

## Aula 15

### *Engenharia da Computação – 3ª série*

## Hashing (E1, E2)

**2023**

### Exercícios



#### Parte A: Hashing Interno sem Tratamento de Colisão

Considere o código **Java** a seguir, que representa uma Classe para **Alunos**:

```
public class Aluno {  
  
    private Integer codAluno;  
    private String nome;  
  
    public Aluno() { }  
  
    public Aluno(Integer codAluno, String nome) {  
        this.codAluno = codAluno;  
        this.nome = nome;  
    }  
  
    public Integer getCodAluno() {  
        return codAluno;  
    }  
  
    public void setCodAluno(Integer codAluno) {  
        this.codAluno = codAluno;  
    }  
  
    public String getNome() {  
        return nome;  
    }  
  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
}
```

## Hashing

### Exercícios



#### Parte A: Hashing Interno sem Tratamento de Colisão

Considere, também, o código **Java** a seguir (1/3), que carrega **10 instâncias** da classe **Aluno** em um **array** chamado **tabAluno**:

```
public class Hash_01 {  
  
    public static void main(String[] args) {  
  
        Aluno[] tabAluno = new Aluno[10];  
  
        tabAluno[0] = new Aluno(10, "Ana");  
        tabAluno[1] = new Aluno(21, "Silas");  
        tabAluno[2] = new Aluno(22, "Ari");  
        tabAluno[3] = new Aluno(24, "Pedro");  
        tabAluno[4] = new Aluno(35, "Jonas");  
        tabAluno[5] = new Aluno(60, "Saul");  
        tabAluno[6] = new Aluno(44, "Josue");  
        tabAluno[7] = new Aluno(57, "Paulo");  
        tabAluno[8] = new Aluno(80, "Sara");  
        tabAluno[9] = new Aluno(90, "Davi");  
  
        Integer hashCode = null, chave;  
        Aluno[] tabHash = new Aluno[10];  
        for (int i=0; i<tabAluno.length; i++ ) {  
            chave = (tabAluno[i].getCodAluno());  
            hashCode = hash(chave);  
            System.out.println("Chave = " + chave +  
                               " mapeada para hascode = " + hashCode);  
        }  
    }  
}
```

### Exercícios



#### Parte A: Hashing Interno sem Tratamento de Colisão

Considere, também, o código **Java** a seguir (2/3), que carrega **10 instâncias** da classe **Aluno** em um **array** chamado **tabAluno**:

```
        if (tabHash[hashCode] == null )
            tabHash[hashCode] = tabAluno[i];

        else {
            System.out.println("** Colisao no slot da Tabela Hash ** ");
            System.out.println("Chave " + tabAluno[i].getCodAluno() +
                " NAO ARMazenada na Tabela Hash ...\n" );
        }
    }
    System.out.println("\nTabela Aluno: ");
    System.out.println("-----");
    for (int i = 0 ; i < tabAluno.length; i++)
        System.out.print ("Slot " + i + " ---> " + tabAluno[i].getCodAluno()
            + " " + tabAluno[i].getNome() + '\n' ) ;

    System.out.println("\nTabela HASH: ");
    System.out.println("-----");
    for (int i = 0 ; i < tabHash.length; i++)
        if (tabHash[i] == null)
            System.out.println("Slot " + i + " ---> Valor nulo");
        else
            System.out.print ("Slot " + i + " ---> " +
                tabHash[i].getCodAluno() + " " + tabHash[i].getNome() + '\n' ) ;
    }
```

## Hashing

### Exercícios



#### Parte A: Hashing Interno sem Tratamento de Colisão

Considere, também, o código **Java** a seguir (3/3), que carrega **10 instâncias** da classe **Aluno** em um **array** chamado **tabAluno**:

```
// -----  
public static Integer hash(Integer key) {  
    }  
}
```

