Estrutura de Dados

Hashing

Profa.: Márcia Sampaio Lima

EST - UEA

Hashing

- É uma forma simples, fácil de se implementar e intuitiva de se organizar grandes quantidades de dados.
  - Permite armazenar e encontrar rapidamente dados por chave.
- Possui como idéia central a divisão de um universo de dados a ser organizado em subconjuntos mais gerenciáveis.

Hashing

- Uma estrutura de dado estática:
  - Alterar seu tamanho tem implicações nas informações armazenadas;
- As EDs usadas são vetor e listas;
- Os elementos são distribuídos usando-se uma função de espalhamento;
  - Define uma regra mas não é posicional.

Hashing

- Objetivo:
  - Mapear um espaço enorme de chaves em um espaço relativamente pequeno.
  - Esse mapeamento é feito através de uma função chamada **hash function.**
  - O valor gerado pela **hash function** é chamado **hash code** e é usado para encontrar a localização do item.

Hashing

- Possui dois conceitos centrais:
  - **Tabela de Hashing:**Estrutura que permite o acesso aos subconjuntos: listas e vetores.
  - **Função de Hashing:** Função que realiza um mapeamento entre valores de chaves e entradas na tabela:função de espalhamento.

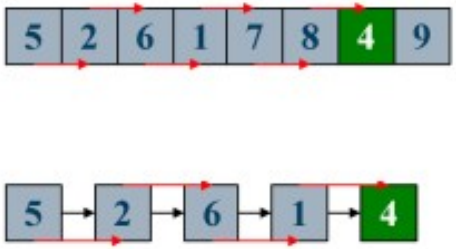
Hashing

- Limitações em relação às árvores:
  - Não permite recuperar/imprimir todos os elementos em ordem de chave nem tampouco outras operações que exijam seqüência dos dados.
  - Não permite operações do tipo recuperar o elemento com a maior ou a menor chave.

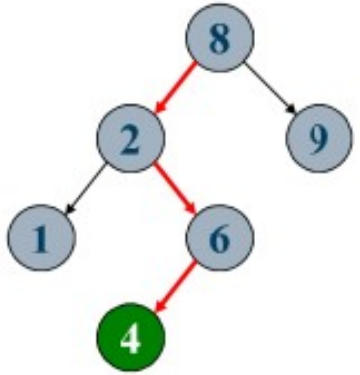
Hashing

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

Ex: Arrays e listas

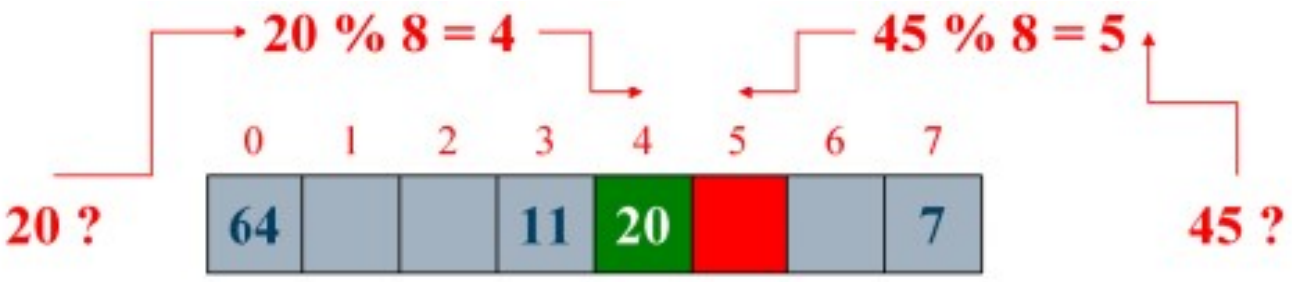


Ex: Árvores



Hashing

- Por que usar *Hashing*?
  - Em algumas aplicações é necessário obter o valor em 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 a **Tabela Hash**

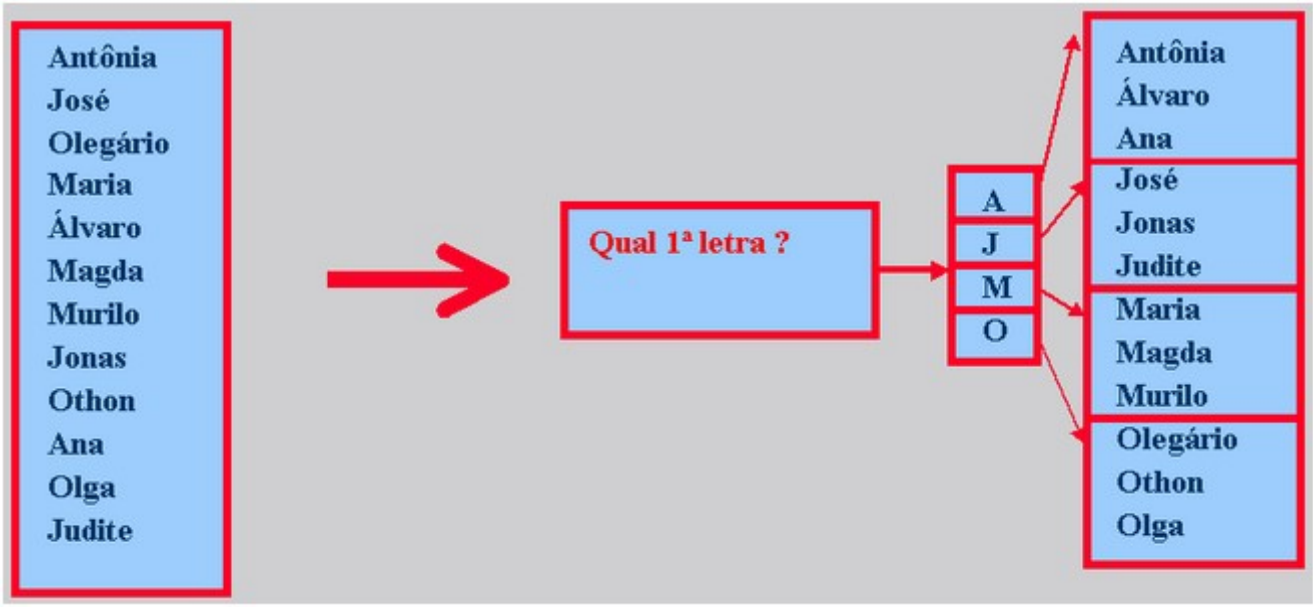


Hashing

- Idéia geral: Se existe um universo de dados classificáveis por chave, podemos:
  - Criar um critério simples para dividir este universo em subconjuntos.
  - Saber em qual subconjunto procurar e colocar uma chave.
  - Gerenciar estes subconjuntos bem menores por algum método simples.

