

TABELA DE DISPERSÃO (*HASH TABLE*)

Prof. Dieisson Martinelli
dieisson.martinelli@udesc.br

Programação

- Tabelas hash
- Funções de espalhamento
- Resolução de colisões
- Atividades

Tabelas hash

▶ Uma **tabela de dispersão** (ou *hash table*) é uma coleção de itens que são armazenados de maneira a serem encontrados com mais facilidade

- Cada **posição da tabela**, geralmente denominada **índice** (ou slot), pode guardar um item e possui um **rótulo inteiro** começando em 0

0	1	2	3	4	5	6	7	8	9	10
None	None	None	None	None	None	None	None	None	None	None

- Inicialmente, a tabela de dispersão **não contém nenhum item**, então **todos os índices estão vazios**. É possível implementá-la usando uma **lista** com cada elemento inicializado pelo **valor especial “None”**

Tabelas hash

- O **mapeamento** entre a **chave** e o **índice** ao qual ela pertence na tabela é conhecido como a **função de espalhamento** (ou *hash function*)
 - A função irá receber qualquer item na coleção e irá retornar um inteiro dentro do intervalo dos índices, isto é, entre 0 e m-1 (onde m é o número de posições da tabela)
- Exemplo: conjunto de inteiros formado por 54, 26, 93, 17, 77 e 31

$$h(\text{item}) = \text{item} \% 11$$

 - Esta primeira **função de hash**, às vezes chamada de “método do resto”, simplesmente **pega um item** e o **divide** pelo **tamanho da tabela**, retornando o **resto da divisão** que é o seu **valor de espalhamento**

Tabelas hash

- Itens: 54, 26, 93, 17, 77 e 31



$$h(\text{item}) = \text{item} \% 11$$

- Resultado:

```
>>> 54 % 11
10
>>> 26 % 11
4
>>> 93 % 11
5
>>> 17 % 11
6
>>> 77 % 11
0
>>> 31 % 11
9
```

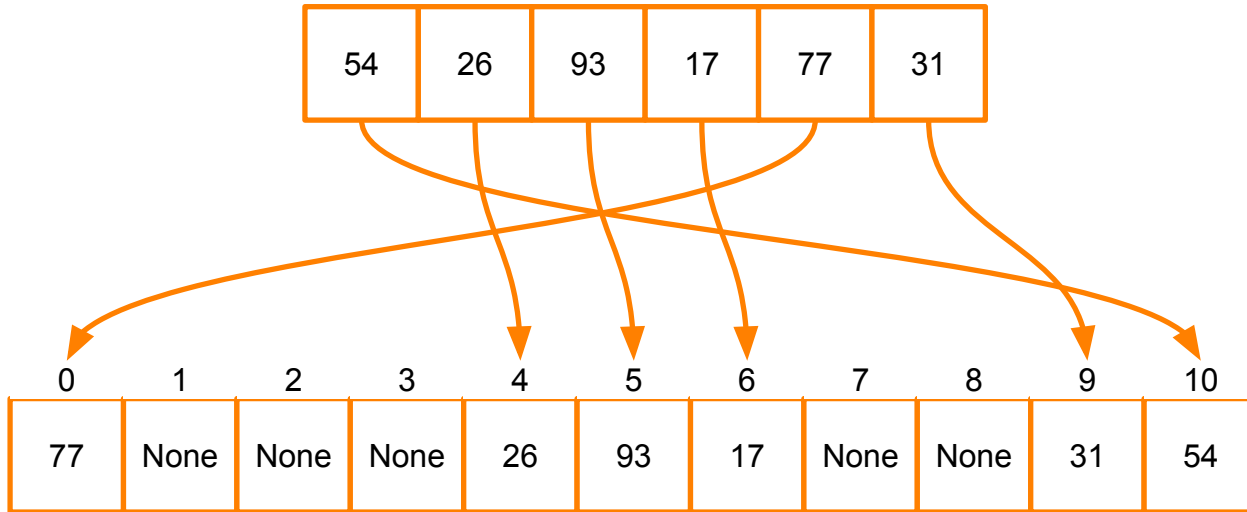
- Uma vez que os valores de espalhamento tenham sido computados, basta inserir cada item na tabela de dispersão na posição designada

