Aula 15: Tabelas de espalhamento

Tabelas de dispersão, tabelas *hash*

David Déharbe
Programa de Pós-graduação em Sistemas e Computação
Universidade Federal do Rio Grande do Norte
Centro de Ciências Exatas e da Terra
Departamento de Informática e Matemática Aplicada

Plano da aula

Referência: Cormen et al. Capítulo 12.

Introdução

- Coleção dinâmica de dados
- Cada dado x tem um atributo chave $x.key$ único
  $x \neq y \Rightarrow x.key \neq y.key$
- Operações
  - Inserção de um dado;
  - Remoção de um dado;
  - Busca na coleção de um dado com uma determinada chave
    - pode ser bem sucedida ou mal sucedida
  - Custo $\Theta(1)$ em média, $\Theta(n)$ no pior caso.

Tabelas de indexação direta

- U: conjunto das chaves possíveis
- U é pequeno
- Encontrar função bijetora i de U para $1..|U|$
- Manter uma tabela A de tamanho $|U|$, onde $A[i(x.key)]$
  - é x, ou uma referência para x, quando o dado x está na coleção;
  - é o valor especial $\textsc{Nil}$, caso contrário.

$\textsc{Buscar}(A, k)$

  return $A[i(k)]$

$\textsc{Inserir}(A, x)$

  $A[i(x.key)] = x$

$\textsc{Remover}(A, x)$

  $A[i(x.key)] = \textsc{Nil}$

1. Assumindo
    - as mesmas hipóteses que para as tabelas de indexação direta, e
    - que não estamos interessados em representar x na coleção, mas apenas registrar a presença ou ausência dele,

   encontrar uma estrutura de dados mais econômica em memória que a tabela.
2. Descrever um procedimento para encontrar o dado com maior chave em uma tabela de indexação direta.
    - Qual o custo deste procedimento?
    - Como alterar a estrutura de dados para ter um custo $\Theta(1)$?

Tabelas de espalhamento

- O tamanho de uma tabela de indexação direta é $\Theta(|U|)$
- Quando U não é pequeno, tabelas de indexação direta não são viáveis.
  - Em um compilador, como representar a tabela dos símbolos?
- Mesmo se $|U|$ pode ser representado em memória, se o tamanho de K, o conjunto dos dados efetivamente presentes da coleção é muito menor que $|U|$, há desperdício de memória.
- Quando $|K|$ é muito menor que $|U|$, recomenda-se considerar tabelas de espalhamento ao invés de tabelas de indexação direta.

- O custo médio das operações é $\Theta(1)$
  - no caso de tabelas de indexação direta, é o custo no pior caso
- A quantidade de memória necessária é $\Theta(|K|)$.
  - O tamanho de uma tabela de indexação direta é $\Theta(|U|)$
- A função de indexação h de U para $1 \mathinner{..} m$ (m tamanho da tabela de espalhamento) é sobrejetora
  - A função da tabela de indexação direta é injetora.
  - A função h é chamada função de espalhamento
    função de dispersão, função *hash*

- Problema: colisões
  - dados x e x', com chaves k e k' são tais que $h(k) = h(k')$, são inseridos na tabela.
- Como resolver as colisões?
  - Encadeamento externo
  - Política de endereçamento aberto
- Como reduzir a probabilidade de colisões?
  - Funções de espalhamento
  - A função ideal mapearia cada chave para uma posição diferente.
  - Em geral: $|U| > m$, e a função ideal não existe.

PPgSC

Tratamento de colisões por encadeamento externo

- Cada posição contem uma referência para a primeira célula de uma lista encadeada
  A lista na posição p armazena $\{x \mid h(x.key) = p\}$;
- ou $\textsc{Nil}$ se a coleção não contem dado x tal que $h(x.key) = p$.

$\textsc{Buscar}(A, k)$

// buscar um dado com chave k na lista $A[h(k)]$

$\textsc{Inserir}(A, x)$

// inserir o dado na cabeça da lista $A[h(x.key)]$

$\textsc{Remover}(A, x)$

// remover o dado x da lista $A[h(x.key)]$

$\textsc{Buscar}(A, k)$

// buscar um dado com chave k na lista $A[h(k)]$

$\textsc{Inserir}(A, x)$

// inserir o dado na cabeça da lista $A[h(x.key)]$

$\textsc{Remover}(A, x)$

// remover o dado x da lista $A[h(x.key)]$

não esquecer de atualizar $A[h(x.key)]$ na inserção e na remoção

Buscar complexidade linear no tamanho da lista $A[h(k)]$

Inserir $\Theta(1)$

Remover
- $\Theta(1)$ se a lista for duplamente encadeada e o dado x contem os atributos *next* e *prev*.
- complexidade linearmente proporcional ao tamanho da lista $A[h(x.key)]$ caso contrário.

