Aula 14 - Tabela hash em JAVA

Hashing é uma tabela composta por um vetor com M posições e cada posição representa um endereço onde os elementos a serem armazenado nele possuem um valor chave que é utilizado para calcular o endereço na tabela onde deverão ser alocados.
Para criação da função hashing é o método da divisão onde uma chave X é mapeada em M endereços da tabela hashing calculando o resto da divisão de X por M, ou seja:

h(x) = x mod m.

Exemplo:

Em uma tabela hashing de tamanho 10 e a chave inserida é 100 será armazenado na posição 0.

Ao aplicar a função de hashing sobre um conjunto de chaves, duas ou mais chaves podem ser mapeadas para o mesmo endereço, esta situação é chamada de colisão, para resolver este problema é utilizado o encadeamento dos endereços da tabela hashing ou se usa de outra alternativa, como a do endereçamento aberto.

Tabela hashing com endereçamento aberto.

Neste tipo de implementação todas as chaves são armazenada na própria tabela sem a necessidade de espaço extras ou ponteiros. Neste método é aplicado quando o número de chaves a serem armazenadas na tabela é reduzida e as posições vazias da tabela são utilizada para o tratamento de colisões.
Quando uma chave X é endereçada na posição H(X) e essa já está ocupada, outras posições vazias na tabela são procuradas para armazenar a chave X, caso nenhuma seja encontrada na tabela é dito que está totalmente preenchido e não pode mais armazenar dados.
Para fazer a busca por uma posição livre pode ser feita de duas maneiras: tentativa linear ou tentativa quadrática.

Tentativa linear

Quando um chave X deve ser inserida e ocorre uma colisão, a seguinte função é utilizada para obter um novo endereço:

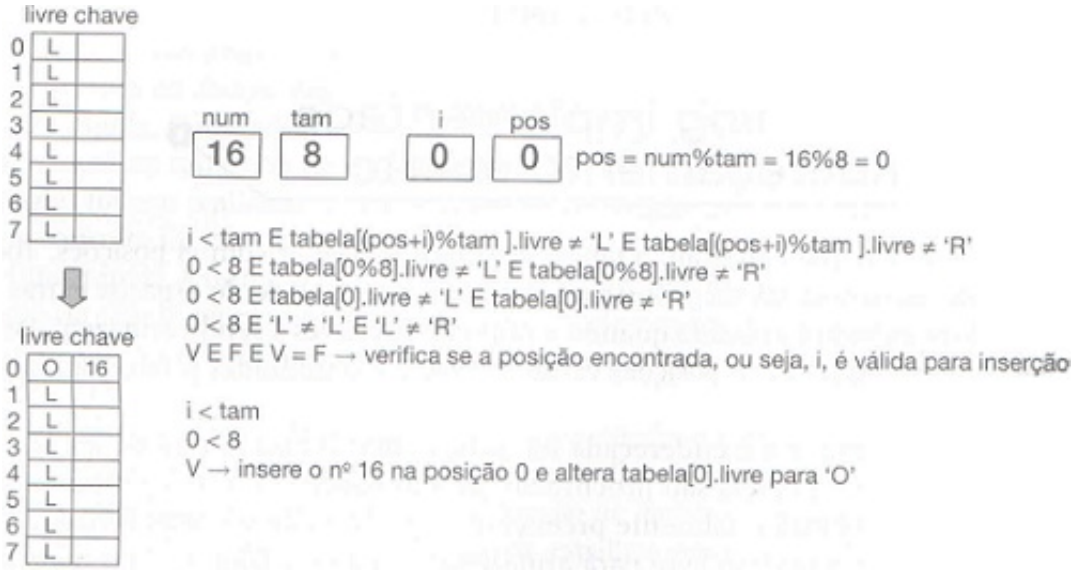
h(x) = (h(x)+j) mod m, 1<= j <= -1, h(x) = x mod m

Onde o objetivo é armazenar a chave no endereço consecutivo h(x)+1,h(x)+2, ... , até que encontrar uma posição vazia.
Quando for tentar remover uma chave X, não se pode remover de fato que irá perder a sequência de tentativas. Para isto cada endereço da tabela é marcado como livre (L), ocupado (O) ou removido (R). Assim uma nova chave poderá ocupar um posição marcada como removido.

