Tabelas de Dispersão (ou Hash)

Nelson Cruz Sampaio Neto
nelsonneto@ufpa.br

05 de maio de 2016

Introdução

- Em geral, os métodos de pesquisa existentes são baseados na comparação da chave de busca com as chaves já armazenadas na estrutura, ou mesmo na utilização de *bits* da chave de pesquisa para escolher o caminho a seguir.

- **Conceito de hash**: Os registros armazenados em uma tabela são endereçados a partir de uma transformação aritmética sobre a chave de pesquisa.

- **Objetivo**: Ter eficiência $O(1)$ nas operações de busca, inserção e remoção. Para isso, as inserções e remoções não devem provocar grandes variações na quantidade de registros armazenados.

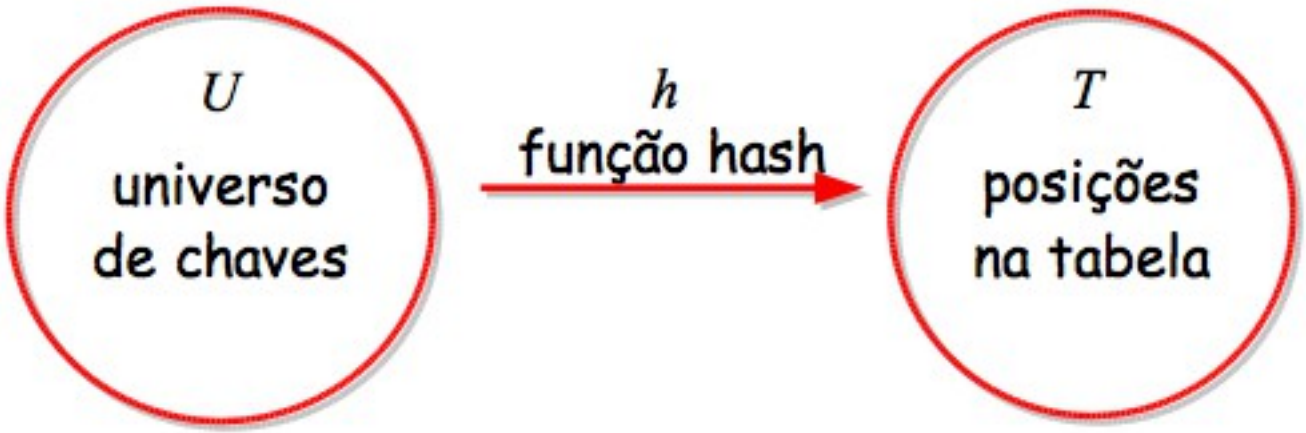
Introdução

- O método de pesquisa conhecido como hash (tabela de dispersão) é constituído de duas etapas principais:

1. Computar o valor da função de dispersão (ou **função hash**), a qual transforma a chave de pesquisa em um endereço da tabela.

2. Considerando que duas ou mais chaves podem ser transformadas em um mesmo endereço de tabela, é necessário existir um método para lidar com as **colisões**.

Introdução

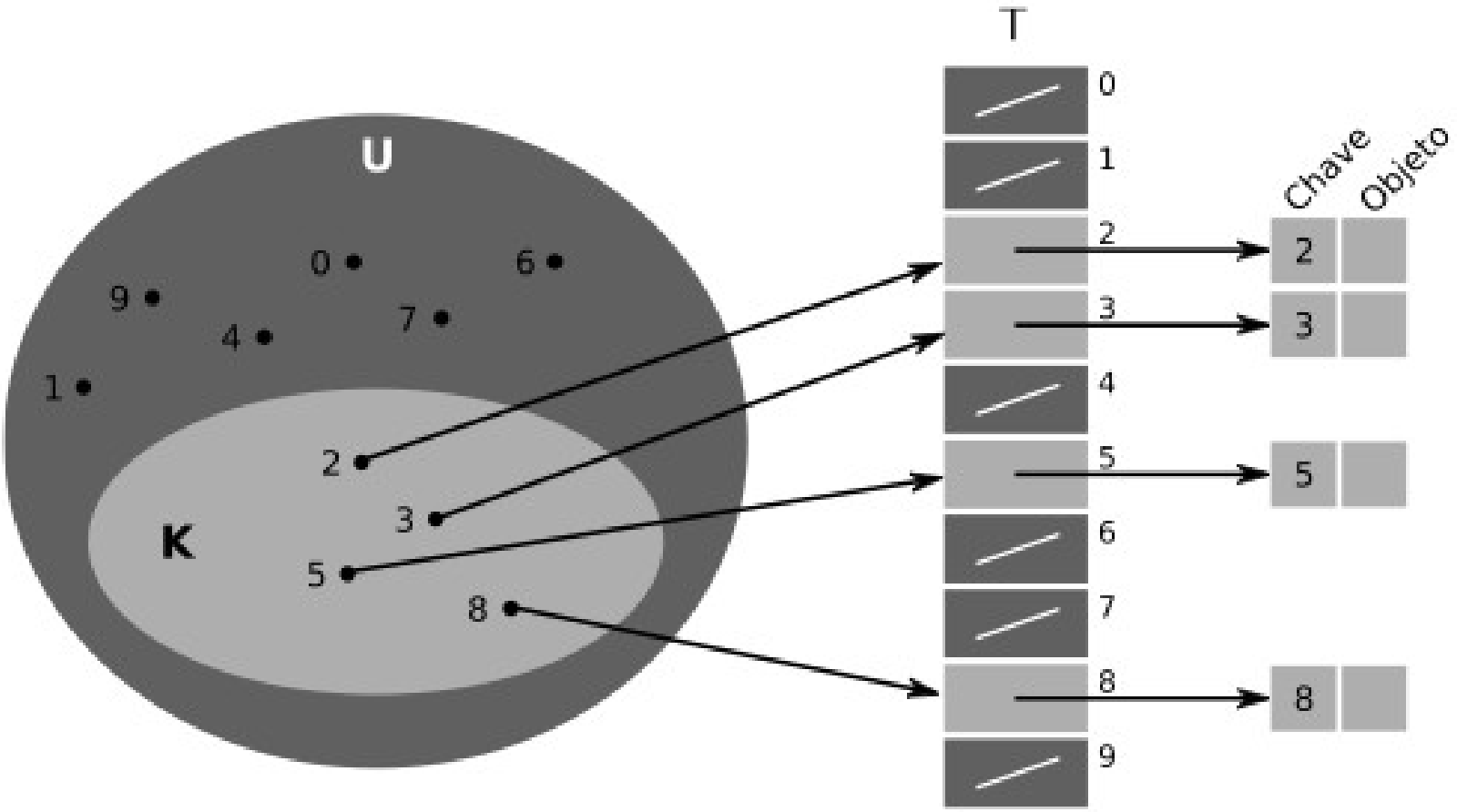


Endereçamento direto

- Quando o universo de chaves *U* é pequeno, podemos alocar uma tabela com uma posição para cada chave, ou seja, |T| = |U|.