Hashing

- Exemplo:

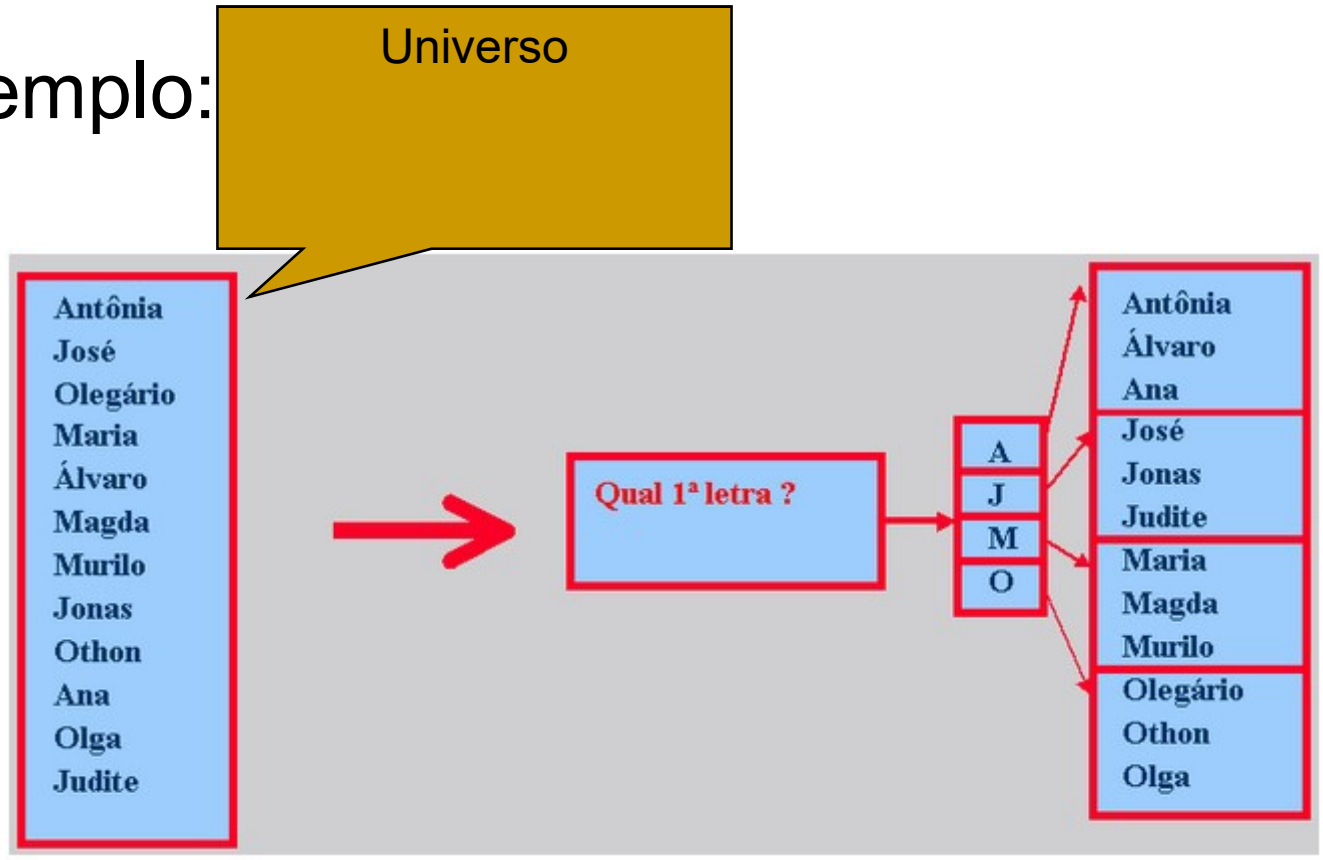


Hashing

- Necessário:
  - Saber quantos subconjuntos.
  - Criar uma regra de cálculo que me diga, dada uma chave, em qual subconjunto devo procurar pelos dados com esta chave ou colocar este dado, caso seja um novo elemento = **função de hashing**.

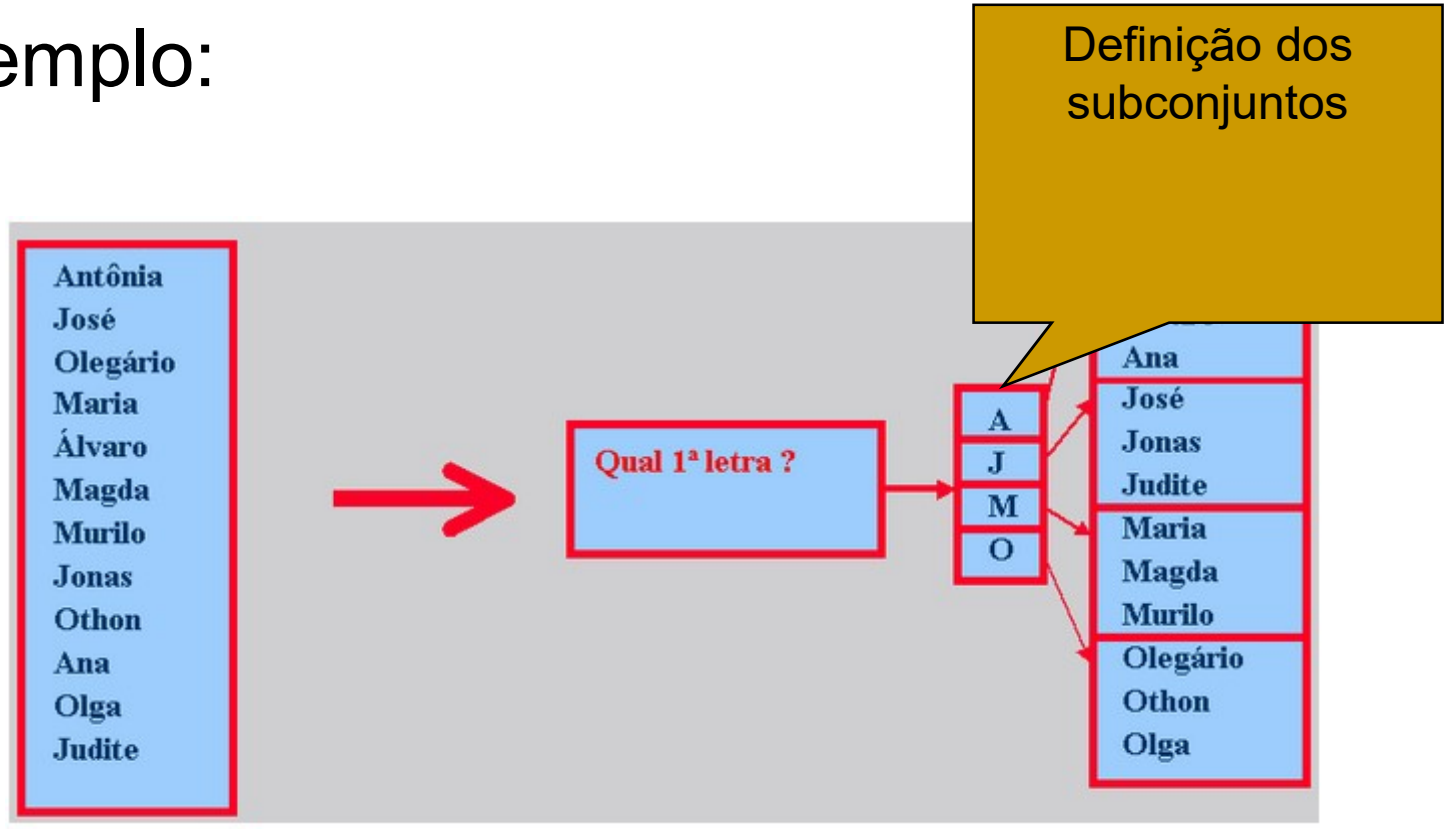
Hashing

- Exemplo:



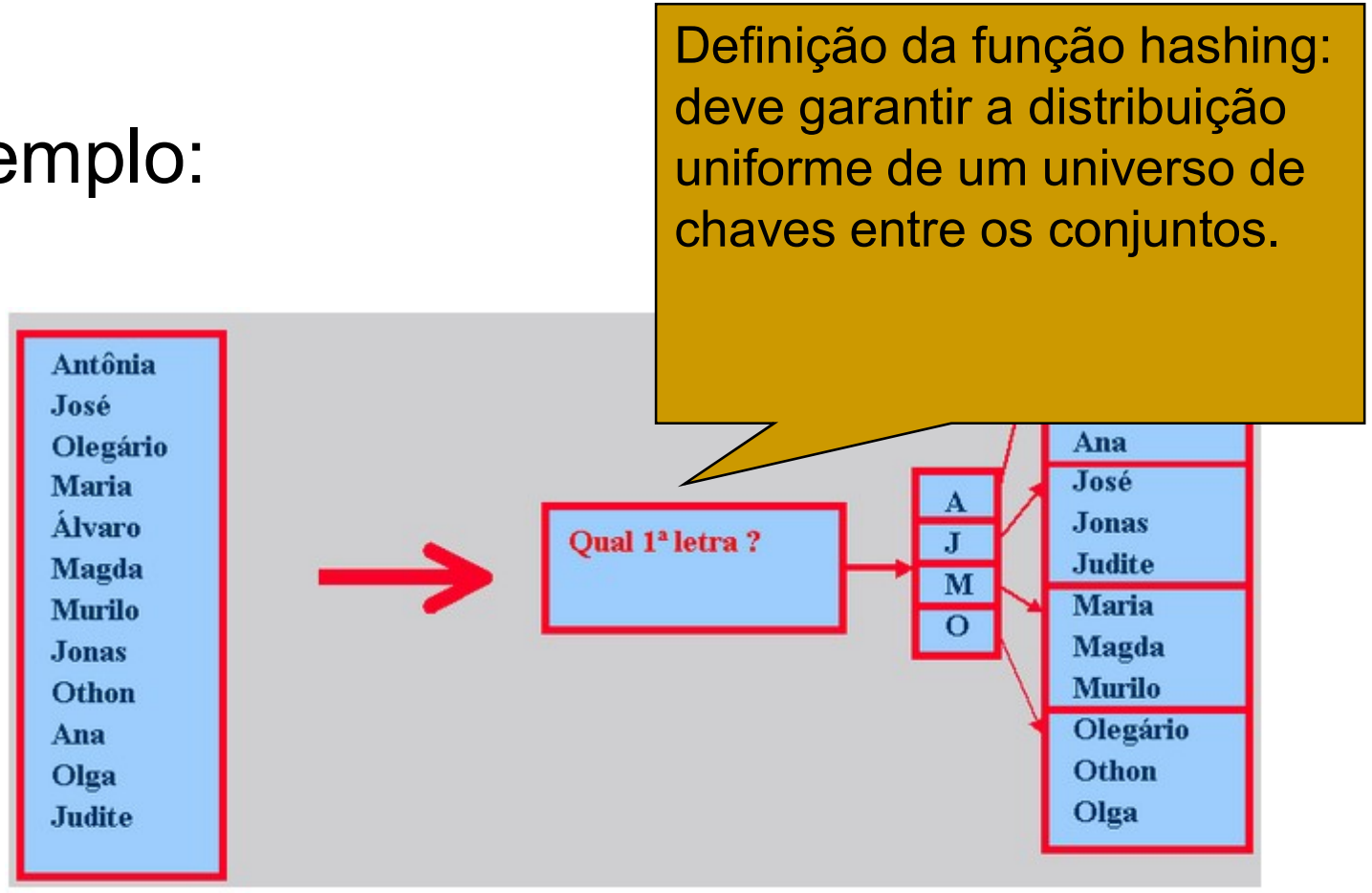
Hashing

- Exemplo:



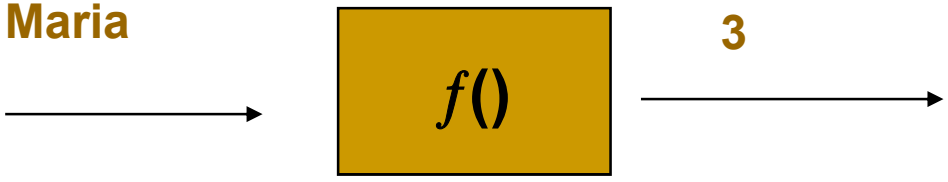
Hashing

- Exemplo:



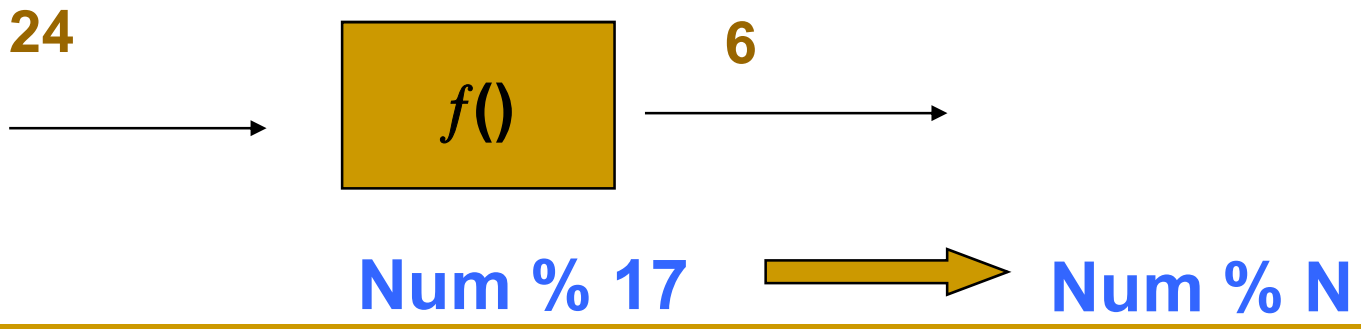
Hash

- Função de Dispersão
  - A função de dispersão ou função de hash mapeia uma chave de busca num índice da tabela.
  - A implementação dessa função recebe como entrada a chave de busca e retorna um índice da tabela:



Hash

- Função de Dispersão
  - A função de dispersão ou função de hash mapeia uma chave de busca num índice da tabela.
  - A implementação dessa função recebe como entrada a chave de busca e retorna um índice da tabela:



Hash

- Transformamos qualquer chave em um numero natural.
- Depois calculamos a posição a ser inserida transformando o número natural em uma das posições possíveis dentro da tabela hash.
- Método da divisão (mais comum):
  - H(k) = k % M
  - M é o tamanho da tabela.

Hash

- Método da divisão (mais comum):
  - H(k) = k % M
  - M é o tamanho da tabela.