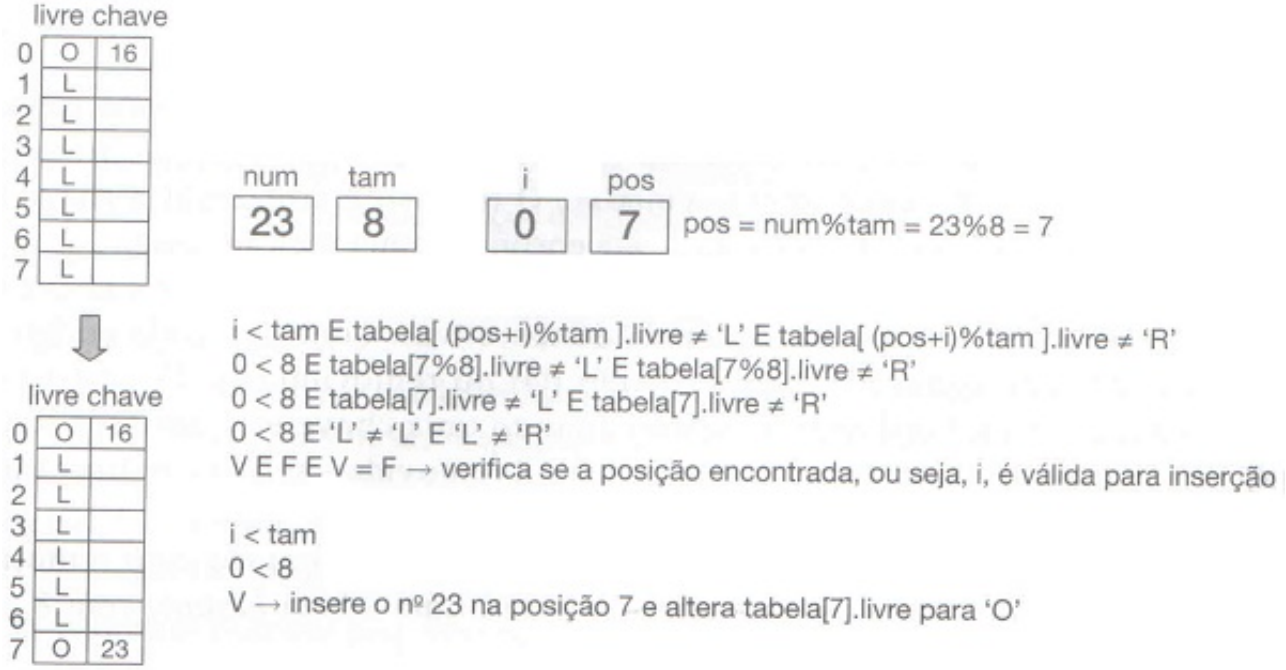
Exemplo:

A baixo segue uma ilustração como seria a inserção dos números, 16, 23, 25, e 41 e depois a remoção do número 41.

