Tabelas de hash

Considerações iniciais:

- Até o momento foram estudados basicamente dois tipos de estruturas de dados para o armazenamento flexível de dados:
  - Listas;
  - Árvores.
- Como vimos, cada um desses grupos possui muitas variantes.

UNIVASF

Tabelas de hash

As Listas são simples de se implementar, mas, com um tempo médio de acesso T = n/2, tornando-se impraticáveis para grandes quantidades de dados.

- Por exemplo: em uma lista com 100.000 dados, para recuperar 3 elementos em sequência faremos um número esperado de 150.000 acessos a nodos.
- Logo: listas são ótimas. Porém, para pequenas quantidades de dados.

UNIVASF

Tabelas de hash

As Árvores são estruturas mais complexas, mas que possuem um tempo médio de acesso $T = \log_G n$, onde G = ordem da árvore.

- A organização de uma árvore depende da ordem de entrada dos dados.
- Para evitar a deterioração, há modelos de árvores que se reorganizam sozinhas. As principais estudas foram AVL e Árvore B.

UNIVASF

Tabelas de hash

Imagine uma estrutura de dados que possibilite o armazenamento de uma tabela onde o acesso a um de seus registros seja efetuado diretamente.

Esta estrutura representa um modelo ideal, contudo em aplicações reais este ideal pode ser aproximado.

Ideal: Indexação direta **(transformação chave-índice)**

pesq(T:tabela, C: chave)

→ baseado em h(C:chave)-> índice

Tabelas de hash

Uma tabela hash, também chamada de tabela de dispersão ou tabela de escrutínio, trata-se de uma forma extremamente simples, fácil de se implementar e intuitiva de se organizar grandes quantidades de dados, permitindo armazenar e encontrar rapidamente dados por meio da utilização de uma chave.

A idéia central propõe a divisão de um universo de dados a ser organizado em subconjuntos mais gerenciáveis.

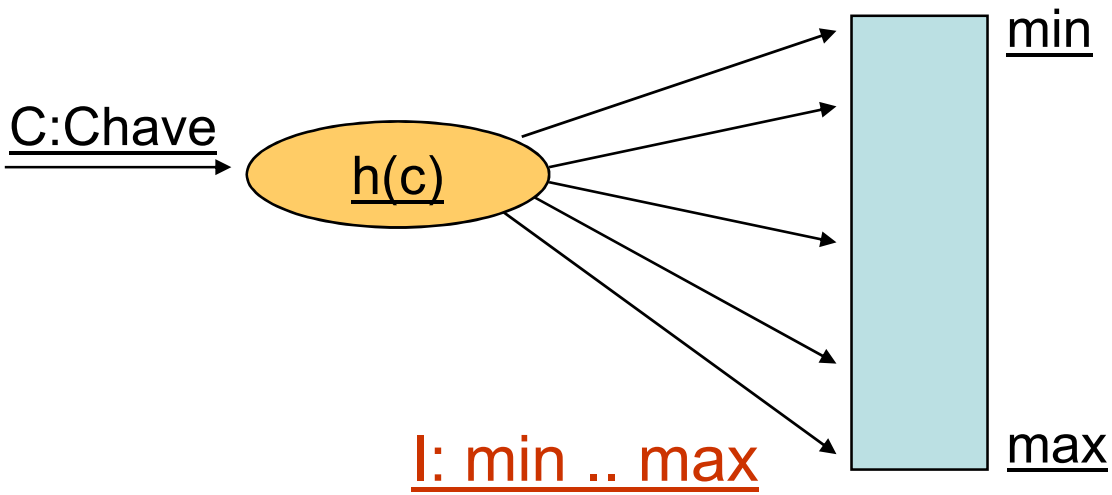
Tabelas de hash

Possui dois conceitos centrais:

Tabela de Hashing: Estrutura que permite o acesso aos subconjuntos.

Função de Hashing: Função que realiza um mapeamento entre valores de chaves e entradas na tabela.

h(chave) -> min .. max



Tabelas de hash

É interessante ressaltar que uma tabela hash possui limitações em relação às árvores, como por exemplo:

- Não permite recuperar/imprimir todos os elementos em ordem de chave nem tão pouco outras operações que exijam sequência dos dados.
- Não permite operações do tipo recuperar o elemento com a maior ou a menor chave.

