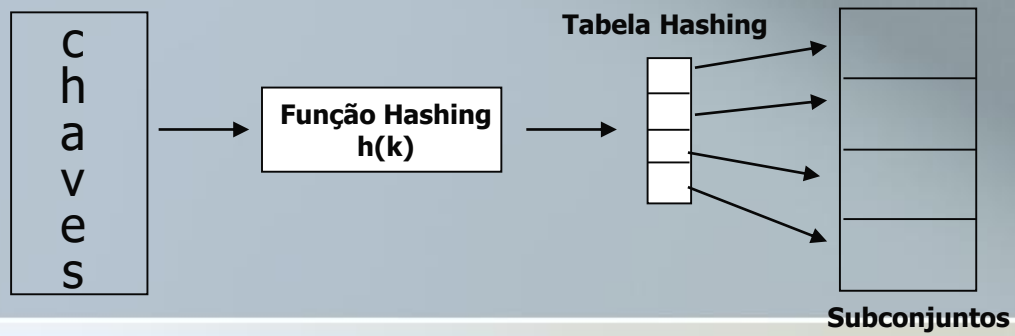
Universidade Federal do Vale do São Francisco
Curso de Engenharia da Computação

Matemática Discreta –12

Prof. Jorge Cavalcanti
jorge.cavalcanti@univasf.edu.br - www.univasf.edu.br/~jorge.cavalcanti

Hashing

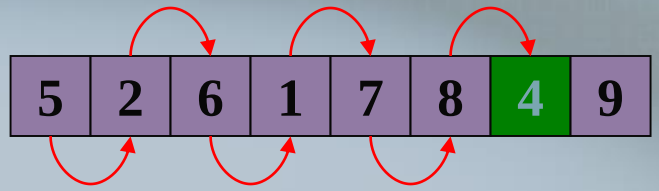
- **Introdução**
  - **Hashing (espalhamento) é uma forma simples, fácil de implementar e intuitiva de se organizar grandes quantidades de dados.**
    - Permite armazenar e encontrar rapidamente dados por chaves.
    - Possui como idéia central a divisão de um universo de dados a ser organizado em subconjuntos mais gerenciáveis.
  - Possui dois conceitos centrais:
    - Tabela de Hashing – estrutura que permite acesso aos subconjuntos;
    - Função de Hashing – função que realiza o mapeamento entre valores da chave e entradas da tabela.

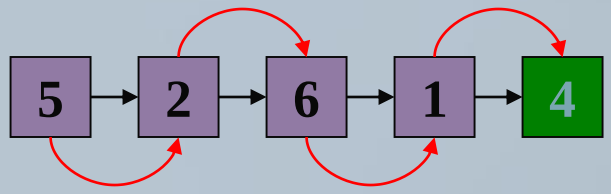


Hashing

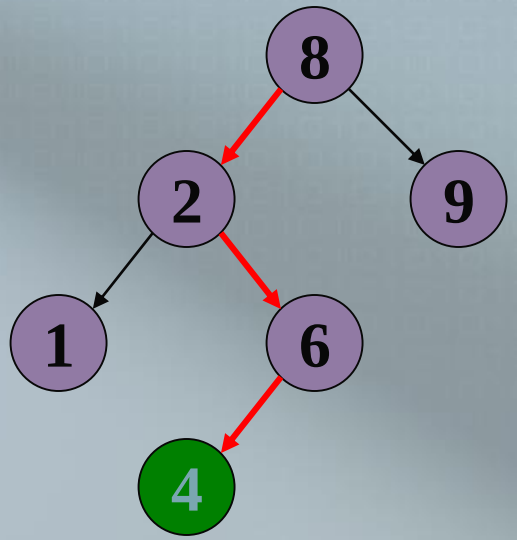
- **Por que usar Hashing?**
  - Estruturas de busca sequencial levam tempo até encontrar o elemento desejado.