Buscar complexidade linear no tamanho da lista $A[h(k)]$

Inserir $\Theta(1)$

Remover
- $\Theta(1)$ se a lista for duplamente encadeada e o dado x contem os atributos *next* e *prev*.
- complexidade linearmente proporcional ao tamanho da lista $A[h(x.key)]$ caso contrário.

O que podemos dizer sobre o tamanho das listas?

Análise de espalhamento com encadeamento externo

$$\begin{array}{rcl} n & : & \text{número de elementos na tabela} \\ m & : & \text{tamanho da tabela} \end{array}$$

- fator de carga: $\alpha = n/m$.
- pior caso:
  - todos os elementos estão na mesma posição $\Theta(n)$
  - lista encadeada
  - função de espalhamento não espalha!
- caso médio depende das propriedades da função de espalhamento

Hipóteses:

1. espalhamento uniforme simples: para qualquer chave k a probabilidade de $h(k) = i$ é $1/m$, para $1 \leq i \leq m$
2. o custo de calcular $h(k)$ é sempre $\Theta(1)$.
3. fator de carga α
4. tratamento de colisões por encadeamento externo

Cenários analizados:

- busca mal-sucedida
- busca bem-sucedida

Hipóteses:

Teorema

Sob as hipóteses enunciadas, o custo de uma busca mal-sucedida é $\Theta(1 + \alpha)$.

Hipóteses:

Teorema

Sob as hipóteses enunciadas, o custo de uma busca mal-sucedida é $\Theta(1+\alpha)$.

Demonstração.

O custo é

- o de uma busca mal-sucedida em uma lista: $\Theta(|\text{lista}|)$ ($=\Theta(\alpha)$ em média).
- acrescentado do cálculo de $h(k)$, e do acesso ao arranjo $(=\Theta(1))$.

ou seja $\Theta(1+\alpha)$.

Hipóteses:

Teorema

Sob as hipóteses enunciadas, o custo de uma busca bem-sucedida é $\Theta(1 + \alpha)$.

Hipóteses:

Teorema

Sob as hipóteses enunciadas, o custo de uma busca bem-sucedida é $\Theta(1+\alpha)$.

Demonstração.

O custo médio é

- o de uma busca bem-sucedida em uma lista: $\Theta(|\text{lista}|/2)$ ($=\Theta(\alpha/2)$ em média).
- acrescentado do cálculo de $h(k)$, e do acesso ao arranjo ($=\Theta(1)$).

ou seja $\Theta(1+\alpha/2)=\Theta(1+\alpha)$. □

PPgSC

- Sob as hipóteses enunciadas, e
- se m é pelo menos proporcionalmente linear a n ($m \in \Omega(n)$),
- então $n \in O(m)$ e $\alpha = n/m \in O(m)/m = O(1)$.
- Logo o custo da busca é $O(1)$.
- Para a inserção, o custo é $O(1)$.
- Para a remoção, o custo é
  - $O(1)$ se a lista for duplamente encadeada e x inclui os attributos de encadeamento;
  - $O(1 + \alpha) = O(1)$ caso contrário (pelo mesmos motivos que a busca).

Exercícios

1. Simule inserir sucessivamente os valores com chaves 5, 28, 19, 15, 20, 33, 12, 17, 10 em uma tabela de espalhamento inicialmente vazia, com 9 posições, e função de espalhamento $h = \lambda k.1 + k \mod 9$.
2. Sob as hipóteses enunciadas, estime a complexidade média da operação de inserção caso a mesma seja realizadda:
   - sempre no final da lista
   - em uma lista ordenada, mantendo a propriedade de ordenação
3. Sob a hipótese de espalhamento uniforme simples, quantas colisões há quando n chaves distintas são inseridos em uma tabela de espalhamento de tamanho m?

