Tabelas de Espalhamento

Prof. Dr. Wesin Ribeiro Alves

Neste capítulo

- Introdução
- Tabelas de endereço direto
- Tabelas de espalhamento
- Função Hash
- Endereçamento Aberto
- Hash perfeito
- revisão

Introdução

Tabela de espalhamento é uma estrutura de dados eficaz para implementar dicionários.

Ela generaliza a noção mais simples de arranjo comum fazendo o uso do endereçamento direto.

Normalmente, a tabela de espalhamento é usada em situações onde precisa-se apenas de operações inserir, buscar e remover.

Dicionários

A implementação das operações é trivial.

DIRECT-ADDRESS-SEARCH(T, k)
1 **return** $T[k] = x$

DIRECT-ADDRESS-INSERT(T, x)
1 $T[x.chave] = x$

DIRECT-ADDRESS-DELETE(T, x)
1 $T[x.chave] = \text{NIL}$

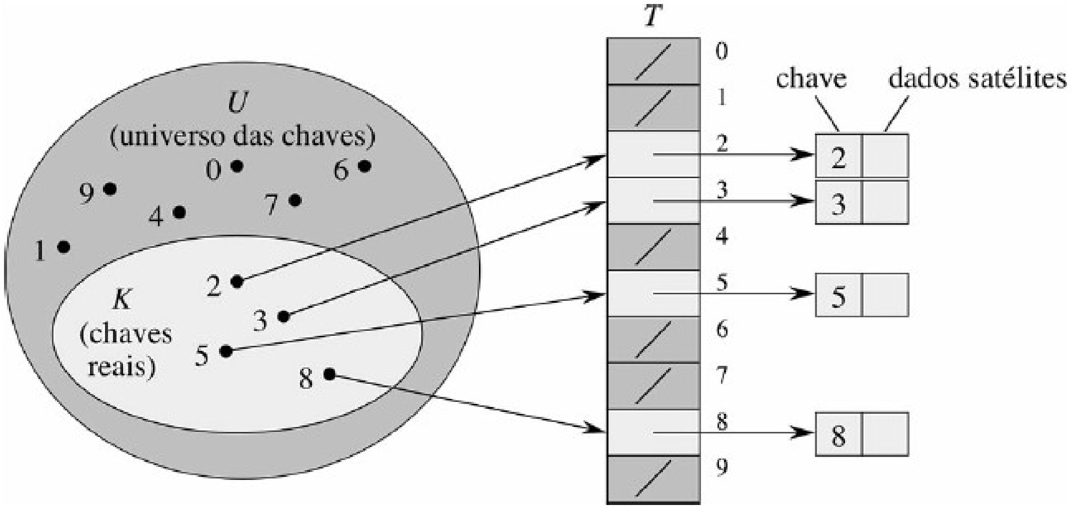
O tempo médio para implementação é O(1).

Tabelas de endereço direto

O endereçamento direto é uma técnica simples que funciona bem quando o número de chaves é razoavelmente pequeno.

T é uma tabela de endereços diretos T=[0..m-1], na qual, cada posição corresponde a uma chave no universo U = {0..m-1}.

Podemos armazenar o objeto na própria posição da tabela e assim economizar espaço.