Ex: Arrays e listas



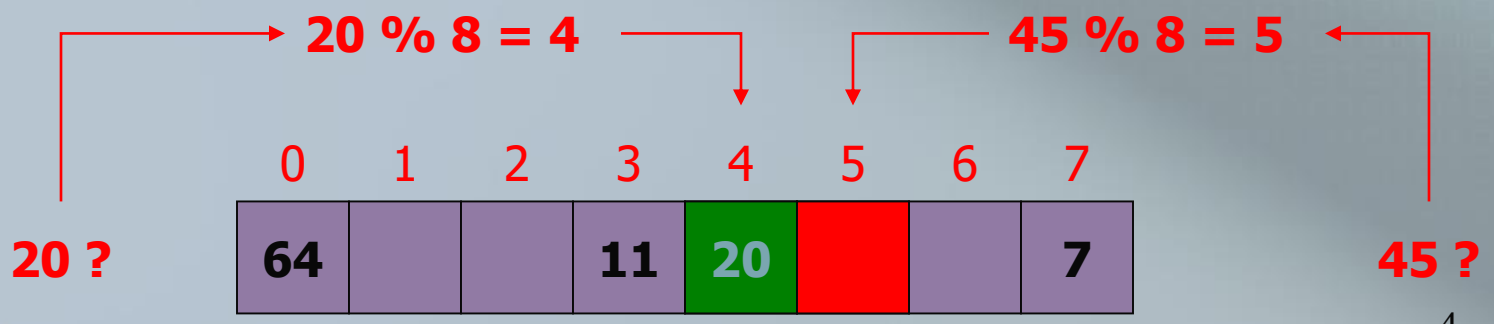


Ex: Árvores



Hashing

- **Por que usar Hashing?**
  - Em algumas aplicações, é necessário obter o valor com poucas comparações, logo, é preciso saber a posição em que o elemento se encontra, sem precisar varrer todas as chaves.
  - A estrutura com tal propriedade é chamada de tabela hashing.
  - A tabela segue a propriedade da função de hashing estabelecida.
  - Ex: h(k)= k mod n, n=8



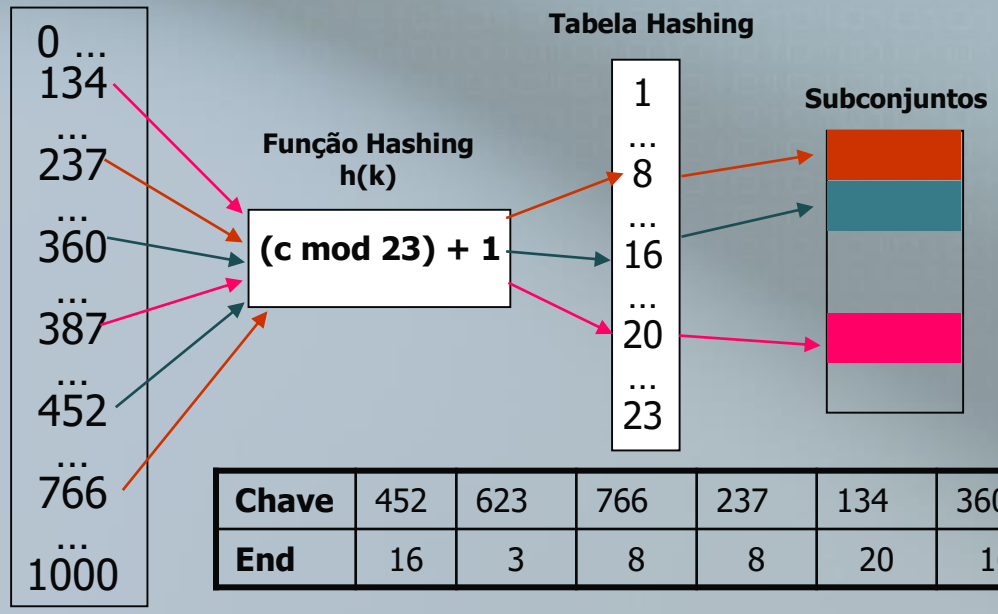
Hashing

- **Ex.:** Considere uma chave de identificação numérica com valores entre 0 e 1000 bem como uma tabela de armazenamento com entradas indexadas de 1 a 23. Uma função de hashing simples e razoável seria:

  hash= h:{0,1,...,1000}→{1,2,3,...,23}, tal que

  para c∈{0,1,...,1000}, tem-se que:

  h(c) = (c mod 23) + 1, onde *mod* calcula o resto da divisão inteira.