- Então, cada posição da tabela, que pode ser implementada como um vetor, representa uma chave de *U* e armazena um elemento ou um ponteiro para o elemento.

Endereçamento direto



Endereçamento direto

- Contudo, nem sempre *U* é pequeno!
- Suponha que a chave seja a matrícula de um aluno da UFPA...
  - Trata-se de um número inteiro de 7 dígitos.
  - Logo, são 10.000.000 chaves (ou elementos).
  - Se cada posição da tabela ocupar míseros 127 bytes, precisamos de mais de 1 Gbyte de memória apenas para a tabela... mesmo que ela não esteja cheia.

Tabelas Hash

- Chamaremos de *K* o conjunto de chaves que serão efetivamente armazenadas na tabela.

- Logo, nossa tabela deveria ter dimensão |K|, mais que isso seria desperdício de memória.

- **Problema**: Na prática, os elementos de *K* não são conhecidos e |U| >> |K|.

- Então, como podemos fazer isso?

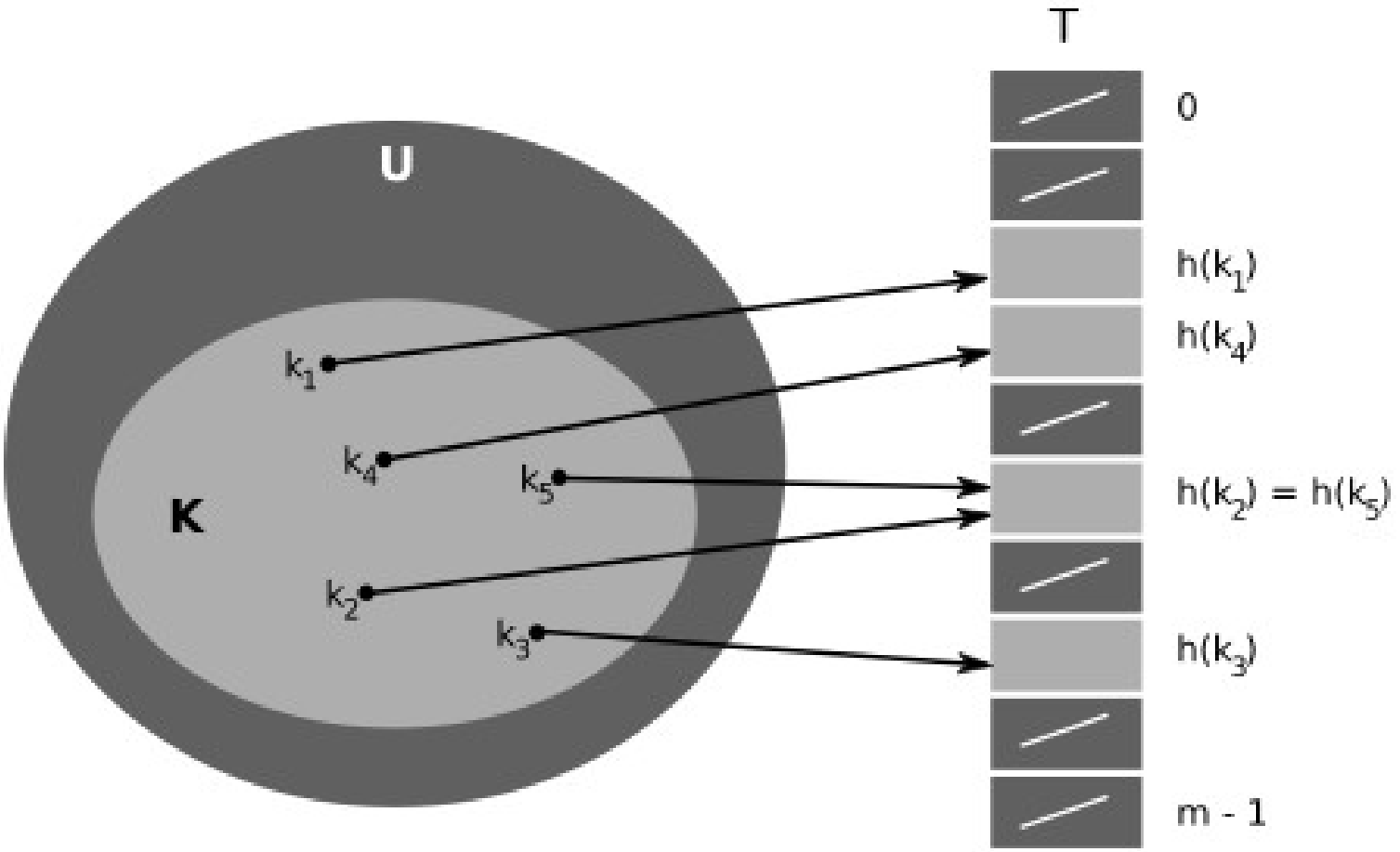
Tabelas Hash

- **Solução**: Utilizar uma **função hash** *h* para mapear chaves em inteiros dentro do intervalo [0..m – 1], no qual *m* é o tamanho da tabela.

- A tabela é implementada como um vetor em que cada posição armazena um subconjunto de *U*.

- Com isso, é possível que mais de uma chave seja mapeada em uma única posição da tabela, o que resulta no que chamaremos de **colisão**.

Tabelas Hash

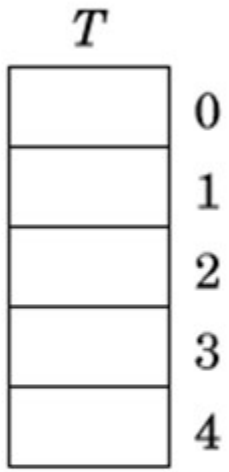


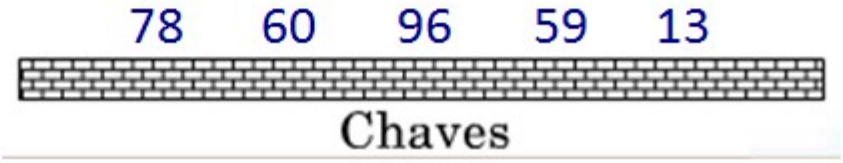
Tabelas Hash

- Idéia: transformar a chave x num endereço-base $h(x)$, que é um valor entre 0 e m-1.
  h é chamada função de dispersão.

- Exemplo

$$h(x) = x \bmod 5$$

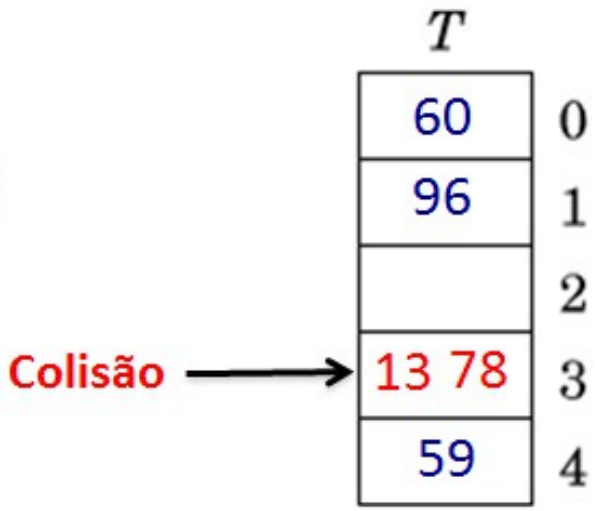




Tabelas Hash

- Exemplo

$$h(x) = x \bmod 5$$



Método da divisão: Para números reais x e y, a operação binária *mod* é definida como *x mod y = x – y * piso(x/y)*, se y ≠ 0.

