Introdução à Ciência da Computação II

Hashing Pt. II: Tratamento de Colisões & Desempenho

Prof. Ricardo J. G. B. Campello

Aula de Hoje

- Tratamento de Colisões
  - Encadeamento Externo
  - Endereçamento Aberto
- Análise de Desempenho
- Re-dispersão
- Variantes e Casos Particulares

Tabela Hash (Revisão)

- Uma **tabela hash** T consiste de:
  - Uma **função hash** h
  - Um vetor (tabela) V de tamanho N
- Uma função hash h mapeia chaves de um dado tipo em inteiros em um intervalo fixo $[0, N-1]$
  - O valor inteiro $h(k) \in [0, N-1]$ é chamado de **valor hash** da chave k

3

Funções Hash (Revisão)

- Uma função hash é composta das seguintes sub-funções:

  Código Hash (hash code):
  h_1: chaves $\rightarrow$ inteiros

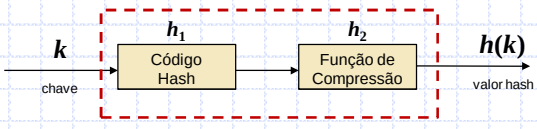
  Função de Compressão:
  h_2: inteiros $\rightarrow [0, N-1]$

- Usualmente, quando se assume que já se dispõe de uma codificação inteira das chaves, refere-se à função de compressão sozinha como "função hash"

- O código hash (quando necessário) é aplicado primeiro; em seguida a função de compressão é aplicada ao resultado:

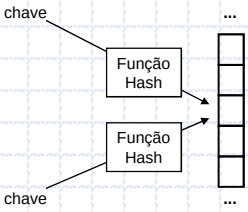
  $h(k) = h_2(h_1(k))$

- A meta da função hash é "dispersar" as chaves de forma que essas ocupem a tabela da forma mais uniforme possível



Colisões

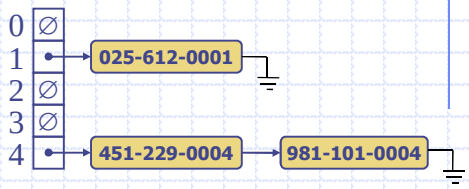
- Colisões ocorrem quando diferentes chaves são mapeadas sem distinção na mesma célula da tabela
- Colisões podem ocorrer tanto na fase de codificação das chaves (h_1) como na fase de compressão (h_2)
- Ocorrendo na fase de codificação, não há como serem revertidas na fase de compressão
- Colisões requerem tratamentos *a posteriori* que demandam esforço computacional
- Funções hash devem ser simples e rápidas de calcular, minimizando ao máximo colisões



- Note que é impossível evitar completamente colisões se o **fator de carga** λ de uma tabela hash for $\lambda > 1$
- Esse fator é dado pela razão entre o no. de chaves, n, e o tamanho da tabela, N: $\lambda = n / N$

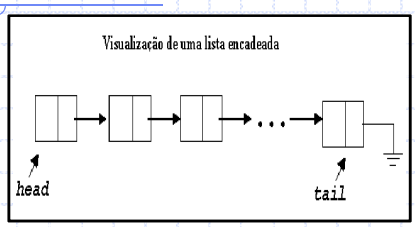
Tratamento de Colisões

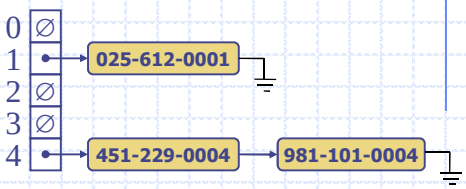
- Um método simples e eficiente de lidar com colisões que não podem ser evitadas é conhecido como **encadeamento externo**, **exterior**, ou **em separado**
  - ***separate chaining*** em inglês
- Nesse método, cada célula da tabela aponta (ponteiro) para uma lista dos itens mapeados naquela célula



- O tempo adicional para percorrer essas listas e o espaço extra para armazená-las são as razões pelas quais colisões devem ser evitadas ao máximo

Encadeamento Externo





- **Lista:**

```
typedef struct {
          int nelem;
          nodo *head, *tail;
        } Lista;
```

- **Tabela Hash:**

```
typedef struct {
          int n;
          Lista V[TAMANHO_N];
        } Tabela_Hash;
```

7

Performance

- Hipótese:
  - função hash eficiente, que distribua de maneira uniforme as chaves, ou seja, distribuição de probabilidade de atribuição de chaves é uniforme sobre o conjunto de células da tabela
- Nesse caso, o tamanho esperado das listas é $O(n/N)$
- Fazendo o tamanho da tabela ser proporcional ao no. de chaves, ou seja, $N \equiv O(n)$, faz-se o tamanho médio das listas no encadeamento externo ser $O(1)$
  - trivial em aplicações estáticas (conjunto fixo de chaves)
  - em aplicações dinâmicas, pode-se estimar o no. típico de chaves que estarão usualmente armazenadas na tabela
    - ou aplicar a técnica de **re-dispersão** (discutida posteriormente)

8

Performance (cont.)

