



Conteúdo

1. Introdução
2. Listas
3. Pilhas e Filas
4. Árvores
5. Árvores de Pesquisa
 - Árvore Binária e Árvore AVL
 - Árvore N-ária e Árvore B
6. Tabelas de Dispersão (Hashing)
7. Métodos de Acesso a Arquivos
8. Métodos de Ordenação de Dados





Tabelas de Dispersão (Hashing)





Dicionário

- ⇒ Um dicionário armazena elementos que são pares chave-valor (ch,v).
- ⇒ A chave é usada como identificador para um objeto valor associado.
- ⇒ O valor são informações adicionais que podem ser acessadas somente através da chave.
- ⇒ Um dicionário permite que múltiplos elementos possam ter a mesma chave.





Mapeamento

⇒ Um mapeamento ou mapa armazena elementos que são pares chave-valor (ch,v).

⇒ A chave é usada como identificador único que é definido para um objeto valor associado.

↳ Um mapeamento requer que cada chave seja única.

⇒ Exemplos:

- mapeamento com estudantes: chave é a matrícula e o valor é o nome, endereço e curso do aluno.
- mapeamento com o números primos: o número pode ser a chave e também o valor.





Dicionário x Mapeamento

- ⇒ Dicionário: permite múltiplos elementos com a mesma chave.
- ⇒ Mapeamento: elementos devem ter chaves únicas.
 - ↳ mais apropriado em situações em que cada chave é para ser vista como um índice único para seu valor, ou seja, um objeto serve como um tipo de localização para um determinado valor.





Dicionário e Mapeamento

- ⇒ Distinguem-se dois tipos de dicionários e mapeamentos:
- ordenados: assume-se que uma relação de ordem total é definida para as chaves.
 - não-ordenados: não é assumida uma relação de ordem para as chaves.





Mapeamento

Exemplos de implementações de mapeamento:

- Estrutura Linear

Lista encadeada → inserção : $O(1)$

consulta : $O(n)$

Lista com Array → inserção : $O(1)$

consulta : $O(n)$

Lista com Array ordenada → inserção : $O(n)$

consulta : $O(\log n)$

- Árvore binária de pesquisa → inserção : $O(\log n)$

consulta : $O(\log n)$

- Tabela hash - mais eficiente!





Problema

⇒ Procura-se uma estrutura de dados para a qual as operações de inserção e consulta tenham performance $O(1)$.





Tabela Hash

- **Hashing:** Método de pesquisa com o objetivo de melhorar a performance de busca e inclusão de chaves

↳ os elementos devem ser localizados rapidamente usando chaves

⇒ As chaves são consideradas como “endereços” dos elementos.

⇒ Não é necessário que os elementos estejam ordenados.





Exemplo

⇒ Tabela de símbolos de um compilador

- Usada, durante a compilação, para referenciar os símbolos definidos pelo programador.
- Sempre que um símbolo é declarado, o compilador insere uma entrada na tabela de símbolos.
- Sempre que um símbolo é utilizado, o compilador verifica se os atributos associados com aquele símbolo estão sendo usados corretamente.
- O nome serve de “endereço” para propriedades sobre o tipo de uma variável ou seu valor.





Tabela Hash

Consiste de dois componentes principais:

➡ **Array de buckets**

↳ permite acesso direto a uma posição ($O(1)$)

