

Banco de Dados – Tarefa T12

Hashing Interno

Parte A – Hashing Interno sem Tratamento de Colisão

1. Considere o código abaixo escrito na Linguagem Java, que representa uma Classe para criar Alunos:

```
package maua;

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;
    }
}
```

2. Considere o código abaixo escrito na Linguagem Java, que carrega 10 instâncias da classe Aluno em um array chamado **tabAluno**. O programa também cria uma tabela Hash (chamada **tabHash**) com 10 elementos e efetua o mapeamento por meio de uma função de Hashing de **tabAluno** para **tabHash**.

```
package maua;

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);
    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')
        ;
}
// -----
public static Integer hash(Integer key) {

}

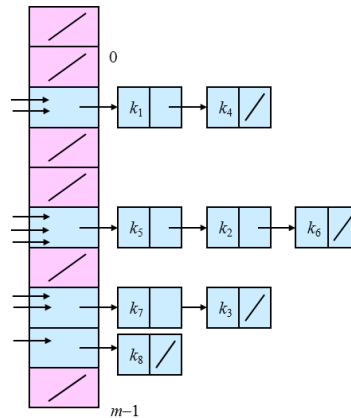
}

```

3. Complementar a função hash no programa. Utilizar a função hash $h = \text{mod}(n)$.
4. Executar o código e avaliar a sua execução.
5. Houve colisões? Em caso afirmativo, quantas e quais colisões ocorreram?
6. Como foi feito o tratamento das colisões?
7. Que sugestões você apresentaria para o tratamento das colisões?

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).

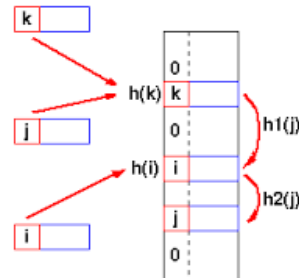


1. Escreva uma classe chamada **TestHash** num package chamado **maua**. Nesse package também estará armazenada a classe **SList** referente à implementação de Listas Simplesmente Encadeadas. A classe **TestHash** deve conter a função **main()** da aplicação a ser executada.
2. Na função **main()**, definir um array chamado **tabKeys** com capacidade para armazenar **20** chaves. Cada chave corresponde a um valor inteiro e portanto o tipo deve ser **Integer**. Desconsiderar a primeira posição do array, visto que a aplicação irá considerar chaves válidas no intervalo de chaves da aplicação varia de 1 a 19. Em cada posição do array, deve estar armazenado o valor correspondente da chave (array associativo).
3. Na função **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 pela função **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 da função **Hash**:

```
public static Integer hash(Integer key) {  
    return (key % 10);  
}
```
5. Na função **main()**, escrever o código para a carga da Tabela Hash. Em cada posição de **tabHash**, deverá ser inserida a chave retornada pela função **hash()**. Para inserção da chave em **tabHash**, chamar a função **Inserelnicio()** existente na classe **SList** correspondente às listas ligadas.
6. Na função **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.

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 (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**.



1. Escrever uma classe **TestHash**, contida no package **maua**, com a função **main()** para execução do código. A função deve inicialmente, criar a tabela hash, representada por um array chamado **tabhash** que irá conter as chaves dos empregados.
2. Considere que inicialmente as chaves 23, 45, 77, 11, 33, 49, 10, 4, 89, 14 deverão ser carregadas na tabela hash.
3. A função **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];  
}
```

4. Escrever a função **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 a função **rehashing** que recebe como parâmetro o endereço da tabela hash e a chave de colisão. A função **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, na função **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, recalculer o índice por meio da chamada da função **rehashing**. Proceder à gravação de todas as chaves e imprimí-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;  
}
```