Tabelas hash

- Itens: 54, 26, 93, 17, 77 e 31
- Resultado: 10, 4, 5, 6, 0 e 9



$$h(\text{item}) = \text{item} \% 11$$



Item	Valor de espalhamento
54	10
26	4
93	5
17	6
77	0
31	9

Tabelas hash

0	1	2	3	4	5	6	7	8	9	10
77	None	None	None	26	93	17	None	None	31	54

- Note que 6 dos 11 índices estão agora ocupados. Essa razão é conhecida como **fator de carga** e é denotada por:

$$\lambda = \text{numero de indices} / \text{tamanho da tabela}$$

- Para o exemplo acima, o fator de carga é $\lambda = 6/11$

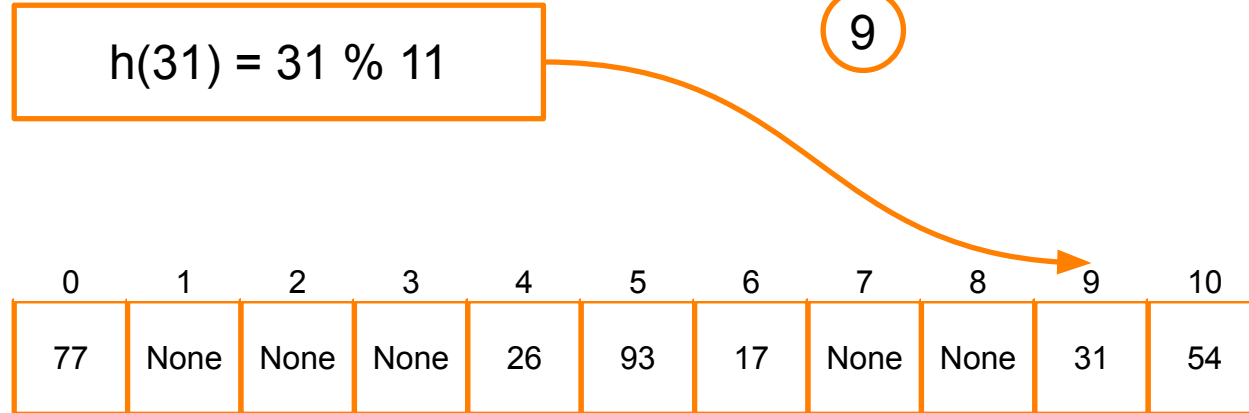
Tabelas hash

- O que o fator de carga indica?
 - Um fator de carga baixo significa que há muitos espaços vazios — a busca e a inserção tendem a ser mais rápidas.
 - Um fator de carga alto (próximo de 1 ou maior) pode causar muitos conflitos (colisões), tornando a tabela hash menos eficiente.

Tabelas hash

- Para **procurar um item** na tabela, basta utilizar a **função de espalhamento** para **calcular o índice** correspondente da **chave** e acessar a tabela de dispersão para verificar se ela está presente

- Exemplo:



Tabelas hash

- Passos para procurar um item em uma tabela hash:
 - Receber a chave que você quer procurar.
 - Aplicar a função de espalhamento (ou função hash) à chave.
 - O resultado será um índice da tabela.
 - Acessar diretamente esse índice na tabela:
 - Se a posição estiver vazia, o item não está presente.
 - Se houver um item, comparar as chaves (para garantir que é o item certo).
 - Se for igual, o item foi encontrado.
 - E se for diferente?

Tabelas hash

- Colisões são inevitáveis em tabelas hash, porque várias chaves podem gerar o mesmo índice.
- Quando isso acontece, usamos estratégias de tratamento de colisão para decidir o que fazer quando duas (ou mais) chaves caem no mesmo lugar
 - Sondagem Linear (Linear Probing)
 - Sondagem Quadrática (Quadratic Probing)
 - Hash Duplo (Double Hashing)
 - Encadeamento (Chaining)

Tabelas hash

- Exemplo: se o item 44 fosse a próxima chave da coleção, ele teria um valor de espalhamento de 0 ($44 \% 11 = 0$)