➡ **Função hash** - $h(ch)$

↳ traduz um valor de chave para uma posição no array

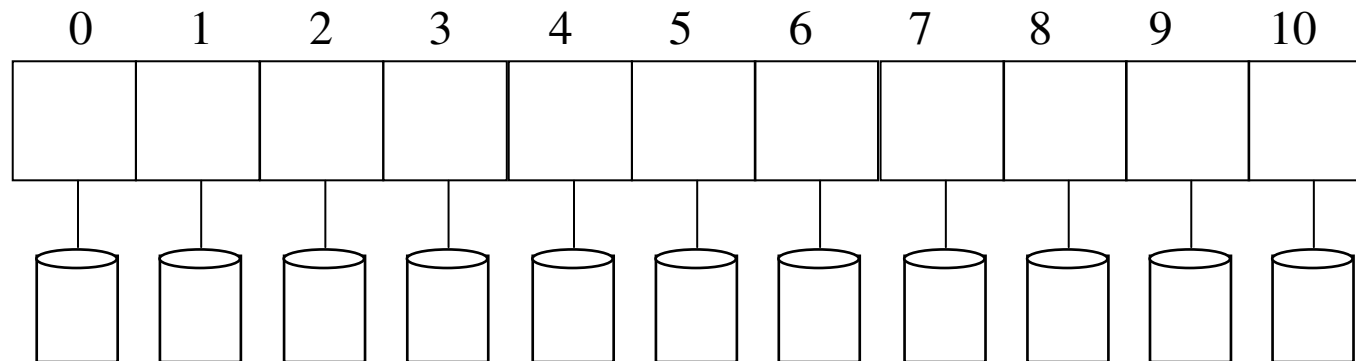
↳ deve ser fácil de computar: complexidade $O(1)$





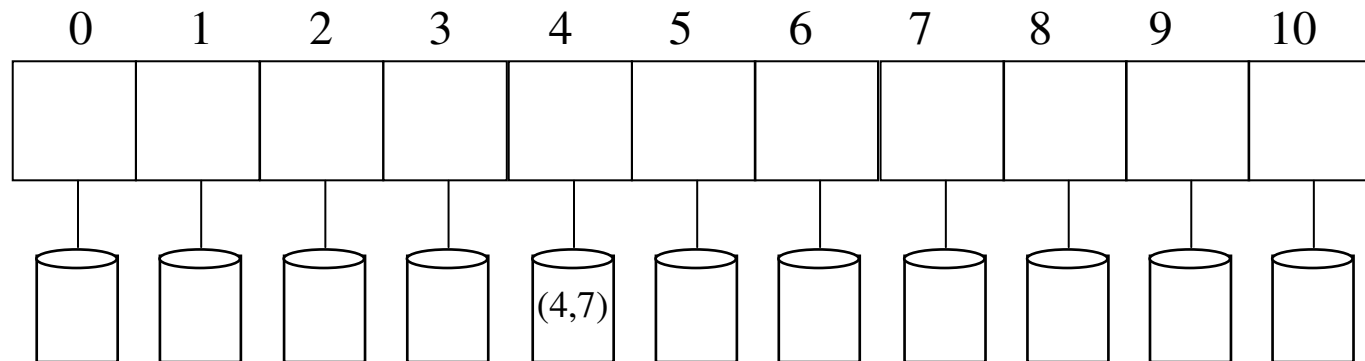
Array de Buckets

Um array de buckets é um array A de tamanho N , em que cada posição de A é considerada um “bucket” para pares chave-elemento.



Array de Buckets

- Se as chaves forem inteiros, o elemento com chave ch é simplesmente inserido no bucket $A[ch]$.



- ➡ o espaço utilizado é proporcional a N (chaves possíveis)
- ➡ as chaves precisam ser inteiros no intervalo $[0, N-1]$



Array de Buckets

➡ Tamanho do array: N

- Valor estimado: N é o número médio de chaves
- N deve ser menor que o número máximo possível de chaves ($N < n$)

Exemplo: chave é código do empregado → numérico com 4 dígitos

- **n**: 0000-9999 → máximo de 10000 chaves possíveis
- **N**: 2000 → média de empregados na empresa

↳ evitar desperdício de espaço





Função Hash

- Traduz um valor de chave para uma posição no array

$$h(ch) \rightarrow \{0, 1, \dots, N - 1\}$$

- O elemento é armazenado na posição $A[h(ch)]$
- Para uma mesma chave, é produzido sempre o mesmo resultado
- $h(ch)$ deve ser fácil de computar: complexidade $O(1)$



Função Hash

Exemplo:

$h(\text{num inscrição}) = \text{num inscrição} - 1$

chave = 4 $\rightarrow h(4) = 3$ (acesso direto) 

Tabela Hash

0	1	inscrição 1
1	2	inscrição 2
2	3	inscrição 3
3	4	inscrição 4
...
N-1	M	inscrição M



Função Hash

⇒ Se existirem duas ou mais chaves com o mesmo valor de hash, então dois diferentes elementos serão mapeados para o mesmo bucket em A.

$$h(ch_x) = h(ch_y)$$

↳ Colisão!