Como evitar colisões?

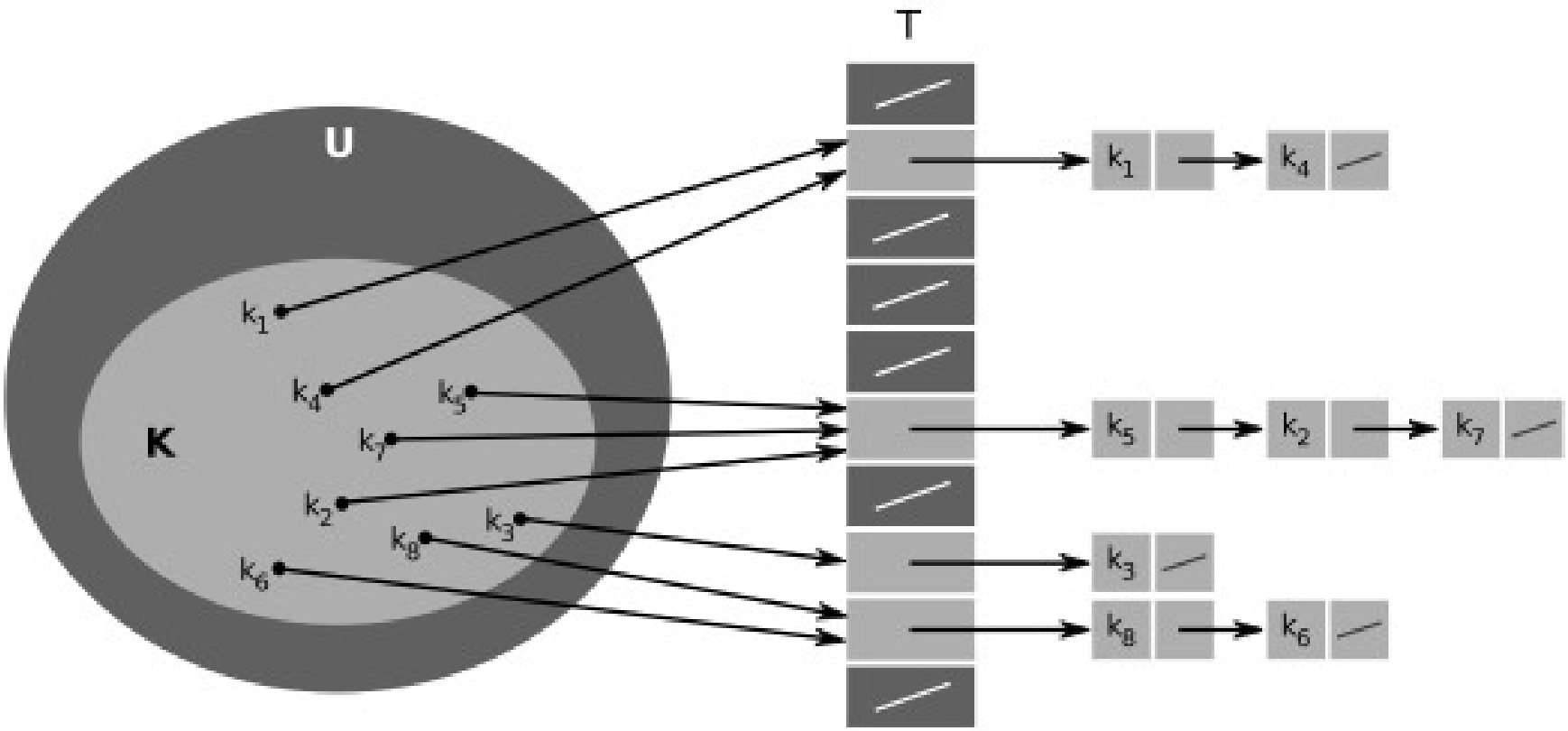
- Claramente, o número de colisões depende de como a função hash mapeia as chaves na tabela.
- P.e. um número primo não muito próximo a uma potência exata de 2 normalmente é uma boa escolha para o tamanho da tabela no método da divisão.
- O fato é que como $|U| > m$, a escolha da função hash apenas minimiza o número de colisões.
- **Solução**: As colisões remanescentes serão tratadas de forma algorítmica, aplicando as técnicas de endereçamento aberto ou encadeamento.

Encadeamento

- Uma das formas de resolver colisões é construir uma **lista linear** para cada endereço da tabela.

- Assim, todas as chaves com mesmo endereço são encadeadas em uma lista linear.

Encadeamento



Exercícios

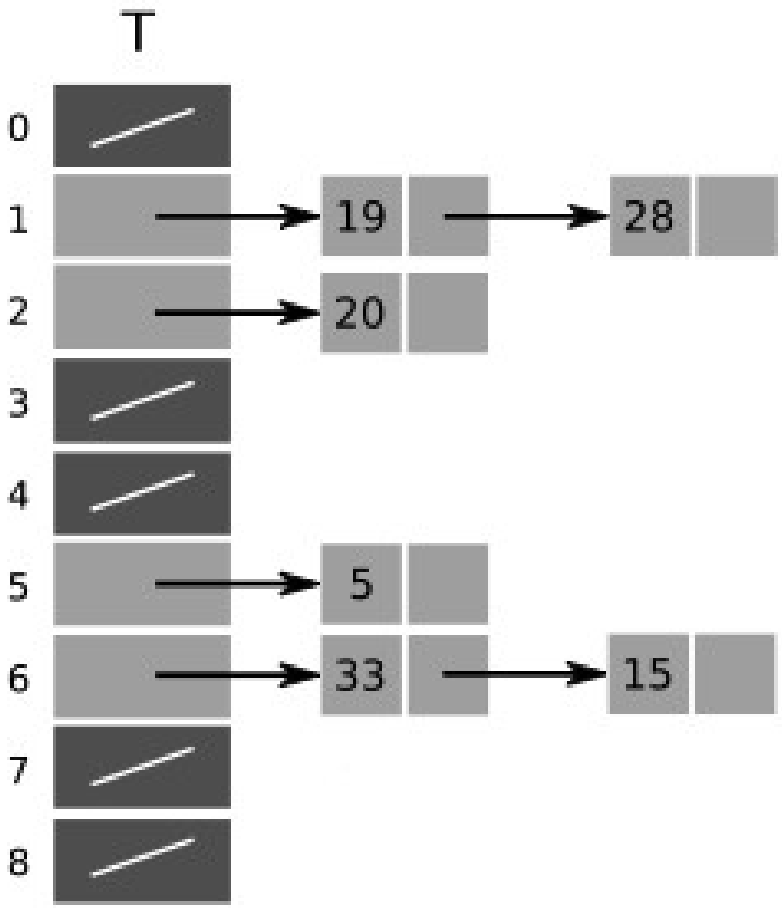
1. Insira as chaves {5, 28, 19, 15, 20, 33} em uma tabela *T* com 9 posições [0..8], utilizando a função hash: **h(k) = k mod 9**.

