

Hashing

Pesquisa, Ordenação e Técnicas de Armazenamento

Prof. Me. Orlando da Silva Junior

Dado um vetor v de n posições, implemente um método chamado *contaNumeros* para contar a quantidade de vezes que os números desse vetor aparecem. Seu método deve receber um vetor v como entrada e devolver um vetor w com essa contagem.

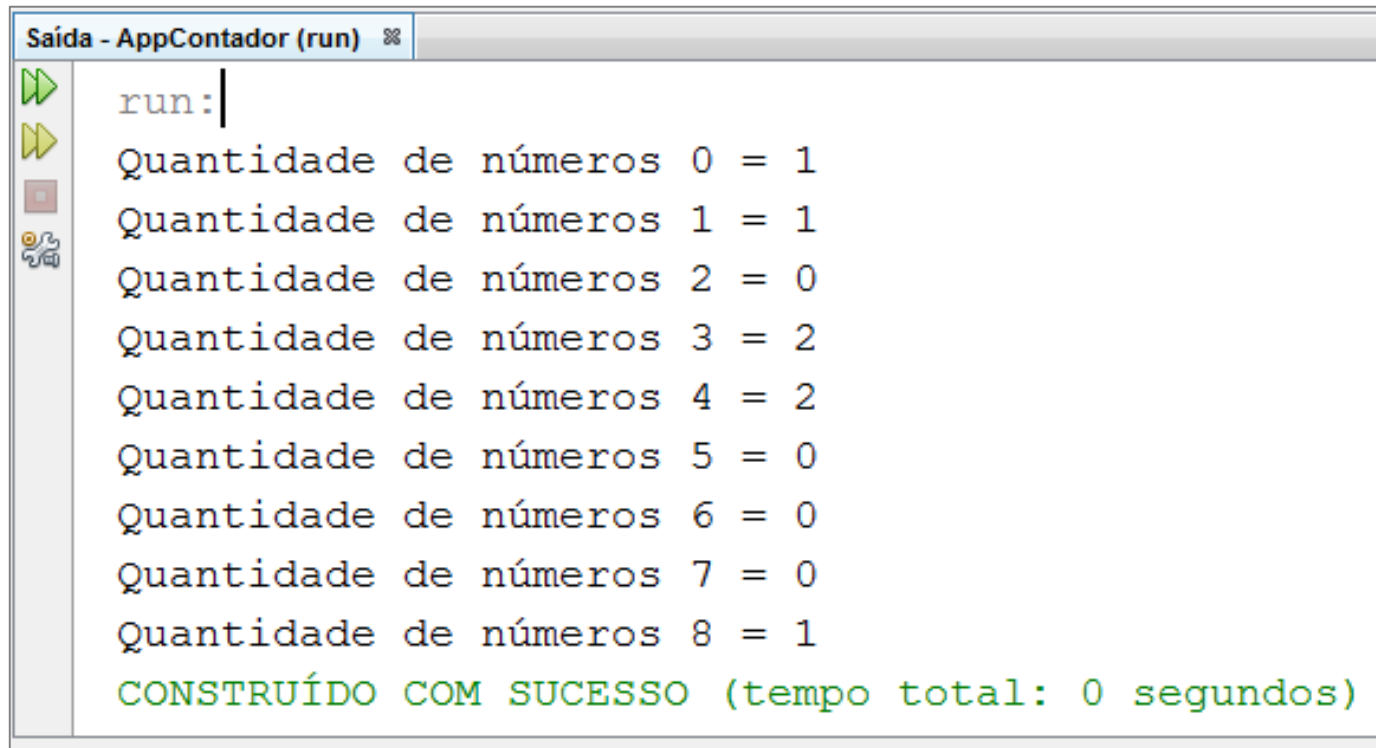
Exemplo:

Para $v = [3\ 4\ 3\ 4\ 1\ 0\ 8]$

Tem-se $w = [1\ 1\ 0\ 2\ 2\ 0\ 0\ 0\ 1]$

***Dica:** faça com que o índice do vetor w seja o valor do vetor v , e o valor do vetor w seja o valor do contador para o i -ésimo número em v .*

```
public static int[] contaNumeros(int[] v) {  
    // encontra maior valor  
    int maior = v[0];  
    for (int i = 1; i < v.length; i++) {  
        if (v[i] > maior) {  
            maior = v[i];  
        }  
    }  
  
    // cria contador  
    int w[] = new int[maior + 1];  
    for (int i = 0; i <= maior; i++) {  
        w[i] = 0;  
    }  
  
    // conta frequência  
    for (int i = 0; i < w.length; i++) {  
        w[v[i]]++;  
    }  
    return w;  
}
```



```
Saída - AppContador (run) ✖
run:
Quantidade de números 0 = 1
Quantidade de números 1 = 1
Quantidade de números 2 = 0
Quantidade de números 3 = 2
Quantidade de números 4 = 2
Quantidade de números 5 = 0
Quantidade de números 6 = 0
Quantidade de números 7 = 0
Quantidade de números 8 = 1
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

```
public static void main(String[] args) {
    int v[] = {3, 4, 3, 4, 1, 0, 8};
    int w[] = contaNumeros(v);

    for (int i = 0; i < w.length; i++) {
        System.out.println(String.format("Quantidade de números %d = %d", i, w[i]));
    }
}
```

O que fizemos?