⇒ Diz-se que uma função hash é “boa” se o mapeamento das chaves minimiza o máximo o número de colisões.





Função Hash

A função hash ($h(ch)$) possui 2 partes:

➡ **Código hash:** mapeamento da chave ch para um inteiro

$f(ch)$: converte ch para uma representação numérica

➡ **Função de compressão:** mapeamento do código hash para um inteiro no intervalo $[0, N-1]$ do array.

$g(ch)$: converte $f(ch)$ para o intervalo $[0, \dots, M-1]$

➡ $h(ch) = g(f(ch))$





Função Hash - Código Hash

⇒ Código hash em Java

- No Java, a classe Object possui o método hashCode() que retorna um int.
- Diferentes objetos possuem diferentes hash codes.

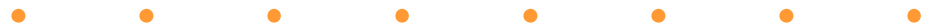




Função Hash - Código Hash

⇒ Conversão para inteiros

- Tipos de dados básicos são convertidos para inteiros.
- Exemplos: tipos Java **byte**, **short**, **int**, **char** e **float** são convertidos para o tipo **int**.



Função Hash - Código Hash

➡ Soma dos Componentes

- Usado com chaves numéricas grandes.
- Soma a representação inteira dos bits de um número
 - número visto como uma k-tupla $(x_0, x_1, \dots, x_{k-1})$ de inteiros
 - código de hash = $\sum_{i=0}^{k-1} x_i$
- Exemplo: *ch* é o RG e existem, no máximo, 50 empregados na empresa

Código hash: somar os dígitos

RG: 0000000000-9999999999

0x10 a 9x10 → [0, 90]

Função Hash - Código Hash

➡ Soma dos códigos numéricos dos caracteres

- Usado com chaves não-numéricas.
- Soma dos códigos numéricos (valores ASCII ou Unicode) dos caracteres
- Exemplo 1: soma de todos os caracteres

A N A S I L V A

$$f(ch) = \text{int}(A) + \text{int}(N) + \dots + \text{int}(V) + \text{int}(A)$$

- Exemplo 2: soma de alguns caracteres

A N A _ S I L V A (soma 1º, 5º e 8º)

$$f(ch) = \text{int}(A) + \text{int}(S) + \text{int}(V)$$

$$\text{se } \text{int}(ch) = \text{ASCII}(ch) \Rightarrow \text{máximo: } 3 \times \text{ASCII}(z) = 3 \times 122 = 366$$

Função Hash - Código Hash

⇒ Códigos hash polinomial

- Usado em chaves não-numéricas:
- Soma dos códigos numéricos dos caracteres levando em conta a posição dos elementos x_i :

$$x_0 a_{k-1} + x_1 a_{k-2} + \dots + x_{k-2} a + x_{k-1}$$



Função Hash

- ⇒ **Código hash**: mapeamento da chave ch para um inteiro
- ⇒ **Função de compressão**: mapeamento do código hash para um inteiro no intervalo $[0, N-1]$ do array.
 - ↳ uma boa função de compressão é uma que minimiza o número de colisões



Função Hash - Compressão

⇒ Método de Divisão:

- Mapeia um inteiro i para uma posição do array de tamanho N :

$$|i| \bmod N$$

- A escolha de N como um número primo ajuda a “espalhar” a distribuição dos valores → poucas divisões exatas

- Exemplo:

$$n = 2500 \qquad M = 1031 \qquad h(ch) = ch \% M$$

$$h(23) = 23 \qquad h(1401) = 370$$

$$h(24) = 24 \qquad h(1402) = 371$$

...

...