UNIVASF

Tabelas de hash

- Idéia geral: Se eu possuo um universo de dados classificáveis por chave, posso:
  - 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 estes subconjuntos bem menores por algum método simples.

UNIVASF

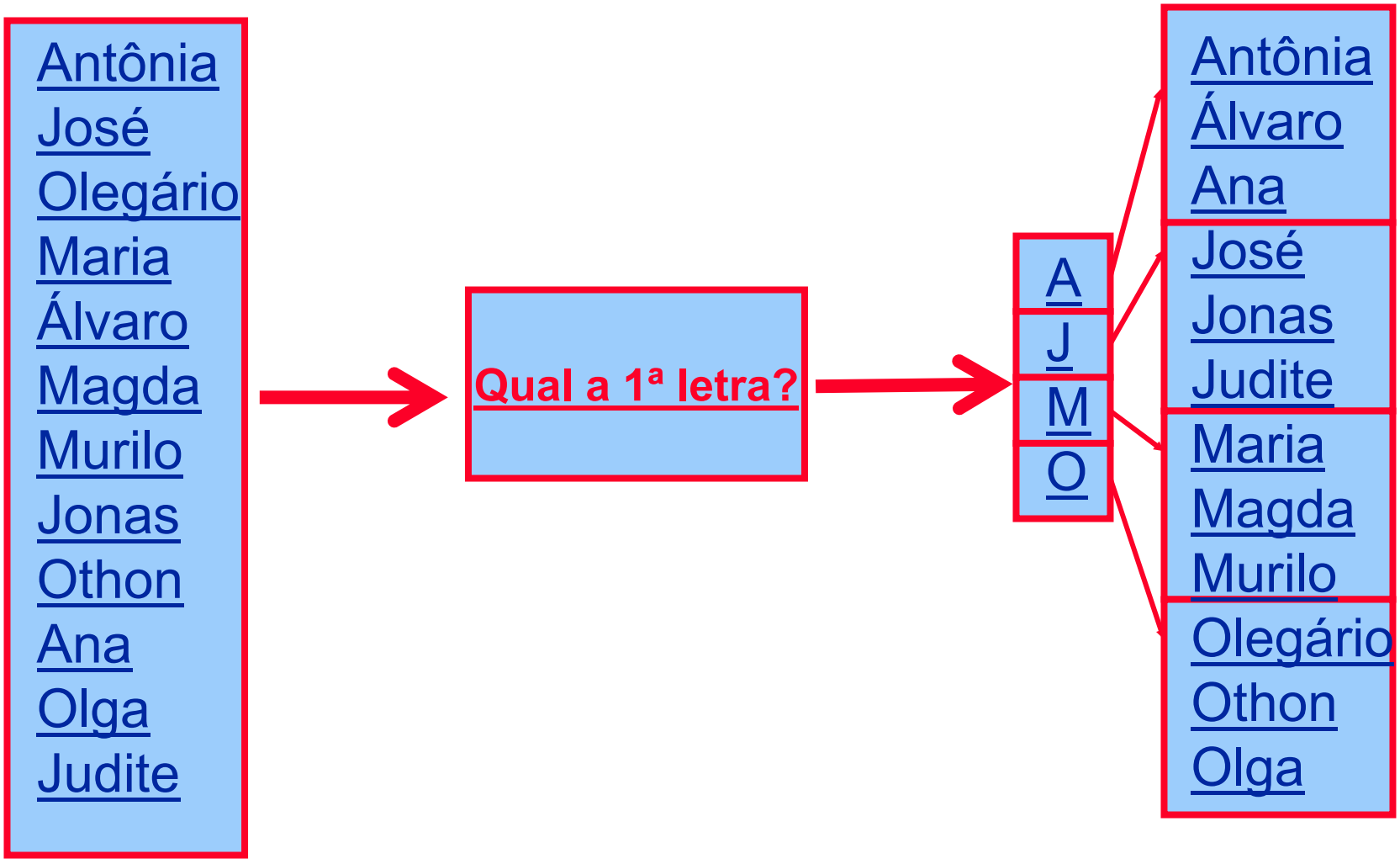
Tabelas de hash

Para isso é necessário:

- Saber quantos subconjuntos são desejados e criar uma regra de cálculo que permita, dada uma chave, determinar em qual subconjunto deve-se procurar pelos dados com esta chave ou colocar este dado, caso seja um novo elemento. (**função de hashing**).
- Possuir um índice que permita encontrar o início do subconjunto certo, depois de calcular o hashing. (**tabela de hashing**).
- Possuir uma ou um conjunto de estruturas de dados para os subconjuntos. **hashing fechado** (***endereçamento aberto)*** ou o **hashing aberto** (***encadeado***).