```
w[v[i]]++;
```

- Utilizamos o valor presente no vetor v para ser o índice do vetor w .
- O valor do vetor w na posição k é igual à quantidade de vezes que o valor $v[i]$ aparece no vetor v .

v	3	4	3	4	1	0	8	$n = 7$
	0	1	2	3	4	5	6	

w	1	1	0	2	2	0	0	0	1	$n = 8 + 1 = 9$
	0	1	2	3	4	5	6	7	8	

Por que precisamos disso?

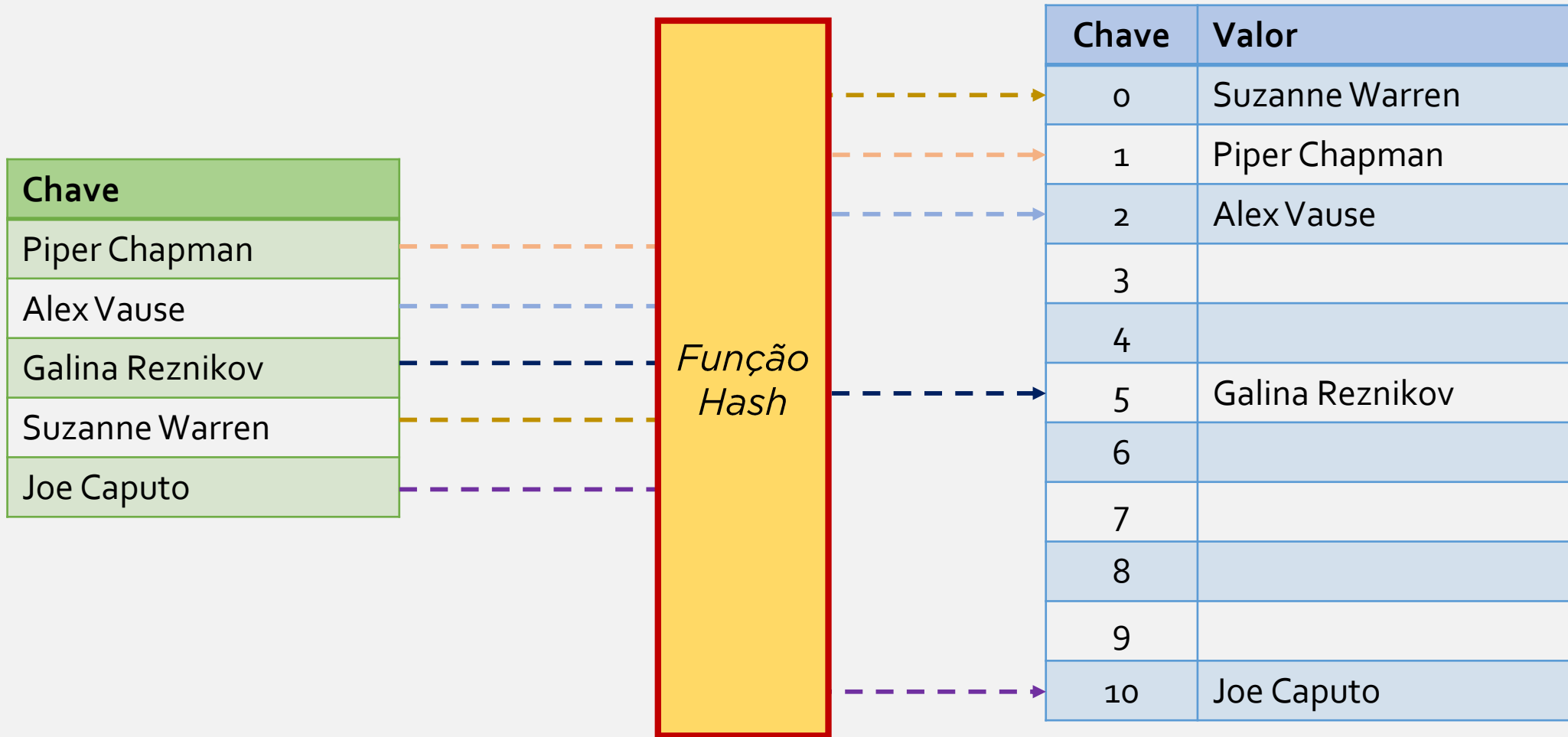
- Em algumas aplicações, é necessário obter o valor procurado com **poucas comparações**.
- Logo, é preciso saber a posição em que o elemento se encontra, sem precisar varrer todas as chaves.
- **Queremos** buscar um elemento em $O(1)$

Tabela de Hash

Ideia fundamental:

- Um vetor em que cada uma das posições armazena nenhuma, uma ou mais de uma chave e valores associados.
- Associa uma **chave** a um **valor** (inteiro, ponto flutuante, objeto).

Objetivo: armazenar os registros em uma tabela por meio de uma **transformação aritmética** sobre a chave de pesquisa (partes da informação).



Representação

Podemos representar as tabelas de espalhamento de duas formas:

1. Utilizando **vetores**: cada posição do vetor armazena uma informação;
2. Utilizando **vetores e listas encadeadas**: o vetor contém referências para listas que representam as informações.

Hashing

Um *método de busca* com *transformação de chave* é formado por 2 etapas principais:

1. Computar o valor da **função de transformação** (função hashing), transformando a chave de pesquisa em um endereço da tabela.
2. Tratar **colisões**, considerando que duas ou mais chaves podem ser transformadas em um mesmo endereço da tabela.

Função de Transformação

Uma função de transformação deve:

- ✓ Ser facilmente computável;
- ✓ Mapear chaves em números inteiros entre 0 e $N - 1$, sendo N o tamanho da tabela;
- ✓ Gerar entradas para a tabela com igual probabilidade.

Função de Transformação

- Considerando que as transformações sobre as chaves são aritméticas, o primeiro passo é transformar as chaves não-numéricas.
- Em Java, basta realizar a **conversão** de cada caractere da chave não-numérica para um número inteiro.

Função de Transformação

O método mais utilizado é o **resto da divisão por N** :

$$h(K) = K \bmod N$$

K é um inteiro correspondente à chave

K é um inteiro correspondente à chave, computado por:

$$K = \sum_{i=0}^{n-1} chave[i] \cdot p[i]$$

- n é o número de caracteres da chave;
- $chave[i]$ corresponde à representação ASCII/Unicode do i -ésimo caractere da chave;
- $p[i]$ é um inteiro de um conjunto de **pesos** gerados aleatoriamente para $0 \leq i \leq n - 1$.

Exercício

Insira as chaves {5, 28, 19, 15, 20, 33} em uma tabela de espalhamento T com 9 posições utilizando a função hash:

$$h(k) = k \bmod 9$$

	T	
0		
1	28	19
2	20	
3		
4		
5	5	
6	15	33
7		
8		

Função Geradora de Pesos

