SBVESDD: Estruturas de Dados

Aula 09: Estruturas de Dados Não-Lineares - Tabelas de Dispersão

Bacharelado em Ciência da Computação
Prof. Dr. David Buzatto

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Campus São João da Boa Vista

Tabelas de Dispersão

- As tabelas de dispersão utilizam o mapeamento de chaves à posições de um array, usando para isso uma função de dispersão (*hash function*);
- Idealmente, cada chave deve mapear a uma posição distinta do array, entretanto, caso haja chaves que mapeiam para posições iguais, há a necessidade de se resolver essa colisão (*collision-resolution*);
- Veremos dois tipos de implementações de tabelas de dispersão:
  - Tabela de dispersão com encadeamento separado (*Separate Chaining Hash Table*);
  - Tabela de dispersão com endereçamento aberto, usando sondagem linear (*Linear Probing Hash Table*).

Tabelas de Dispersão
Função de Dispersão

- A função de dispersão transforma chaves em índices de um array;
  - Qualquer chave ao ser processada deve gerar um índice de 0 a $M-1$ para uma tabela de dispersão que tem inicialmente M posições disponíveis para armazenar pares chave-valor;
  - Para qualquer chave buscamos uma função de dispersão que seja fácil de computar e que distribua uniformemente as chaves;
- A função de dispersão depende do tipo da chave, ou seja, para cada tipo de chave, precisamos de uma implementação distinta da função;
- Usaremos algumas estratégias para a computação do índice, mas sempre que possível, utilizaremos a implementação padrão fornecida pela plataforma Java.

Tabelas de Dispersão

Função de Dispersão

- **Inteiros:** a estratégia mais comum é a dispersão modular (*modular hashing*), que consiste em obter o resto da divisão inteira da chave pelo tamanho do array;
- **Ponto flutuante:** dispersão modular da representação binária do valor;
- **Strings:** R é maior que o valor de qualquer caractere e M é o tamanho do array;

```
int hash = 0;
for ( int i = 0; i < s.length(); i++ ) {
    hash = ( R * hash + s.charAt(i) ) % M;
}
```

- **Chaves compostas:** mesma estratégia anterior.

```
int hash = ( ( ( dia * R + mes ) % M ) * R + ano ) % M;
```

Tabelas de Dispersão

Função de Dispersão

- **Convenções em Java:**
  - Toda classe em Java é subclasse da classe **`Object`**, que por sua vez possui o método **`int hashCode()`** que fornece a implementação padrão da função de dispersão;
  - A implementação do método **`hashCode()`** deve ser consistente com o método **`equals`**:
    - Se **`a.equals(b)`** for verdadeiro, então **`a.hashCode()`** deve ser igual a **`b.hashCode()`**;
    - Se **`a.hashCode()`** for diferente de **`b.hashCode()`**, então **`a`** e **`b`** são necessariamente diferentes;
    - Caso **`a.hashCode()`** e **`b.hashCode()`** forem iguais, pode ser que **`a`** e **`b`** sejam iguais ou diferentes;
  - Na prática implementaremos os dois métodos utilizando o assistente de inserção de código do NetBeans.

Tabelas de Dispersão

Função de Dispersão

- **Convenções em Java:**
  - O método **`hashCode()`** retorna um inteiro, sem se preocupar com o mapeamento a uma posição, para isso, precisamos mapear manualmente, usando a dispersão modular;

```
private int hash( Key x ) {
    return ( x.hashCode() & 0x7fffffff ) % M;
}
```

  - Mascaramento do inteiro de 32-bits com sinal para um inteiro de 31-bits sem sinal.

Tabelas de Dispersão

Função de Dispersão

- **Mascaramento:**

```
private int hash( Key x ) {
    return ( x.hashCode() & 0x7fffffff ) % M;
}
```

```
x.hashCode() = 19723 = 00000000000000000100110100001011
          0x7fffffff = 01111111111111111111111111111111 &
                       00000000000000000100110100001011 = 19723
```