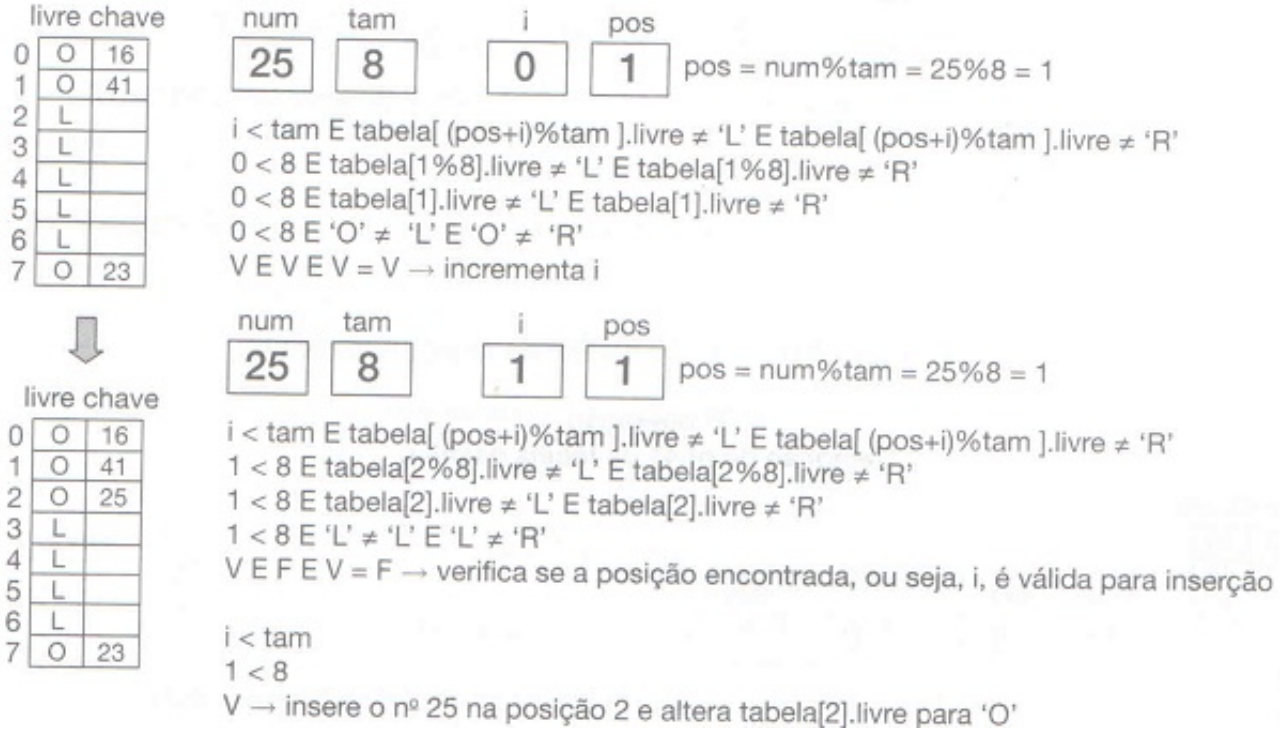
Inserção do número 16 na tabela hashing



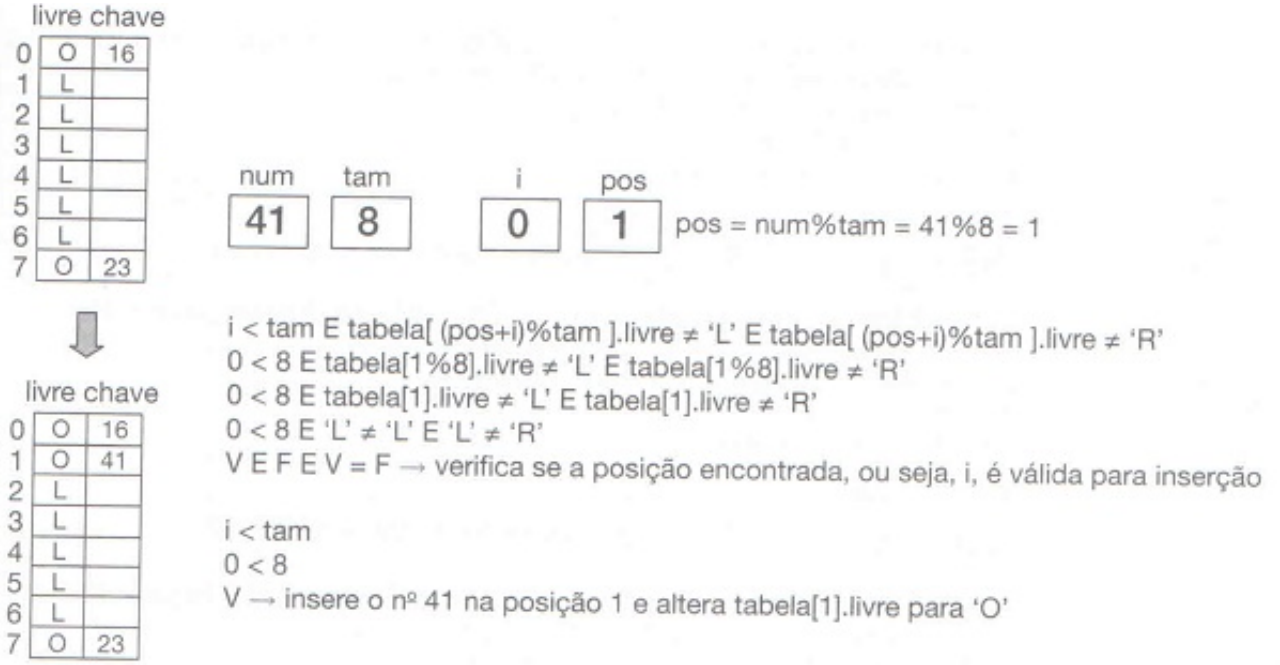
Inserção do número 23 na tabela hashing



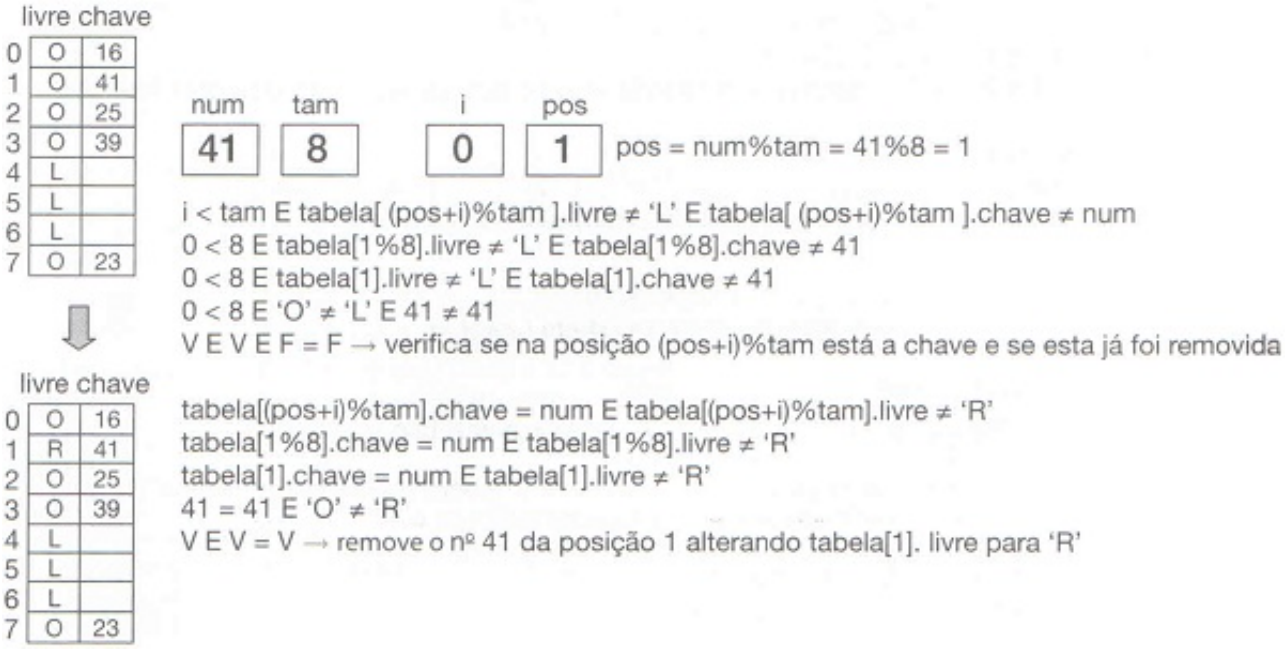
Inserção do número 25 na tabela hashing



Inserção do número 41 na tabela hashing



Remoção do número 41 na tabela hashing



Abaixo segue um exemplo em Java de um algoritmo usando hashing com endereçamento aberto linear:

```
package hashaberto;

import java.util.Scanner;

public class HashAberto {

   public static class hash {

      int chave;
      char livre;
   }
   static int tam = 8;
   static hash tabela[] = new hash[tam];
   static Scanner entrada = new Scanner(System.in);

   public static void inserir(int pos, int n) {
      int i = 0;
      while (i < tam
            && tabela[(pos + i) % tam].livre != 'L'
            && tabela[(pos + i) % tam].livre != 'R') {
         i = i + 1;
      }
      if (i < tam) {
         tabela[(pos + i) % tam].chave = n;
         tabela[(pos + i) % tam].livre = 'O';
      } else {
         System.out.println("Tabela cheia");
      }
   }

   public static void remover(int n) {
      int posicao = buscar(n);
      if (posicao < tam) {
         tabela[posicao].livre = 'R';
      } else {
         System.out.println("Elemento não está presente");
      }
   }
```

```
public static int buscar(int n) {
   int i = 0;
   int pos = funcaohashing(n);
   while (i < tam
         && tabela[(pos + i) % tam].livre != 'L'
         && tabela[(pos + i) % tam].chave != n) {
      i = i + 1;
