Tabelas de Espalhamento (Tabelas Hash)

Kelly Rosa Braghetto

30 de março 2012

Operações em um Dicionário

Estrutura de Dados	Busca	Inserção/Remoção
Vetor	não ordenado: $O(n)$ ordenado: $O(\lg n)$	$O(n)$
Listas Encadeadas	$O(n)$	$\mathrm{T}(\text{Busca}) + \Theta(1)$
Árvores AVL	$O(\lg n)$	$O(\lg n)$

Operações em um Dicionário

Estrutura de Dados	Busca	Inserção/Remoção
Vetor	não ordenado: $O(n)$ ordenado: $O(\lg n)$	$O(n)$
Listas Encadeadas	$O(n)$	$\mathrm{T}(\text{Busca}) + \Theta(1)$
Árvores AVL	$O(\lg n)$	$O(\lg n)$

$\Rightarrow$ Esses algoritmos de busca se baseiam em **comparações** feitas entre a chave buscada e as chaves dos elementos do dicionário.

Operações em um Dicionário

Estrutura de Dados	Busca	Inserção/Remoção
Vetor	não ordenado: $O(n)$ ordenado: $O(\lg n)$	$O(n)$
Listas Encadeadas	$O(n)$	$\mathrm{T}(\mathrm{Busca}) + \Theta(1)$
Árvores AVL	$O(\lg n)$	$O(\lg n)$

$\Rightarrow$ Esses algoritmos de busca se baseiam em **comparações** feitas entre a chave buscada e as chaves dos elementos do dicionário.

$\Rightarrow$ Mas esses são os melhores tempos de execução para buscas em dicionários que podemos ter?

Busca em um Dicionário

Exemplo: tabela de símbolos de um compilador

- Armazena informações sobre os identificadores de um código-fonte:
  - nomes de constantes
  - nomes de variáveis
  - nomes dos tipos de dados
  - nomes de funções

Identificador	Tipo
max_Tam	constante inteira
fila_cheia	variável do tipo boolean
BuscaBinária	função
⋮	⋮

- É usada nas diferentes fases de uma compilação (análise sintática, análise semântica, otimização, geração do código de máquina, etc.)

Busca em um Dicionário em Tempo Constante

É possível implementar estruturas de dados para dicionários nas quais a busca por um elemento pode ser executada [em média] em tempo constante ($O(1)$).

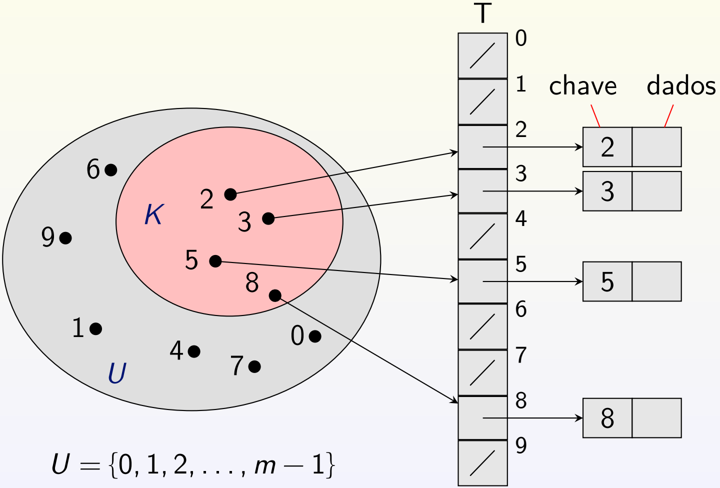
Hoje veremos duas estruturas com essa propriedade:

- **Tabelas de Endereçamento Direto**
- **Tabelas de Espalhamento** ($=$ Tabelas *Hash*)

Algumas definições

- U: é o universo das chaves, ou seja, o conjunto de todas as possíveis chaves
- K: é conjunto das chaves efetivamente usadas (logo, $K \subseteq U$)

Tabelas de Endereçamento Direto



$U = \{0, 1, 2, \ldots, m-1\}$

K: chaves usadas

Tabelas de Endereçamento Direto

Operações

BUSCA-POR-ENDEREÇAMENTO-DIRETO(T, k)
    return $T[k]$

INSERE-POR-ENDEREÇAMENTO-DIRETO(T, x)
    $T[x.chave] \leftarrow x$

REMOVE-POR-ENDEREÇAMENTO-DIRETO(T, x)
    $T[x.chave] \leftarrow$ NIL

$\Rightarrow$ **As três operações têm tempo de execução $O(1)$.**

Tabelas de Endereçamento Direto

Problemas

- Dois elementos diferentes do dicionário não podem possuir um mesmo valor de chave.
- Se $|U|$ é muito grande, manter a tabela T é impraticável.
- Quando $|K|$ é muito menor que $|U|$, grande parte do espaço alocado para T é desperdiçado.