0	1	2	3	4	5	6	7	8	9	10
77	None	None	None	26	93	17	None	None	31	54

- Como 77 também possui 0 como valor de espalhamento, teríamos um problema que é chamado de **colisão**

Funções de espalhamento

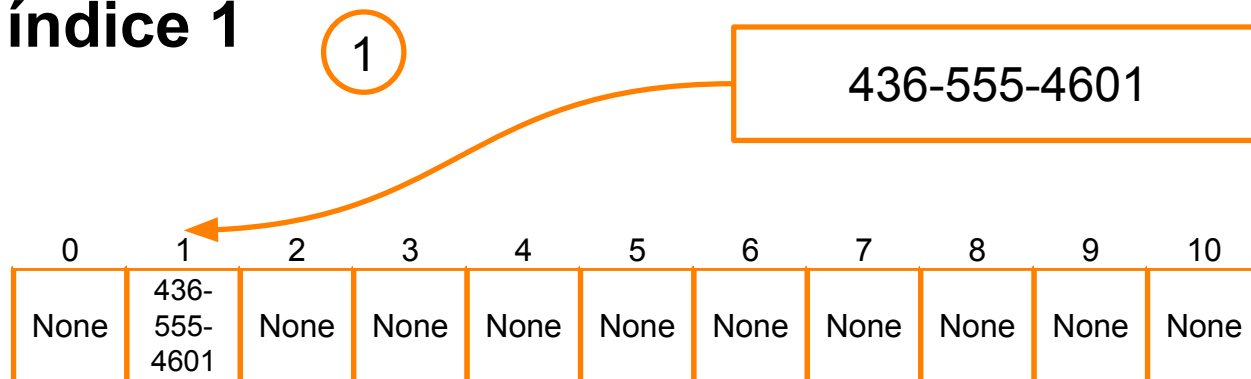
- Uma **função de espalhamento** (*hash function*) capaz de mapear cada item para um índice único é chamada de **função de espalhamento perfeita**
- Se soubermos que os itens e a coleção nunca irá mudar, então é possível construir uma função de espalhamento perfeita
 - Se os itens forem escolhidos de forma aleatória ou mudarem com o tempo, não existe uma fórmula garantida que consiga evitar colisões para todos os casos.
 - Felizmente, não precisamos que a função de espalhamento seja perfeita para **obter ganhos de desempenho**
- Uma **função hash** precisa minimizar o número de **colisões**, ser fácil de computar e distribuir os itens uniformemente na **tabela hash**

Funções de espalhamento

- Há algumas maneiras conhecidas de estender o método do resto da divisão
- **método de *folding***: Transforma a chave em partes menores, soma essas partes e usa o total como base para o índice na tabela hash.
- O **método de *folding*** começa dividindo o item em pedaços de tamanhos iguais (o último pedaço pode não ser de tamanho igual). Esses pedaços são **somados** para então gerar o valor de espalhamento resultante
- Por exemplo
 - Se o item fosse um número de telefone 436-555-4601, iríamos extrair os dígitos e dividi-los em grupos de 2 (43, 65, 55, 46, 01). Depois de **somar tudo**, $43+65+55+46+01$, **teríamos 210**

Funções de espalhamento


- Se a tabela de dispersão tiver **11 índices**, então é necessário **dividir o resultado por 11** e pegar o resto da divisão. Nesse caso, $210 \% 11$ é 1, então o número de telefone seria mapeado para o **índice 1**



- Alguns **métodos de *folding*** vão além e trocam a ordem dos pedaços antes de somá-los. Para o exemplo acima, teríamos $43+56+55+64+01=219$, o que resulta em $219 \% 11=10$

Funções de espalhamento

- ▶ O **método do quadrado do meio** funciona assim: você pega o valor da chave, faz o quadrado (multiplica por ele mesmo), e depois pega os dígitos do meio do resultado para usar como índice na tabela.
- Por exemplo
 - Se o item for **44**, primeiro calculamos $44^2=1.936$. Extraíndo os dois dígitos do meio, **93**, e realizando o passo de pegar o resto da divisão, ficamos com **5** ($93 \% 11$)