2. Se a *i*-ésima letra do alfabeto é representada pelo número *i* e a função dispersão **h(chave) = chave mod M** é utilizada para M = 7, então mostre o resultado da inserção das seguintes chaves na tabela: P E S Q U I S A.

Obs: Para números reais *x* e *y*, a operação *mod* é definida como

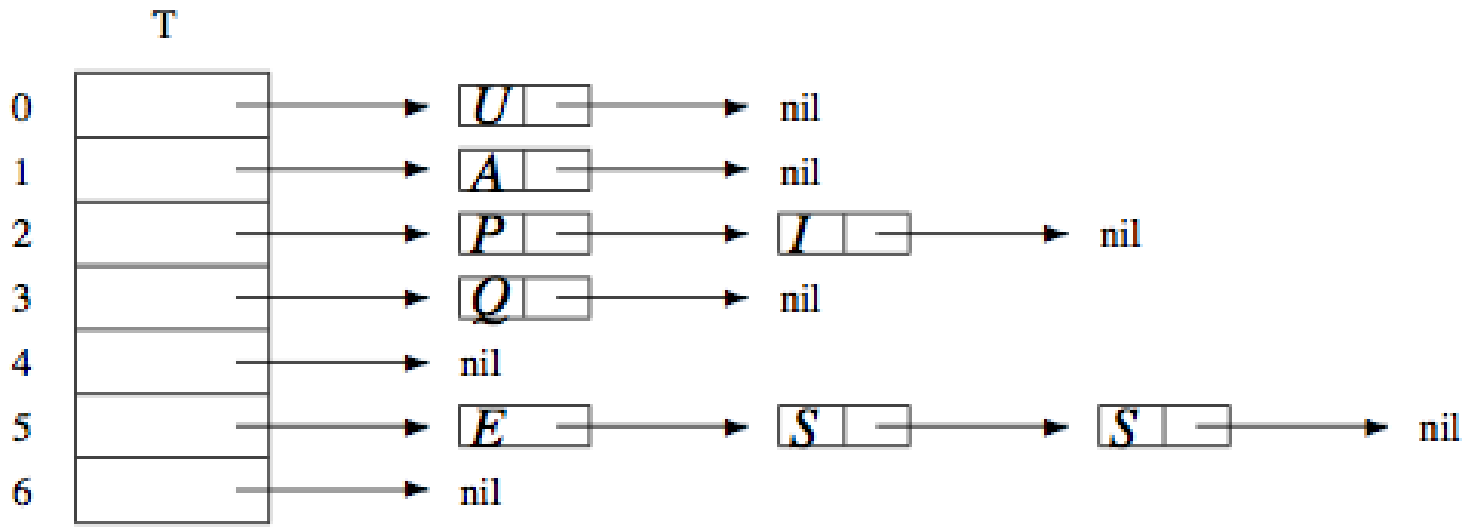
x mod y = x – y * piso(x/y)

Exercício 1



Exercício 2

- Por exemplo, $h(A) = h(1) = 1$, $h(E) = h(5) = 5$, $h(S) = h(19) = 5$, e assim por diante.



Encadeamento

- É usual efetuar-se a inclusão de uma nova chave *x* no final da lista correspondente ao endereço *h(x)*.

- A ideia é que a lista será percorrida de qualquer maneira, para assegura que *x* não pertence à mesma.

- Mas, caso chaves repetidas sejam aceitas, essa condição pode ser relaxada, e a chave inserida no início da lista.

Encadeamento

- **Análise**: Com uma “boa” função hash, assume-se que qualquer item do conjunto de chaves tem igual probabilidade de ser endereçado para qualquer entrada da tabela.

- Então, o comprimento esperado de cada lista encadeada é n/m, chamado de **fator de carga**, em que n representa o número de registros na tabela e m o tamanho da tabela.

Encadeamento

- Pela análise anterior, as operações pesquisa, insere e retira custam $O(1 + n/m)$ **em média**, sendo que a constante 1 representa o tempo para encontrar a entrada da tabela (ou seja, calcular a função hash), e n/m o tempo para percorrer a lista.

- Para valores de m próximos de n, o tempo torna-se **constante**, isto é, independente de n.

Encadeamento

- No **pior caso**, todas as chaves são mapeadas para a mesma posição e a busca custa $\theta(n)$ mais o cálculo da função hash.

- Qual a validade dessa análise?
  - A função hash tem que ser terrível.
  - Pouquíssimas chances de ocorrer na prática.

Encadeamento interno

- Em algumas aplicações não é desejável a manutenção de uma estrutura exterior à tabela de dispersão.

- Nessa situação, ainda é possível resolver as colisões usando listas encadeadas, desde que estas compartilhem o mesmo espaço de memória que a tabela.

Encadeamento interno

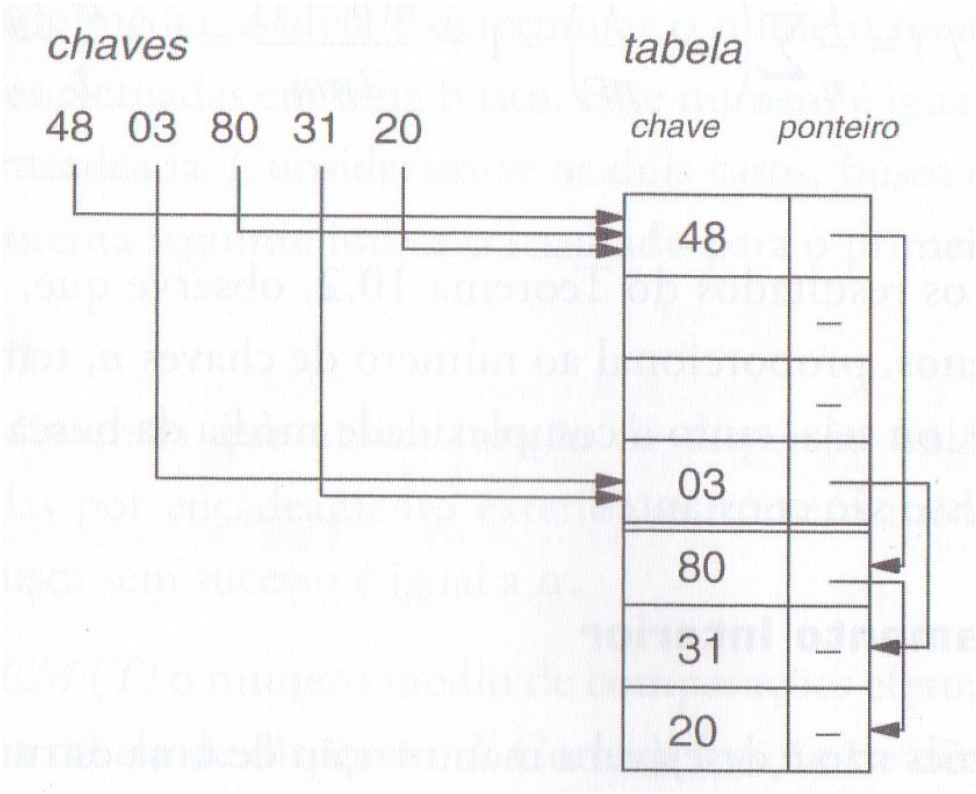
- O encadeamento interno prevê a divisão da tabela em duas zonas, uma de endereço-base (tamanho *p*) e outra de colisões (tamanho *s*).