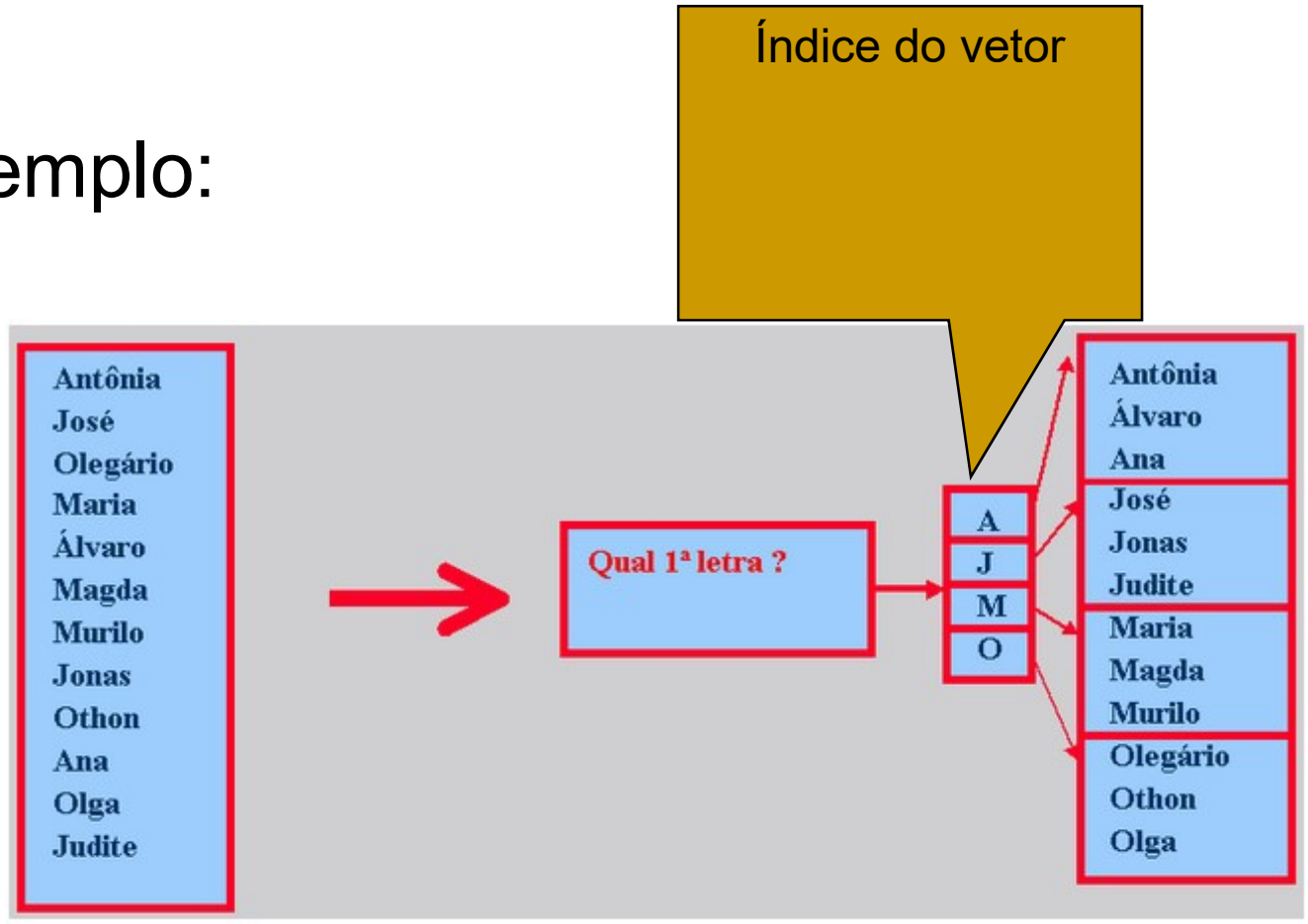
  - M é um número primo, pois melhora a distribuição dos elementos dentro da tabela.

Hashing

- Necessário:
  - Possuir um índice que me permita encontrar o início do subconjunto certo, depois de calcular o hashing. Isto é a **tabela de hashing**.

Hashing

- Exemplo:

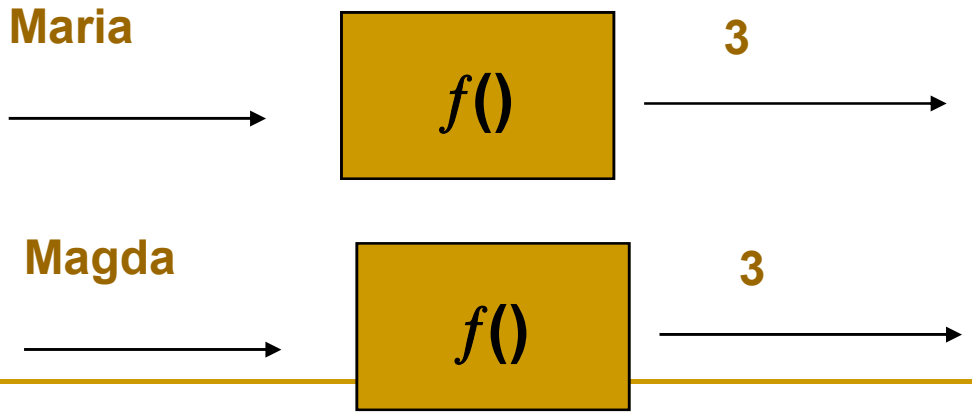


Hash

- Elemento com chave k é armazenado na posição h(k), onde h é a função de dispersão.
- Se a função h der o mesmo resultado, para duas chaves então temos uma colisão.
- O que fazer nesses casos?

Hash

- Colisão
  - Por haver mais chaves que posições na tabela hash, é comum que várias chaves sejam mapeadas para a mesma posição.
  - Quando isso ocorre dizemos que houve uma colisão.



Hash

- Colisão
  - 45 % 8 = 5
  - 21 % 8 = 5
  - 93 % 8 = 5

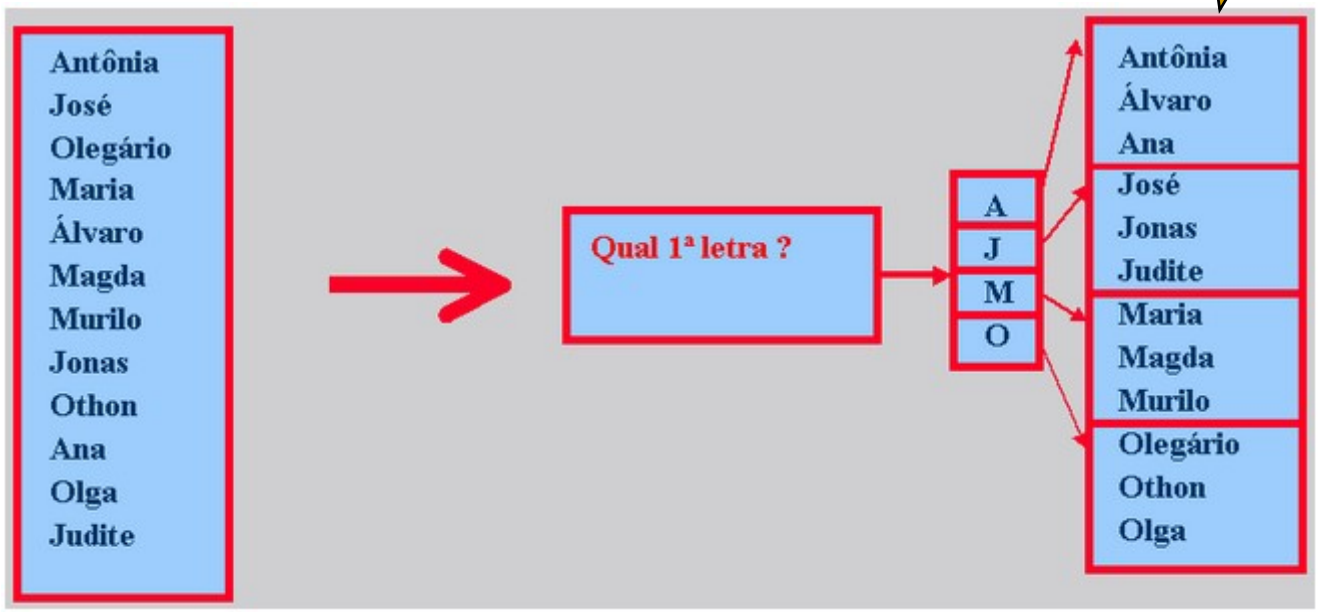
  - O que fazer?
    - Resolver essas colisões!!!

Hashing

- Necessário:
  - Possuir uma ou um conjunto de estruturas de dados para os subconjuntos.
  - Existem duas filosofias: **hashing fechado** ou o **hashing aberto**.

