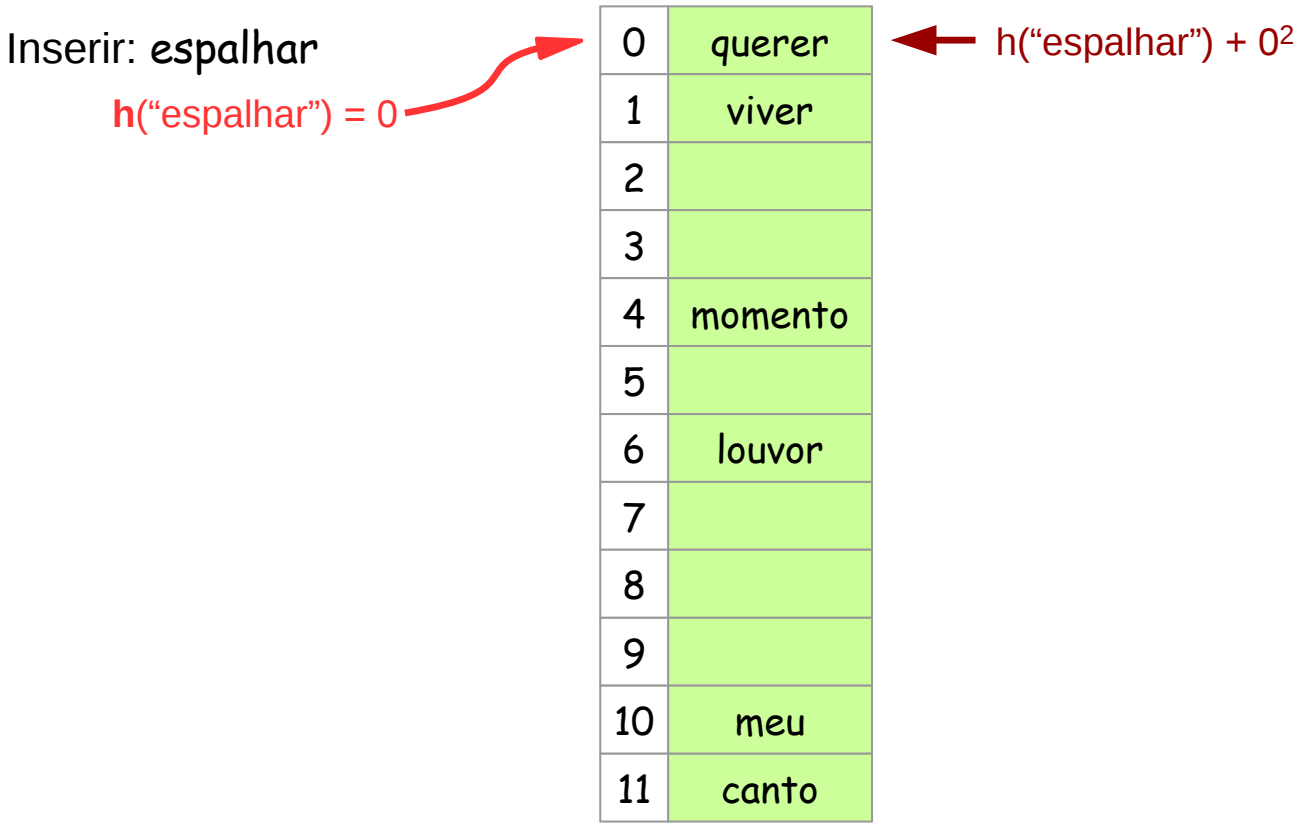
Colisão: reespalhamento quadrático



Inserindo espalhar

- se não houver espaço, procuramos posição livre

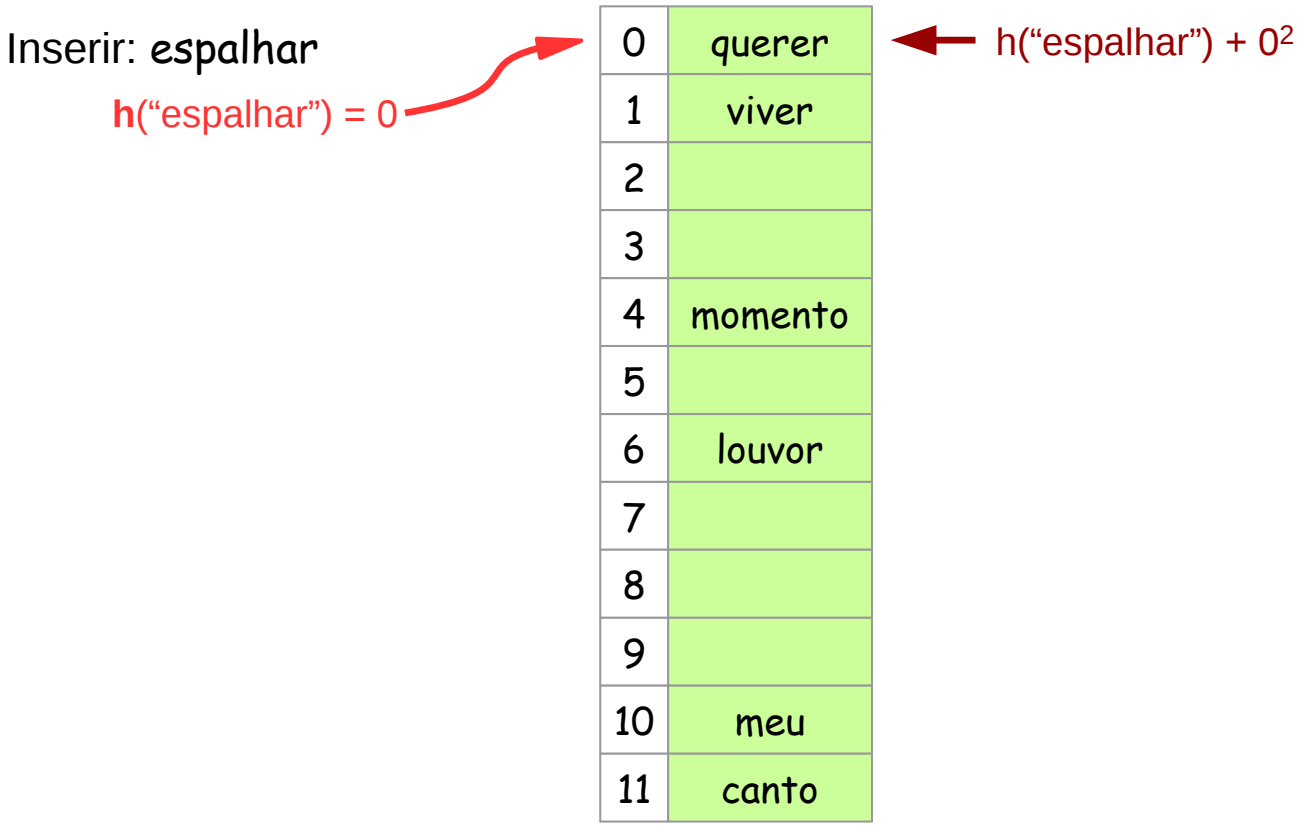
Colisão: reespalhamento quadrático



Inserindo espalhar

- se não houver espaço, procuramos posição livre "pulando" quadrados

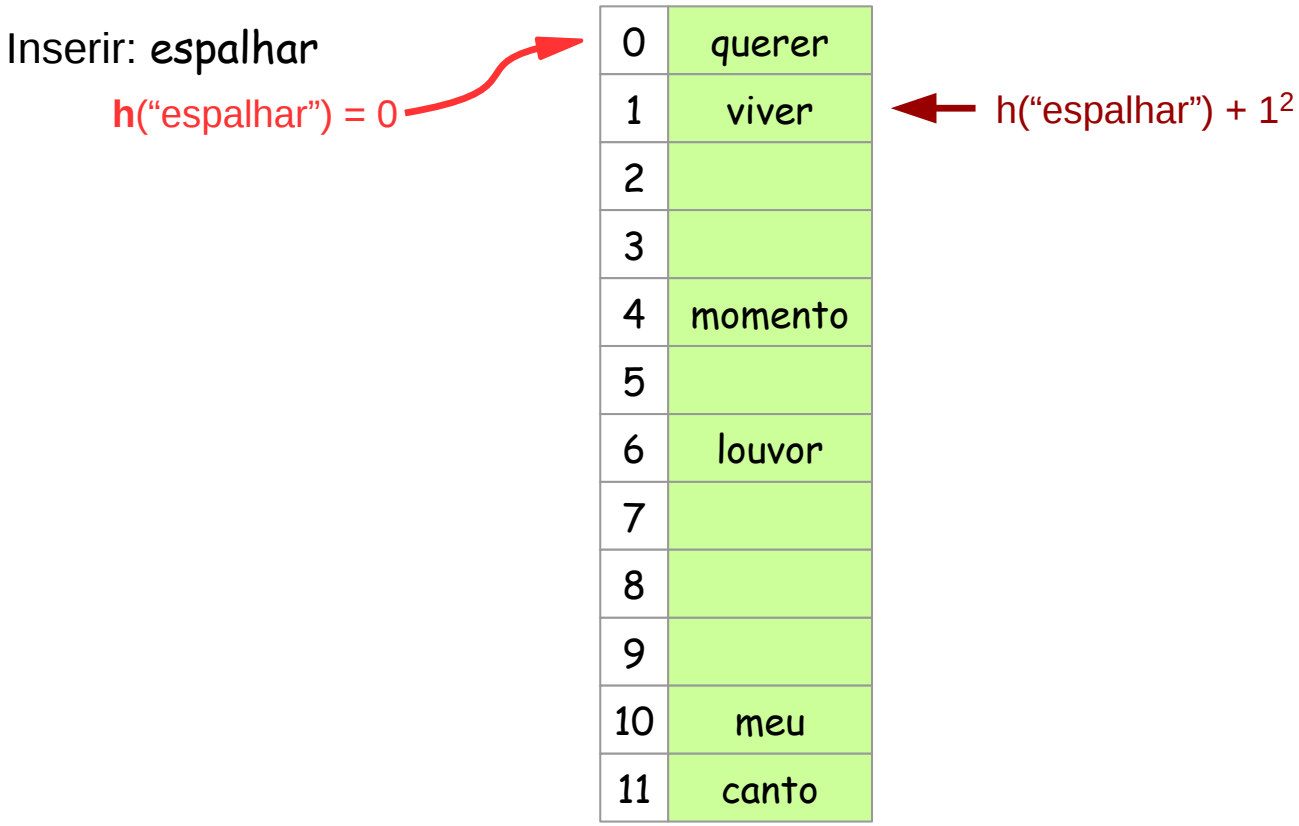
Colisão: reespalhamento quadrático