0	1	2	3	4	5	6	7	8	9	10
None	None	None	None	None	44	None	None	None	None	None

Funções de espalhamento

- Comparação do método de resto e quadrado do meio

Item	Resto	Quadrado do meio	
54	10	3	$54^2 = 2916$ ($91 \% 11 = 3$)
26	4	7	$26^2 = 676$ ($7 \% 11 = 7$)
93	5	9	$93^2 = 8649$ ($64 \% 11 = 9$)
17	6	8	$17^2 = 289$ ($8 \% 11 = 8$)
77	0	4	$77^2 = 5929$ ($92 \% 11 = 4$)
31	9	6	$31^2 = 961$ ($6 \% 11 = 6$)
44	0	5	$44^2 = 1936$ ($93 \% 11 = 5$)

colisão



Funções de espalhamento

- Podemos criar também **funções de espalhamento** para itens baseados em **caracteres**, como **strings**. A palavra “**cat**” pode ser entendida como uma sequência de **valores ordinais**
- Exemplo:

```
>>> ord('c')
99
>>> ord('a')
97
>>> ord('t')
116
```

- Podemos pegar esses três valores, somá-los e usar o **método do resto da divisão** para extrair um valor de **espalhamento**
- $99+97+116 = 312$ ($312 \% 11 = 4$)

Funções de espalhamento

- Entretanto, no método do exemplo anterior os **anagramas** terão o **mesmo valor de dispersão**:
 - **CAT**: $(99+97+113) \% 11 = 4$; **ACT**: $(97+99+113) \% 11 = 4$; **TAC**...
- Para contornar isso, nós podemos utilizar a **posição de cada caractere** do anagrama como um **peso**

c a t

↓ ↓ ↓

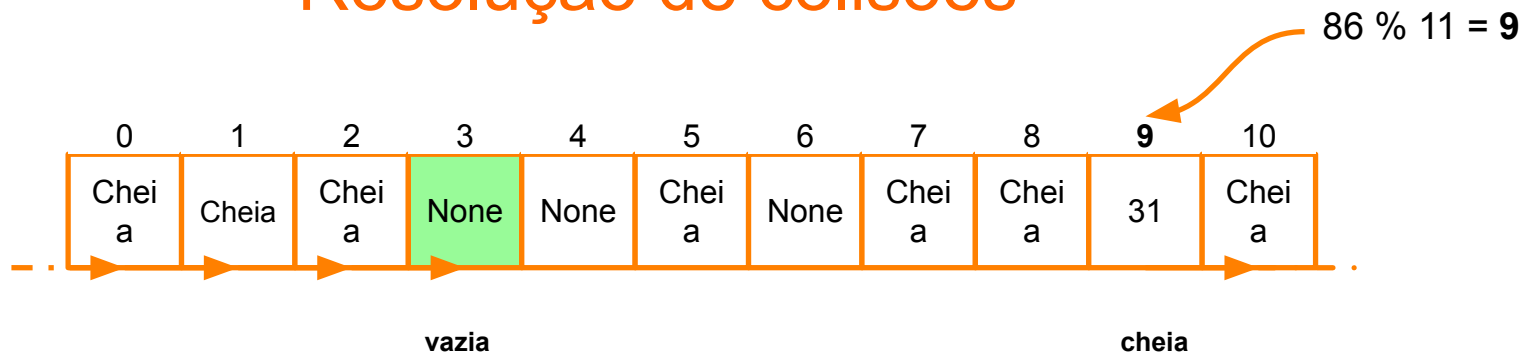
$$99*1 + 97*2 + 113*3 = 641 \quad (641 \% 11 \rightarrow 3)$$

- A função de espalhamento precisa ser eficiente, de modo que ela não se torne a parte dominante no processo de armazenamento e busca
- Se a função de espalhamento for muito complexa, então computar posições na tabela se torna mais trabalhoso do que simplesmente realizar uma **busca sequencial** ou **binária**

Resolução de colisões

- Quando dois itens são levados à **mesma posição** pela função de espalhamento, é preciso ter uma forma sistemática de colocar o segundo item na **tabela *hash***. Esse processo é chamado de **resolução de colisões**
- Um **método** para resolução de colisões olha para a **tabela *hash*** e tenta encontrar **outra posição** aberta que possa armazenar o item que causou a colisão
 - Uma forma simples de fazer isso é começar pela posição do **valor de espalhamento original** e mover de forma sequencial pelas entradas até encontrar a primeira que esteja vazia
 - Por exemplo: armazenar o valor 86 em uma tabela de 11 posições