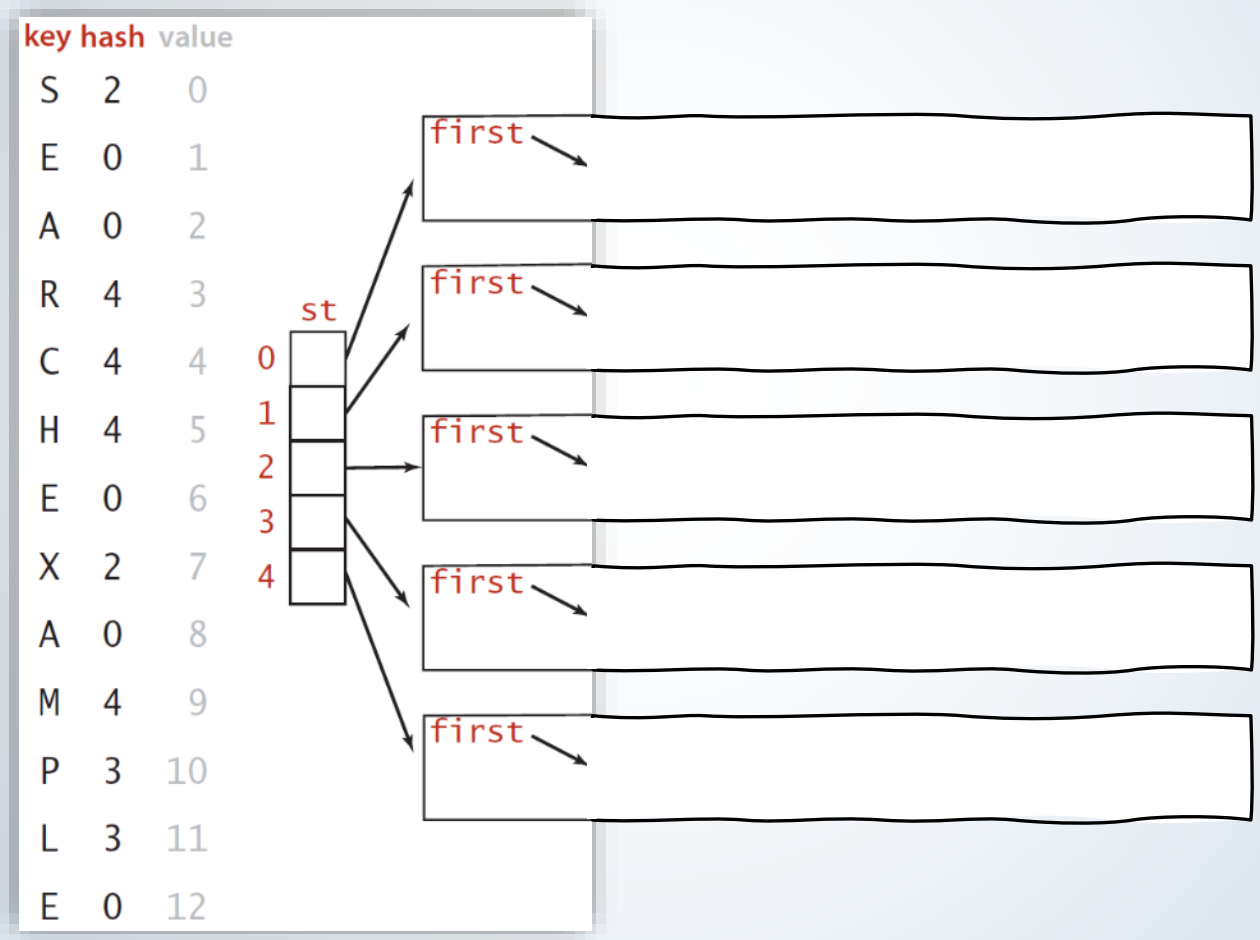
```
x.hashCode() = -19723 = 11111111111111111011001011110101
           0x7fffffff = 01111111111111111111111111111111 &
                        01111111111111111011001011110101 = 2147463925
```

Tabelas de Dispersão
Separate Chaining Hash Table

- A função de dispersão deverá converter as chaves em posições de um array. O segundo componente do algoritmo de dispersão é a resolução de colisão, que é a estratégia usada para gerenciar o caso em que duas ou mais chaves serão inseridas no mesmo índice;
- Uma abordagem geral para isso é construir, para cada índice do array, uma lista encadeada de pares chave-valor, sendo que nessa lista serão armazenadas as chaves sozinhas ou em colisão, associadas a seus valores;
- Esse método é conhecido como encadeamento separado, pois itens em colisão são encadeados juntos numa lista encadeada separada;
- A ideia básica é escolher um tamanho de array suficientemente grande para permitir que as listas sejam suficientemente curtas para tornar a busca mais eficiente: executa-se a função de dispersão para encontrar o índice da lista que contém a chave, e então, sequencialmente, busca-se na lista pela chave desejada.