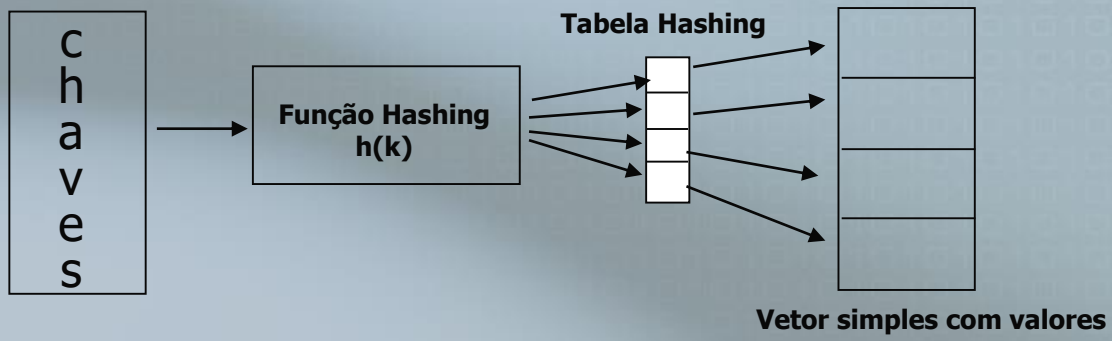
Chave	452	623	766	237	134	360	285
End	16	3	8	8	20	16	10

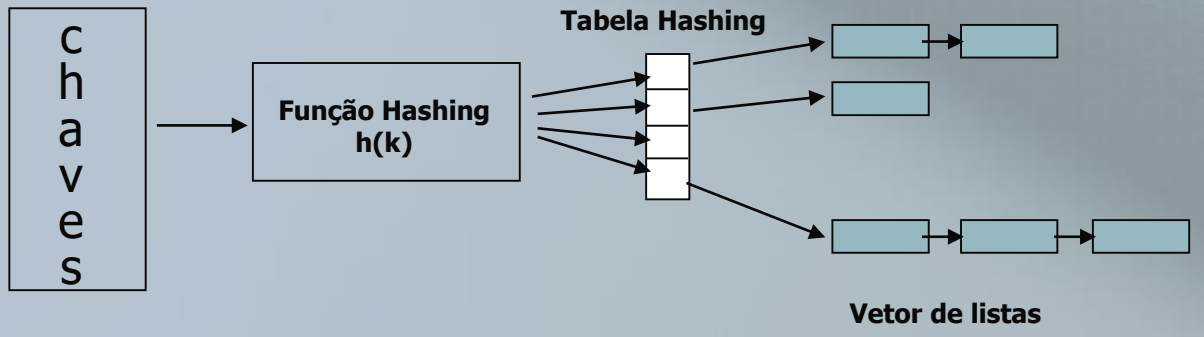
Hashing

- Então, se possuímos um universo de dados classificáveis por chaves, podemos:
  - Criar um critério simples para dividir este universo em subconjuntos com base em alguma qualidade do domínio das chaves;
  - Saber em qual subconjunto procurar e colocar uma chave;
  - Gerenciar esses subconjuntos menores por algum método simples;
- Para isso, precisamos:
  - Saber quantos subconjuntos eu quero criar e uma regra de cálculo que nos diga, dada uma chave, em qual subconjunto devo procurar pelos dados com esta chave ou colocar este dado, caso seja um novo elemento;
    - Esta regra é uma *Função de Hashing*, também chamada de função de cálculo de endereço, função de randomização ou função de aleatorização.
  - Possuir um índice que permita encontrar o início do subconjunto certo, depois de calcular o hashing. Isto é uma tabela de hashing.

Hashing

- A função hashing é a responsável por gerar um índice a partir de determinada chave.
- A Tabela hashing pode apontar para posições na forma de vetores simples ou listas.





Hashing

- Ex.:

Vicente
Elisa
Maria
Antonio
João
Olga
Márcio
Edson
Artur
José
Ana

Função de Hash:
Qual a 1ª letra do nome?

A	Antonio Artur Ana
E	Elisa Edson
J	João José
M	Maria Márcio
O	Olga
V	Vicente

Hashing

- **Função de Hashing**
  - Possui o objetivo de transformar o valor de chave de um elemento de dados em uma posição para este elemento em um dos ***b*** subconjuntos definidos.
  - Deve procurar dividir o universo de chaves $K = \{k_0..,k_m\}$ em ***b*** subconjuntos de mesmo tamanho.
  - A probabilidade de uma chave k_j **pertencente a** ***K*** aleatória qualquer cair em um dos subconjuntos b_i; ***i pertencente a*** **[1,*b*]** deve ser **uniforme**.
  - Se a função de Hashing não dividir ***K*** uniformemente entre os b_i, a tabela de hashing pode degenerar.
    - O pior caso de degeneração é aquele onde todas as chaves caem em um único conjunto b_i.
  - A função "primeira letra" do exemplo anterior é um exemplo de uma função ruim.
    - A letra do alfabeto com a qual um nome inicia não é distribuída uniformemente. Quantos nomes começam com "X"?

