TABELAS DE DISPERSÃO

Gustavo Carvalho
(ghpc@cin.ufpe.br)

Universidade Federal de Pernambuco
Centro de Informática, 50740-560, Brazil

Agenda

Dicionários: ADT

Uma coleção de registros, permitindo armazenamento e recuperação

Registros indexados por uma chave (*key*)

- Chaves precisam ser comparáveis; idealmente, ordem total

Operações:

- void clear(Dictionary d);
- void insert(Dictionary d, Key k, E e); // refletir: múltiplas entradas
- E remove(Dictionary d, Key k); // refletir: múltiplas entradas
- E removeAny(Dictionary d); // alternativa: getKeys
- E find(Dictionary d, Key k); // refletir: múltiplas entradas
- int size(Dictionary d);

Implementações: array, lista ligada,
tabelas de dispersão, e árvores balanceadas

Agenda

Hashing

Trade-off entre eficiência espacial e temporal

- Hashing: mais espaço para privilegiar eficiência temporal

Maneira extremamente eficiente para implementar dicionários

- Tabela de dispersão (*hash table*): $H[0..m-1]$
- Função hash: $h : Key \rightarrow 0..m-1$

Requisitos desejáveis

- Relação entre m e $\mid Key \mid$: viável sem comprometer efic. temporal
- h deve distribuir chaves de forma uniforme
- h deve ser fácil de computar

Hashing

Desvantagens:

- Não é ideal se houver múltiplos registros por chave
- Não é ideal para percorrer registros em ordem por chave
- Não é ideal para pesquisas baseadas em faixas de valores

Uma das estruturas mais utilizadas para armazenar grandes volumes de dados em disco (alternativa: árvores B)

Hashing: funções hash para números

Uma primeira abordagem[1]

- $h(K) = K \bmod m$

Exemplo:

- Seja $m = 100$ (índices de 0..99)
- Se $K = 4567$, então $h(K) = 4567 \bmod 100 = 67$

Uma abordagem melhor (método mid-square):

- Calcula o quadrado de K, seleciona or r dígitos do meio do resultado, tal que $10^r - 1 < m$

Exemplo:

- Seja $m = 100$ (índices de 0..99), então $r = 2$
- Se $K = 4567$, $K^2 = 20857489$, então $h(K) = 57$

[1] **Cuidado** com implementação de mod : $\mathbb{Z} \to \mathbb{N}$

Hashing: funções hash para strings

Uma primeira abordagem (*fold*):

Algoritmo: int h(string x, int M)

```
1  s ← length(x);
2  sum ← 0;
3  for i ← 0 to s − 1 do
4  |    sum ← sum + x[i];
5  return sum % M;
```

A distribuição pode ser dependendo de M e s

- Considere que na média $s = 10$
- Seja x composta por letras maiúsculas
- Como $A = 65$ e $Z = 90$, $sum \in [650..900]$
- Se $M \leq 100$, a distribuição é razoável
- Se $M \geq 1000$, a distribuição é ruim

Hashing: funções hash para strings

Uma abordagem melhor (*sfold*):

Algoritmo: int h(string x, int M)

```
1  intLength ← length(x)/4;
2  sum ← 0;
3  for j ← 0 to intLength − 1 do
4      sub ← substring(x, j ∗ 4, (j ∗ 4) + 4);      // pos. inicial e final
5      mult ← 1;
6      s ← length(sub);
7      for k ← 0 to s − 1 do
8          sum ← sum + sub[k] ∗ mult;
9          mult ← mult ∗ 256;
10 sub ← substring(x, intLength ∗ 4);      // pos. inicial até o final
11 mult ← 1;
12 s ← length(sub);
13 for k ← 0 to s − 1 do
14     sum ← sum + sub[k] ∗ mult;
15     mult ← mult ∗ 256;
16 return abs(sum)%M// abs = overflow e %
```

Hashing: colisões

Se $m < |\ Key\ |$, teremos necessariamente colisões

Colisões mesmo se $m \geq |\ Key\ |$

Política de resolução de colisões:

- Encadeamento (open hashing ou *separate chaining*)
- Endereçamento aberto (closed hashing ou *open addressing*)

Hashing perfeito

- Ausência de colisões
- Conjunto de chaves: conhecido e disponível previamente
- Pode ser muito custoso
- Exemplo de aplicação: acesso a dados em CD *read-only*