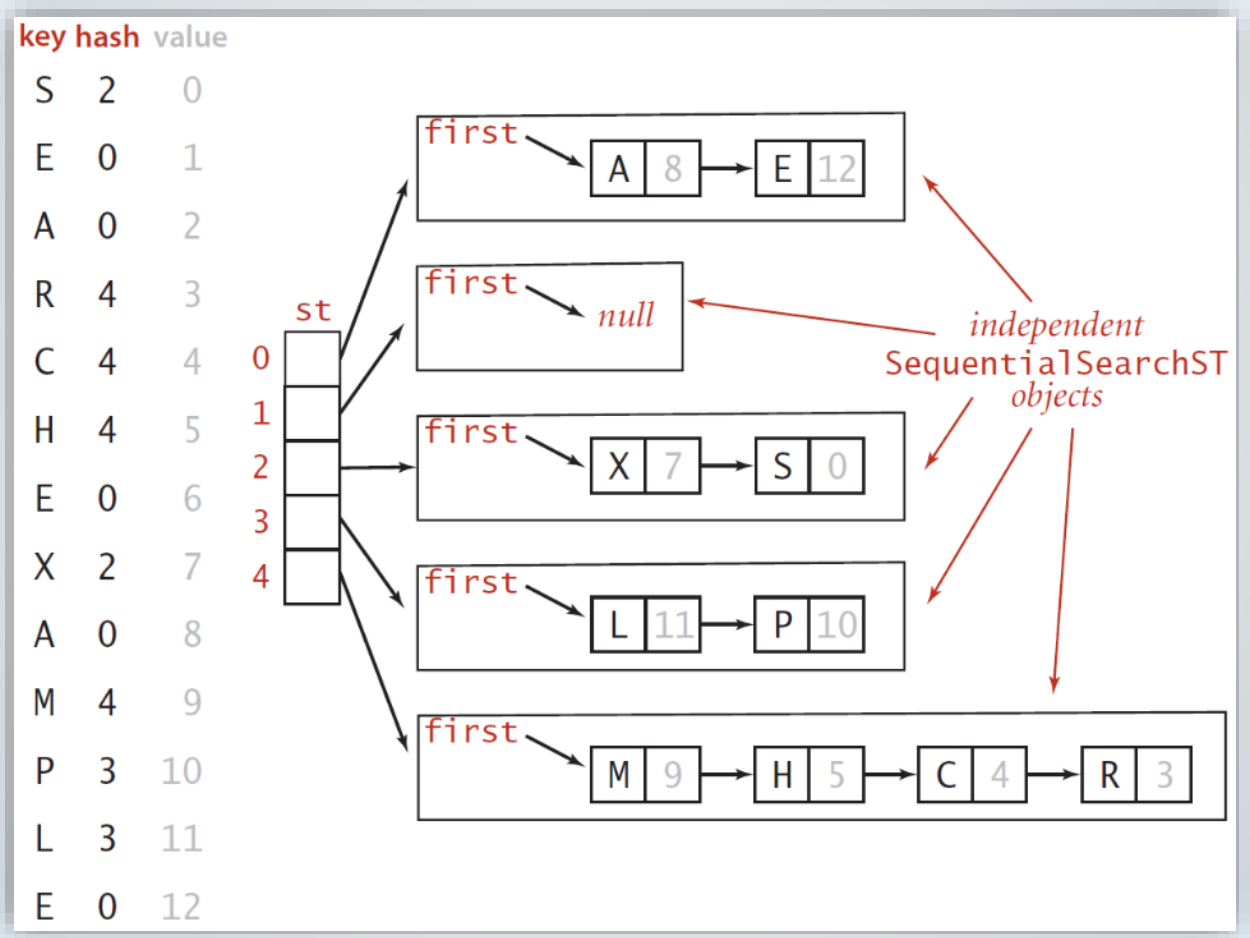
Tabelas de Dispersão

Separate Chaining Hash Table



Tabelas de Dispersão

Separate Chaining Hash Table



Tabelas de Dispersão

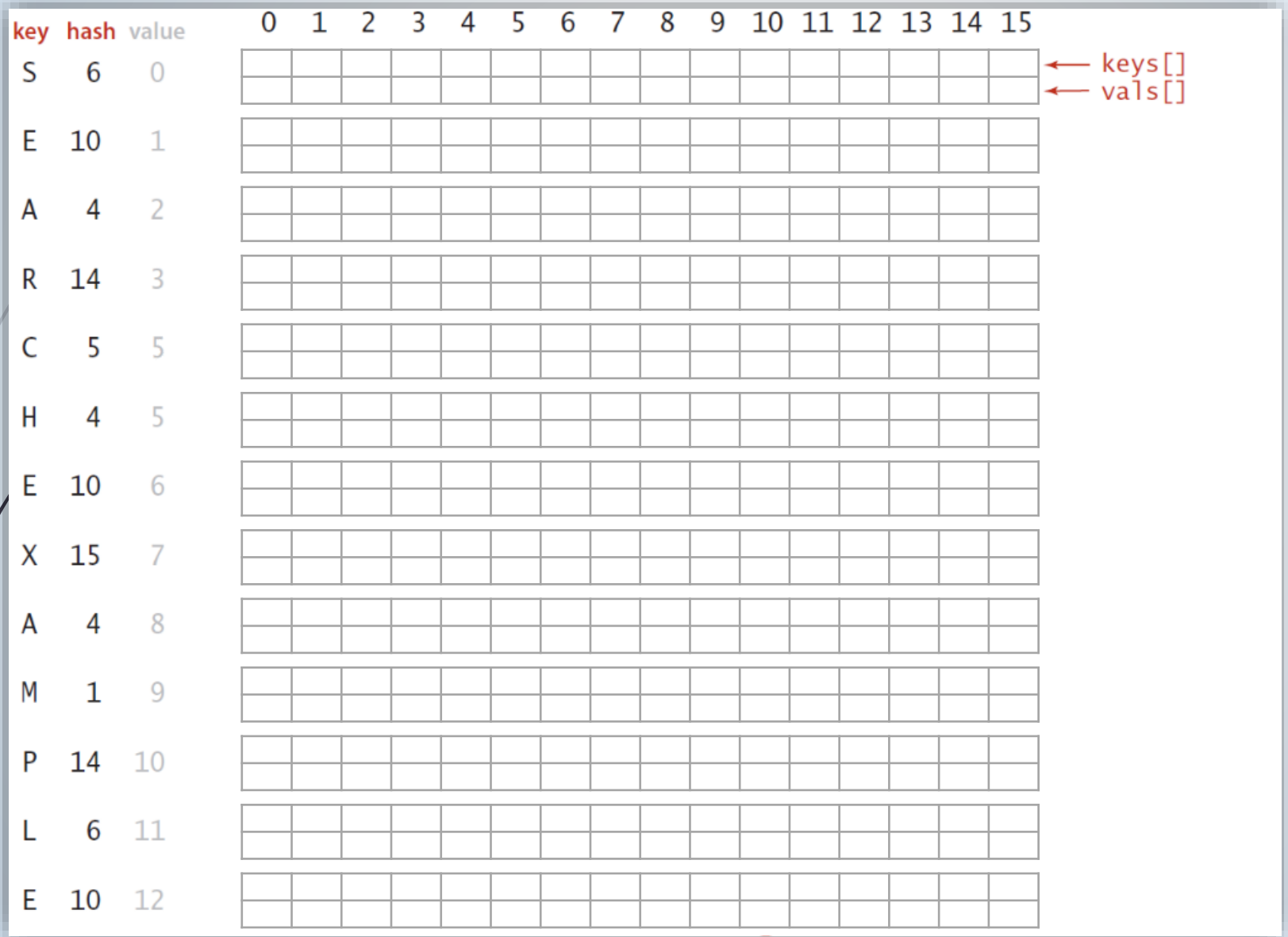
Linear Probing Hash Table

- Outra abordagem para implementar a dispersão é armazenar N pares chave-valor em uma tabela de dispersão de tamanho $M > N$, confiando em entradas vazias na tabela para auxiliar na resolução de colisões. Esse tipo de método é chamado de dispersão com endereçamento aberto;
- O método de endereçamento aberto mais simples é chamado de sondagem linear: quando há colisão, verifica-se a próxima entrada da tabela. A sondagem linear é caracteriza-se em três possibilidades:
  - A chave é igual à chave buscada: encontrou;
  - Posição vazia: não encontrou;
  - Chave é diferente à chave buscada: tentar a próxima entrada.
- Ao calcular a dispersão da chave para gerar um índice da tabela, verifica-se se a chave buscada é igual à chave que está naquela posição e continua-se a buscar pela chave ou por uma posição vazia, mesmo que isso implique em partir para o início da tabela.