Hashing

- Uma função de Hashing deve procurar satisfazer as seguintes condições:
  - Ser simples de calcular;
  - Assegurar que elementos distintos tenham índices distintos;
  - Gerar uma distribuição equilibrada para os elementos dentro do *array* ou subconjuntos;
  - Deve ser aleatória, ou pseudo-aleatória, para prevenir adivinhações do valor original;
  - Deve ter mão única, o que significa ser muito difícil a partir do endereço e dos valores originais obter a função.
  - A forma de transformação mais simples e utilizada é a **Divisão,** como vista anteriormente:
    - **$h(k_j) = mod(k_j,b) + 1$**, onde b (preferencialmente um número primo) é o número de subconjuntos em dividimos os dados.

Hashing

- A função ideal é aquela que gera um endereço diferente para cada um dos possíveis valores das chaves (*Função injetora*).
- Porém nem sempre é possível e ai geram *colisões*, ou seja, em alguns casos, podem ser atribuídos mesmos endereços a chaves com valores diferentes.
- **Diferenças entre hashing e indexação:**
  - No espalhamento, os endereços parecem ser aleatórios – não existe conexão óbvia entre a chave e o endereço, apesar da chave ser utilizada no cálculo do endereço
  - No espalhamento, duas chaves podem levar ao mesmo endereço (colisão) – portanto as colisões devem ser tratadas.

Hashing

- **Chaves não-numéricas:**
  - Ex.:Suponha que foi reservado espaço para manter 1.000 registros e considere a seguinte h(K):
    - Obter as representações ASCII dos dois primeiros caracteres do nome;
    - Multiplicar estes números e usar os três dígitos menos significativos do resultado para servir de endereço.

Nome	Cod ASCII para 2 primeiras letras	Produto	Endereço
BALL	66 65	66 x 65 = 4.290	290
LOWELL	76 96	76 x 96 = 6.004	004
TREE	84 82	84 x 82 = 6.888	888

Hashing

- **Continuação Exemplo:**

- O ideal é usar uma função de espalhamento perfeita, que não produz colisão, mas...
- Duas palavras diferentes podem produzir o mesmo endereço (colisão), pois as chaves são sinônimas
  - Temos, h(LOWELL)=h(LOCK)=h(OLIVER)

Hashing

- **Colisões**
  - Colisão: quando duas ou mais chaves são mapeadas na mesma posição da tabela de hash;
  - Tipicamente, o número de posições numa tabela de hash é pequeno comparado com o universo de chaves possíveis;
  - A maioria das funções de hash usadas na prática mapeiam várias chaves na mesma posição na tabela de hash.
- **Problemas de hashing:**
  - Encontrar funções de hash que distribuam as chaves de modo uniforme e minimizem o número de colisões;
  - Resolver colisões.

Hashing

Colisões

- Devido ao fato de existirem mais chaves que posições, é comum que várias chaves sejam mapeadas na mesma posição.
- **Ex: 45 % 8 = 5** **1256 % 15 = 11**
  21 % 8 = 5 **356 % 15 = 11**
  93 % 8 = 5 **506 % 15 = 11**
- O que fazer quando mais de um elemento for inserido na mesma posição de uma tabela *hash?*

Hashing

Endereçamento Fechado *(Closed Addressing)*

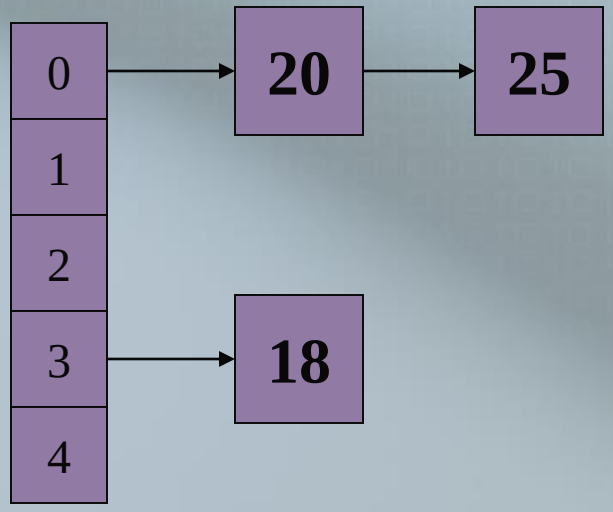
- No endereçamento fechado, a posição de inserção não muda, logo, todos devem ser inseridos na mesma posição, através de uma lista ligada em cada uma.