UNIVASF

Tabelas de hash: Exemplo



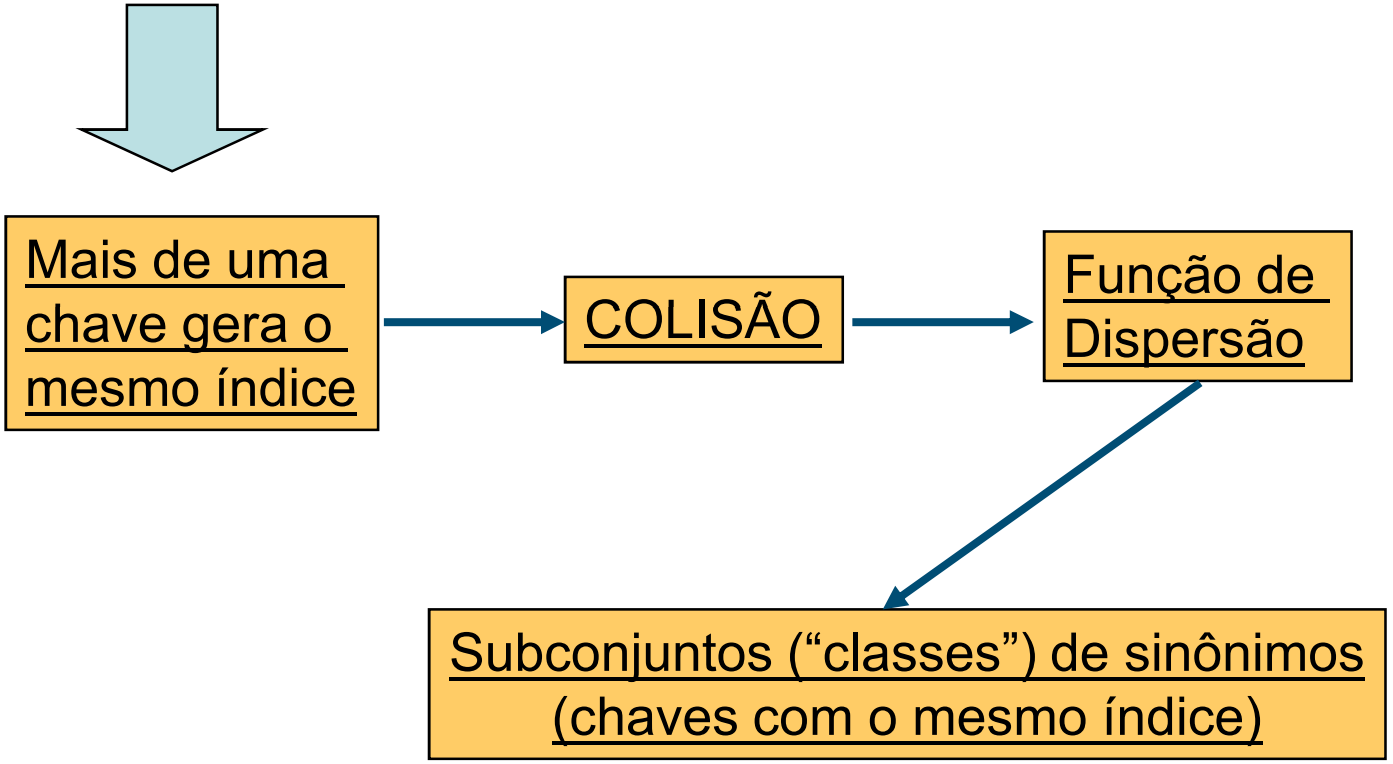
Tabelas de hash

- Devemos garantir um tratamento coerente das operações de inserção e pesquisa na tabela
  - **Inserir**: calcula-se endereço da nova entrada usando a função h
  - **Pesquisa**: mesma função deve informar o mesmo índice para a mesma entrada

UNIVASF

Tabelas de hash

- chaves -> índices



Tabelas de hash

- Implementação de tabelas por cálculo de endereço envolve dois problemas
  - Escolha da função de dispersão adequada para o tipo de chave usada
    - gerar endereços em todo o intervalo [min..max]
    - estabelecer distribuição uniforme na tabela
  - Adoção de um processo de solução das colisões

Tabelas de hash: Nomenclatura

n: Tamanho do universo de dados.

b: Número de subconjuntos em que dividimos os dados: **depósitos**.

s: **Capacidade de cada depósito** (quando for limitada. Aplica-se somente a hashing fechado ou endereçamento aberto).