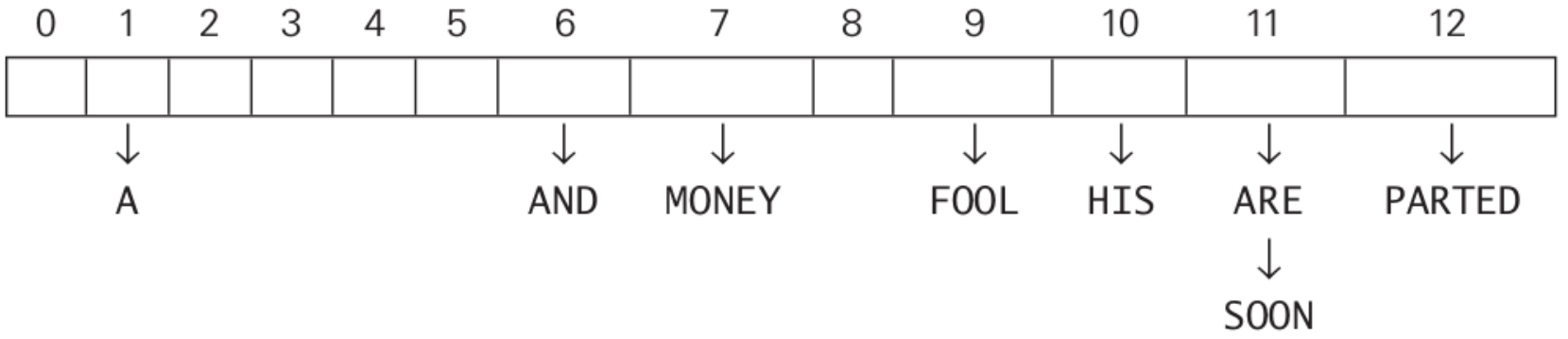
Agenda

Política de encadeamento

Uma lista ligada é associada a cada posição da tabela de dispersão

Exemplo (considerando $h(A) = 1$ e *fold*):

keys	A	FOOL	AND	HIS	MONEY	ARE	SOON	PARTED
hash addresses	1	9	6	10	7	11	11	12



[2]

[2] Fonte: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Política de encadeamento

Algoritmo de inserção:

- Calcula $h(K)$, então insere na lista ligada
- Inserção ordenada: menor custo busca, maior custo inserção
- Inserção no final: maior custo busca, menor custo de inserção

Algoritmo de busca:

- Calcula $h(K)$, se a lista correspondente for "vazia", registro associado a K não está na tabela; se a lista correspondente não estiver "vazia", percorre a lista.

Algoritmo de remoção:

- Calcula $h(K)$, então remove da lista ligada

Política de encadeamento

Eficiência da inserção (considerando inserção no final)

- No caso médio: $\Theta(1)$

Eficiência da busca: depende do tamanho das listas

- Tamanho das listas: depende das chaves, m e h
- Seja fator de carga: $\alpha = n/m$, então $S \approx 1 + \frac{\alpha}{2}$ e $U = \alpha$
  - n = quantidade de chaves armazenadas na tabela
- Considerando $\alpha \approx 1$, então $\Theta(1)$ na média

Eficiência da remoção (considerando $\alpha \approx 1$)

- No caso médio: $\Theta(1)$

Desvantagem: quando usada com armazenamento em disco

Agenda

Política de endereçamento aberto

Registros armazenados na própria tabela

- Requisito: $m \geq n$

Estratégias:

- Linear probing
- Pseudo-random probing
- Quadratic probing
- Double hashing

Política de endereçamento aberto

Linear probing:

- Inserção: havendo colisão, tenta a próxima posição até conseguir
  - Probe sequence | Probe function (**offset**): $p(K, i) = i$
  - Tabela trada como array circular | Só pode inserir se $m > n$

Exemplo (considerando $h(A) = 1$ e *fold*):

keys	A	FOOL	AND	HIS	MONEY	ARE	SOON	PARTED
hash addresses	1	9	6	10	7	11	11	12

0	1	2	3	4	5	6	7	8	9	10	11	12
	A											
	A								FOOL			
	A					AND			FOOL			
	A					AND			FOOL	HIS		
	A					AND	MONEY		FOOL	HIS		
	A					AND	MONEY		FOOL	HIS	ARE	
	A					AND	MONEY		FOOL	HIS	ARE	SOON
PARTED	A					AND	MONEY		FOOL	HIS	ARE	SOON

[3]

[3] Fonte: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Política de endereçamento aberto

Linear probing:

- Busca: (1) calcula $h(K)$, se a célular estiver vazia, registro não está na tabela; (2) caso contrário, se for o que se busca, encontrou; (3) caso contrário, tentar a próxima posição até encontrar uma posição vazia ou o que se procura

Exemplo (considerando $h(A) = 1$ e *fold*):

- $h(LIT) = (12 + 9 + 20) \bmod 13 = 2$,
  como a posição 2 está vazia, então *LIT* não está na tabela
- $h(KID) = (11 + 9 + 4) \bmod 13 = 11$,
  comparações com *ARE*, *SOON*, *PARTED* e *A*
  até declarar que *KID* não está na tabela

Política de endereçamento aberto

Linear probing:

- Remoção: não pode remover o valor; usar símbolo especial
  - Alterar a inserção/remoção para considerar este símbolo

Análise da eficiência temporal é mais complicada de ser feita:

- $S \approx \frac{1}{2}(1 + \frac{1}{1-\alpha})$
- $U \approx \frac{1}{2}(1 + \frac{1}{(1-\alpha)^2})$

Política de endereçamento aberto

Linear probing: evolução de S e U em função de α

α	$\frac{1}{2}(1+\frac{1}{1-\alpha})$	$\frac{1}{2}(1+\frac{1}{(1-\alpha)^2})$
50%	1.5	2.5
75%	2.5	8.5
90%	5.5	50.5

[4]

Quando $\alpha \approx 1$, eficiência deteriora: primary clustering

- Maior chance de adicionar um elemento a um cluster
- Maior chance de unificar dois clusters

[4] Fonte: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Política de endereçamento aberto

Pseudo-random probing: quando houver colisão

- $p(K, i) = Perm[i - 1]$, onde $Perm$ é uma permutação de $1..m - 1$

Quadratic probing: quando houver colisão

- $p(K, i) = c_1 i^2 + c_2 i + c3$
- Se $m = 2^k$, $p(K, i) = (i^2 + i)/2$ garante visitar todas as posições

Estas duas abordagens não impedem: secondary clustering

- Se $h(k_1) = h(k_2)$, k_1 e k_2 compartilham mesma probe sequence
- Motivo: p ignora o parâmetro K

Política de endereçamento aberto

Alernativa: double hashing

- Uma nova função hash determina o incremento: $s(K)$
  - $p(K, i) = i * s(K)$
- Sugestões da literatura
  - Tabelas pequenas
    - $s(K) = m - 2 - K \bmod (m - 2)$
    - $s(K) = 8 - (K \bmod 8)$
  - Tabelas grandes
    - $s(K) = K \bmod 97 + 1$

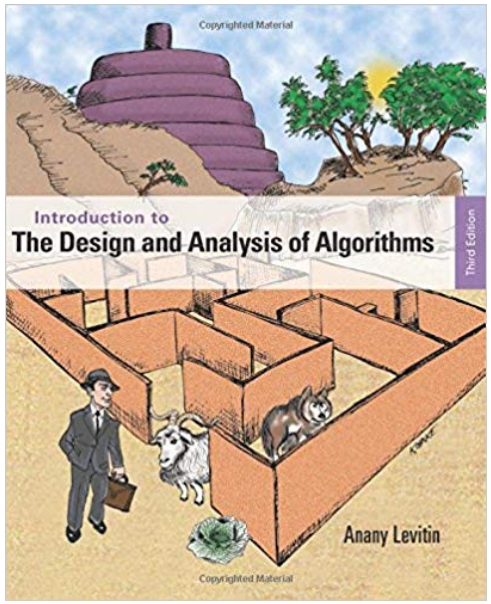
Eficiência temporal

- Boas escolhas de h e s: melhor que linear probing
- Também deteriora quando $\alpha \approx 1$
- Alternativa: aumentar a tabela e fazer rehashing

Solução para armazenamento em disco: bucket hashing

Agenda

Bibliografia + leitura recomendada



Capítulo 1 (pp. 35–37)
Capítulo 7 (pp. 253–254)
Capítulo 7 (pp. 269–274)
Anany Levitin.
Introduction to the Design and Analysis of Algorithms.
3a edição. Pearson. 2011.

Capítulo 4 (pp. 131–138)
Capítulo 9 (pp. 314–336)
Clifford Shaffer.
Data Structures and Algorithm Analysis.
Dover, 2013.

TABELAS DE DISPERSÃO

Gustavo Carvalho
(ghpc@cin.ufpe.br)

Universidade Federal de Pernambuco
Centro de Informática, 50740-560, Brazil