Inserindo espalhar

- se não houver espaço, procuramos posição livre "pulando" quadrados

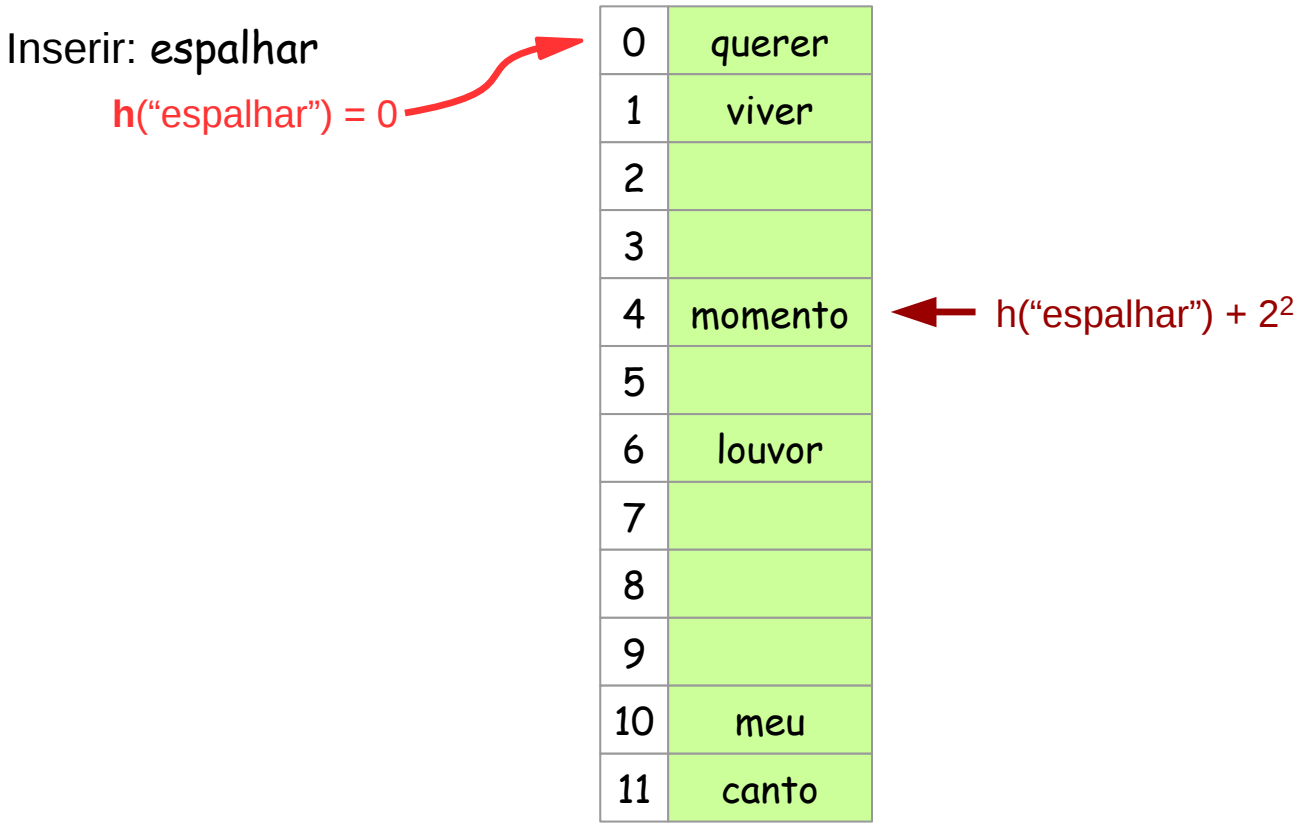
Colisão: reespalhamento quadrático



Inserindo espalhar

- se não houver espaço, procuramos posição livre "pulando" quadrados

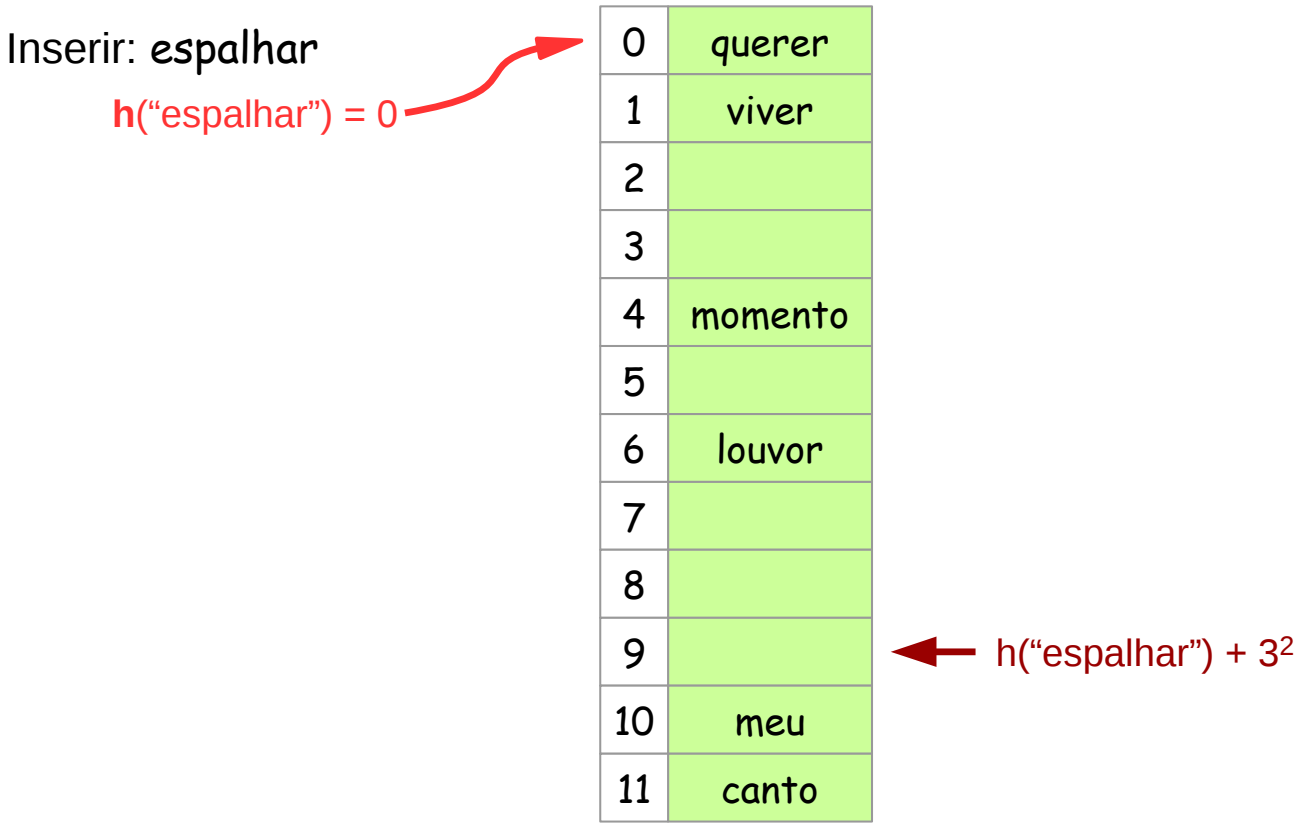
Colisão: reespalhamento quadrático



Inserindo espalhar