### Exercícios



#### Parte A: Hashing Interno sem Tratamento de Colisão

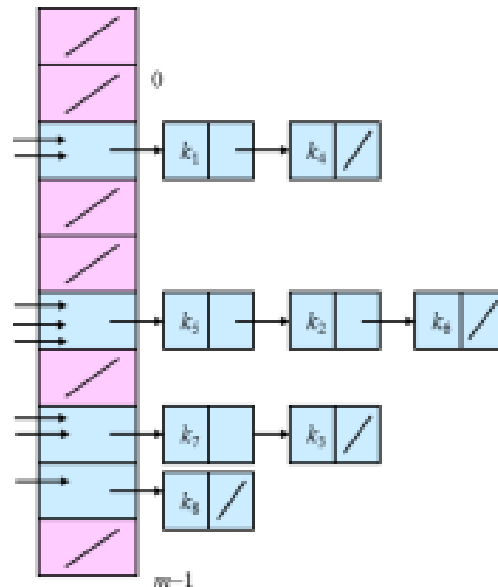
1. Completar o método ***hash()*** no programa, para isso, utilizando a função ***hash***  $h = \text{mod}(n)$ ;
2. Executar o código e avaliar a sua execução;
3. Responda: Houve colisões? Em caso de afirmativo, quantas e quais colisões ocorreram? Justifique!
4. Como foi feito o tratamento de colisões?
5. Que sugestões você apresentaria para o tratamento das colisões?

### Exercícios



#### Parte B: Hashing Interno com Tratamento de Colisão – Encadeamento

Considere uma aplicação que utiliza **20 chaves**, com valores de 0 até 19. Para essa aplicação será construída uma **tabela Hash** com **10 elementos**. Será empregada uma função **hash** definida pelo **método da divisão** e o tratamento de colisões será feito pelo **método do encadeamento** (listas ligadas irão absorver os elementos de colisão).





### Exercícios



#### Parte B: Hashing Interno com Tratamento de Colisão – Encadeamento

1. Armazenando num mesmo pacote, escreva uma classe chamada **TestHash** e a classe **SList** referente à implementação de **Listas Simplesmente Encadeadas**. A classe **TestHash** deve conter o método **main()** da aplicação a ser executada;
2. No método **main()**, definir um **array** chamado **tabKeys** com capacidade para armazenar **20 chaves**. Cada chave corresponde a um valor **inteiro**. Desconsiderar a primeira posição do **array**, visto que a aplicação irá considerar **chaves válidas** entre **1 e 19** no intervalo de chaves da aplicação. Em cada posição do **array**, deve estar armazenado o valor correspondente da chave (**array associativo**).



### Exercícios



#### Parte B: Hashing Interno com Tratamento de Colisão – Encadeamento

3. No método ***main()***, definir um **array** chamado **tabHash** com capacidade para armazenar **10 chaves**. Em cada posição do **array**, deve estar armazenado o endereço de uma lista ligada que conterá a chave retornada pelo método ***hash()*** com as suas respectivas colisões. Inicializar essa tabela com listas ligadas inicialmente vazias (referências às listas devem ter valores ***null***);
4. Escrever o código do método ***hash()***:

```
public static Integer hash(Integer key) {  
    return (key % 10);  
}
```

### Exercícios



#### Parte B: Hashing Interno com Tratamento de Colisão – Encadeamento

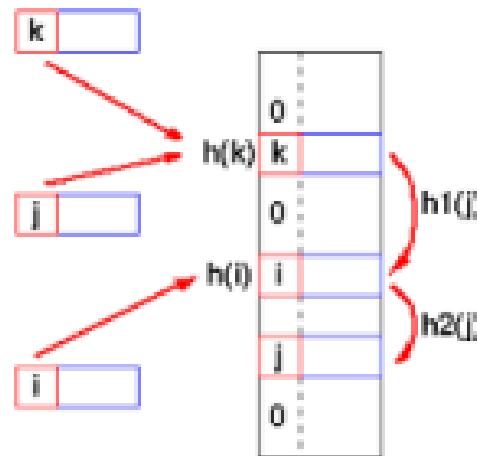
5. No método **main()**, escrever o código para a carga da Tabela **Hash**. Em cada posição de **tabHash**, deverá ser inserida a chave retornada pelo método **hash()**. Para inserção da chave em **tabHash**, chamar o método **inserirInicio()** existente na classe **SList** correspondente às listas ligadas;
6. No método **main()**, escrever o código para imprimir em cada posição de **TabHash**, a lista com as chaves armazenadas (colisões);
7. Modificar o exercício, para um total de **100.000 chaves**, com valores de **1 a 99.999**. Para esse universo de chaves, considerar a tabela **hash** com capacidade para armazenar **1000 chaves**.

### Exercícios



#### Parte C: Hashing Interno com Tratamento de Colisão – *Rehashing*

Considere uma aplicação que utiliza uma tabela **hash** para armazenar empregados de uma grande rede de empresas, com milhares de Empregados. Considere que a tabela **hash**, a ser criada em memória, terá capacidade para **10 empregados** e irá armazenar apenas o código do empregado, ou seja, sua **chave**. Será empregada uma **função hash** definida pelo método da divisão e o tratamento de colisões será feito pelo método do endereçamento aberto, ou **rehashing**.