- Naturalmente, $p + s = m$, e os valores de *p* e *s* são fixos.

- Nesse caso, n/m é menor ou igual a 1.

Encadeamento interno

- Exemplo: Há um total de *n = 5* chaves, com uma tabela de tamanho *m = 7*, sendo dividida em *p = 4* e *s = 3*. Sendo h(x) = x mod 4.



Encadeamento interno

- Aumentando-se o tamanho da zona de colisão pela correspondente diminuição da zona de endereço-base, diminui a possibilidade de ocorrência de "falso" overflow.

- Porém, a eficiência da tabela de dispersão também diminui. No caso limite de $p = 1$ e $s = m - 1$, a tabela se reduz a uma lista encadeada, cujo tempo médio de busca é $O(n)$.

Encadeamento interno

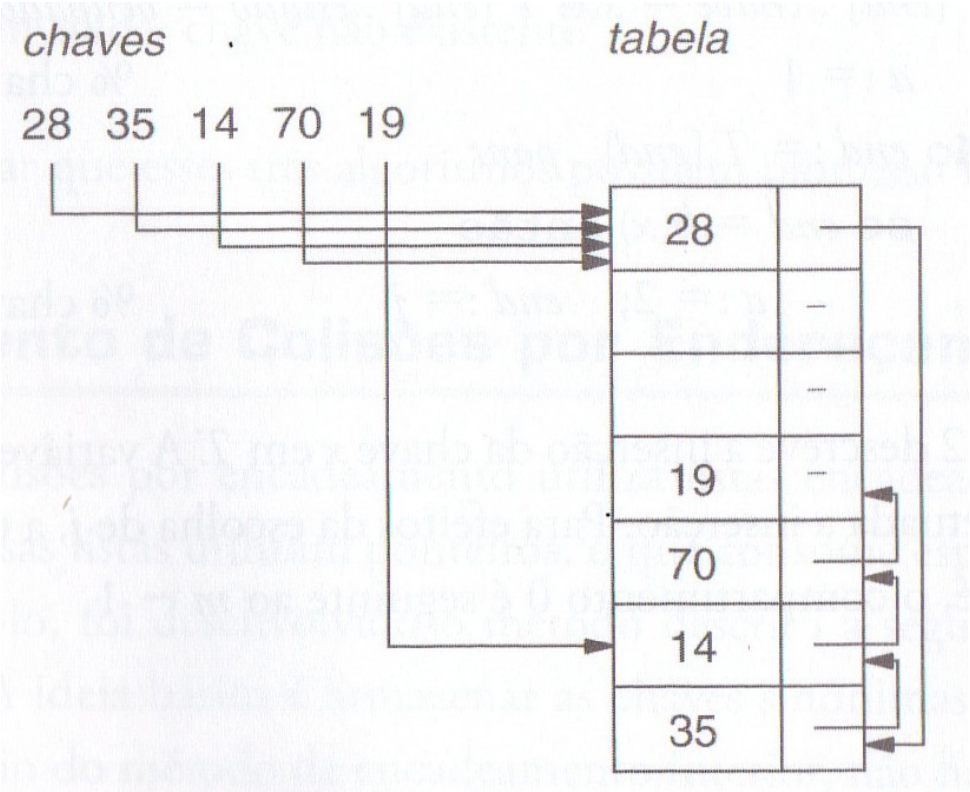
- Uma outra técnica consiste em não diferenciar as duas zonas da tabela, ou seja, qualquer endereço da tabela pode ser de base ou de colisão.

- Essa técnica pode gerar o efeito indesejado conhecido como **colisões secundárias**, ou seja, aquelas provenientes da coincidência de endereços para chaves que não possuem a mesma reposta para a função hash.

- A operação de remoção exige cuidados!

Encadeamento interno

- Exemplo: A colisão secundária se verifica na inclusão da chave 19. Existe a fusão de listas que contêm chaves com diferentes endereços-base. Considerar h(x) = x mod 7.



Endereçamento aberto

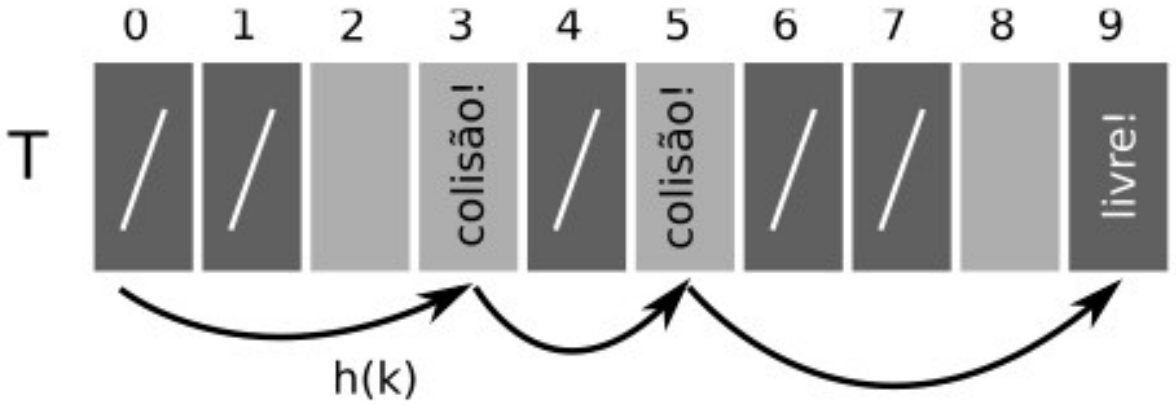
- Em uma tabela de hash com endereçamento aberto, todos os elementos são armazenados na própria tabela e sem o uso de listas encadeadas.

- Logo, o fator de carga não pode exceder o valor 1.

- O espaço gasto com encadeamento é economizado e a colisão é tratada com a busca de uma posição vazia na própria tabela para inserção.