```
private int[] geraPesos(int n){  
    int p[] = new int[n];  
    java.util.Random rand = new java.util.Random();  
    for (int i = 0; i < n; i++){  
        p[i] = rand.nextInt(n) + 1;  
    }  
    return p;  
}
```

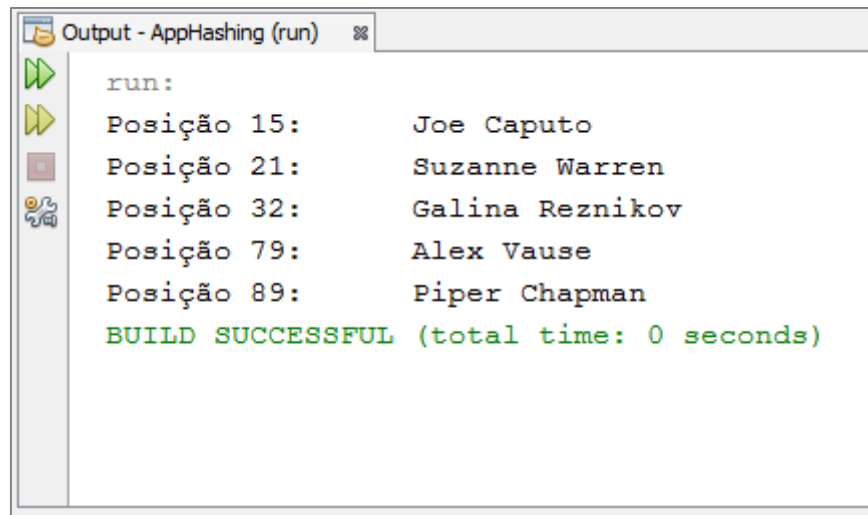
Função *K*

```
private int k(String chave, int[] pesos){  
    int soma = 0;  
    for (int i = 0; i < chave.length(); i++){  
        soma = soma + ((int)chave.charAt(i)) * pesos[i];  
    }  
    return soma;  
}
```


Função de Transformação