Hashing

- Exemplo:

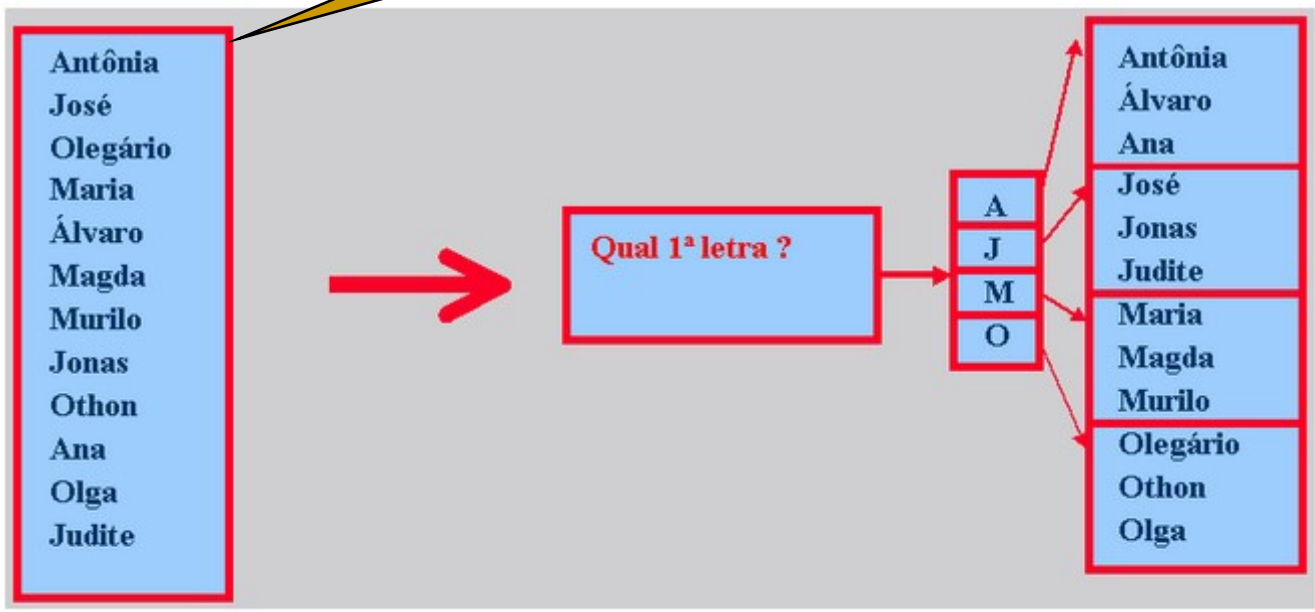


Hashing

- Colisão
  - Ocorrem quando dois ou mais elementos do conjunto universo mapeados na mesma entrada:
  - Independente da função de espalhamento, algumas colisões irão ocorrer;
  - Uma distribuição uniforme das entradas da tabela não impede as colisões;

Hashing

- Exemplo:



Colisão ocorre quando temos nomes que comecem com a mesma letra

Hashing

- Tratamento de Colisões:
  - Hashing Aberto ou de Encadeamento Separado
- **Hashing Fechado ou de Endereçamento Aberto**

Hashing

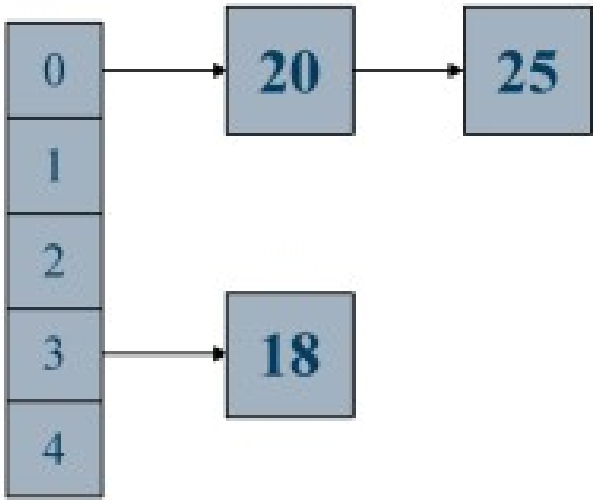
- **Hashing Aberto ou de Encadeamento Separado :**
- Nesse hash, a posição de inserção não muda, logo, todos devem ser inseridos na mesma posição, através de uma lista ligada em