Exemplo: registros dos 5000 alunos de uma universidade

- O registro de um aluno contém:
  - Chave: número de matrícula do aluno (contendo 5 dígitos)
  - Demais dados: nome, endereço, etc.
- $T[00000..99999] \Rightarrow$ apenas 5% das posições de T são usadas.

Tabelas de Endereçamento Direto

Possível solução:

$\Rightarrow$ Usar uma tabela $T[0..m-1]$, com $m < |U|$.

Novo problema:

$\Rightarrow$ Se não vamos ter mais em T uma posição "exclusiva" para cada valor de chave $k \in U$, como saberemos onde um elemento deve ser armazenado na tabela?

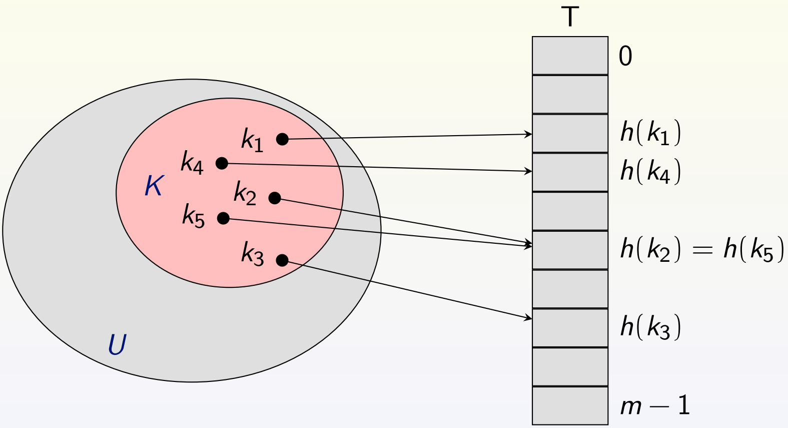
Tabelas de Espalhamento

- Quando o conjunto K é muito menor que U, uma estratégia melhor é usar uma **tabela de espalhamento**.
- Em uma tabela de espalhamento, um elemento com chave k é armazenado na posição $h(k)$, onde h é uma função que mapeia as chaves em U em posições da tabela $T[0..m-1]$:

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

- Chamamos h de **função hash** (ou, **função de espalhamento**).

Tabelas de Espalhamento



Tabelas de Espalhamento

Colisões

- É possível haver colisões no mapeamento das chaves para posições em T.
- Uma colisão ocorre quando, para duas chaves distintas $k_1, k_2 \in U$,
$$h(k_1) = h(k_2)$$
- É possível diminuir a chance colisões por meio da escolha de uma "boa" função hash.
- Entretanto, é muito difícil evitar colisões completamente.

Probabilidade de Colisões

Paradoxo do Aniversário

$\Rightarrow$ Em um grupo de apenas 23 pessoas, a probabilidade de que duas pessoas façam aniversário no mesmo dia é maior do que 50%.

Se uma função hash distribui os elementos de forma uniforme entre as posições da tabela, a probabilidade p de inserirmos n itens consecutivos sem colisão em uma tabela de tamanho m é:

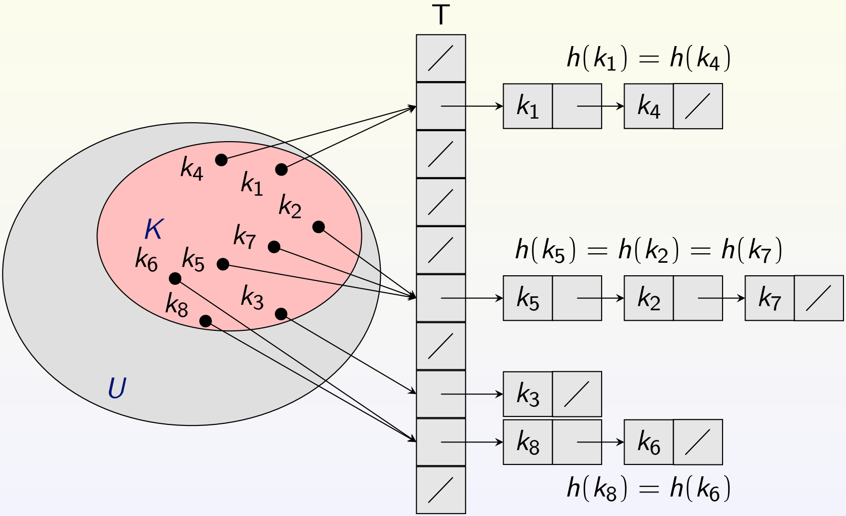
$$\bar{p} = \frac{m-1}{m} \times \frac{m-2}{m} \times \ldots \times \frac{m-n+1}{m}$$

Logo, a probabilidade de colisão é: $p = 1 - \bar{p}$.