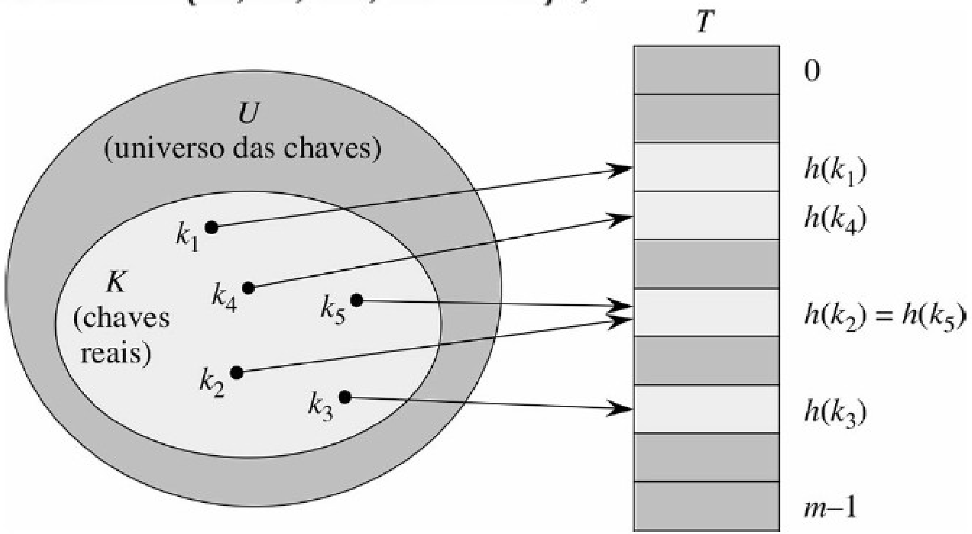


Tabelas de espalhamento

A função hash h pode ser usada para calcular a posição da chave k no arranjo.

Quando o conjunto K de chaves armazenadas em um dicionário é muito menor que o universo U de todas as chaves possíveis, uma tabela de espalhamento requer armazenamento muito menor que uma tabela de endereços diretos.

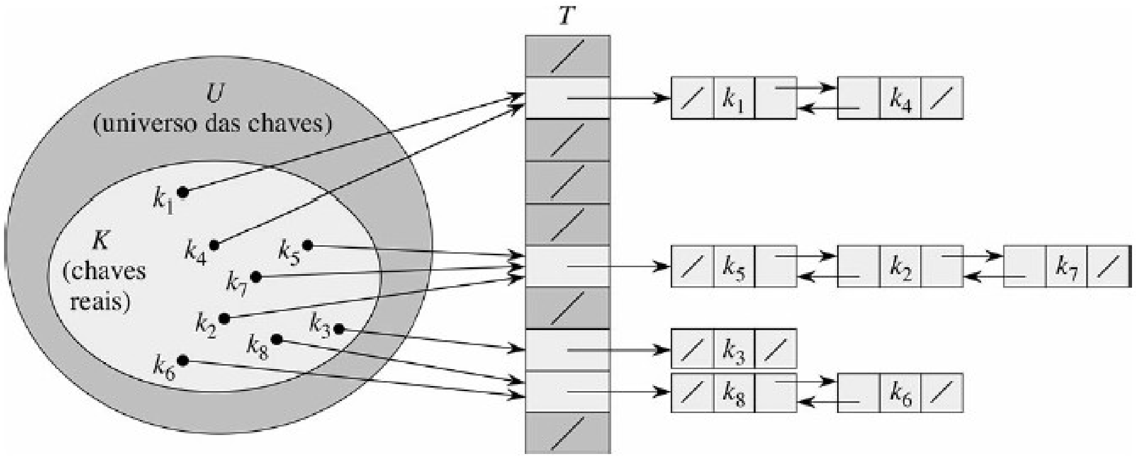
$$h : U \rightarrow \{0, 1, \ldots, m-1\} \,,$$



Resolução de colisões por encadeamento

após o hash, duas chaves podem ser mapeadas para a mesma posição. Chamamos essa situação de ***colisão.***

No encadeamento, todos os elementos resultantes do hash vão para a mesma posição em uma lista ligada.
As operações de dicionário são fáceis de implementar quando as colisões são resolvidas por encadeamento.



Análise do hash com encadeamento

- Complexidade de tempo
  - Inserção: O(1)
  - Remoção O(1) se usar LDE
  - Busca O(1 + alpha)
  - alpha => fator de carga = n/m
  - X_ij = I{h(k_i),h(k_j)}

$$\Pr\{h(k_i) = h(k_j)\} = 1/m$$

$$E[X_{ij}] = 1/m.$$

$$
\begin{aligned}
E\left[\frac{1}{n}\sum_{i=1}^{n}\left(1+\sum_{j=i+1}^{n} X_{ij}\right)\right]
&= \frac{1}{n}\sum_{i=1}^{n}\left(1+\sum_{j=i+1}^{n} E[X_{ij}]\right) \\
&= \frac{1}{n}\sum_{i=1}^{n}\left(1+\sum_{j=i+1}^{n} \frac{1}{m}\right) \\
&= 1+\frac{1}{nm}\left(\sum_{i=1}^{n} n-\sum_{i=1}^{n} i\right) \\
&= 1+\frac{1}{nm}\left(n^2-\frac{n(n+1)}{2}\right) \\
&= 1+\frac{n-1}{2m} \\
&= 1+\frac{\alpha}{2}-\frac{\alpha}{2n}
\end{aligned}
$$

Implementação

Resolução de colisões por encadeamento.