Função Hash - Compressão

⇒ Método de Multiplicação, Adição e Divisão (MAD)

- Mapeia um inteiro i para uma posição do array de tamanho N :

$$|ai + b| \bmod N$$

Onde:

- N é um número primo
- $a > 0$ e $b \geq 0$ são constantes inteiras escolhidas aleatoriamente



Tabela Hash - Implementação

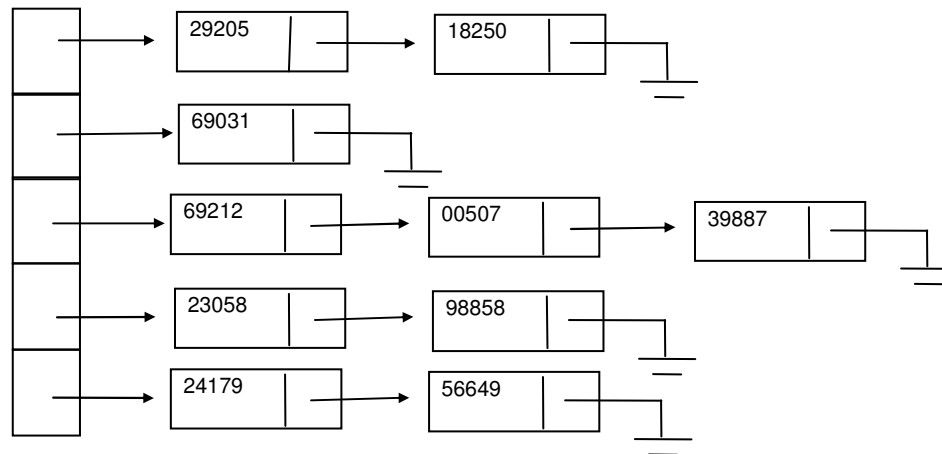
- Se cada bucket só puder possuir 1 elemento, é necessário tratar a colisão
- Algumas alternativas de implementação de acordo com o tratamento de colisões:
 - ➡ Encadeamento Externo ou Separado: todas as chaves mapeadas para a mesma posição estarão juntas em uma lista encadeada.
 - ➡ Encadeamento Interno ou Aberto: ocorrendo a colisão, a chave é colocada na próxima posição livre do array.





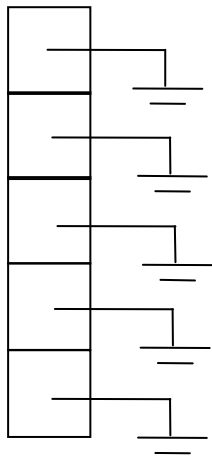
Encadeamento Externo

- ➡ Cada posição $A[i]$ possui uma referência para uma lista encadeada que contém todos os elementos (ch, v) tais que $h(ch) = i$.
- Para inserir um elemento na tabela, coloca-se o elemento no final de alguma das listas encadeadas.