Funções de espalhamento: características desejadas

m : tamanho da tabela de espalhamento
h : função de espalhamento
U : universo das chaves

- A função de espalhamento ideal é uniforme:
  - Todas as listas encadeadas tem o mesmo tamanho n/m.
  - A probabilidade de qualquer operação de inserção ocorrer na posição j seja $1/m$.
  - Seja $P(k)$ a probabilidade de um dado ter a chave $k \in U$.

$$\sum_{\{k|h(k)=j\}} P(k) = 1/m$$

- Exemplo: $U = \{k \in \mathbb{R} \,|\, 0 \leq k \leq 1\}$, P é uniforme.
  $h(k) = 1 + \lfloor k \cdot m \rfloor$

PPgSC

Funções de espalhamento: características desejadas

- A distribuição das chaves geralmente não é conhecida.
- Existe heurísticas com resultado prático satisfatório.
  - análise estatística do domínio de aplicação
  - análise qualitativa do domínio de aplicação
  - o resultado do espalhamento deve ser independente de qualquer padrão que possa ocorrer nos dados
    Exemplo de chaves comum em tabela de símbolos: `"i"`, `"j"`, `"pt"`, `"ptr"`, `"pt1"`

Tratando chaves como números naturais

- É comum funções de espalhamente considerar que as chaves são números naturais.
- É simples satisfazer esta hipótese:
  - qualquer chave tem uma representação binária
  - por interpretação como número na base 2, qualquer código binário é mapeado para um número natural.
  - Exemplo (codificação ASCII, base 128):
    - `"pt"` é interpretada como o par de naturais (112, 116), através da norma ASCII, a qual tem 128 códigos diferentes.
    - Logo `"pt"` é mapeado para $112 \cdot 128^1 + 116 \cdot 128^0 = 14452$.
  - O mapeamento é uma função bijetora.

PPgSC

Espalhamento pelo método da divisão

m : tamanho da tabela de espalhamento
h : função de espalhamento

- $h = \lambda k \cdot k \mod m$
- Potências de 2 são valores a evitar para m
  - Motivo: $h(k)$ só depende de $\log_2 m$ bits de k.
- Potências de 10 são valores a evitar para m se as chaves são números decimais
- Receita para um m geralmente bom:
  - número primo
  - não vizinho de uma potência de 2
- Sempre verificar experimentalmente a escolha com dados representativos da aplicação.

PPgSC

Espalhamento por multiplicação

$$
\begin{aligned}
k &: \text{chave} \\
m &: \text{tamanho da tabela de espalhamento} \\
h &: \text{função de espalhamento}
\end{aligned}
$$

- Escolher uma constante $A \in \mathbb{R}$ tal que $0 < A < 1$
- Seja $\varphi(k) = k \cdot A - \lfloor k \cdot A \rfloor$ a parte fracionária do produto da chave por A.
- $h = \lambda k.1 + \lfloor m \cdot \varphi(A) \rfloor$ é a função de espalhamento obtida.
- O valor de m não é crítico: pode ser uma potência de 2.
- A escolha ótima do valor de A depende da aplicação
  - O valor $(\sqrt{5} - 1)/2 = 0.61803...$ foi sugerido por ter boa probabilidade de funcionar bem (Knuth).
- $h = \lambda k.k \times \lfloor A \cdot 2^w \rfloor / 2^{w-p}$, onde w é o tamanho da palavra do computador e $2^p = m$.

Espalhamento universal

Carter & Wegman, 1979

- O usuário maliocoso pode escolher uma série de dados que vão todos ser “espalhados” para a mesma posição.
- O desempenho da aplicação será afetado negativamente
- Espalhamento universal é uma solução a este problema
- Há uma coleção de funções de espalhamento.
- Em tempo de execução uma das funções de espalhamento é escolhida aleatoriamente.
- Reduz a probabilidade do desempenho ser ruim.
- Uma coleção H de funções de espalhamento é universal quando, para cada par de chaves distintas x, y, $h(x) = h(y)$ com probabilidade $|H|/m$.
  - Escolhendo uma função de espalhamento h ao acaso em H, a chance de colisão entre x e y é $1/m$.
  - Esta probabilidade é a mesma que se $h(x)$ e $h(y)$ ter sido escolhidos aleatoriamente em $1 \mathinner{.\,.} m$.

Espalhamento universal

Teorema

Se h é escolhido de uma coleção universal de funções de espalhamento para espalhar $n \leq m$ chaves, o número esperado de colisões para uma chave x é menos que 1.

Demonstração.

- Seja $x \neq y$ duas chaves quaisquer.
- Por definição, a probabilidade de colisão é $1/m$.
- Se há n chaves distintas $x_1, \ldots x_n$,
  - a probabilidade de colisão de x_1 com x_2 é $1/m$, com x_2 é $1/m$, etc.
  - a probabilidade global de colisão é $(n-1)/m$;
  - se $n \leq m$, a probabilidade de colisão $(n-1)/m < 1$.