   }
   if (tabela[(pos + i) % tam].livre == n
         && tabela[(pos + i) % tam].livre != 'R') {
      return (pos + i) % tam;
   } else {
      return tam;
   }
}

static int funcaohashing(int num) {
   return num % tam;
}

static void mostrarhash() {
   for (int i = 0; i < tam; i++) {
      if (tabela[i].livre == 'O') {
         System.out.println("Estrada " + i + ": " + tabela[i].chave + " " + tabela[i].livre);
      }
   }
}

public static void main(String[] args) {
   // TODO code application logic here
   int op, pos, num, i;

   for (i = 0; i < tam; i++) {
      tabela[i] = new hash();
      tabela[i].livre = 'L';
   }
   do {
      System.out.println("\nMENU DE OPÇÕES\n");
      System.out.println("1 - Inserir elemento");
      System.out.println("2 - Mostar tabela hashing");
      System.out.println("3 - Excluir elemento");
      System.out.println("Digite sua opção:");
      op = entrada.nextInt();

      if (op < 1 || op > 4) {
         System.out.println("Opção inválida!");
      } else {
         switch (op) {
            case 1:
               System.out.println("Digite um número: ");
               num = entrada.nextInt();
               pos = funcaohashing(num);
               inserir(pos, num);
               break;
            case 2:
               mostrarhash();
               break;
            case 3:
               System.out.println("Digite um número: ");
               num = entrada.nextInt();
               remover(num);
               break;
         }
      }
   } while (op != 4);
```

```
  }
}
```