Endereçamento aberto

- Quando uma chave *k* é endereçada para uma entrada da tabela que já esteja ocupada (**colisão**), uma sequência de localizações alternativas $h_1(k)$, $h_2(k)$, ... é escolhida.



- Mas se nenhuma das $h_1(k)$, $h_2(k)$, ... posições está vazia, então a tabela está cheia e não podemos inserir *k*.

Endereçamento aberto

- Existem várias propostas para a escolha de localizações alternativas.
- A mais simples é a **hash linear**, na qual a posição h_j na tabela é dada por:

$$h_j = (h(k) + j) \bmod m , \text{ para } 1 \leq j \leq m - 1$$

- Desvantagem: É suscetível ao **agrupamento primário**, isto é, são construídas longas sequências de posições ocupadas, o que degrada o desempenho da busca.

Exercícios

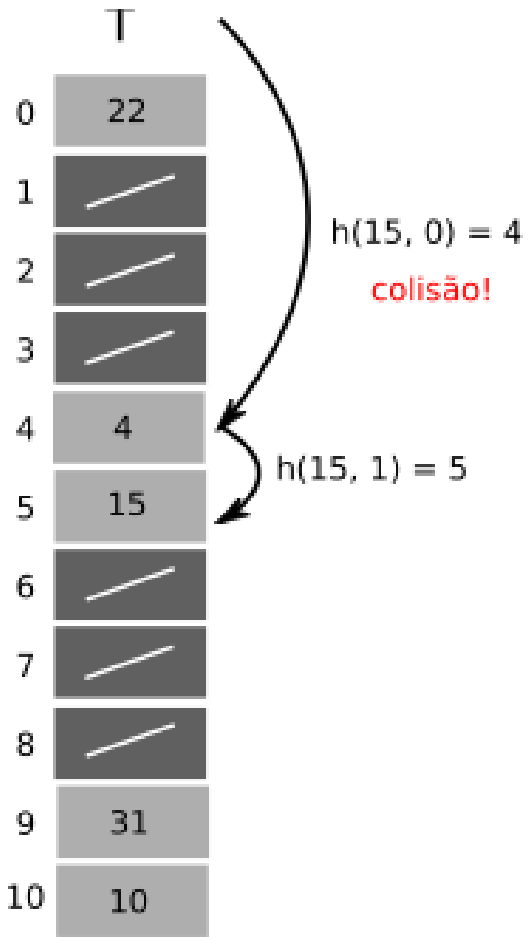
1. Insira as chaves {10, 22, 31, 4, 15, 28, 59} em uma tabela *T* de tamanho 11 com hash linear e função: **h(k) = k mod 11**.

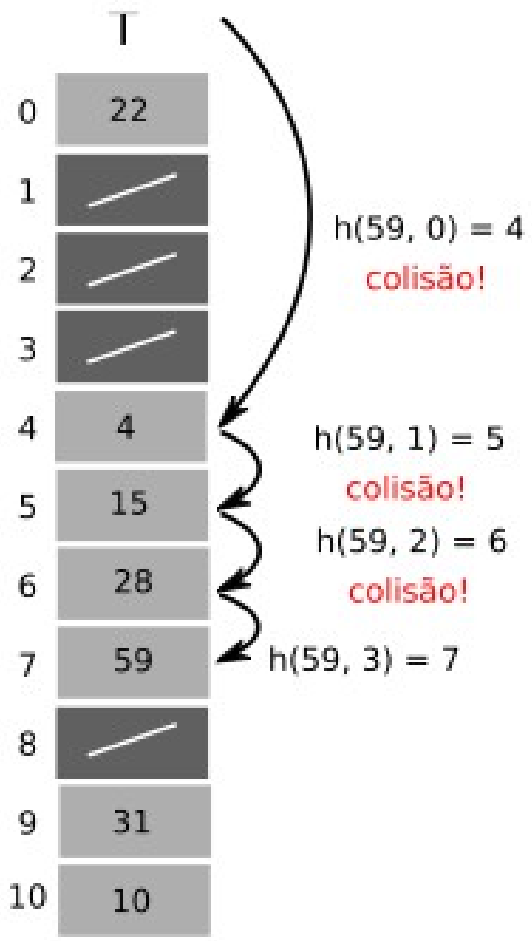
2. Se a *i*-ésima letra do alfabeto é representada pelo número *i* e a função dispersão **h(chave) = chave mod M** é utilizada para M = 7, então apresente a inserção das chaves L U N E S na tabela usando hash linear para resolver colisões.

Obs: Para números reais *x* e *y*, a operação *mod* é definida como

x mod y = x – y * piso(x/y)

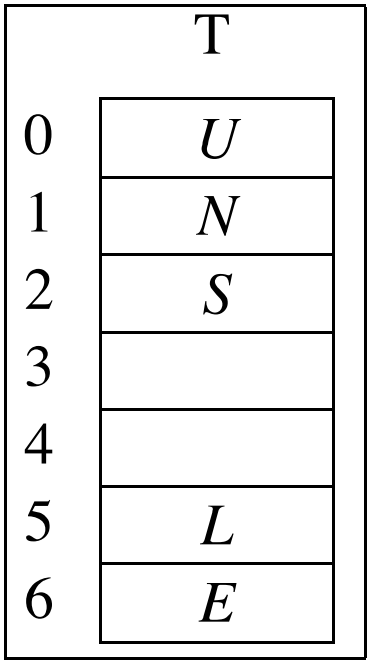
Exercício 1


T
0
22
1
2
3
4
4
5
15
6
7
8
9
31
10
10
h(15, 0) = 4
colisão!
h(15, 1) = 5


T
0
22
1
2
3
4
4
5
15
6
28
7
59
8
9
31
10
10
h(59, 0) = 4
colisão!
h(59, 1) = 5
colisão!
h(59, 2) = 6
colisão!
h(59, 3) = 7

Exercício 2

- Por exemplo, $h(L) = h(12) = 5$, $h(U) = h(21) = 0$, $h(N) = h(14) = 0$, $h(E) = h(5) = 5$, e $h(S) = h(19) = 5$.



Endereçamento aberto

- Outro método bastante conhecido para encontrar as localizações alternativas é o **hash quadrático**.

$$h_j = (h(k) + c_1 j + c_2 j^2) \bmod m , \text{ para } 1 \leq j \leq m - 1$$

sendo c_1 e c_2 constantes.

- Evita o agrupamento primário. Porém, as sequências de teste ainda são idênticas para duas chaves com o mesmo mapeamento, é o chamado **agrupamento secundário**.

Endereçamento aberto

- Os valores de m, c_1 e c_2 devem ser escolhidos de tal forma que as localizações alternativas correspondam a varrer toda a tabela.

- A equação abaixo fornece uma maneira de calcular, diretamente, esses endereços:

$$h_j = (h_{j-1} + j) \bmod m , \text{ para } 1 \leq j \leq m - 1$$

- Se m for potência de 2, os endereços obtidos por essa equação correspondem à varredura de toda a tabela.