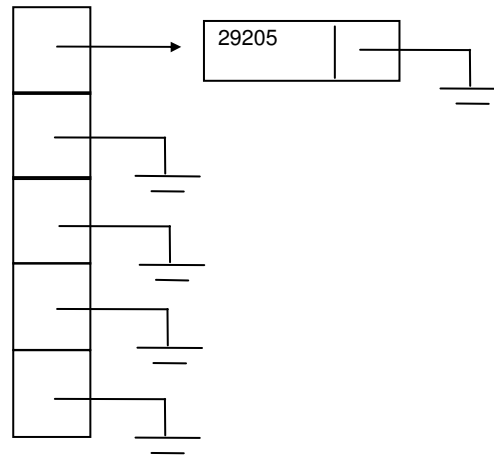


Encadeamento Externo - Exemplo

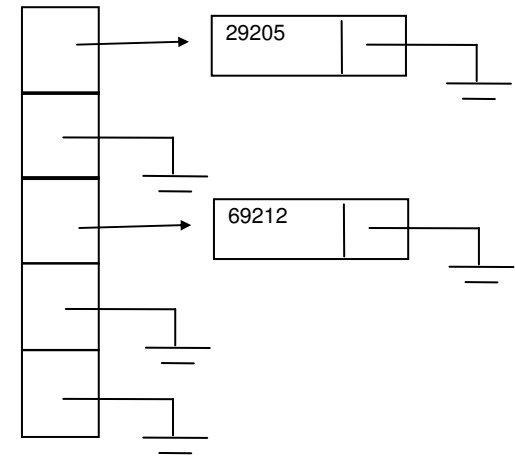
início



entra 29205



entra 69212



chave	$hash(chave)$
-------	---------------

29205	0
-------	---

69212	2
-------	---

$$hash(chave) = chave \text{ MOD } 5$$

Encadeamento Externo - Exemplo

entra 18250

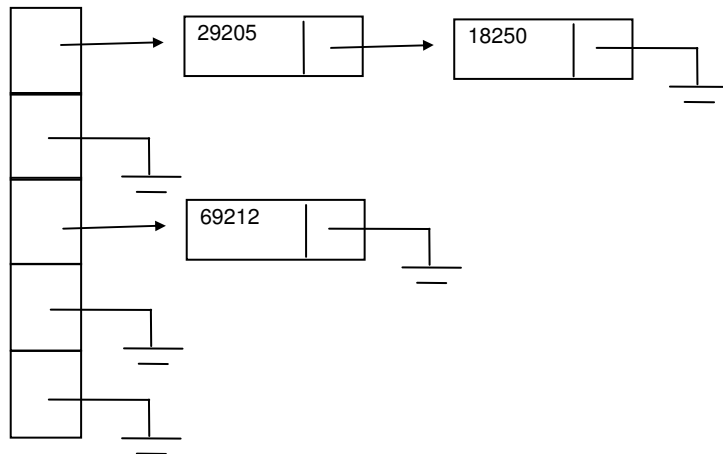
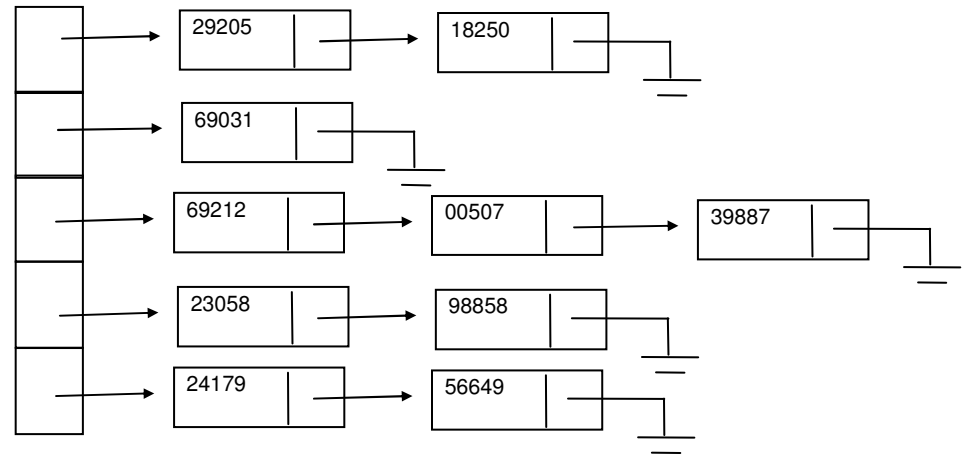


tabela completa



chave *hash* (chave)

18250	0	24179	4
69031	1	39887	2
23058	3	98858	3
00507	2	56649	4

$$\text{hash (chave)} = \text{chave MOD } 5$$



Encadeamento Externo

- Complexidade das Operações

- Inserção de elemento

$O(1)$

- Exclusão de elemento

$O(n)$: pior caso

$O(n/N)$: caso médio

- Pesquisa de elemento

$O(n)$: pior caso

$O(n/N)$: caso médio

- Considerando que n/N seja menor do que 1 e que tenhamos uma boa função de hash, o tempo de execução esperado das operações de inserção, exclusão e pesquisa é $O(1)$.

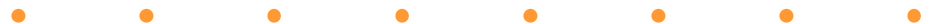




Encadeamento Externo

Desvantagem:

- requer o uso de uma estrutura de dados auxiliar para armazenar os elementos com colisões.



Encadeamento Interno

⇒ Cada elemento é armazenado diretamente no array, com um elemento em cada posição.