- Nesse caso, todas as operações na tabela executam em tempo constante no **melhor caso** e **em média**
  - ou seja, inserção, remoção e busca executam em tempo $O(1)$
    - assumindo que a função hash executa em tempo constante
- No **pior caso**, quando todas as chaves são mapeadas em uma mesma célula, recai-se em uma lista:
  - busca e remoção em $O(n)$
  - inserção em $O(1)$
    - insere-se no início da lista

9

Performance (cont.)

	Busca	Inserção	Remoção	Notas
Tabela Hash	$O(1)$ esperado	$O(1)$	$O(1)$ esperado	$O(n)$ no pior caso para busca e remoção
Árvores AVL	$O(\log n)$ pior caso	$O(\log n)$ pior caso	$O(\log n)$ pior caso	implementação complexa

Comparação com AVL

10

Re-dispersão

- Vimos que é possível alcançar tempo esperado $O(1)$ para busca, inserção e remoção em uma tabela hash com encadeamento externo, fazendo N ser $O(n)$
  - A idéia é manter o tamanho da tabela aproximadamente proporcional ao número de chaves a serem manipuladas
    - em outras palavras, manter o fator de carga aprox. constante
    - uma boa prática, quando possível, é manter $\lambda < 0.9$
- Como a maioria das aplicações são dinâmicas, ou seja, n é variável, torna-se necessário o uso de algum mecanismo para ajuste do fator de carga

Re-dispersão

- Um mecanismo eficiente é a **re-dispersão**:
  - *rehashing* em inglês
  - consiste em alocar um novo vetor toda vez que o fator de carga exceder um limite preestabelecido e re-mapear os elementos da tabela antiga para a nova tabela

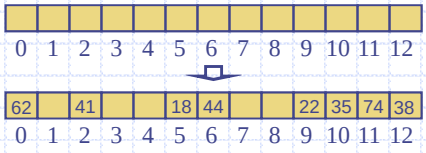
Endereçamento Aberto

- Também denominada **open addressing** em inglês, trata-se de uma outra estratégia de tratamento de colisões, na qual um item em colisão é alocado a uma outra célula disponível da tabela
- É uma abordagem mais voltada a aplicações onde se tem restrições de memória, que podem ser minimizadas ao preço de um maior esforço de processamento
- A idéia é, em caso de colisão ao tentar inserir um novo item, percorrer (sondar) a tabela buscando por uma célula não ocupada
- O fator de carga nunca pode exceder 1
  - pois cada célula da tabela armazena um único item

13

Sondagem Linear

- Re-aloca o item em colisão na próxima célula disponível da tabela (circularmente)
  - ou seja, se a célula $i = h(k)$ estiver ocupada, tenta-se novamente em $(i + j) \bmod N$, $j = 1, 2, \ldots$
- Exemplo
  - dada a função hash para chaves inteiras $h(k) = k \bmod 13$, insira as chaves 18, 41, 22, 44, 38, 35, 74, 62 nesta ordem



14

Inserção com Sondagem Linear

1. Verifica-se se a tabela não está cheia, encerrando caso $n = N$
2. Calcula-se o valor hash $h(k)$ da chave k a ser inserida na tabela
3. Inicia-se a sondagem linear na célula de índice $h(k)$, até que
   - uma célula inexplorada (**I**) ou desocupada (**D**) seja encontrada
4. Armazena-se a chave k na célula encontrada
5. Marca-se a célula em questão como ocupada (**O**)
6. Incrementa-se o número de chaves n

* **NOTA:** Assume-se que, inicialmente, ao se declarar a tabela, todas as células são marcadas como inexploradas (**I**) (através de um flag)

15

Remoção com Sondagem Linear

1. Executa-se a rotina de busca pela chave k a ser removida
   - encerrando caso a chave não seja encontrada
   - vide rotina de busca a seguir
2. Se a chave for encontrada, simplesmente marca-se a respectiva célula como desocupada (**D**), indicando que as demais infos. são lixo
3. Decrementa-se o número de chaves n

16

Busca com Sondagem Linear

1. Verifica-se se a tabela não está vazia, retornando **–1** caso $n = 0$
2. Calcula-se o valor hash $h(k)$ da chave k que se deseja encontrar
3. Inicia-se a sondagem linear na célula de índice $h(k)$, até que
   - uma célula ocupada (**O**) contendo a chave procurada seja encontrada; **ou**
   - uma célula inexplorada (**I**) seja encontrada; **ou**
   - n células ocupadas (**O**) sejam encontradas (nenhuma com a chave desejada)
4. No primeiro caso, retorna-se o índice da célula contendo a chave
   - nos demais casos, retorna-se **–1** (busca falhou)

* **NOTA:** Assume-se que, inicialmente, ao se declarar a tabela, todas as células são marcadas como inexploradas (**I**)

Exemplos

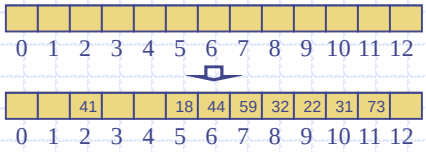
No quadro...