- se não houver espaço, procuramos posição livre "pulando" quadrados

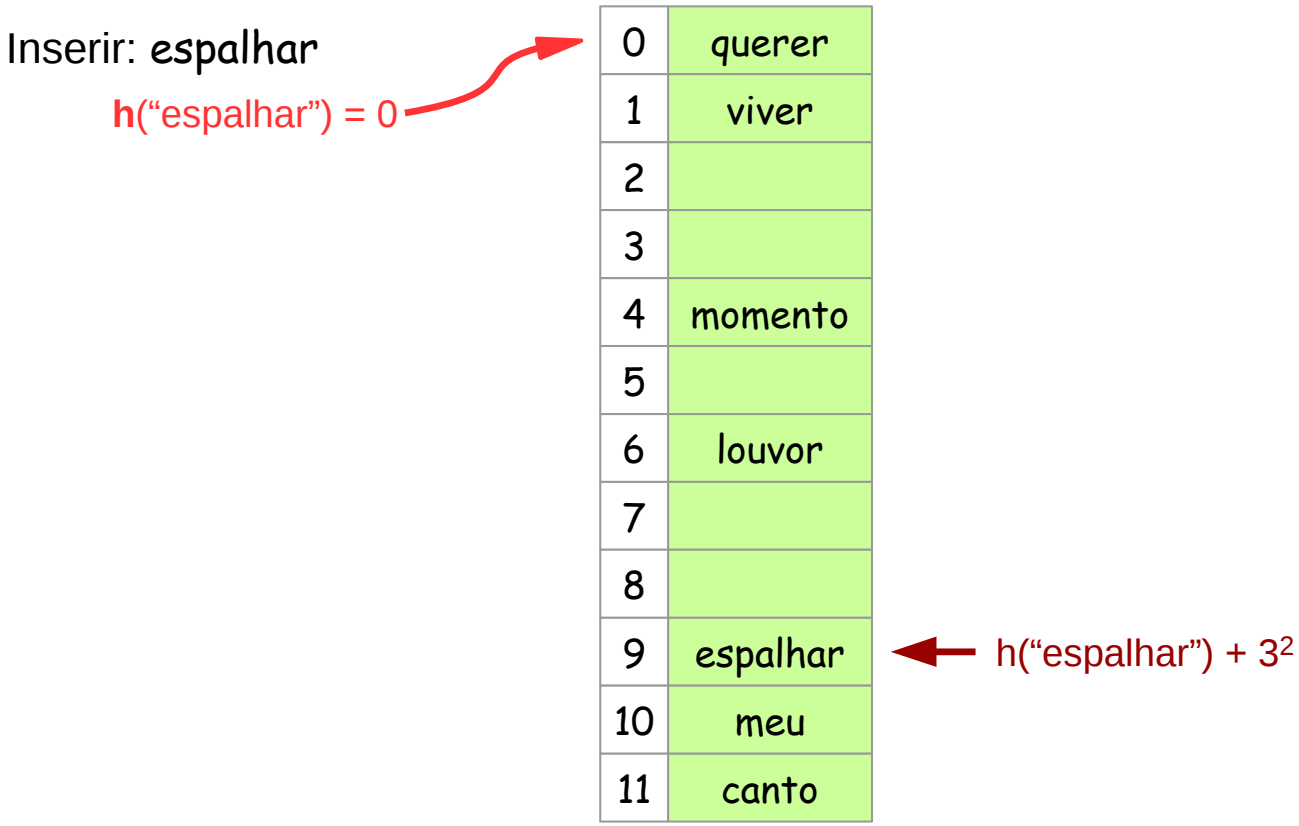
Colisão: reespalhamento quadrático



Inserindo espalhar

- se não houver espaço, procuramos posição livre "pulando" quadrados

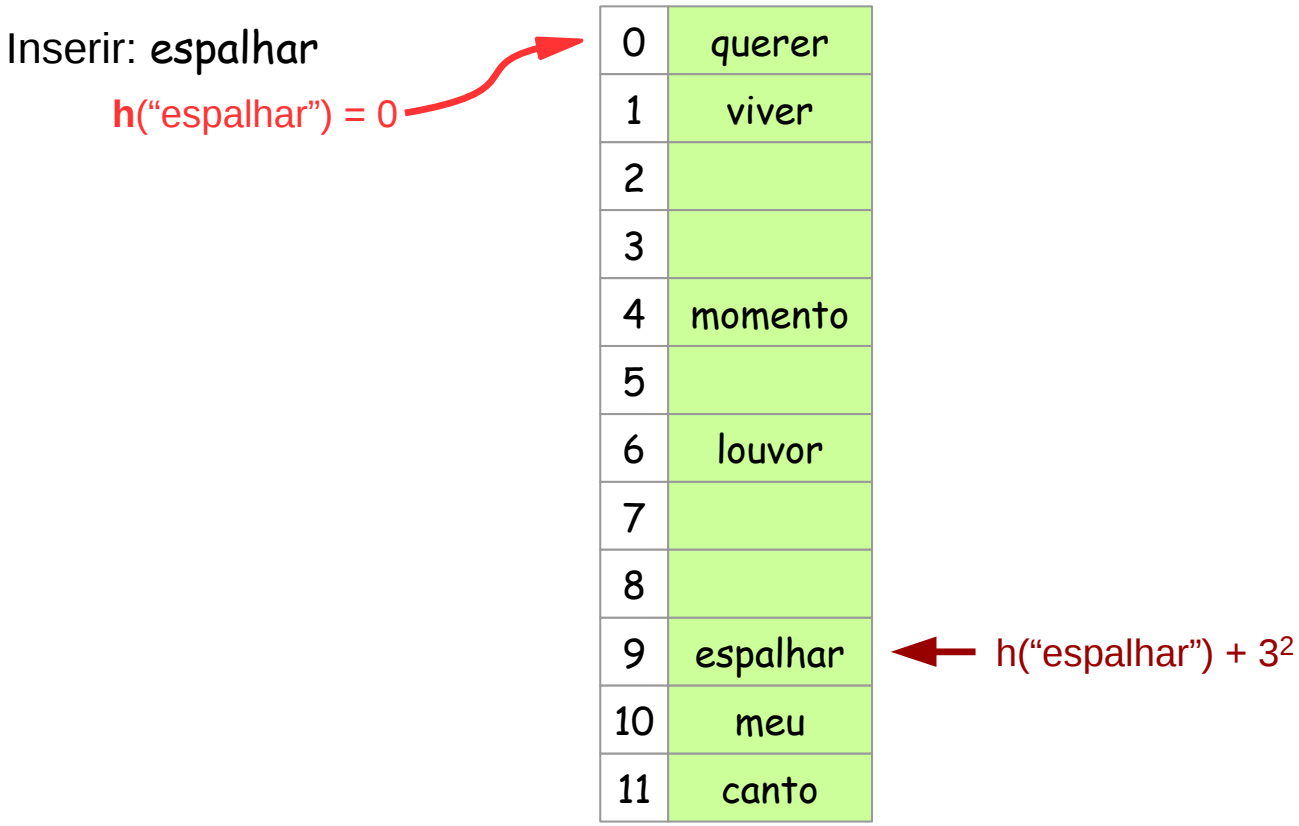
Colisão: reespalhamento quadrático



Inserindo espalhar

- se não houver espaço, procuramos posição livre "pulando" quadrados