Exercício

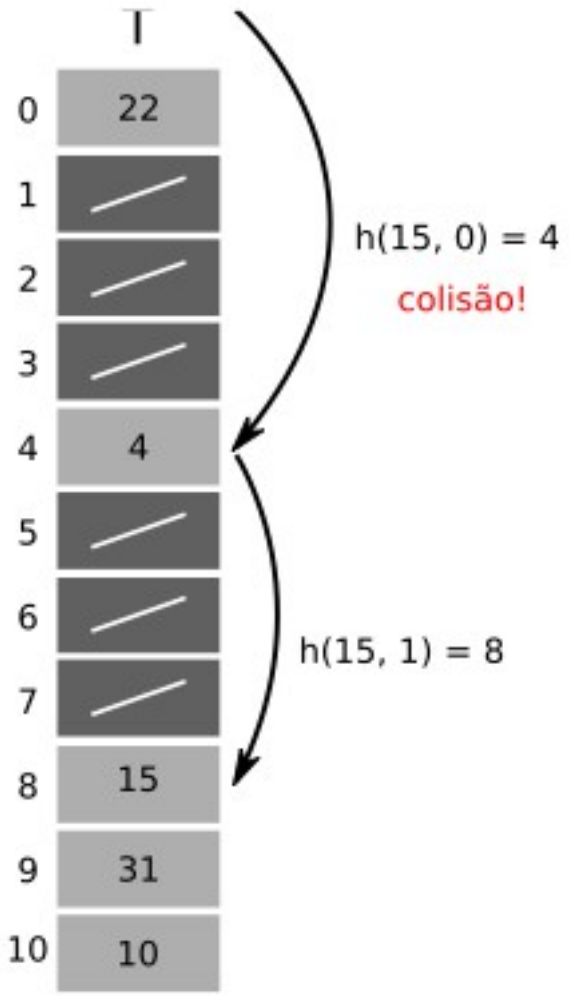
- Insira as chaves {10, 22, 31, 4, 15, 28, 59} em uma tabela *T* de tamanho 11 com hash quadrático e função: **h(k) = k mod 11**.

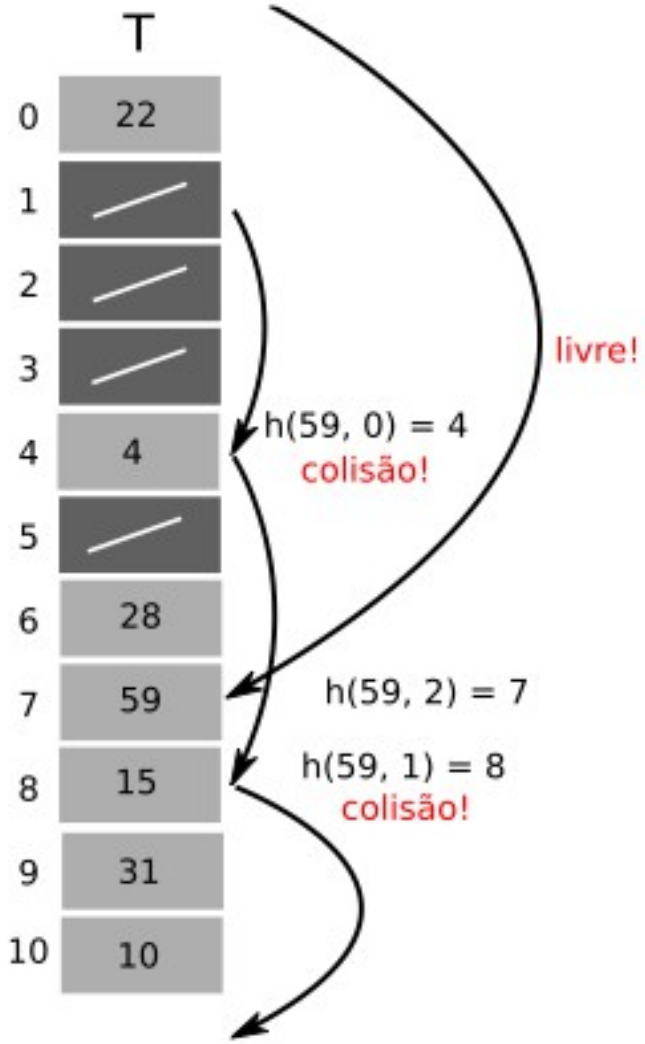
  As constantes c_1 e c_2 são iguais a 1 e 3, respectivamente.

Obs: Para números reais *x* e *y*, a operação *mod* é definida como

x mod y = x – y * piso(x/y)

Exercício


T
0
22
1
2
3
4
4
5
6
7
8
15
9
31
10
10
h(15, 0) = 4
colisão!
h(15, 1) = 8


T
0
22
1
2
3
4
4
5
6
28
7
59
8
15
9
31
10
10
livre!
h(59, 0) = 4
colisão!
h(59, 2) = 7
h(59, 1) = 8
colisão!

Endereçamento aberto

- Outro método é o **hash duplo**:

$$h_j = (h(k) + j\, d(k)) \bmod m \text{ , para } 1 \leq j \leq m - 1$$

- Projeto e implementação mais difíceis que os métodos apresentados anteriormente.

- No entanto, não causa agrupamento do tipo produzido pelo teste linear ou pelo teste quadrático, e apresenta melhor desempenho na média.

Endereçamento aberto

- Para varrer toda a tabela, é necessário que *d(k)* e *m* sejam primos entre si, ou seja, o único divisor comum a eles é o número 1.

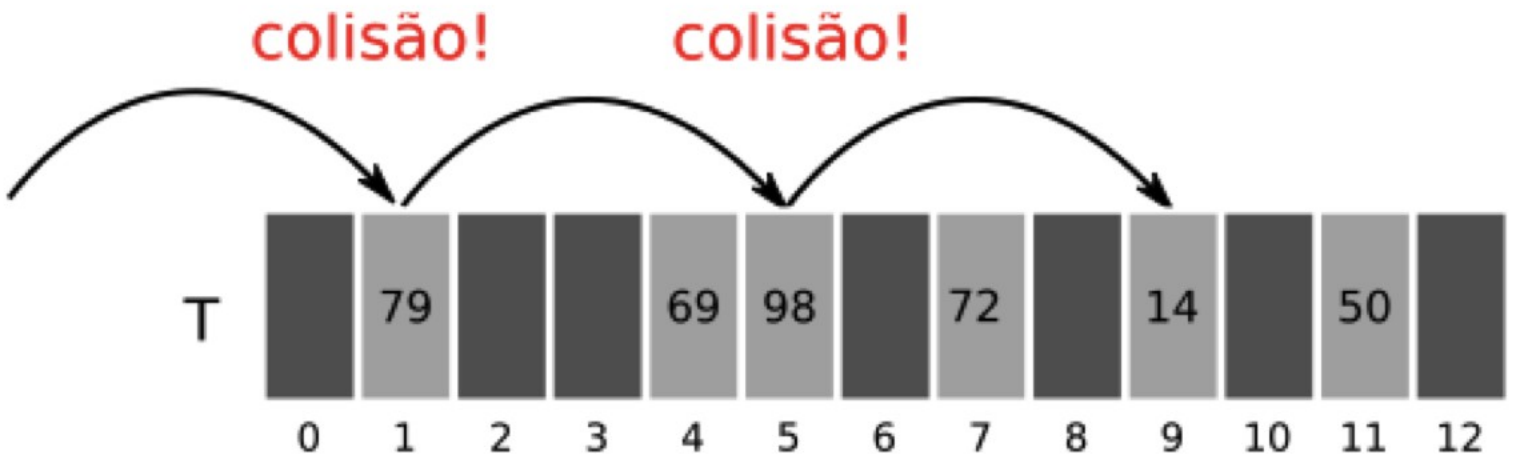
- Por exemplo, se *m* for potência de 2, basta definir *d(k)* de forma a produzir números ímpares.