Implementação em C

- Tabela Hash completa com endereçamento aberto via sondagem linear
  - No quadro...

Notas (Endereçamento Aberto)

- O procedimento de sondagem linear possui um inconveniente conhecido como **clustering linear**
  - longas seqüências de células ocupadas que tendem a se agrupar
  - retarda as sondagens
  - Exemplo
    - inserir as chaves 18, 41, 22, 44, 59, 32, 31, 73 com $h(k) = k \bmod 13$



20

Notas (Endereçamento Aberto)

- Existem estratégias de sondagem alternativas para implementação de endereçamento aberto:
  - Sondagem Quadrática
  - Double Hashing
- De formas distintas, essas estratégias buscam amenizar o problema de clustering discutido anteriormente

21

Notas (Endereçamento Aberto)

- Independentemente da estratégia de sondagem adotada, o fator de carga da tabela hash, $\lambda = n / N$, afeta em maior ou menor grau o seu desempenho
- Claramente, existe um compromisso forte entre desempenho em termos de processamento (tempo) e espaço alocado (memória):
  - $\uparrow N$ $\downarrow$ no. de colisões
- Uma heurística para endereçamento aberto é, sempre que possível, manter o fator de carga como $\lambda \leq 0.5$
  - pode-se mostrar que a perda de desempenho das sondagens acentua muito quando mais da metade da tabela está cheia !

Variantes e Casos Particulares

- **Endereçamento Direto**:
  - Denominado **direct addressing** em inglês, trata-se de um caso muito particular de hashing onde as chaves são naturalmente inteiros em $\{0, 1,..., N-1\}$
  - Nesse caso, a função hash é dispensável, não há colisões se as chaves forem primárias e todas as ops. (inserção, remoção e busca) são garantidamente $O(1)$

23

Variantes e Casos Particulares

- **Endereçamento Direto** (cont.)**:**
  - O problema é que, como não se pode reduzir N, pois os valores das chaves estão naturalmente entre 0 e $N-1$, perde-se o controle sobre o fator de carga
    - Nesse caso, se o no. de chaves é muito menor que N (ou seja, $n << N$) o endereçamento direto implica uma perda de espaço pela alocação de uma tabela de tamanho muito superior ao necessário

24

Variantes e Casos Particulares

- **Hashing Perfeito:**
  - Em aplicações muito particulares onde o conjunto de chaves é estático e conhecido *a priori*, pode-se projetar um hashing perfeito
    - por exemplo, conjunto de palavras reservadas de uma linguagem de programação (aplicação em compiladores)
  - Existem algoritmos para gerar funções hash perfeitas, que não produzem colisões quando aplicadas sobre um dado conjunto de chaves
  - Esse tipo de hashing executa todas as operações (busca, inserção e remoção) em tempo $O(1)$

25

Aplicações

- As aplicações de tabelas hash são variadas
- Basicamente, qualquer aplicação onde se queira acesso imediato a um conjunto de informações através de uma chave, sem ter que procurá-la !
- Alguns exemplos:
  - Compiladores
  - Indexação de Arquivos (hashing externo)
  - etc

26

Exercícios

- Modifique a implementação em C de tabela hash vista em aula, substituindo o tratamento de colisões via endereçamento aberto pela utilização de **encadeamento externo**
- Assuma uma tabela hash (para chaves k inteiras) com N = 11 e função hash dada por h(k) = (2k + 5) mod 11. Desenhe a tabela hash (inicialmente vazia) após a inserção das chaves 12, 44, 13, 88, 23, 94, 11, 39, 20, 16 e 5, assumindo que colisões são tratadas através de **encadeamento externo**
- Repita o exercício anterior assumindo que colisões são tratadas através de **endereçamento aberto com sondagem linear**
- Remova algumas das chaves inseridas no exercício anterior e insira outras que permitam praticar os diferentes fluxos de execução dos algoritmos de inserção, busca e remoção

27

Para saber mais...

- Capítulo 8 (Goodrich & Tamassia, 2002)
- Capítulo 8 (Szwarcfiter & Markenzon, 1994)
- Capítulo 5 (Ziviani, 2004)

28

Bibliografia

- M. T. Goodrich & R. Tamassia, *Data Structures and Algorithms in C++/Java*, John Wiley & Sons, 2002/2005
- M. T. Goodrich & R. Tamassia, *Estruturas de Dados e Algoritmos em Java*, Bookman, 2002
- J. L. Szwarcfiter & L. Markenzon, *Estruturas de Dados e seus Algoritmos*, LTC, 1994
- N. Ziviani, *Projeto de Algoritmos*, Thomson, 2a. Edição, 2004
- T. H. Cormen et al., *Introduction to Algorithms*, MIT Press, 2nd Edition, 2001