T: Cardinalidade do domínio das chaves. Quantas chaves podem existir ?

n*/*T: **Densidade identificadora**.

$\alpha = n/(b \cdot s)$: **Densidade de carga**. $b \cdot s$ fornece a capacidade máxima (quando existir explicitamente). $n/(b \cdot s)$ indica o fator de preenchimento. Aplica-se somente a hashing fechado (endereçamento aberto).

Hashing Aberto ou Encadeado

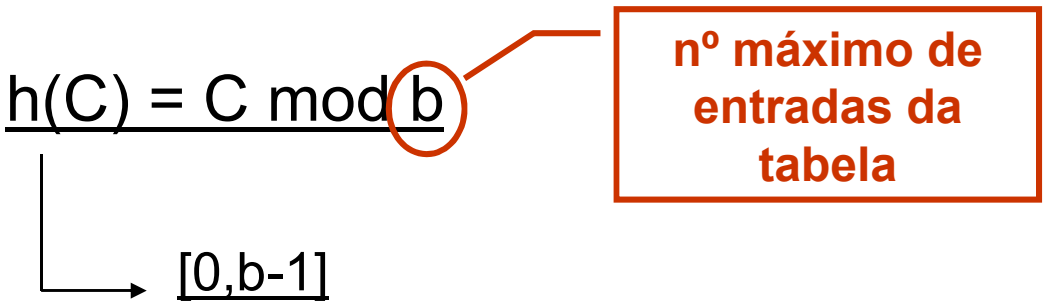
- Forma mais intuitiva de se implementar o conceito de Hashing.
- Utiliza a idéia de termos uma tabela com ***b*** entradas, cada uma como cabeça de lista para uma lista representando o conjunto b_i.
  - Calculamos a partir da chave qual entrada da tabela é a cabeça da lista que queremos.
  - Utilizamos uma técnica qualquer para pesquisa dentro de b_i. Tipicamente será a técnica de pesquisa sequencial em lista encadeada.
  - **Podemos utilizar qualquer outra estrutura para representar os b_i. Uma árvore, por exemplo, poderia ser uma opção.**

UNIVASF

Propostas de Função de Dispersão

- Um dos métodos mais simples para criar funções hashing é o método da divisão. No qual uma chave **C** é mapeada para um dos **b** endereços da tabela hashing.

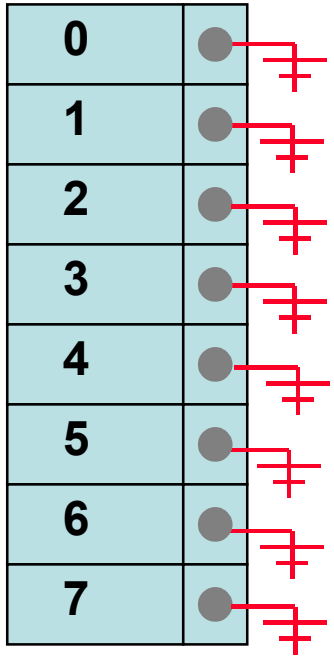
- **Resto da Divisão**



Hashing Aberto ou Encadeado: Exemplo

Para uma melhor compreensão analisaremos um exemplo no qual uma tabela com 8 entradas é utilizada para acomodar registros cujas chaves são valores pertencentes ao conjunto dos números naturais.