20 % 5 = 0

18 % 5 = 3

25 % 5 = 0

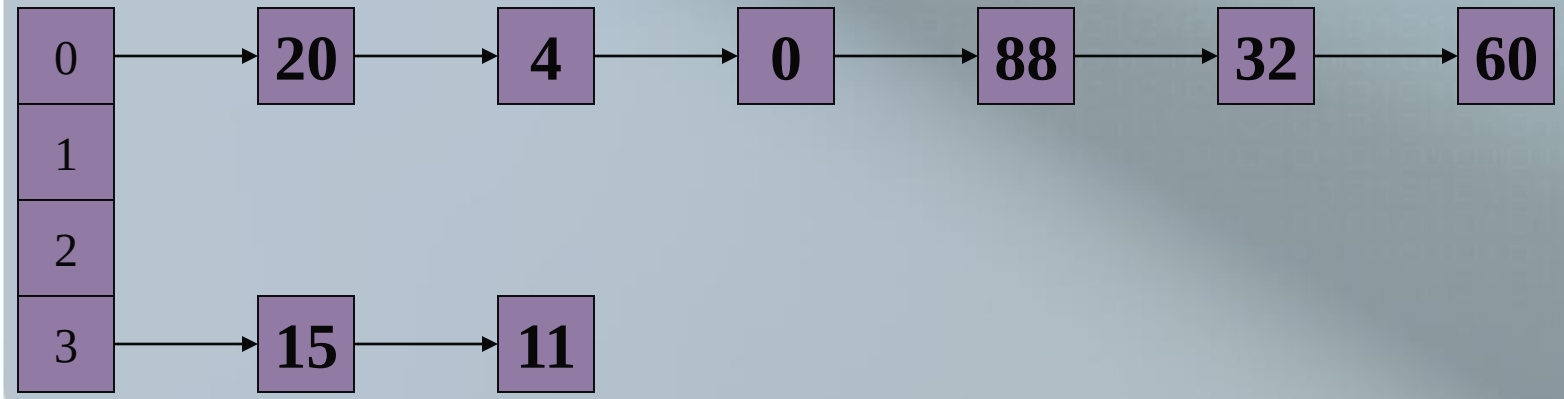
colisão com 20



Hashing

Endereçamento Fechado *(Closed Addressing)*

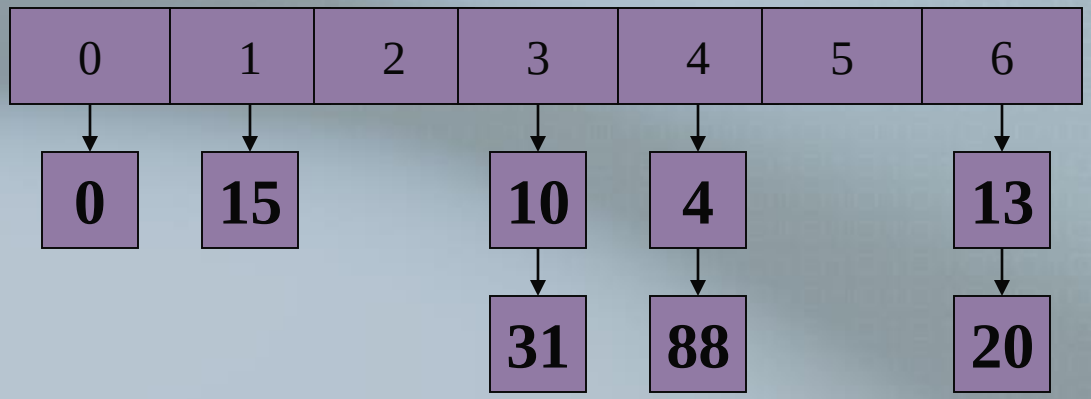
- A busca é feita do mesmo modo: calcula-se o valor da função hash para a chave, e a busca é feita na lista correspondente.
- Se o tamanho das listas variar muito, a busca pode se tornar ineficiente, pois a busca nas listas é seqüencial:



Hashing

Endereçamento Fechado *(Closed Addressing)*

- Por esta razão, a função hash deve distribuir as chaves entre as posições uniformemente:

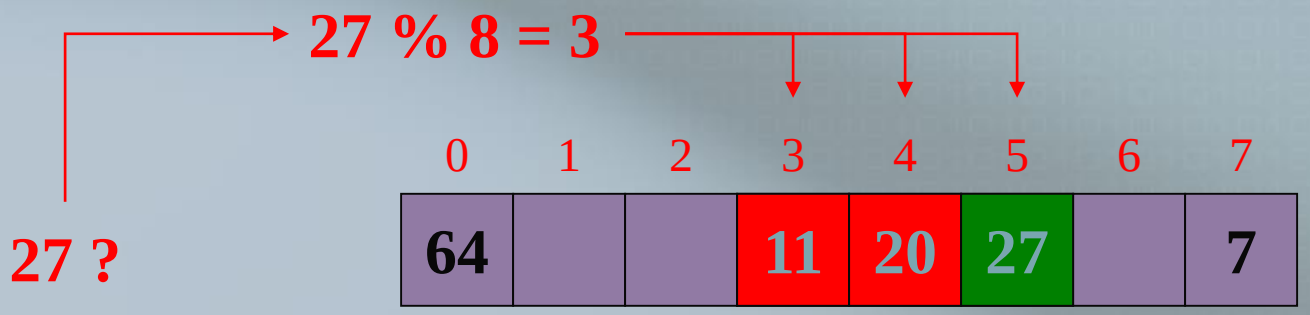


- Se o tamanho da tabela for um número primo, há mais chances de ter uma melhor distribuição.

Hashing

Endereçamento Aberto *(Open Addressing)*

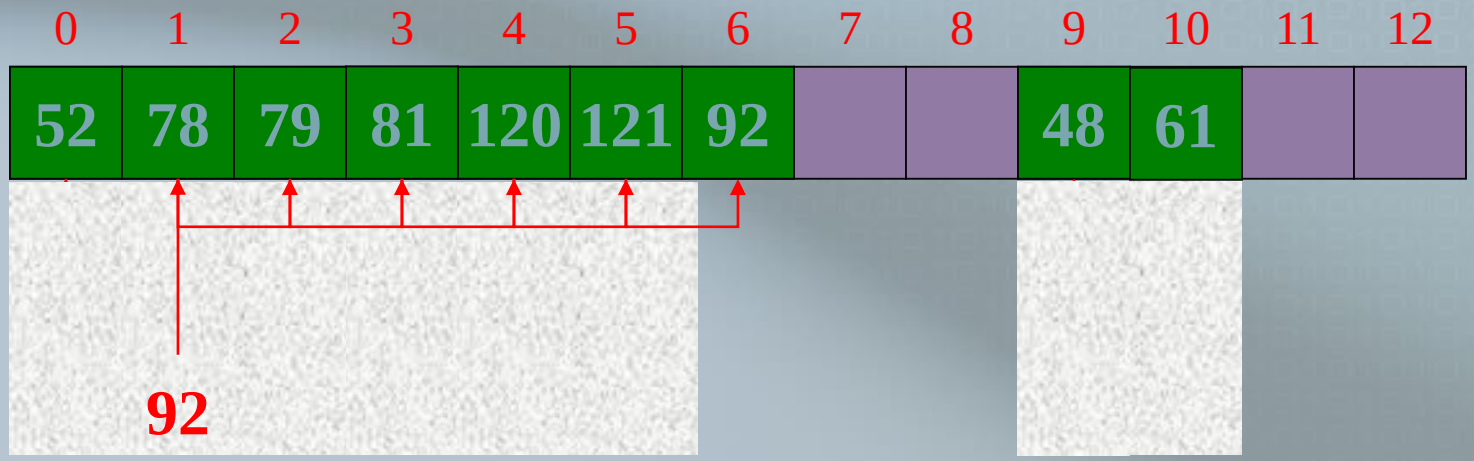
- No endereçamento aberto, quando uma nova chave é mapeada para uma posição já ocupada, uma nova posição é indicada para esta chave.
- A nova posição é incrementada até que uma posição vazia seja encontrada (*linear probing*):