- No caso de ocorrer uma colisão, procura-se em uma seqüência de posições da própria tabela através de alguma estratégia:
 - teste linear
 - teste quadrático
 - hashing duplo
 - outra função para calcular o incremento do novo endereço
- Quando o endereço produzido pela função hash está ocupado por uma chave que não é sinônimo, ocorre uma pseudo-colisão.

Encadeamento Interno

➡ Teste linear

A chave que colidiu é armazenada na primeira entrada livre a partir do endereço calculado, ou seja, se $A[h(ch)]$ estiver ocupada, tenta-se em $A[h(ch)+1]$, e assim por diante.

chave	hash(chave)		
JOAO	5		0
MARIA	1		1
FERNANDO	5		2
CLAUDIA	0		3
MARCIA	7		4
JORGE	6		5
PEDRO	1		6
			7



	CLAUDIA
	MARIA
	JORGE
	PEDRO
	JOAO
	FERNANDO
	MARCIA



Encadeamento Interno

⇒ Teste quadrático

Envolve o teste de posições $A[(i + f(j)) \bmod N]$, para $j = 0, 1, 2, \dots$

onde $f(j) = j^2$, até que seja achada uma posição vazia.





Encadeamento Interno

⇒ Hashing duplo

Escolhe-se uma função de hash secundária h'

Se a posição $A[h(ch)]$ já está ocupada, então tenta-se as posições

$A[(h(ch) + f(j)) \bmod N]$ para $j = 1, 2, 3, \dots$ onde $f(j) = j \cdot h'(ch)$





Encadeamento Interno

Complexidade das Operações

→ Inserção de elemento

$O(n)$: pior caso

→ Exclusão de elemento

$O(n)$: pior caso

→ Pesquisa de elemento

$O(n)$: pior caso





Rehashing

⇒ Fator de carga = n/N

↳ deseja-se um fator de carga mantido abaixo de 1.

Se o fator de carga de uma tabela de hash está significativamente acima do limite:

- a tabela é redimensionada e todos os elementos são inseridos na nova tabela.
- geralmente o tamanho da nova tabela é o dobro do tamanho da tabela anterior.





Tabela Hash - Interface

```
public interface TabelaHash<C,V> extends EstruturaDeDados{  
    public void insere (C chave, V valor);  
    public V retorna (C chave);  
    public V remove (C chave);  
    public boolean contem (C chave);  
    public Iterator<C> retornaChaves ();  
    public Iterator<V> retornaValores ();  
}  
  
public interface EstruturaDeDados {  
    public boolean estaVazia();  
    public void esvazie();  
    public int numeroElementos();  
}
```





Implementação

Implemente uma tabela hash utilizando encadeamento externo.

Observações:

- Esta classe deverá implementar a interface TabelaHash.java.
- O tamanho da tabela deverá ser passado como parâmetro no construtor.
- O encadeamento externo deverá ser implementado com uma lista encadeada.
- A função hash poderá ser calculada como o resto da divisão do hashCode da chave pelo tamanho do array.
- A função hashCode() do Java pode retornar um valor negativo...





Classe TabelaHashEncadeamentoExterno

```
public class TabelaHashEncadeamentoExterno<C,V> implements
    TabelaHash<C,V> {

    private ListaEncadeada<Associacao<C,V>>[] tabela;

    private int M; //tamanho do array tabela

    private int numElementos;

    public TabelaHashEncadeamentoExterno(int tamanhoTabela) {
        if (tamanhoTabela < 1)
            throw new RuntimeException ("tamanho invalido");
        else {
            tabela = new ListaEncadeada[tamanhoTabela];

            ...
        }
    }
}
```



Exercício

Implemente o seguinte problema: a partir de uma frase, conte o número de ocorrências das palavras que aparecem na frase. Utilize a tabela hash para armazenar a quantidade de ocorrência de cada palavra.

Exemplo: os testes de unidade e os testes de módulo normalmente são de responsabilidade dos programadores que desenvolveram o componente

os : 2	normalmente : 1	desenvolveram : 1
testes: 2	são : 1	o : 1
de : 3	responsabilidade : 1	componente : 1
unidade : 1	dos : 1	
e : 1	programadores : 1	
módulo : 1	que : 1	