20 % 5 = 0

18 % 5 = 3

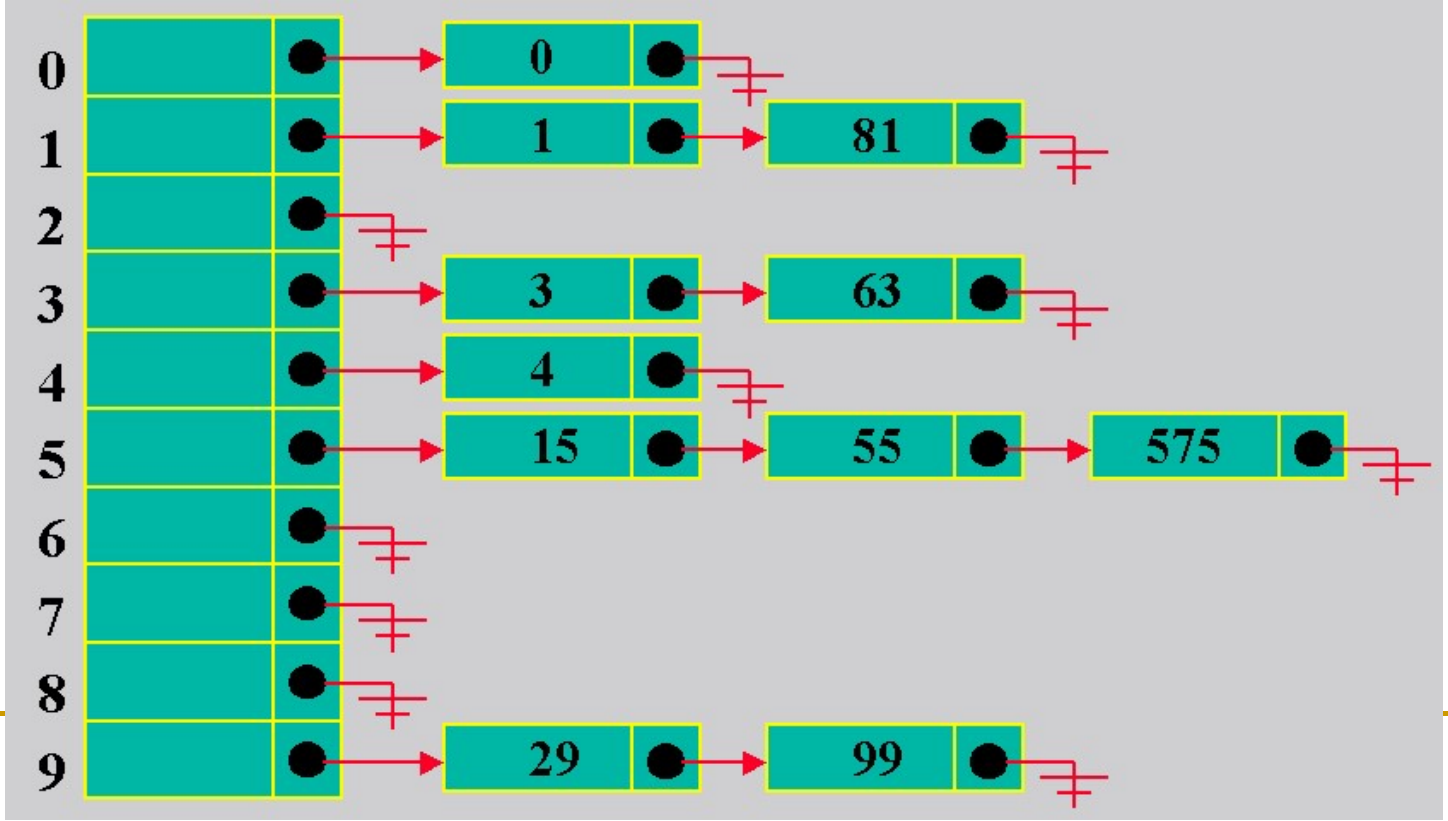
25 % 5 = 0

colisão com 20

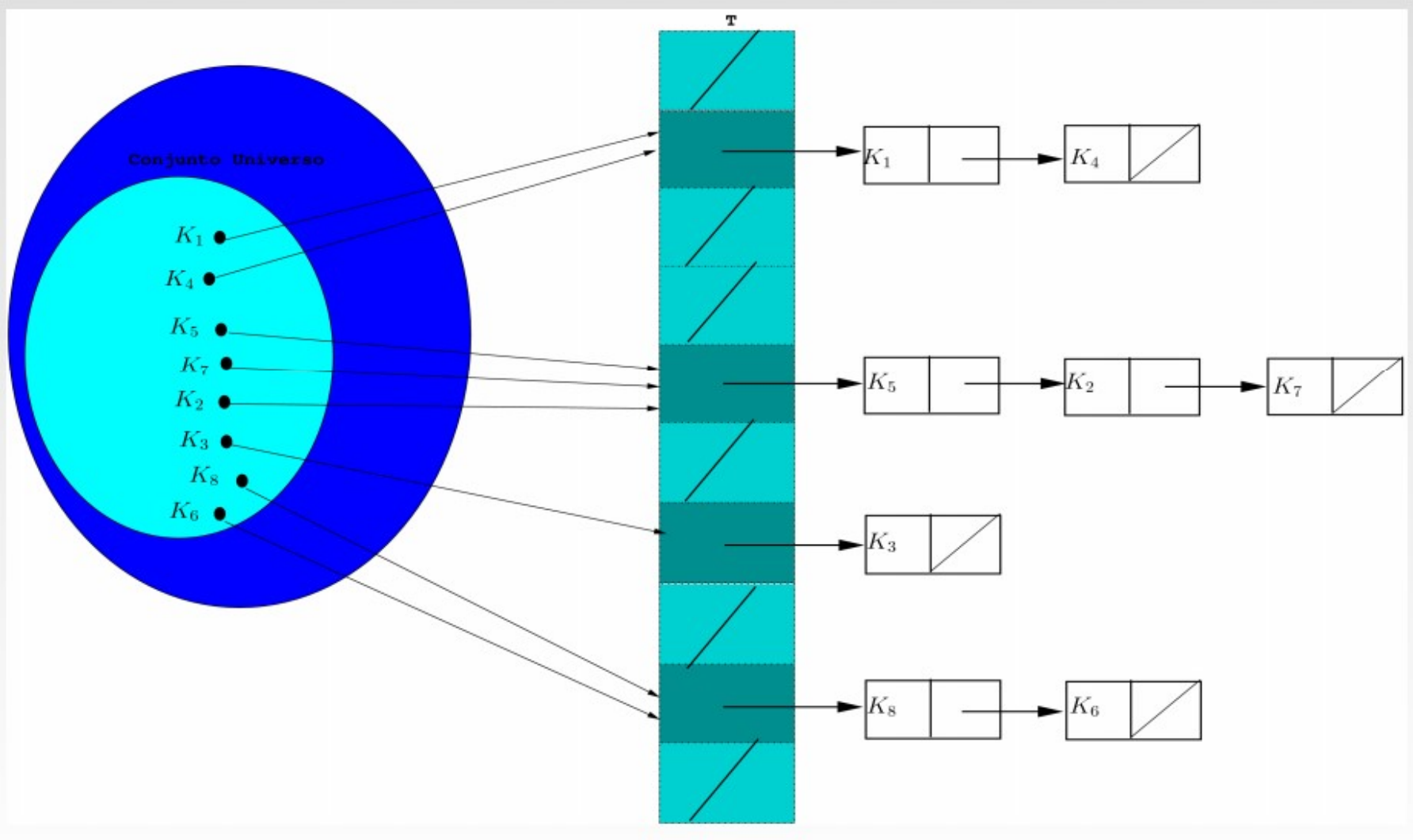


Hashing

- **Exemplo:**
  - Vetores e Lista encadeada

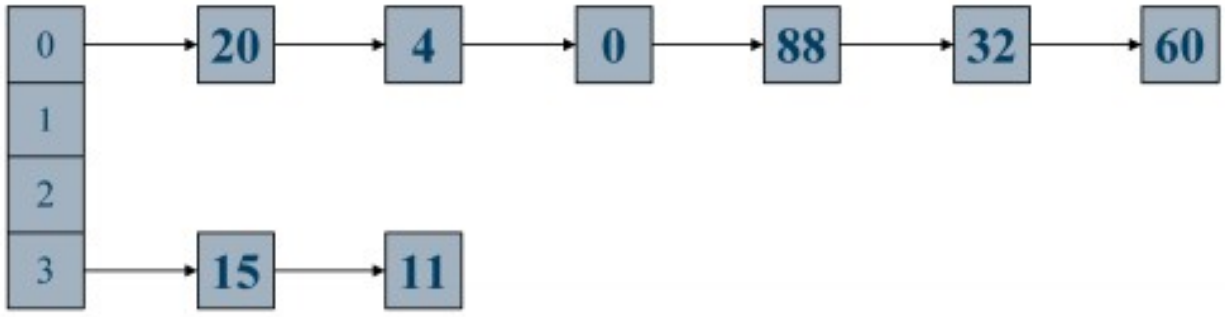


Hashing



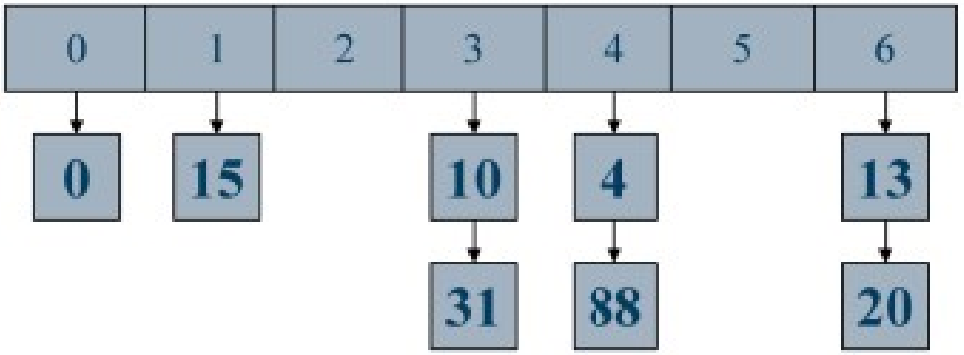
Hash

- A busca é feita da seguinte forma:
  - Calcula o valor da função hash.
  - A busca é feita na lista correspondente.
  - Se o tamanho das listas variar muito, a busca pode se tornar ineficiente, pois a busca na lista é seqüencial.