Hashing

Endereçamento Aberto *(Open Addressing)*

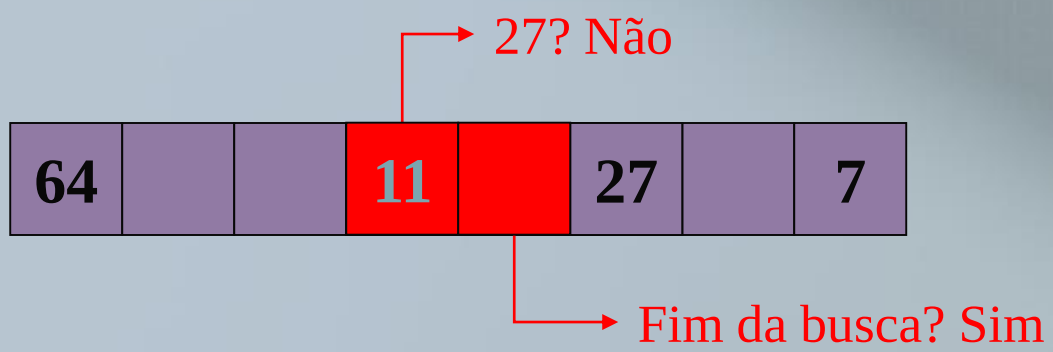
Valores: 52, 78, 48, 61, 81, 120, 79, 121, 92
Função: hash(k) = k % 13
Tamanho da tabela: 13



Hashing

Endereçamento Aberto – Remoção

- Para fazer uma busca com endereçamento aberto, basta aplicar a função hash, e a função de incremento até que o elemento ou uma posição vazia sejam encontrados.
- Porém, quando um elemento é removido, a posição vazia pode ser encontrada antes, o que significaria fim de busca, MESMO que o elemento PERTENÇA à tabela:



Inserção do 27

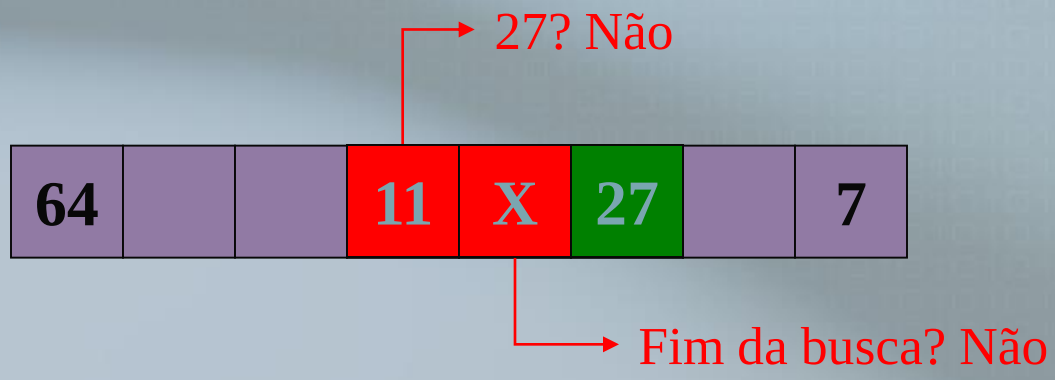
Remoção do 20

Busca pelo 27

Hashing

Endereçamento Aberto – Remoção

- Para contornar esta situação, mantemos um bit (ou um campo booleano) para indicar que um elemento foi removido daquela posição:

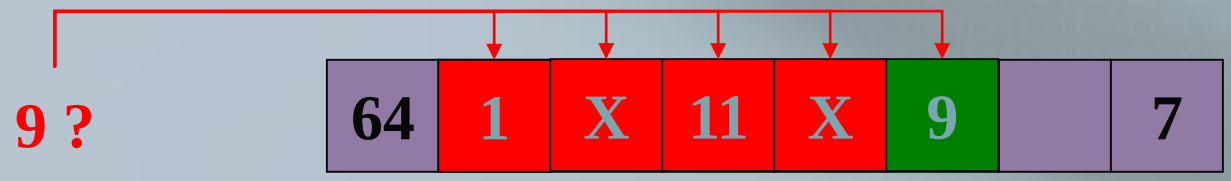


- Esta posição estaria livre para uma nova inserção, mas não seria tratada como vazia numa busca.