Tabelas de Dispersão

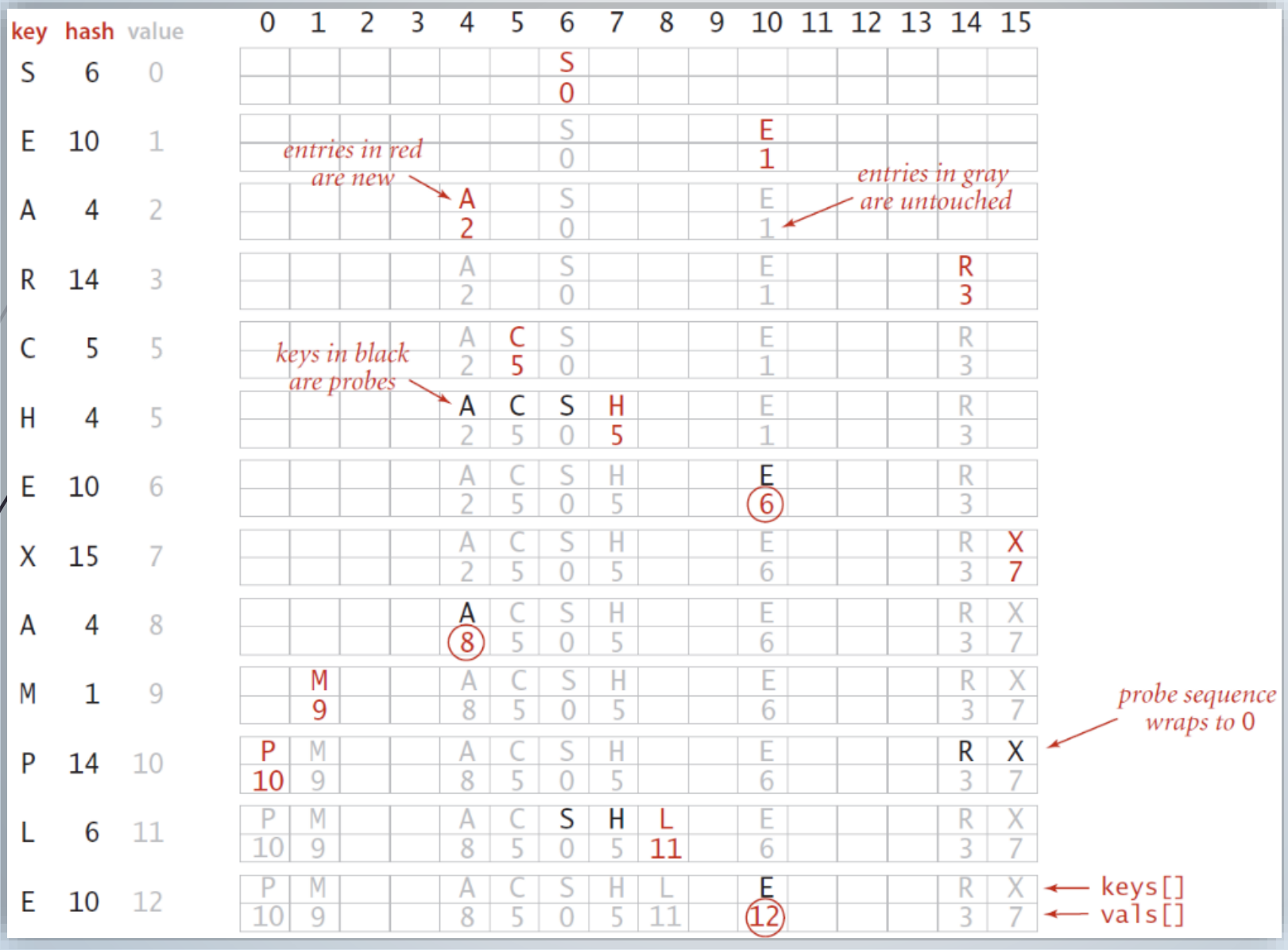
Linear Probing Hash Table

key	hash	value
S	6	0
E	10	1
A	4	2
R	14	3
C	5	5
H	4	5
E	10	6
X	15	7
A	4	8
M	1	9
P	14	10
L	6	11
E	10	12



Tabelas de Dispersão

Linear Probing Hash Table



Bibliografia

SEDGEWICK, R.; WAYNE, K. **Algorithms**. 4. ed. Boston: Pearson Education, 2011. 955 p.

WEISS, M. A. Data Structures and **Algorithm Analysis in Java**. 3. ed. Pearson Education: New Jersey, 2012. 614 p.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. **Algoritmos – Teoria e Prática**. 3. ed. São Paulo: GEN LTC, 2012. 1292 p.