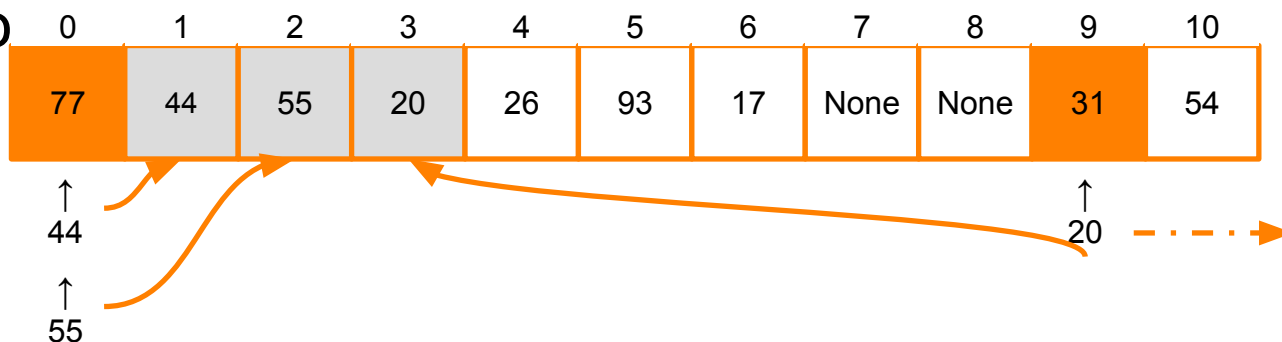
Resolução de colisões



- Note que poderemos ter que voltar para a primeira entrada (**circularmente**) para cobrir a tabela de dispersão inteira
- Esse processo de resolução de colisões é conhecido como **endereçamento aberto**, já que ele procura encontrar a próxima entrada disponível na tabela
- Ao visitar sistematicamente uma posição por vez, estamos realizando uma técnica de endereçamento aberto chamada **sondagem linear**

Resolução de colisões

- Exemplo: armazenar os inteiros (54, 26, 93, 17, 77, 31, **44**, **55**, **20**) submetidos à função de espalhamento baseada no resto da divisão



- Ao tentar colocar o **44** na entrada 0 (zero) ocorre uma colisão. Usando a sondagem linear percorre-se sequencialmente as entradas até encontrar uma posição disponível (**entrada 1**). Novamente, o **55** cai em 0 (zero) e é colocado na **entrada 2** (próxima entrada disponível). Por fim, o valor **20** leva à entrada 9 (que já está ocupada). Então a sondagem visita as entradas 10, 0, 1 e 2, até encontrar a **posição 3** (vazia) na tabela

Resolução de colisões

- Uma vez construída uma tabela de dispersão usando **endereçamento aberto** e **sondagem linear**, é essencial que se utilize os mesmos métodos para recuperar estes itens

0	1	2	3	4	5	6	7	8	9	10
77	44	55	20	26	93	17	None	None	31	54

- Para buscar o **item 93** basta computar o seu valor de espalhamento ($93 \% 11 = 5$), retornando **True**. Mas para computar o **item 20** ($20 \% 11 = 9$) é necessário **percorrer sequencialmente as entradas**, começando pela posição 10, até encontrar o item 20 (retornando **True**) ou uma entrada vazia (retornando **False**)

Resolução de colisões

- Uma **desvantagem** da sondagem linear é a tendência de **aglutinação** (agrupamento de itens devido às muitas colisões pelo mesmo valor de espalhamento)
 - Isso terá um impacto sobre os outros itens que forem inseridos, como quando foi adicionado o **item 20**. Uma aglutinação de valores **sendo levados a 0** teve que ser vencida até achar a **posição vazia**

0	1	2	3	4	5	6	7	8	9	10
77	44	55	20	26	93	17	None	None	31	54

- Uma forma de lidar com a aglutinação é estender a **técnica de sondagem linear** para que em vez de procurar sequencialmente a próxima entrada aberta, alguns **slots sejam pulados**