Para $\mathbf{m = 365}$, temos:

n	10	23	30	50	57
p	11,7%	50,7%	70,6%	97%	99%

Tratamento de Colisões por Encadeamento



Tratamento de Colisões por Encadeamento

Operações

BUSCA-NO-ENCADEAMENTO(T, k)

buscar por um elemento com chave k na lista $T[h(k)]$

INSERE-NO-ENCADEAMENTO(T, x)

inserir x no início da lista $T[h(x.chave)]$

REMOVE-NO-ENCADEAMENTO(T, x)

remover x da lista $T[h(x.chave)]$

Tratamento de Colisões por Encadeamento

Análise do Tempo de Execução das Operações

⇒ Sob a suposição de que qualquer elemento do dicionário tem igual probabilidade de ser direcionado para qualquer posição da tabela, temos que:

- O tamanho esperado para cada lista encadeada é $\frac{n}{m}$, onde n é o número de elementos armazenados na tabela e m é o tamanho da tabela.
- As operações de busca, inserção e remoção consomem um tempo de execução $O(1 + \frac{n}{m})$, sendo que:
  - $O(1)$ é o tempo para encontrar a posição na tabela ($=$ tempo para o cálculo da função hash);
  - $\frac{n}{m}$ é o tempo para percorrer a lista encadeada.

⇒ Para um m próximo a n, o custo das operações se torna constante.

Funções Hash

Características

- As funções hash devem mapear chaves em números inteiros dentro do intervalo $[0..m-1]$ (onde m é o tamanho da tabela).
- Uma função hash ideal é aquela que:
  1. é simples de ser computada;
  2. "espalha" de forma uniforme as chaves do dicionário entre as posições da tabela.

Funções Hash – Método da Divisão

Neste método, a função hash é da forma:

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

Características

- Método rápido (requer uma única operação: a divisão).
- Não é recomendado o uso de valores para m que sejam da forma $m = 2^p$, para algum inteiro p. Caso contrário, a valor da função hash dependerá somente dos p bits de mais baixa ordem da chave. É desejável que a função hash leve em conta todos os bits da chave.
- Uma boa sugestão de valor para m é um número primo não muito próximo a uma potência de 2.

Funções Hash – Método da Multiplicação

Neste método, a função hash opera em dois passos:

1. A chave é multiplicada por uma constante A tal que $0 < A < 1$ e depois extrai-se a parte racional da multiplicação (o que resulta em um número x tal que $0 \leqslant x < 1$)
2. x é multiplicado por m e toma-se o piso do resultado, o que gera um número em $[0..m-1]$.

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

onde "$(kA \bmod 1)$" equivale à parte fracionária de kA $(= kA - \lfloor kA \rfloor)$.

Características

- Calculada de forma eficiente usando operações bit a bit.
- Uma grande vantagem do método é que ele independe do valor de m.

Na próxima aula...

Mais sobre tabelas de espalhamento:

- Endereçamento Aberto
- *Hashing* Perfeito

Exercícios

1. Suponha que um dicionário D é representado por uma tabela de endereçamento direto T de tamanho n.
   Descreva um procedimento que encontra o elemento com a maior chave em D.
   Qual é o desempenho do seu algoritmo no pior caso?
2. Implemente uma tabela de espalhamento usando uma função de hash definida pelo método da divisão. As chaves no seu dicionário serão 5000 números aleatoriamente gerados entre $[00000..99999]$. Teste sua tabela de espalhamento para diferentes valores apropriados para m. Para cada valor de m, informe o número de colisões que ocorreram em cada posição da tabela.

Referências para Estudo

- *Projeto de Algoritmos* [Ziviani], Capítulo 5
- *Introduction to Algorithms* [Cormen et al.], Capítulo 11
- *Algorithms in C (Parts 1-4)* [Sedgewick], Capítulo 14