Projeto de classe universal de funções de espalhamento

- m é um número primo
- cada chave k é considerada como uma sequência $\langle k_1, \ldots, k_r \rangle$ de r cadeias de t bits (condição $2^t < m$)
- Seja $a = \langle a_1, \ldots, a_r \rangle$ uma sequência formada por r elementos escolhidos do conjunto $1 \mathinner{..} m$
- A função de espalhamento $h_a = \lambda k. \sum_{i=1}^{r} a_i k_i$
- $H = \bigcup_a \{h_a\}$ tem m^r elementos.

Teorema

A classe H é universal.

Projeto de classe universal de funções de espalhamento

Demonstração.

Seja $x \neq y$ e $h(x) = h(y)$. Assumimos que $x_1 \neq y_1$ (poderia ser qualquer sub-sequência).

- $h_a(x) = h_a(y) \Longrightarrow \sum_{i=1}^{r} a_i x_i \mod m = \sum_{i=1}^{r} a_i y_i \mod m$.
- Logo $\sum_{i=1}^{r} a_i(x_i - y_i) \mod m = 0$.
- Logo $a_1(x_1 - y_1) \equiv \sum_{i=1}^{r} a_i(x_i - y_i) \mod m = 0$.
- Teoria dos números: como $x_1 - y_1 \neq 0$, possui um inverso multiplicativo módulo m.
- Logo $a_1 = -\sum_{i=2}^{r} a_i(x_i - y_i) \cdot (x_1 - y_1)^{-1} \mod m$
  ... e existe um único $a_0 \mod m$ tal que $h(x) = h(y)$.
- Tem m^{r-1} valores de a (uma para cada $\langle a_2, \ldots, a_r \rangle$) tais que x e y colidem.
- De m^r combinações possíveis, há m^{r-1} colisões. A probabilidade é $m^{r-1}/m^r = 1/m$: H é universal.

Exercícios

1. Aplicando o espalhamento por multiplicação com $A = (\sqrt{5} - 1)/2$ e $m = 1000$, quais são as posições para as chaves 61, 62, 63, 64 e 65?
2. Em uma aplicação onde comparar duas chaves é custoso, como adaptar as estruturas de dados envolvidas em uma tabela de espalhamento para acelerar as operações?

Resolução de colisões por endereçamento aberto

- Sem encadeamento externo
- Se houver uma colisão, uma nova posição é calculada
- Função de espalhamento: $h(k, i)$
  - k chave
  - i tentativa

    até m tentativas
- capacidade limitada ($\neq$ encadeamento externo)

Inserção

Resolução de colisões por endereçamento aberto

```
INSERIR(T, x)
i = 1
repeat
    p = h(x.key, i)
    if T[p] == NIL
        T[p] = x
        return p
    else i = i + 1
until i == m + 1
// tratamento de tabela cheia
```

```
BUSCAR(T, k)
i = 1
repeat
    p = h(k, i)
    if T[p].key == k
        return p
    if T[p] == NIL
        return NIL
    i = i + 1
until i == m + 1
return NIL
```

Remover

Resolução de colisões por endereçamento aberto

```
REMOVER(T, x)
i = 1
repeat
    p = h(x.key, i)
    if T[p].key == k
        T[p] = NIL
        return
    i = i + 1
until i == m + 1
// tratar erro
```

Remover

Resolução de colisões por endereçamento aberto

```
REMOVER(T, x)
i = 1
repeat
    p = h(x.key, i)
    if T[p].key == k
        T[p] = NIL
        return
    i = i + 1
until i == m + 1
// tratar erro
```

- Assumindo $m = 10$, $h = \lambda k, i \cdot 1 + ((k + i - 1) \mod m)$, $x.key = x$,
- INSERIR(5), INSERIR(15), REMOVER(5), BUSCAR(15)

Remover

Resolução de colisões por endereçamento aberto

REMOVER(T, x)

```
i = 1
repeat
    p = h(x.key, i)
    if T[p].key == k
        T[p] = NIL
        return
    i = i + 1
until i == m + 1
// tratar erro
```

- Assumindo $m = 10$, $h = \lambda k, i \cdot 1 + ((k + i - 1) \mod m)$, $x.key = x$,
- INSERIR(5), INSERIR(15), REMOVER(5), BUSCAR(15)
- ...problema...