Hashing

Endereçamento Aberto – Expansão

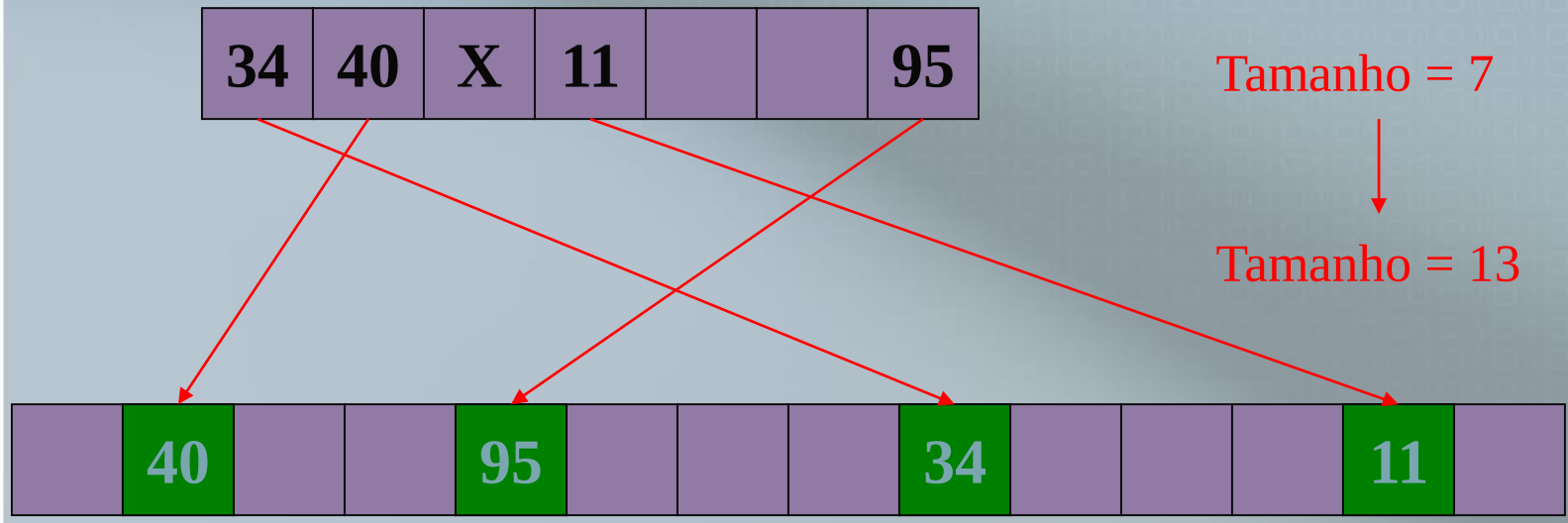
- Nesta política de hashing, há o que chamamos de **fator de carga** (*load factor*). Este fator indica a porcentagem de células da tabela hash que estão ocupadas, incluindo as que foram removidas.
- Quando este fator fica muito alto (ex: excede 50%), as operações na tabela passam a demorar mais, pois o número de colisões aumenta.



Hashing

Endereçamento Aberto – Expansão

- Quando isto ocorre, é necessário expandir o array que constitui a tabela, e reorganizar os elementos na nova tabela. Como podemos ver, o tamanho atual da tabela passa a ser um parâmetro da função hash.



Hashing

Endereçamento Aberto – Expansão

- O momento ou critério para expandir a tabela pode variar:
  - Impossibilidade de inserir um elemento
  - Metade da tabela está ocupada
  - O *load factor* atingiu um valor limite escolhido
- A terceira opção é a mais comum, pois é um meio termo entre as outras duas.

Hashing

Quando não usar Hashing?

- Muitas colisões diminuem muito o tempo de acesso e modificação de uma tabela hash. Para isso é necessário escolher bem:
  - a função hash
  - o tratamento de colisões
  - o tamanho da tabela
- Quando não for possível definir parâmetros eficientes, pode ser melhor utilizar árvores balanceadas (como AVL), em vez de tabelas hash.

- \+ **Hashing em Estruturas de Dados...**

Hashing

Exercícios:

1 - Ilustre a organização final de uma Tabela Hash após a inserção das seguintes chaves: 35, 99, 27, 18, 65, 45. Considere a tabela com tamanho 6 (posições 0 a 5), o *método da divisão inteira* como **função de hashing** e tratamento de colisão por ***endereçamento fechado.*** Considere também que os números possíveis de chaves estão no intervalo entre 1 a 100.

2 - Idem à questão anterior, porém o tratamento de colisão será por ***endereçamento aberto (linear probing)***.

Hashing

Exercícios:

3 – Seja uma função h(k) = k % 11 e os dados abaixo obtidos para uma sequência de chaves:

key	82	31	28	4	45	27	59	79	35
h(key)	5	9	6	4	1	5	4	2	2

Resolva as colisões por endereçamento aberto. Qual o fator de carga da tabela?

4 – Sejam as seguintes entradas para uma tabela hash de 07 posições: **0, 4, 6, 11, 15, 7, 9, 3, 25, 22**. Estabeleça uma função hash, ilustre como seria a distribuição dos dados na tabela a partir dessa função e trate as possíveis colisões por endereçamento fechado.