Resolução de colisões

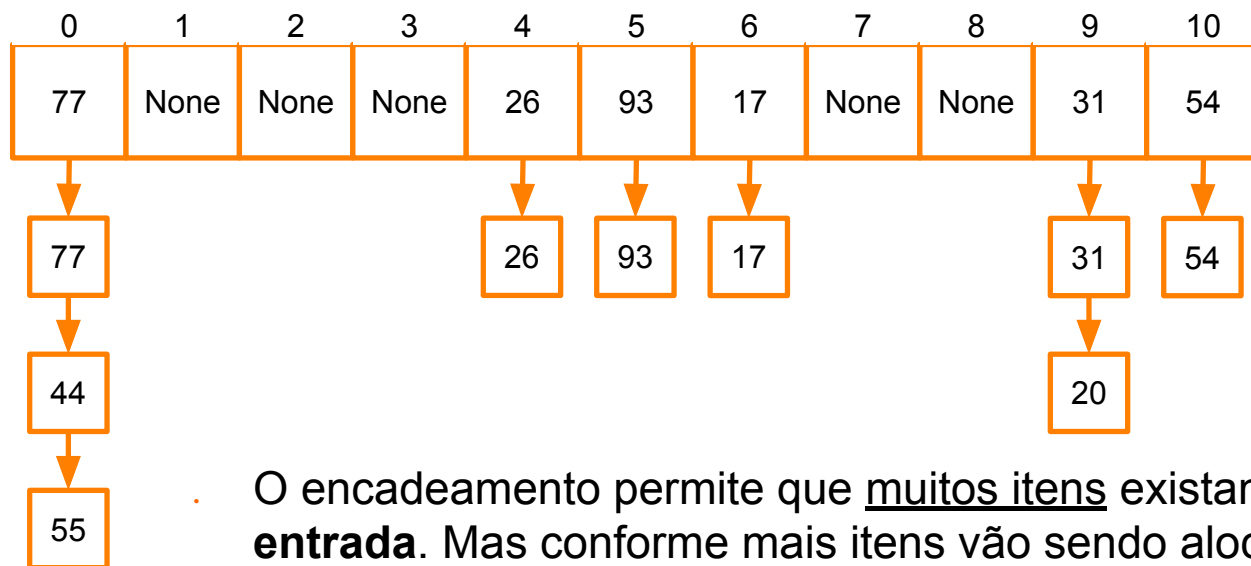
- **Exemplo:** itens dispostos com uma **sondagem “mais 3”**. Significa que uma vez ocorrida a colisão, uma procura de **três em três entradas** é feita até encontrar uma entrada vazia

0	1	2	3	4	5	6	7	8	9	10
77	55	None	44	26	93	17	20	None	31	54

- O nome geral do processo de procurar por outro *slot* depois de uma colisão é **rehashing**. O rehash “mais 3” pode ser definido como **$rehash(pos) = (pos + 3) \% tamanhodataabela$**
- De modo geral, temos **$rehash(pos) = (pos + pulo) \% tamanhodataabela$** . É importante notar que o tamanho do “pulo” precisa assumir um valor tal que todas as entradas da tabela serão visitadas em algum momento

Resolução de colisões

- Um método alternativo para lidar com o **problema da colisão** é permitir que cada entrada tenha uma referência para uma **coleção** (ou sequência) de itens



- O encadeamento permite que muitos itens existam na **mesma entrada**. Mas conforme mais itens vão sendo alocados para a mesma posição, a dificuldade de encontrar um item na coleção **aumenta**

Exercícios

- 1 Na tabela de dispersão de tamanho 13, para quais entradas as chaves 27 e 30 seriam mapeadas?
- 2 Suponha que seja dado o seguinte conjunto de chaves a serem inseridas em uma tabela de dispersão com 11 posições: 113 , 117 , 97 , 100 , 114 , 108 , 116 , 105 , 99. Apresente o conteúdo da tabela depois de todas as chaves serem inseridas com sondagem linear

0	1	2	3	4	5	6	7	8	9	10