Hash

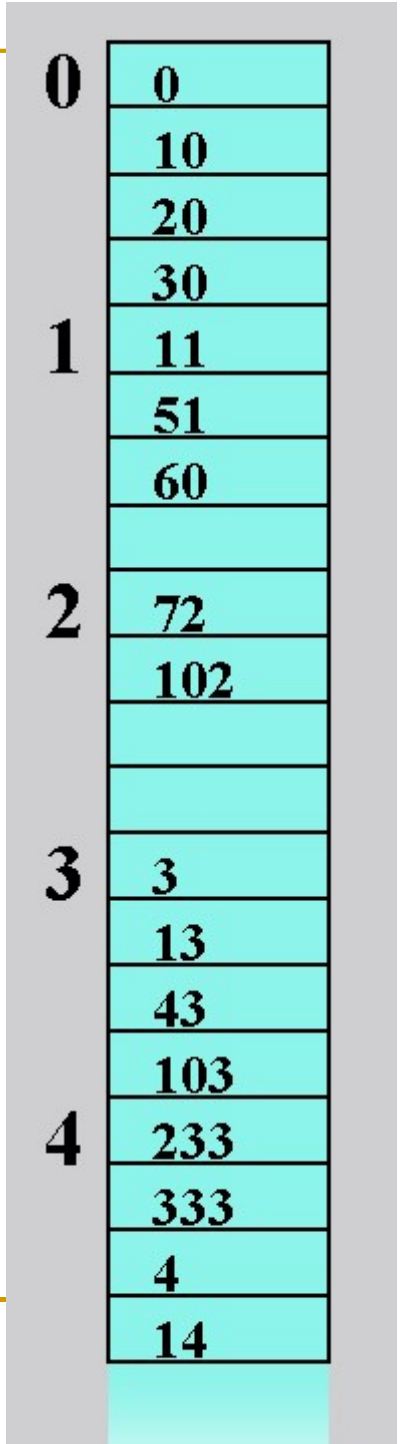
- A busca é feita da seguinte forma:
  - Por isso, a função hash deve distribuir as chaves entre as posições da tabela **uniformemente**:



  - Se o tamanho da tabela for um numero primo há mais chances de melhor distribuição.

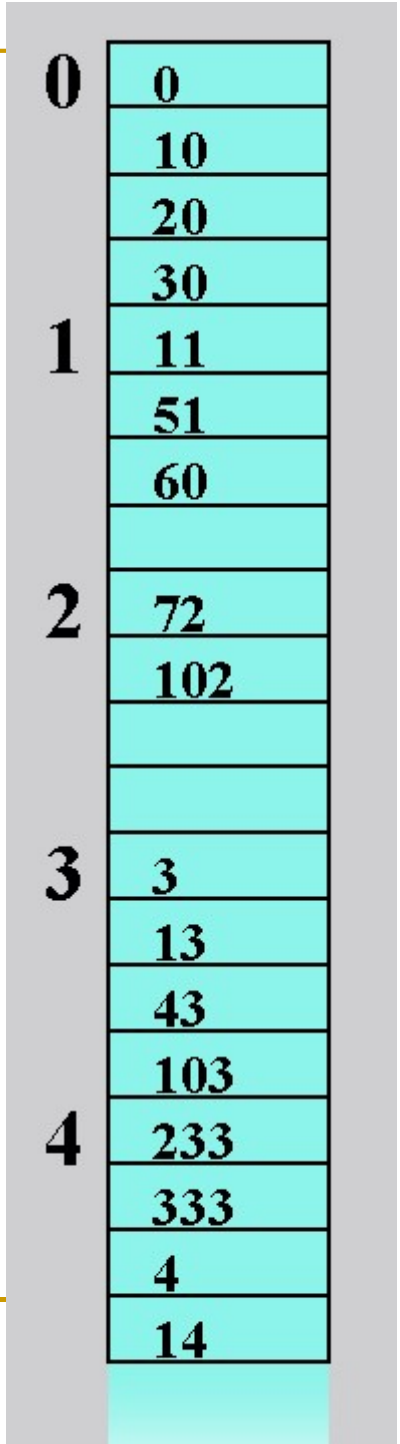
Hashing

- **Hashing Fechado ou de Endereçamento Aberto**
- Utiliza somente uma estrutura de dados de tamanho fixo para implementar o hashing.
  - Não necessita de ponteiros para a sua implementação.
  - Somente uma tabela dividida em partes iguais.
    - Áreas de tamanho fixo para cada subconjunto.
    - Cada repositório é examinado seqüencialmente na busca por uma chave
    - Quando um subconjunto está cheio, há ***estouro***.



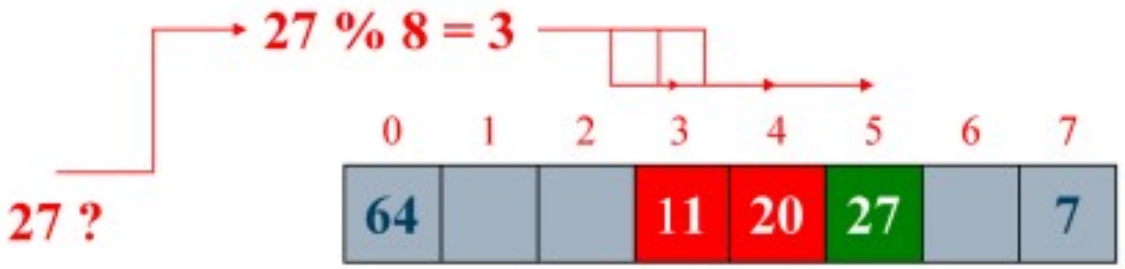
Hashing

- **Hashing Fechado ou de Endereçamento Aberto**
  - Filosofias para tratamento de estouros
    - Utilização do primeiro espaço vazio



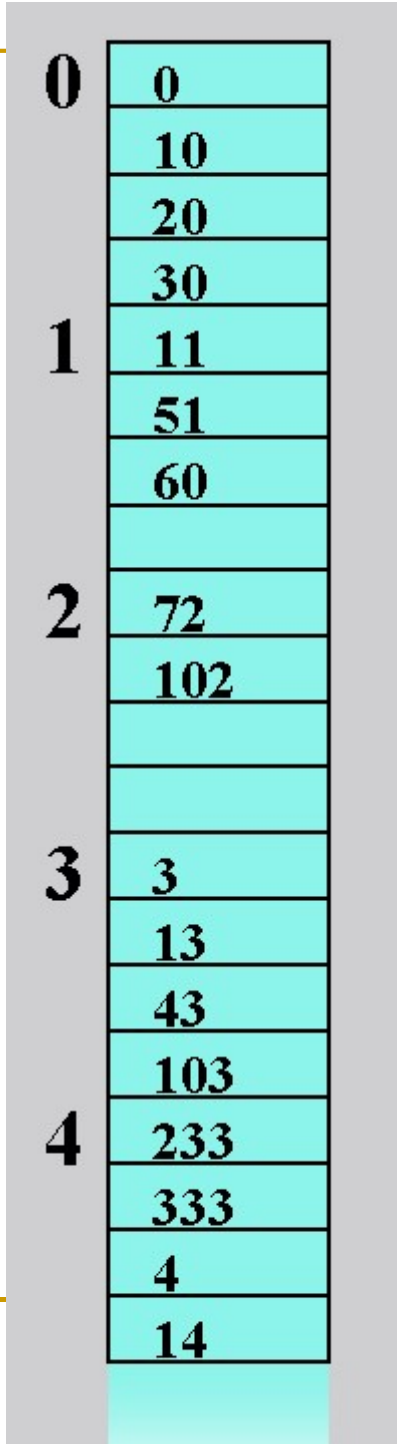
Hashing

- **Hashing *aberto***
  - Filosofias para tratamento de estouros
    - Utilização do primeiro espaço vazio