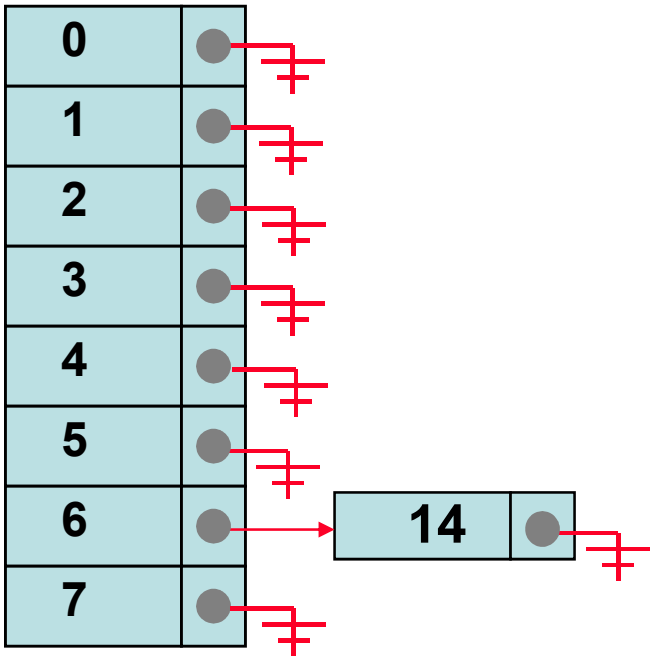
Inicialmente a tabela está vazia.



Hashing Aberto ou Encadeado: Exemplo

Inserir chave 14.

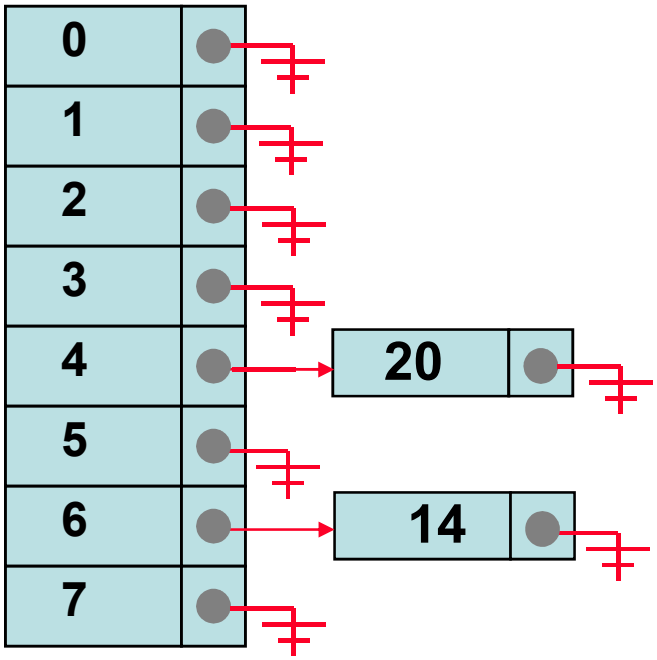
14%8=6



Hashing Aberto ou Encadeado: Exemplo

Inserir chave 20.

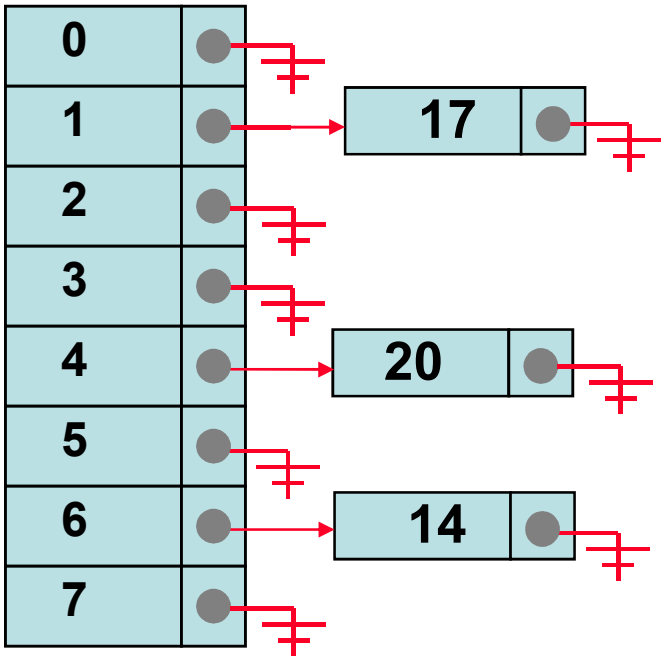
20%8=4



Hashing Aberto ou Encadeado: Exemplo

Inserir chave 17.