```
public int h(String chave, int n) {  
    int tamanhoChave = chave.length();  
    int[] pesos = geraPesos(tamanhoChave);  
    int k = k(chave, pesos);  
    return k % n;  
}
```

Exemplo 1: Resultado considerando 5 entradas de dados e uma tabela com 100 posições

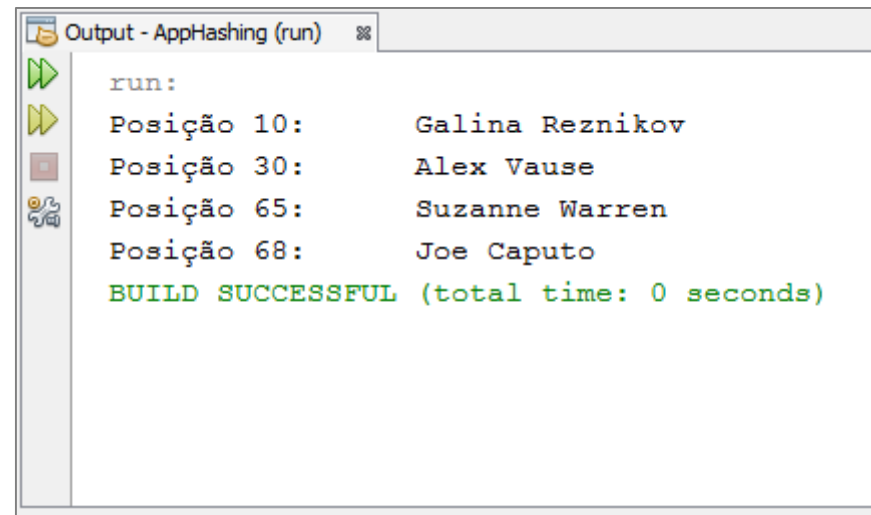


The screenshot shows an IDE output window titled "Output - AppHashing (run)". The output text is as follows:

```
run:
Posição 15:      Joe Caputo
Posição 21:      Suzanne Warren
Posição 32:      Galina Reznikov
Posição 79:      Alex Vause
Posição 89:      Piper Chapman
BUILD SUCCESSFUL (total time: 0 seconds)
```

The output window includes a vertical toolbar on the left with icons for running (green play), stepping (yellow play), stopping (red square), and debugging (bug icon).

Exemplo 2: Resultado considerando 5 entradas de dados e uma tabela com 100 posições



```
run:  
Posição 10:      Galina Reznikov  
Posição 30:      Alex Vause  
Posição 65:      Suzanne Warren  
Posição 68:      Joe Caputo  
BUILD SUCCESSFUL (total time: 0 seconds)
```

Um dado de entrada foi perdido. Por quê?

COLISÃO!

Colisões

ACONTECEM QUANDO

1. Duas ou mais chaves geram o mesmo endereço da tabela hash; ou
Não se pode garantir que as funções de hashing possuam um bom potencial de distribuição (espalhamento).
2. O vetor de dados tem mais chaves que o espaço disponível na tabela de espalhamento.
o número N de chaves possíveis é muito maior que o número de entradas disponíveis na tabela.

Tratamento de Colisões

As principais técnicas para o são:

- ❑ **Encadeamento:** a informação é armazenada em estruturas encadeadas fora da tabela de hash. Ou seja, manter as chaves em listas encadeadas que levam a um mesmo índice na tabela;
- ❑ **Endereçamento aberto:** utiliza os lugares vazios da tabela para problemas de colisão.

Atenção! *Devemos sempre ter um método para o tratamento de colisões, não importando a qualidade da função de hashing.*

Endereçamento Aberto

Quando o número de registros a serem armazenados na tabela puder ser previamente estimado, então não haverá necessidade de usar listas encadeadas.

Estratégia: *quando a função hash gera para uma chave uma posição que já está ocupada, o procedimento de armazenamento verifica se a posição seguinte também está ocupada; se estiver ocupada, verifica a posição seguinte e assim por diante, até encontrar uma posição livre.*

Quando usar?

- Existem menos registros a serem armazenados que o tamanho total da tabela de espalhamento.
- Se a tabela está cheia, o elemento não pode ser inserido.

Como encontrar uma **posição livre**?

A proposta mais simples é utilizando **hashing linear**.

Uma nova **posição** h_j na tabela será dada por:

$$h_j = (h(K) + j) \bmod N$$

$$1 \leq j \leq N - 1$$

Devo usar Hashing?

Vantagens

- O algoritmo é simples e eficiente nas operações de inserção e remoção;
- A busca é realizada em $O(1)$.

Desvantagens

- A tabela pode ficar esparsa para sempre (espaço subutilizado);
- O grau de espalhamento é sensível à função de hashing utilizada e ao tipo de informação usada como chave.

Aplicações no mundo

☐ Integridade de bancos de dados (SGBDs)

☐ Criptografia

☐ Compactação de dados (ZIP / RAR)

☐ Jogos