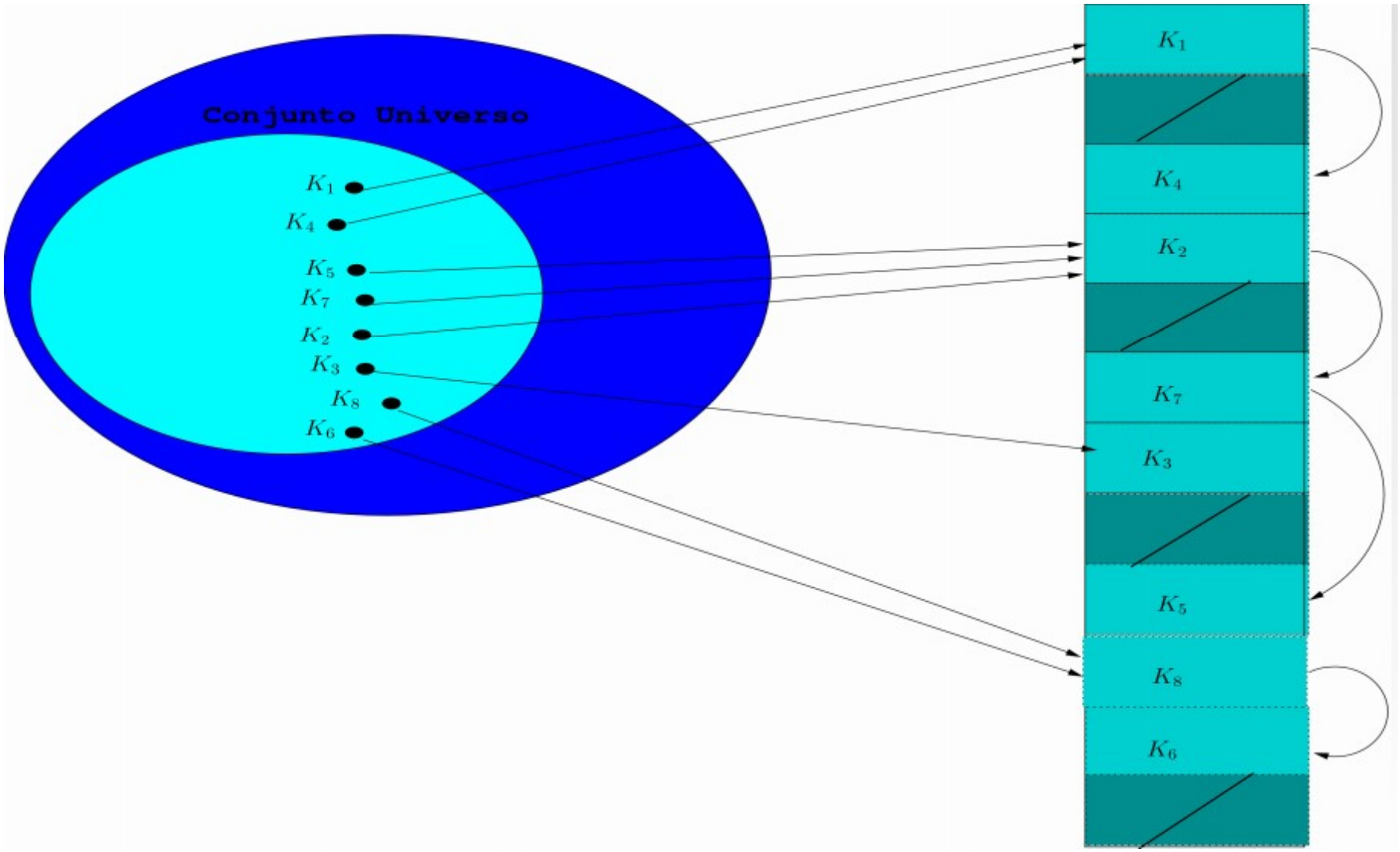


Hashing

- **Hashing Fechado ou de Endereçamento Aberto**
  - Função hash
    - ***h(k)* = *mod(k, 10)***
  - Manipulação de estouro:
    - Usamos o primeiro espaço livre.
    - No caso do 60, isto foi logo apos duas entradas em "1".
  - Critério de parada de busca:
    - Consideramos o arquivo inteiro como uma lista circular e
      paramos ao chegar de novo ao ponto de partida.
    - Somente aí consideramos uma chave como não existente.



Hashing



Hashing

- Vantagens:
- Simplicidade
  - É fácil de imaginar um algoritmo para implementar hashing.
- Escalabilidade
  - Podemos adequar o tamanho da tabela de hashing ao tamanho do universo dos dados esperado em nossa aplicação.
- Eficiência para ***n*** grandes
  - Para trabalharmos com problemas envolvendo tamanho do universo de entrada grande.

Hashing

- Desvantagens:
  - Dependência da escolha de função de hashing:
    - Para que o tempo de acesso ideal seja mantido, é necessário que a função de hashing divida o universo dos dados de entrada em ***b*** conjuntos de tamanho aproximadamente igual.

Hashing

- Função de Espalhamento:
  - Deve mapear chaves dentro do intervalo [0..m-1], onde m é o tamanho da tabela.
  - O ideal é que:
    - Seja simples de ser computada;
    - Para cada chave de entrada, qualquer uma das saídas possíveis, i.e índice na tabela, é igualmente provável de ocorrer.

Hashing

- Função de Espalhamento:
  - E quando a chave é alfabética?

$$K = \sum_{i=1}^{N} chave[i] \times p[i]$$

  - onde,
    - n é o número de caracteres da chave;
    - chave[i] corresponde à representação ASCII do i-ésimo caractere da chave.
    - p[i] é um inteiro de um conjunto de pesos gerados de forma aleatória.

Hashing

- Função de Espalhamento:
  - E quando a chave é numero?
  - Usa o resto da divisão por m;
    - Ou seja:

$$H(k) = k \bmod m$$

    - k: é o inteiro que identifica o elemento;
    - m: melhor que seja um número primo distante de uma potência de dois;

Hashing

- Exercício:
  - Tabelas de dispersão (tabelas hash) armazenam elementos com base no valor absoluto de suas chaves e em técnicas de tratamento de colisões. As funções de dispersão transformam chaves em endereços-base da tabela, ao passo que o tratamento de colisões resolve conflitos em casos em que mais de uma chave é mapeada para um mesmo endereço-base da tabela.

    Suponha que uma aplicação utilize uma tabela de dispersão com 23 endereços-base (índices de 0 a 22) e empregue h(x) = x mod 23 como função de dispersão, em que x representa a chave do elemento cujo endereço-base deseja- se computar. Inicialmente, essa tabela de dispersão encontra-se vazia. Em seguida, a aplicação solicita uma seqüência de inserções de elementos cujas chaves aparecem na seguinte ordem: 44, 46, 49, 70, 27, 71, 90, 97, 95.

Hashing

- Exercício:
- Com relação à aplicação descrita, faça o que se pede a seguir.

  - Escreva, no espaço reservado, o conjunto das chaves envolvidas em colisões.

  - Assuma que a tabela de dispersão trate colisões por meio de encadeamento exterior. Esboce a tabela de dispersão para mostrar seu conteúdo após a seqüência de inserções referida.

- Exercício 2:
- Suponha uma tabela de hash de tamanho M=10 com endereçamento aberto para armazenar chaves no intervalo [1, 999]. Insira as seguintes chaves nessa tabela: 371, 121, 173, 203, 11, 24, nessa ordem, considerando função hash: h(k) = k%M