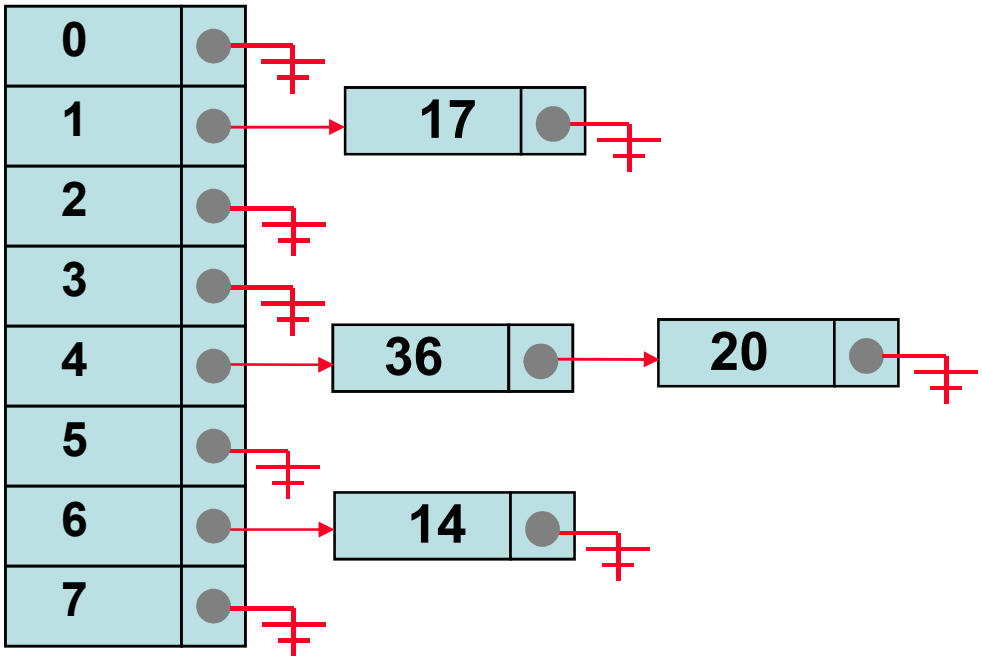
17%8=1



Hashing Aberto ou Encadeado: Exemplo

Inserir chave 36.

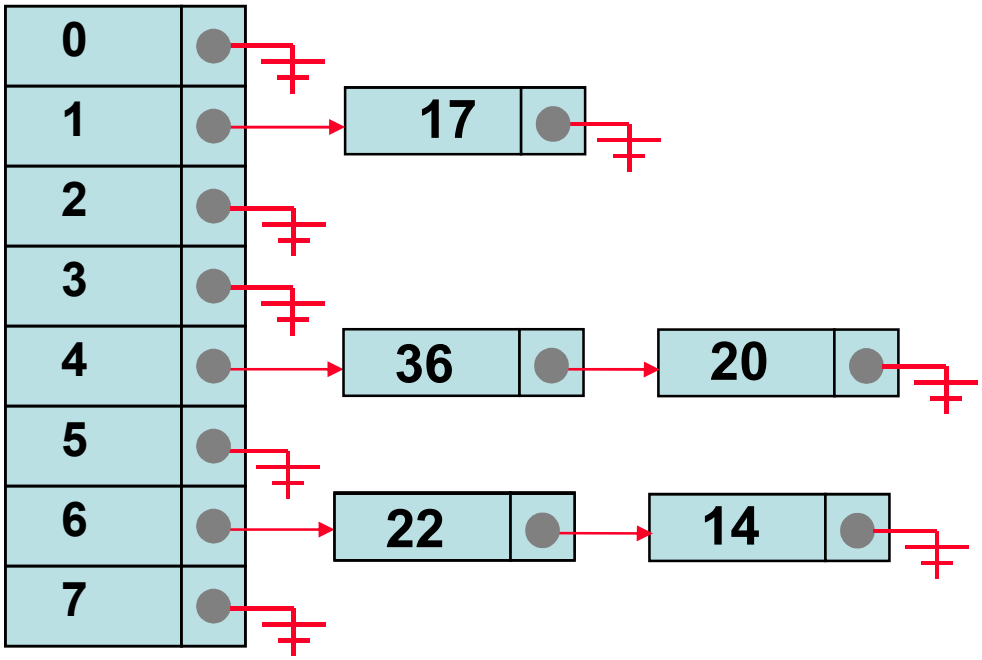
36%8=4



Hashing Aberto ou Encadeado: Exemplo

Inserir chave 22.