Remover

Resolução de colisões por endereçamento aberto

REMOVER(T, x)

$i = 1$
repeat
    $p = h(x.key, i)$
    if $T[p].key == k$
        $T[p] =$ NIL
        return
    $i = i + 1$
until $i == m + 1$
// tratar erro

- Assumindo $m = 10$, $h = \lambda k, i \cdot 1 + ((k + i - 1) \mod m)$, $x.key = x$,
- INSERIR(5), INSERIR(15), REMOVER(5), BUSCAR(15)
- ...problema... solução: DELETED

Inserção

Resolução de colisões por endereçamento aberto

```
INSERIR(A, x)
i = 1
repeat
      p = h(x.key, i)
      if T[p] == NIL or T[p] == DELETED
            T[p] = x
            return p
      else i = i + 1
until i == m + 1
// tratamento de tabela cheia
```

Busca

Resolução de colisões por endereçamento aberto

```
BUSCAR(A, k)
i = 1
repeat
    p = h(k, i)
    if T[p].key == k
        return p
    if T[p] == NIL
        return NIL
    i = i + 1
until i == m + 1
return NIL
```

Remover

Resolução de colisões por endereçamento aberto

```
REMOVER(A, x)
i = 1
repeat
    p = h(x.key, i)
    if T[p].key == k
        T[p] = DELETED
        return
    i = i + 1
until i == m + 1
// tratar erro
```

Tentativas lineares

$$\begin{array}{rcl} h & : & \text{função de espalhamento da chave} \\ m & : & \text{tamanho da tabela} \end{array}$$

$$h_L = \lambda k, i \cdot 1 + (h(k) + (i - 1) \mod m)$$

- tentativas são realizadas em posições sucessivas
- problema: agrupamentos primários
  - quando há uma sub-faixa ocupada de tamanho $n < m$,
  - a probabilidade de colisão é n/m,
  - cria uma sub-faixa de tamanho $n' \geq n + 1$
  - a probabilidade de colisão nesta sub-faixa torna-se n'/m.
- não é uma boa sub-solução: desempenho degrada-se rapidamente.

Tentativas quadráticas

h	:	função de espalhamento da chave
m	:	tamanho da tabela

$$h_Q = \lambda k, i \cdot 1 + (h(k) + c_1 \cdot (i-1) + c_2 \cdot (i-1)^2 \mod m)$$

$c_1 \neq 0, c_2 \neq 0$ são constantes

- elimina o problema dos agrupamentos lineares
- para que todas as posições sejam visitadas em m tentativas, os valores de m, c_1 e c_2 não podem ser quaisquer
- note que se $h_Q(k_1, 1) = h_Q(k_2, 1)$ então $h_Q(k_1, i) = h_Q(k_2, i)$ para $i > 1$
- a mesma sequência de tentativas é calculada para chaves com o mesmo valor de espalhamento inicial
  - aparecem agrupamentos secundários

Espalhamento duplo

$$\begin{array}{rcl} h_1,\, h_2 & : & \text{função de espalhamento da chave} \\ m & : & \text{tamanho da tabela} \end{array}$$

$$h_D = \lambda k, i \cdot 1 + (h_1(k) + (i-1) \cdot h_2(k) \mod m)$$

- a sequência de tentativas depende de k para a posição inicial,
- o deslocamento para as próximas tentativas também depende de k.
- para m tentativas visitar m posições diferentes, o valor de $h_2(k)$ deve ser relativamente primo com relação a m.
  - dica 1: $m = 2^p$ e $\forall k \cdot h_2(k)$ é ímpar.
  - dica 2: m primo, e $\forall k \cdot h_2(k) < m$ é ímpar
    exemplo: $h_1 = \lambda k \cdot 1 + (k \mod m)$, $h_2 = \lambda k \cdot 1 + (kmodm')$, com $m' = m - 1$.

PPgSC

Comparação das tentativas

- tentativa linear: m sequências de tentativas possíveis
- tentativa quadrática: m sequências de tentativas possíveis
- espalhamento duplo: m^2 sequência de tentativas possíveis
- ideal: $m!$ sequências de tentativas possíveis

$\Longrightarrow$ espalhamento duplo é teoricamente bem superior

PPgSC

Exercícios

1. Seja uma tabela de dispersão de tamanho $m = 11$ com endereçamento aberto. Os valores $10, 22, 31, 4, 15, 28, 17, 88, 59$ são inseridos, nesta ordem. Qual o resultado destas operações quando é usada a função de espalhamento $h = \lambda k \cdot 1 + k \mod m$, e as colisões são tratadas por:
   - tentativas lineares
   - tentativas quadráticas
   - espalhamento duplo, e $h_2 = \lambda k \cdot 1 + (k \mod (m-1))$, $c_1 = 1$, $c_2 = 3$.