CHAINED-HASH-INSERT(T, x)
1 insere x no início da lista $T[h(x.chave)]$

CHAINED-HASH-SEARCH(T, k)
1 procura um elemento com a chave k na lista $T[h(k)]$

CHAINED-HASH-DELETE(T, x)
1 elimina x da lista $T[h(x.chave)]$

Funções hash

Existem três tipos de esquemas para criação de boas funções hash: por divisão, por multiplicação e por hash universal.

Uma boa função hash satisfaz (aproximadamente) a premissa do hashing uniforme simples.

cada chave tem igual probabilidade de passar para qualquer das m posições por uma operação de hash.

Uma boa função hash minimiza a chance de pequenas variações nos símbolos passarem para a mesma posição após o hashing.

Uma boa abordagem deriva o valor hash de um modo que esperamos seja independente de quaisquer padrões que possam existir nos dados.

Funções hash

Existem três tipos de esquemas para criação de boas funções hash: por divisão, por multiplicação e por hash universal.

Método da divisão

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

Método da multiplicação

$$h(k) = \lfloor m(k\,A \bmod 1) \rfloor$$

$0 < A < 1$

Método universal

$$h_{ab}(k) = ((ak + b) \bmod p) \bmod m$$

$$\Pr\{h_{ab}(k) = h_{ab}(l)\} \leq 1/m$$

$$\mathcal{H}_{pm} = \{h_{ab} : a \in \mathbb{Z}_p^* \text{ and } b \in \mathbb{Z}_p\}$$

Endereçamento Aberto

Todos os elementos ficam na própria tabela de espalhamento. Isto é, cada entrada da tabela contém um elemento do conjunto dinâmico ou nulo.

- Não usa lista ligada
- A tabela pode ficar cheia
- Evita por completo a utilização de ponteiros
- + memória, - colisões

Sequência de sondagem

Pseudo código para operações de inserção e busca.

```
HASH-INSERT(T, k)
i = 0
repeat j = h(k, i)
        if T[j] == NIL
            T[j] = k
            return j
        else i = i + 1
until i == m
error "estouro da tabela"
```

```
HASH-SEARCH(T, k)
i = 0
repeat
    j = h(k, i)
        if T[j] == k
            return j
        i = i + 1
until T[j] == NIL ou i == m
return NIL
```

Sondagem linear

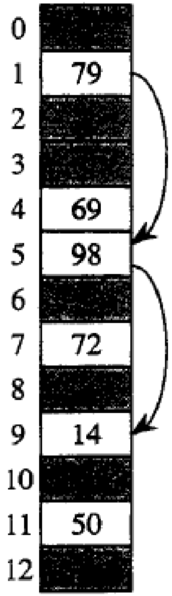
- $h(k,i) = (h'(k)+1) \bmod m$
- Fácil de implementar
- Problema de agrupamento primário
- Sondagem inicial em $T[h'(k)]$

Sondagem quadrática

- h(k,i) = (h'(k) + c_1*i + c_2*i^2) mod m
- C_1 e c_2 são constantes
- I em {0..m-1}
- Problema de agrupamento secundário
- Sondagem inicial em T[h'(k)]

Hash duplo

- h(k,i) = (h_1(k) + i*h_2(k)) mod m
- H_1 e h_2 são funções de hash auxiliares
- Sondagem inicial em T[h_1(k)
- Melhor quando m é primo ou potência de 2



O Hash perfeito

Ocorre quando forem exigidos *O*(1) acessos à memória para executar uma busca no pior caso.

- Dois níveis de hash universal
  - Primeiro nível igual ao hashing com encadeamento
  - Segundo nível usa uma tabela hashing secundário S_j
  - m_j = n_j ^ 2
  - Complexidade de espaço O(n)