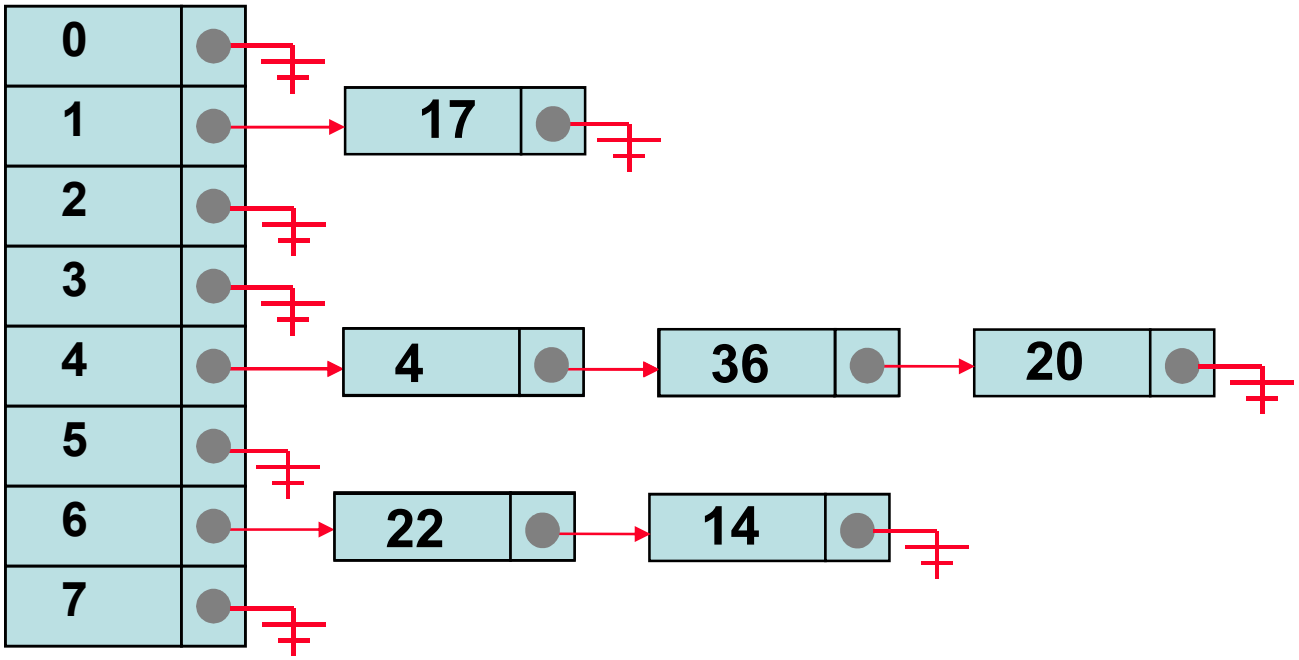
22%8=6



Hashing Aberto ou Encadeado: Exemplo

Inserir chave 4.