- Ou então, mais simples ainda, basta definir *m* como um número primo e projetar *d(k)* de forma que ele sempre retorne um inteiro positivo menor que *m*.

Exemplo

Sejam $h(k) = k \bmod 13$ e $\mathrm{d}(k) = 1 + (k \bmod 11)$.

Então $h(14, 0) = 1$, $h(14, 1) = 5$ e $h(14, 2) = 9$.



Endereçamento aberto

- O algoritmo de pesquisa (ou busca) percorre a mesma sequência de posições examinada pelo algoritmo de inserção quando a chave *k* foi inserida.

- Após a remoção, a posição **não** pode ser deixada como uma célula vazia, pois pode interferir nas buscas.

- A posição deve ser marcada de alguma maneira (com uma variável booleana, por exemplo) para que na busca possa-se saber que havia algo lá.

Endereçamento aberto

- Considerando um mapeamento uniforme, o número **médio** de comparações em uma busca sem sucesso e na inserção é limitado por:

$$\sum_{i=1}^{\infty} \alpha^{i-1} = \frac{1}{1-\alpha} = O(1).$$

sendo $\alpha = n/m$ o fator de carga da tabela.

- O aspecto negativo está relacionado com o **pior caso**, que é *O*(n), se a função hash não conseguir espalhar os registros de forma razoável pelas estradas da tabela.

Conclusões

- Considerando um mapeamento uniforme, cada operação toma tempo constante no caso médio. Mas é raro conhecer a distribuição de probabilidade segundo a qual as chaves são obtidas.

- Na prática, podemos usar heurísticas de fácil implementação para criar uma função hash que provavelmente terá um bom desempenho.

- Para resolução de colisões, encadeamento é o método mais simples, mas gasta mais espaço.

- Endereçamento aberto tem implementação mais difícil ou que pode ser suscetível a efeitos de agrupamento.

Conclusões

- **Vantagens**:
  - Simplicidade de implementação.
  - Considerando K o conjunto de chaves armazenadas, a tabela requer espaço $\theta(|K|)$ ao invés de $\theta(|U|)$.
  - A busca na tabela requer $O(1)$ no caso médio.

- **Desvantagens**:
  - Colisão: Efeito que acontece quando duas chaves são mapeadas para a mesma posição na tabela.
  - A busca na tabela requer $O(|K|)$ no pior caso.

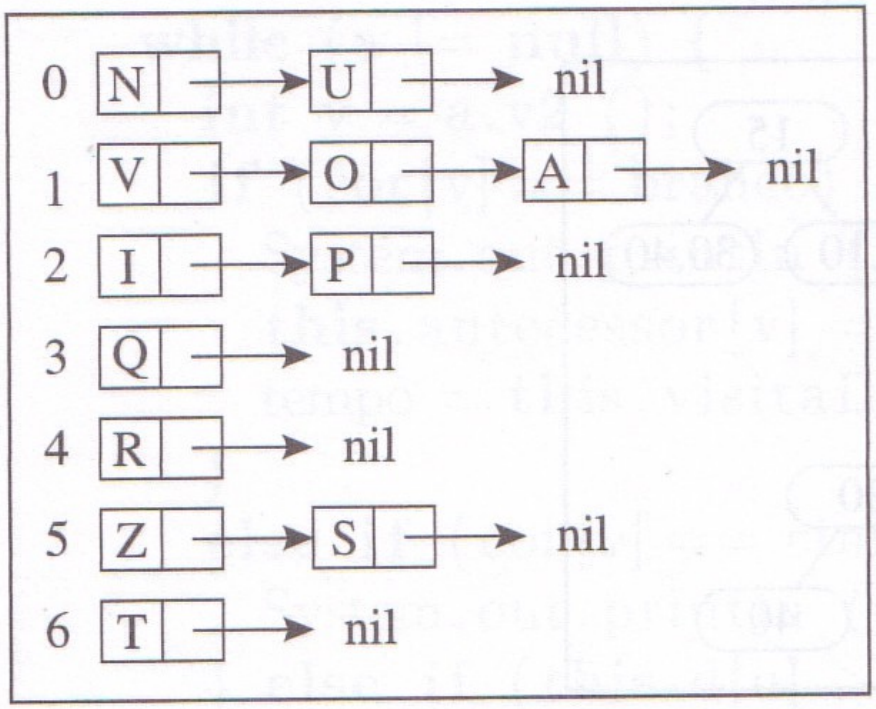
Exercícios

a) Desenhe o conteúdo da tabela hash resultante da inserção de registros com as chaves N I V O Z A P Q R S T U, nesta ordem, em uma tabela inicialmente vazia de tamanho 7 (sete), usando listas encadeadas. Use a função hash **h(k) = k mod 7** para a *k*-ésima letra do alfabeto.

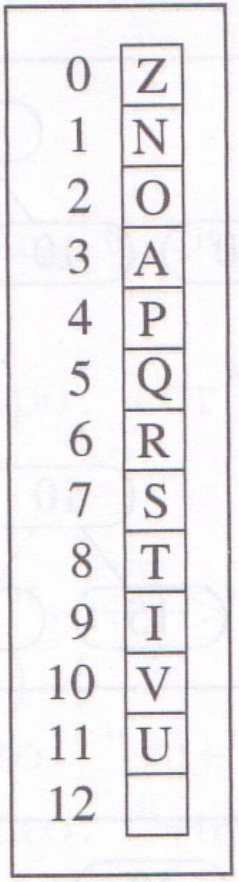
b) Desenhe o conteúdo da tabela hash resultante da inserção de registros com as chaves N I V O Z A P Q R S T U, nesta ordem, em uma tabela inicialmente vazia de tamanho 13 (treze), usando endereçamento aberto e hash linear para resolver as colisões. Use a função **h(k) = k mod 13** para a *k*-ésima letra do alfabeto.

Soluções

a)


0 N U nil
1 V O A nil
2 I P nil
3 Q nil
4 R nil
5 Z S nil
6 T nil

b)


0 Z
1 N
2 O
3 A
4 P
5 Q
6 R
7 S
8 T
9 I
10 V
11 U
12