### Exercícios



#### Parte C: Hashing Interno com Tratamento de Colisão – Rehashing

1. Escrever uma classe **TestHash**, com o método **main()** para execução do código. O método deve, inicialmente, criar a **tabela hash**, representada por um **array** chamado **tabHash**, que irá conter as chaves dos empregados;
2. Considere inicialmente as chaves 23, 45, 77, 11, 33, 49, 10, 4, 89 e 14, que deverão ser carregadas na **tabela hash**;
3. O método **main()** terá o seguinte código inicial:

```
public static void main(String[] args) {  
  
    Integer[] tabChaves = new Integer[] { 23, 45, 77, 11, 33, 49, 10, 4, 89, 14} ;  
  
    Integer[] tabhash = new Integer[10];  
}
```

### Exercícios



#### Parte C: Hashing Interno com Tratamento de Colisão – Rehashing

4. Escrever o método **hash()**, que receberá uma chave como parâmetro e retornará o índice correspondente a essa chave na **tabela hash**. Considerar o método da divisão para a escrita do código da função **hash**:

*Integer indiceHash = hash(codigoEmpregado);*

5. Escrever o método **rehashing()**, que recebe como parâmetro o endereço da **tabela hash** e a chave de colisão. O método **rehashing()** deverá percorrer a **tabela hash** passada como parâmetro e retornar a primeira posição da **tabela hash** que esteja livre para armazenar a chave. Caso a tabela não tenha índices livres, retornar **null**;
6. Uma vez conhecido o índice da **tabela hash** correspondente ao empregado passado como parâmetro, no método **main()** proceder à gravação da chave na **tabela hash** no índice retornado pela função **hash**. Caso a posição da **tabela hash** já estiver preenchida com outra chave, recalcular o índice por meio da chamada do método **rehashing()**.

### Exercícios



#### Parte C: Hashing Interno com Tratamento de Colisão – Rehashing

6. Uma vez conhecido o índice da **tabela hash** correspondente ao empregado passado como parâmetro, no método **main()** proceder à gravação da chave na **tabela hash** no índice retornado pela função **hash**. Caso a posição da **tabela hash** já estiver preenchida com outra chave, recalcular o índice por meio da chamada do método **rehashing()**. Proceder à gravação de todas as chaves e imprimi-las:

```
public static Integer rehashing(Integer[] tabhash, Integer indice) {  
    for (Integer i = indice + 1 ; i < tabhash.length ; i ++ ) {  
        if (tabhash[i] == null )  
            return i;  
    }  
    for (Integer i = 0 ; i < indice ; i++ ) {  
        if (tabhash[i] == null )  
            return i;  
    }  
    return null;  
}
```

# ECM306 – Tópicos Avançados em Estrutura de Dados

## Referências bibliográficas



- CORMEN, T.H. et al. Algoritmos: Teoria e Prática (Caps. 13). Campus. 2002.
- ZIVIANI, N. Projeto de algoritmos: com implementações em Pascal e C (Cap. 1). 2.ed. Thomson, 2004.
- FEOFILOFF, P. Minicurso de Análise de Algoritmos, 2010. Disponível em:  
<http://www.ime.usp.br/~pf/livrinho-AA/>
- DOWNEY, A.B. *Analysis of algorithms* (Cap. 2), Em: *Computational Modeling and Complexity Science*. Disponível em:  
<http://www.greenteapress.com/compmod/html/book003.html>
- ROSA, J.L. Notas de Aula de Introdução a Ciência de Computação II. Universidade de São Paulo. Disponível em:  
<http://coteia.icmc.usp.br/mostra.php?ident=639>



# ECM306 – Tópicos Avançados em Estrutura de Dados

## Referências bibliográficas

- GOODRICH, Michael T. et al: *Algorithm Design and Applications*. Wiley, 2015.
- LEVITIN, Anany. *Introduction to the Design and Analysis of Algorithms*. Pearson, 2012.
- SKIENA, Steven S. *The Algorithm Design Manual*. Springer, 2008.
- Série de Livros Didáticos. *Complexidade de Algoritmos*. UFRGS.
- BHASIN, Harsh. *Algorithms – Design and Analysis*. Oxford University Press, 2015.
- FREITAS, Aparecido V. de – 2022 – Estruturas de Dados: Notas de Aula.
- CALVETTI, Robson - 2015 – Estruturas de Dados: Notas de Aula.



# ECM306 – Tópicos Avançados em Estrutura de Dados

## Aula 15

FIM