Hashing

ACH2002 - Introdução à Ciência da Computação II

Delano M. Beder

Escola de Artes, Ciências e Humanidades (EACH)
Universidade de São Paulo
dbeder@usp.br

11/2008

Material baseado em slides do professor Marcos L. Chaim

Hash

- Considere que queiramos criar um dicionários cujos elementos possuem como chaves Strings de tamanho máximo 8.
- Quantos elementos a tabela de endereçamento direto deveria ter para que todos os possíveis Strings desse tipo possam ser armazenados?
  - para um total de 26 letras.
  - Número de diferentes Strings de no máximo 8 letras é $26 + 26^2 + 26^3 + 26^4 + 26^5 + 26^6 + 26^7 + 26^8$
  - Olha que não estamos considerando algarismos!

Hash

- Apesar do número de chaves possíveis ser grande, o número de palavras armazenadas no dicionário é bem menor na prática.
- Por exemplo, quantas são as variáveis declaradas em um programa?
  - Com certeza é menor do que o número de variáveis possíveis.
- O endereçamento direto não serve porque cada elemento com uma chave k é armazenado na posição k. $\Rightarrow$ Seria necessário um arranjo muito grande, estouraria a memória!
- Como resolver esse problema?

Hash

- Pontos a considerar:
  - Apesar do universo U de chaves ser muito grande, o conjunto de chaves armazenado K é pequeno. $\Rightarrow |K| << |U|$. $\Rightarrow$ Memória necessária é $\Theta(|K|)$.
  - O problema é *espalhar* as chaves contidas no conjunto K em um arranjo de tamanho $|K|$
- Seja K tal que $|K| = m$ então o problema é :
  - Criar uma tabela $T[0...m-1]$ para armazenar as chaves k que ocorrem na prática.
  - Encontrar uma função $h(k)$ tal que: $h(k) = i$ e $0 \leq i \leq m-1$.
- A tabela $T[0...m-1]$ é chamada de *tabela hash*; e $h(k)$, função hash.

Hash

- Vamos generalizar o exemplo da aula anterior:
  - Queremos construir um dicionário dinâmico que pode conter pares do seguinte tipo: (String nome, String significado) onde a chave é *nome*.
  - Os Strings podem ter no máximo 8 letras.
  - Este dicionário dinâmico, apesar de possuir como chave qualquer String de 8 de letras, na prática vai armazenar em média uns 500 elementos.
  - Este dicionário deve ter as seguintes funções: insere, elimina e busca.

Hash

- Problemas:
  - Criar a tabela hash $\Rightarrow$ fácil!
  - Encontrar uma função hash $h(s)$ que mapeia qualquer String s para um número entre 0 e 500.
  - Esta função deve ter complexidade assintótica $\Theta(1)$, como o endereçamento direto.
  - Tratar as possíveis colisões... porque o número de 500 elementos é uma média. Se houver, mais de 500 inevitavelmente haverá colisões.

Hash

Criando a função *hash*:

```
int hash (String nome) {
  int somaValCar = 0;
  int i;

 // Soma os valores dos caracteres

  for(i = 0; i < nome.length() && i < 8; ++i) {
    somaValCar += nome.charAt(i);
  }

  return (somaValCar % 501);
}
```


Hash

```
void insere(String nome, String significado) {
  // Tem que tratar colisão
  ...
 }

 void elimina(String nome) {
   // Tem que tratar colisão
   ...
 }

 TermoDicionario busca(String nome) {
   // Tem que tratar colisão
   ...
   return null;
 }
```

Hash

Tratamento de colisão utilizando listas ligadas:

insere(String nome, String descrição) : insere o objeto TermoDicionario(nome,descrição) no início da lista dic[hash(nome)].

busca(String nome) : procura por um elemento TermoDicionario(nome,descrição) na lista dic[hash(nome)].

elimina(String nome) : elimina o objeto TermoDicionario(nome,descrição) da lista dic[hash(nome)].

Como criar as listas ligadas?

Listas

- Uma maneira de tratar as colisões que ocorrem em tabelas hash é utilizando *listas ligadas*.
- Mas o que são listas ligadas?
  - É alternativa para a implementação de um *tipo abstrato de dados* (TAD) chamado *lista linear*.
- *Lista linear* é uma estrutura em que as operações inserir, eliminar e buscar estão definidas.
- São estruturas muito flexíveis, porque podem crescer ou diminuir de tamanho durante a execução do programa.

Listas

- Listas são adequadas para aplicações nas quais não é possível prever a demanda por memória.
- Permite a manipulação de quantidades imprevisíveis de dados de formato também imprevisível.
  - Exemplo: número de variáveis declaradas e utilizadas em um programa.
  - Exemplo: número de palavras no dicionário mapeadas para o mesmo valor em uma tabela hash.
    - "casa", "saca", "scaa", "asca".

Listas

Definição de **Lista Linear**:

- seqüência de zero ou mais elementos $x_1, x_2, ..., x_n$ na qual x_i é de um determinado tipo e n representa o tamanho da lista linear.
- Os elementos da lista linear possuem uma posição relativa em uma dimensão.
- Assumindo $n \geq 1$, x_1 é o primeiro elemento da lista e x_n é o último.
- Em geral, x_i precede x_{i+1} para $i = 1, 2, 3, \ldots, n-1$. Analogamente, x_i sucede x_{i-1} para $i = 2, 3, \ldots, n$.
- O elemento x_i é dito ser o i-ésimo elemento da lista.

Listas

Para criar um TAD **Lista**, é necessário definir um conjunto de operações sobre os objetos do tipo **Lista**.

Um conjunto simplificado de operações necessário para o uso em tabelas hash é o seguinte:

1. `insereNaLista(x)`: insere x no início da lista.
2. `eliminaDaLista(x)`: verifica se x pertence à lista e retira-o da lista.
3. `buscaDaLista(x)`: verifica se o elemento x pertence à lista.
4. `estaVazia()`: retorna verdeiro ou falso dependendo se a lista está vazia ou não.
5. `tamanhoDaLista()`: retorna o número de elementos da lista.

A linguagem Java tem uma classe **LinkedList** que implementa esse conjunto de operações.

LinkedList – Java

Um conjunto simplificado de operações necessário para o uso em tabelas hash é o seguinte:

1. `insereNaLista(x)`: insere x no início da lista.
   - **void addFirst(Object o)**
2. `eliminaDaLista(x)`: verifica se x pertence à lista e retira-o da lista.
   - **boolean remove(Object o)**
3. `buscaDaLista(x)`: verifica se o elemento x pertence à lista.
   - **boolean contains(Object o)**
4. `tamanhoDaLista()`: retorna o número de elementos da lista.
   - **int size()**
5. `estaVazia()`: retorna verdeiro ou falso dependendo se a lista está vazia ou não.
   - poderia ser implementada: **return size() == 0;**

Resumo

- Tabelas hash: extensão do endereçamento direto; permite tratar uma grande possibilidade de chaves sem ocupar muito espaço.
- Tratamento de colisão em tabelas hash utilizando listas ligadas.

Referências Utilizadas

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest & Clifford Stein.
  Algoritmos - Tradução da 2a. Edição Americana. Editora Campus, 2002
- Michael T. Goodrich & Roberto Tamassia.
  Estrutura de Dados e Algoritmos em Java. Bookman, 2007.