Tentativa quadrática

Nesta tentativa, é gerado, o agrupamento secundário, mas as degradações são reduzidas se comparadas com a tentativa linear. No entanto, se duas chaves possuem o mesmo endereço inicial, todas as tentativas seguintes serão iguais nos dois métodos. Para a aplicação deste método os endereços calculados pela função de hashing h'(x,k) devem corresponder à varredura de toda a tabela, para k = 0,…,m−1. Equações recorrentes para calcular os endereços diretamente:

$$h'(x,0) = h(x)$$
$$h'(x,k) = (h(x,k-1)+k) \bmod m,\ 1 \leq k \leq m-1$$

Tabela hashing com lista encadeada

Neste tipo de implementação a tabela hashing é um vetor com M posições, sendo que cada posição possui um ponteiro para uma lista encadeada onde estarão contidos todos os elementos que possuem o mesmo endereço mapeado.

Abaixo segue um exemplo em Java de um algoritmo usando hashing com lista encadeada:

```
import java.util.Scanner;

public class HashLista {

  public static class hash {

    int chave;
    hash prox;
  }

  static int tam = 10;
  static hash tabela[] = new hash[10];
  static Scanner entrada = new Scanner(System.in);

  public static void inserir(int pos, int n) {
    hash novo;
    novo = new hash();
    novo.chave = n;
    novo.prox = tabela[pos];
    tabela[pos] = novo;
  }

  static int funcaohashing(int num) {
    return num % tam;
  }

  static void mostrarhash() {
    hash aux;
    for (int i = 0; i < tam; i++) {
      aux = tabela[i];
      while (aux != null) {
        System.out.println("Entrada " + i + ": " + aux.chave);
        aux = aux.prox;
      }
    }
  }

  static void excluirhash(int num) {
```

```
    int pos = funcaohashing(num);
    hash aux;
    if (tabela[pos] != null) {
      if (tabela[pos].chave == num) {
        tabela[pos] = tabela[pos].prox;
      } else {
        aux = tabela[pos].prox;
        hash ant = tabela[pos];
        while (aux != null && aux.chave != num) {
          ant = aux;
          aux = aux.prox;
        }
        if (aux != null) {
          ant.prox = aux.prox;
        } else {
          System.out.println("Número não encotrado");
        }
      }
    } else {
      System.out.println("Número não encontrador");
    }
  }

  public static void main(String[] args) {
    int op, pos;
    int num;

    do {
      System.out.println("\nMENU DE OPÇÕES\n");
      System.out.println("1 - Inserir elemento");
      System.out.println("2 - Mostar tabela hashing");
      System.out.println("3 - Excluir elemento");
      System.out.println("Digite sua opção:");
      op = entrada.nextInt();

      if (op < 1 || op > 4) {
        System.out.println("Opção inválida!");
      } else {
        switch (op) {
          case 1:
            System.out.println("Digite um número: ");
            num = entrada.nextInt();
            pos = funcaohashing(num);
            inserir(pos, num);
            break;
          case 2:
            mostrarhash();
            break;
          case 3:
            System.out.println("Digite um número: ");
            num = entrada.nextInt();
            excluirhash(num);
            break;
        }
      }
    } while (op != 4);

  }
}
```

Exercícios

1) Uma companhia de gestão de navios deseja controlar melhor os tripulantes que embarcam em um cruzeiro. Ela possui 250 tripulantes no total, mas, no máximo, 50 tripulantes embarcam

em um navio-cruzeiro por viagem. Os tripulantes são identificados por um número entre 1 e 250, chamado código do tripulante. Faça uma tabela de endereçamento aberto para guardar a tripulação que segue num determinado navio-cruzeiro. O programa deve realizar as seguintes operações:

a) Lê os dados da tripulação de um navio-cruzeiro.
b) Consulta se um dado tripulante x está no navio, onde x é o código do tripulante.
c) Se x for encontrado, mostra os seus dados.

A tabela deve guardar código, nome e idade do tripulante.