Tabelas *Hash* – Endereçamento Direto

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

Tabela *Hash*

- Muitas aplicações exigem um conjunto dinâmico que admita apenas as operações de dicionário *insere*, *busca* e *elimina*.
- Uma tabela hash é uma estrutura de dados eficiente para implementar dicionários.
- Embora a busca por um elemento em tabela hash possa demorar tanto quanto procurar por um elemento em um arranjo – o tempo $\Theta(n)$ no pior caso – na prática, o hash funciona extremamente bem.
- Sob hipóteses razoáveis, o tempo esperado para a busca por um elemento em uma tabela hash é $O(1)$.
- Uma tabela hash é uma generalização da noção mais simples de um arranjo comum utilizando *endereçamento direto*.

Endereçamento Direto

Considere o seguinte problema:

- Queremos construir um dicionário dinâmico que pode conter somente os seguintes elementos:
- D = {(1, "um"), (2, "dois"), (3, "três"), (4,"quatro"), (5,"cinco"), (6, "seis"), (7, "sete"), (8, "oito") }.
- o dicionário é dinâmico: pode estar vazio, pode conter somente {(3, "três"), (8, "oito")} e {(5,"cinco"), (7, "sete")} ou pode estar completo com os 8 elementos.
- Este dicionário deve ter as seguintes funções: insere, elimina e busca.

Como podemos implementar esse dicionário simplificado?

Endereçamento Direto

- Primeiramente, precisamos criar uma classe do tipo `TermoDicionarioNumero`.

```
class TermoDicionarioNumero {
  int num;
  String descricao;

  TermoDicionarioNumero(int num, String descricao) {
    this.num = num;
    this.descricao = descricao;
  }
}
```

- E agora, como podemos criar o dicionário propriamente dito?

Endereçamento Direto

- O tamanho do dicionário é conhecido, 8 *palavras*.

```
class DicionarioEnderecamentoDireto {
  TermoDicionarioNumero[] conjuntoDinamico;
  conjuntoDinamico = new TermoDicionarioNumero [9];

  void insere(int num, String descricao) {
  ...
  }

  void elimina(int num) {
  ...
  }

  TermoDicionarioNumero busca(int num) {
  ...
  return null;
  }
}
```


Endereçamento Direto

- Vamos começar pelo método `busca`, o ideal seria ter algo assim:
  - conjuntoDinamico[1] : retorna a referência para o objeto `TermoDicionarioNum` igual a { 1, "um"}.
- Como podemos obter isto?
- Podemos criar um mapeamento como o seguinte:
  - 1 → conjuntoDinamico[1];
  - 2 → conjuntoDinamico[2];
  - 3 → conjuntoDinamico[3];
  - ...
  - 8 → conjuntoDinamico[8].

Endereçamento Direto

- Isto é, uma função *chave(int num)* que retorna a posição no arranjo da chave *num*.
- Como seria implementada esta função em Java?

```
  int chave (int num) {
   if(num > 0 && num < 9)
     return num;
   else
     return 0;
  }
```

- Agora com o método `chave()` ficou fácil implementar a busca.

```
  TermoDicionarioNumero busca(int num) {
    return conjuntoDinamico[chave(num)];
  }
```


Endereçamento Direto

- Como fazemos agora para inserir um elemento no conjunto dinâmico?

```
void insere(String nome, String significado) {
   if(chave(nome) != 0)
     dic[chave(nome)] = new TermoDicionario(nome,significado);
}
```

- E para eleminar um elemento no conjunto dinâmico?

```
 void elimina(int num) {
   conjuntoDinamico[chave(num)] = null;
 }
```

- Quando o *endereçamento direto* é aplicável?
- Qual o custo das operações insere, elimina e busca usando endereçamento direto?

Endereçamento Direto

- O endereçamento direto funciona bem quando o universo U de chaves é relativamente pequeno.
- O conjunto dinâmico é tal qual cada elemento tem uma chave definida a partir do universo $U = \{0, 1, ..., m-1\}$, onde m não é muito grande, e não há dois elementos com a mesma chave.
- Para representar o conjunto dinâmico, usamos um arranjo ou uma *tabela de endereço direto* $T[0...m-1]$, na qual cada abertura (slot), ou *posição*, corresponde a uma chave no universo U.
- Se o conjunto não contém nenhum elemento com chave k, então T[k] = null.
- A implementação das operações de dicionário é trivial.

Endereçamento Direto

- Operações de dicionário:
  - buscaEnderecamentoDinamico: return T[k]
  - insereEnderecamentoDinamico: return T[chave(x)] ← x
  - deleteEnderecamentoDinamico: T[chave(x)] ← null

Endereçamento Direto

Considere agora o seguinte problema:

- Queremos construir um dicionário dinâmico simplificado, com 4 palavras de no máximo 8 letras.
- Este dicionário dinâmico pode ter somente as seguintes palavras: "concha", "casa", "hospital"e "time".
- o dicionário é dinâmico: pode estar vazio, pode conter somente "casa"e "concha"ou pode estar completo com as 4 palavras.
- Os termos do dicionário são compostos de um *nome* e um *significado*.
- Este dicionário deve ter as seguintes funções: insere, elimina e busca.

Endereçamento Direto

Podemos implementar utilizando endereçamento direto porém com a função chave seguinte:

```
int chave (String nome) {
  if(nome.equals("concha"))
       return 0;
     else
       if(nome.equals("casa") )
            return 1;
          else
            if(nome.equals("hospital"))
                 return 2;
               else
                 if(nome.equals("time"))
                      return 3;
                    else
                      return 4;
}
```

Endereçamento Direto

Problemas:

- O cálculo da chave não é $\Theta(1)$. Na verdade, é $O(n)$.

Comentários:

- Então não serve porque o custo é igual ao da busca seqüêncial!
- O endereçamento direto requer um mapeamento da chave na sua localização na *Tabela de Endereçamento Direto*. $\Rightarrow$ Isto não acontece neste exemplo.
- Em poucas aplicações isto ocorre, isto é, encontrar um endereçamento direto.

Resumo

- Endereçamento direto: permite inserção, buscas e eliminação a um custo $\Theta(1)$ em conjuntos dinâmicos.
- É um método limitado, pois requer:
  - um universo de chaves limitado $\Rightarrow$ Tabela de endereçamento direto é do tamanho do universo;
  - chaves que possam ser mapeadas para o elementos de um arranjo com custo $\Theta(1)$.
- A idéia do endereçamento direto é estendida nas tabelas *hash*.

Referências utilizadas:

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest & Clifford Stein. *Algoritmos - Tradução da 2a. Edição Americana*. Editora Campus, 2002.