João Guilherme Martins Jatobá
18.01790-8
Tópicos Avançados em Estruturas de Dados - Atividade 17

Parte A

3-)

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

4-)

Chave = 10 mapeada para hascode = 0
Chave = 21 mapeada para hascode = 1
Chave = 22 mapeada para hascode = 2
Chave = 24 mapeada para hascode = 4
Chave = 35 mapeada para hascode = 5
Chave = 60 mapeada para hascode = 0
** Colisao no slot da Tabela Hash **
Chave 60 NAO ARMAZENADA NA TABELA HASH ...

Chave = 44 mapeada para hascode = 4
** Colisao no slot da Tabela Hash **
Chave 44 NAO ARMAZENADA NA TABELA HASH ...

Chave = 57 mapeada para hascode = 7
Chave = 80 mapeada para hascode = 0
** Colisao no slot da Tabela Hash **
Chave 80 NAO ARMAZENADA NA TABELA HASH ...

Chave = 90 mapeada para hascode = 0
** Colisao no slot da Tabela Hash **
Chave 90 NAO ARMAZENADA NA TABELA HASH ...

Tabela Aluno:

Slot 0 ---> 10 Ana
Slot 1 ---> 21 Silas
Slot 2 ---> 22 Ari
Slot 3 ---> 24 Pedro
Slot 4 ---> 35 Jonas
Slot 5 ---> 60 Saul
Slot 6 ---> 44 Josue
Slot 7 ---> 57 Paulo
Slot 8 ---> 80 Sara
Slot 9 ---> 90 Davi

Tabela HASH:

Slot 0 ---> 10 Ana
Slot 1 ---> 21 Silas
Slot 2 ---> 22 Ari
Slot 3 ---> Valor nulo
Slot 4 ---> 24 Pedro
Slot 5 ---> 35 Jonas
Slot 6 ---> Valor nulo
Slot 7 ---> 57 Paulo
Slot 8 ---> Valor nulo
Slot 9 ---> Valor nulo

5-)
Houve algumas colisões, que aconteceram quando o dígito unitário do códigos dos alunos eram iguais.
Aconteceram 4 colisões, com os números 10-60, 24-44, 10-80 e 10-90.

6-)
O tratamento foi simples e ineficaz: os números que vinham depois, e colidiram, eram eliminados da tabela, logo há muitos espaços que ficaram sem valor atribuído na tabela hash.

7-)
Quando ocorrerem colisões, pode-se colocar o último valor na próxima casa da tabela hash que está livre.
Pode-se também criar uma tabela para cada índice da tabela hash, logo vários valores podem estar no mesmo índice. Com isso não haverá mais colisões.

****

Parte B

```
package maua.parte_B;

import java.util.List;

public class SList {

   public static void insereInicio(List<Integer> teste,Integer key){
        teste.add(key);
}
}
```

```
package maua.parte_B;

import java.util.*;
```

```java
public class TestHash {
    public static void main(String[] args) {
        System.out.println("Quer com 20 chaves (1) ou 100.000 (2)?");
        Scanner sc = new Scanner(System.in);
        int opcao = sc.nextInt();
        boolean flag = true;
        while (flag) {
            switch (opcao) {
                case 1:
                    metodo(10);
                    flag = false;
                    break;
                case 2:
                    metodo(1000);
                    flag = false;
                    break;
                default:
                    System.out.println("Escolher 1 ou 2.");
                    opcao = sc.nextInt();
                    break;
            }
        }
    }
    public static void metodo(int tamanho){
        int chaves;
        if(tamanho==10){chaves = 20;}
        else{chaves=100000;}

        List<Integer> tabKeys = Arrays.asList(new Integer[chaves]);
        for (int i=1;i< tabKeys.size();i++){
            tabKeys.set(i, i);
        }
        System.out.println();

        List<List<Integer>> tabHash = new ArrayList<>(tamanho);
        for (int j=0;j< tamanho;j++){
            tabHash.add(new LinkedList<>());
        }

        System.out.println();
        for (int m=1;m< tabKeys.size();m++){
            SList.insereInicio(tabHash.get(hash(m,tamanho)),tabKeys.get(m));
        }
        for (List<Integer> hash : tabHash) {

            System.out.println(hash);
        }
    }


    public static Integer hash(Integer key,int tamanho) {
```

```
        return (key % tamanho);
    }
}
```

****

Parte C

```
package maua.parte_C;

public class TestHash {
    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];
        for (int i=0; i<tabChaves.length; i++ ) {
            Integer chave = tabChaves[i];
            Integer indiceHash = hash(chave);
            System.out.println("Chave = " + chave +
                    " mapeada para hascode = " + indiceHash);
            if (tabHash[indiceHash] == null )
                tabHash[indiceHash] = tabChaves[i];
            else {
                System.out.println("** Colisao no slot da Tabela Hash ** "
);
                tabHash[rehashing(tabHash,indiceHash)] = tabChaves[i];
            }

        }
        for (int i = 0 ; i < tabHash.length; i++)
            System.out.print ("Slot " + i + " ---> " +
                        tabHash[i]+ '\n');
    }
    public static Integer hash(Integer key) {
        return key%10;
    }

    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;
    }

}
```

//

Chave = 23 mapeada para hascode = 3
Chave = 45 mapeada para hascode = 5
Chave = 77 mapeada para hascode = 7
Chave = 11 mapeada para hascode = 1
Chave = 33 mapeada para hascode = 3
** Colisao no slot da Tabela Hash **
Chave = 49 mapeada para hascode = 9
Chave = 10 mapeada para hascode = 0
Chave = 4 mapeada para hascode = 4
** Colisao no slot da Tabela Hash **
Chave = 89 mapeada para hascode = 9
** Colisao no slot da Tabela Hash **
Chave = 14 mapeada para hascode = 4
** Colisao no slot da Tabela Hash **
Slot 0 ---> 10
Slot 1 ---> 11
Slot 2 ---> 89
Slot 3 ---> 23
Slot 4 ---> 33
Slot 5 ---> 45
Slot 6 ---> 4
Slot 7 ---> 77
Slot 8 ---> 14
Slot 9 ---> 49