Colisão: reespalhamento quadrático



Inserindo espalhar

- se não houver espaço, procuramos posição livre "pulando" quadrados
- uma função de hashing secundária poderia ser utilizada

Exercícios

1. Suponha que queremos guardar os dados de uma turma de de alunos ingressantes em uma tabela de espalhamento. Escreva uma boa função de hashing para isso.
2. Quais as complicações para a remoção de tabelas de espalhamento que utilizam endereçamento aberto? Especifique um TAD "Tabela de Espalhamento" com endereçamento aberto.
3. Suponha que $h(\text{"há"}) = 6$. Insira essa chave na tabela do slide 17 usando reespalhamento linear.
4. Suponha que $h(\text{"vão"}) = 11$. Insira essa chave na tabela do slide 18 usando reespalhamento quadrático.

Exercícios 2 - Extra: outras aplicações

Funções de hashing podem ser aplicada em outros contextos:

- **Verificação de paridade:** para evitar erros de transmissão, podemos, além de informar uma chave, transmitir o resultado da função de hashing. Exemplos:
  - dígitos verificadores
  - sequências de verificação para arquivos (*checksum* MD5 e SHA)
- **Segurança:** por definição, calcular o resultado da função de hashing para um objeto deve ser rápido; o problema inverso é o seguinte: dado um número n, queremos encontrar algum objeto x tal que $f(x) = n$. Uma função de hashing cuja inversa seja "difícil" de calcular é considera segura. Tente explicar o porquê dessa definição.

Considere um número primo grande p. Explique porque é fácil inverter a função de hashing $h_1(x) = x \bmod p$, enquanto $h_2(x) = x^2 \bmod p$ não é tão fácil.