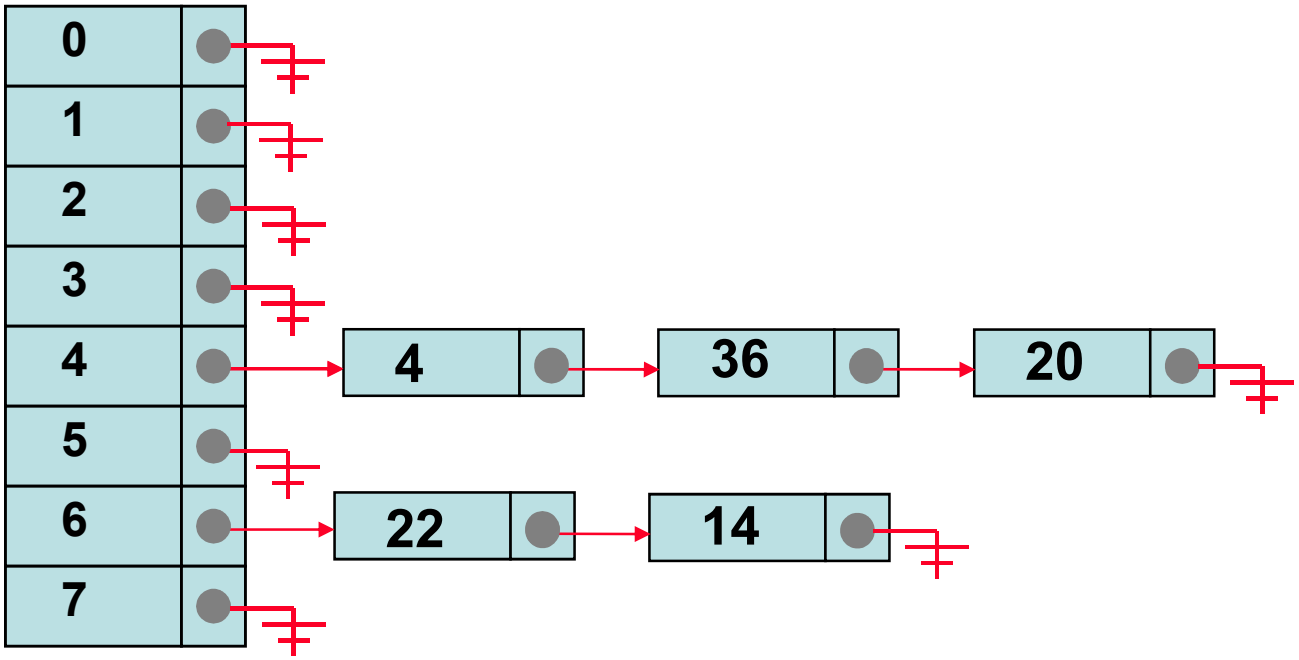
4%8=4



Hashing Aberto ou Encadeado: Exemplo

Remover chave 17.

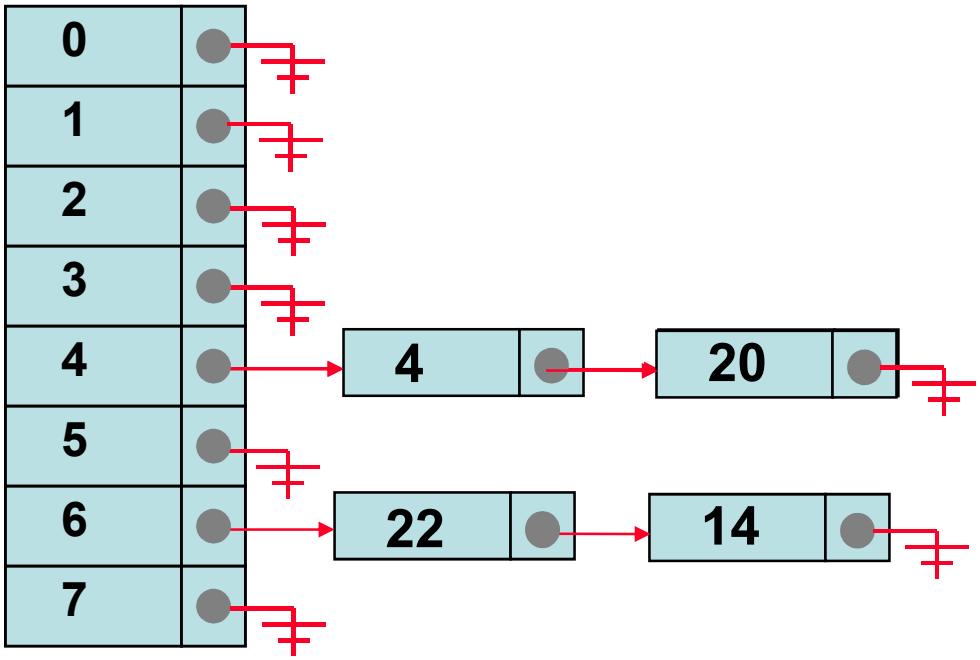
17%8=1



Hashing Aberto ou Encadeado: Exemplo

Remover chave 36.

36%8=4



Hashing Aberto ou Encadeado: Exercício

Com base no que foi apresentado defina a(s) estrutura(s) de dados necessária(s) para a implementação de uma tabela hashing aberta.

```
#define numEntradas 8
typedef struct _Hash
{
   int chave;
   struct _Hash *prox;
} Hash;
typedef Hash* Tabela[numEntradas];
```


UNIVASF

Hashing Aberto ou Encadeado: Exercício

Agora, implemente a função de inserção de uma chave na tabela hashing aberta em questão.

```
void inserirHash(Tabela tabela, int n)
{
  Hash* novo;
  int pos = funcaoHashing(n);
  novo = (Hash *) malloc (sizeof (Hash));
  novo->chave = n;
  novo->prox = tabela[pos];
  tabela[pos] = novo;
}
```