Algoritmos e Estrutura de Dados

Aula 11 – Estrutura de Dados: Tabelas Hash

Parte II

Prof. Tiago A. E. Ferreira

Introdução

- Como visto na aula passada as tabelas hash:
  - Diminuem a quantidade de endereçamento quando comparadas a uma tabela de endereçamento direto
  - Calcula os índices (ou endereços) a partir da função hash (h)

Funções Hash

- O que faz uma função hash de boa qualidade?
  - Satisfaz, aproximadamente, à hipótese do hash uniforme
    - Cada chave tem igual probabilidade de efetuar o hash para qualquer das *m* posições, não importando o hash de qualquer outra chave.
  - Contudo, de forma geral, a distribuição de probabilidade das chaves não é conhecida
    - Impossibilitando a obtenção de um hash uniforme!
  - Ocasionalmente conhecemos a distribuição
    - Por exemplo: **Chaves são números reais aleatórios *k*, tal que 0≤ k ≤ 1, então: *h(k)* = ⌊*km*⌋**

Funções Hash

- Na prática utiliza-se:
  - Heurísticas para definir as funções hash
- De forma geral, as funções hash supoẽ que o universo de chaves é o conjunto de números naturais **N**.
  - E a partir de uma chave qualquer realizam alguma operação para representá-la por um número inteiro!

Método de Divisão

- Neste método mapeia-se uma chave **k** para uma posição **m** tomando o resto da divisão inteira:

$$h(k) = k \mod m$$

  - Por exemplo:
    - Se m=12 e k=100, então h(k)=4
- Valores de **m**
  - De forma geral evita-se potencias de 2
  - Uma boa escolha é um primo não muito próximo de uma potência de 2

Método de Divisão

- Neste método mapeia-se uma chave **k** para uma posição **m** tomando o resto da divisão inteira:

$$h(k) = k \mod m$$

  - Por exemplo:
    - Se m=12 e k=100, então h(k)=4
- Valores de **m**
  - De forma geral evita-se potencias de 2
  - Uma boa escolha é um primo não muito próximo de uma potência de 2

Método de Divisão

- Exemplo:
  - Suponha uma tabela hash, com colisões resolvidas com encadeamento, para conter aproximadamente 2000 cadeias de caracteres
  - Suponha também tolera-se uma pesquisa malsucedida com uma média de 3 elementos
  - Assim, **m=701**
    - Primo próximo de 2000/3 e distante de qualquer potencia de 2.

Método da Multiplicação

- Este método opera em duas etapas:
  - Primeira:
    - Multiplica-se a chave **k** por uma constante **A** (**0 < A < 1**) e extrai-se a parte fracionária de **kA**.
  - Segunda:
    - Multiplica-se o valor encontrado por **m** e toma-se o piso.
- Função Hash:

$$h(k) = \lfloor m(kA \bmod 1) \rfloor$$

Método da Multiplicação

- Uma vantagem deste método é que o valor de **m** não é crítico!
  - Pode-se escolher uma potência de 2
- Quanto ao valor de **A**:
  - Irá funcionar com qualquer valor de **A**. Contudo, funciona melhor com determinados valores que outros.
    - Por exemplo: um valor sugerido é

$$A \approx (\sqrt{5} - 1) = 0{,}6180339887\ldots$$

Hash Universal

- Dado que uma tabela hash tenha uma função hash fixa
  - Assim, um possível ataque a esta máquina seria a escolha das chaves que geram o mesmo hash!
- A única maneira eficaz de melhorar a situação é escolher uma função hash aleatoriamente
  - Esta abordagem é chamada de **hash universal**.
  - Ou seja, escolhe-se uma função hash ao acaso a partir de uma classe de funções cuidadosamente projetada no início da aplicação.

Hash Universal

- Seja **H** uma coleção finita de funções hash que mapeia um dada universo de chaves no intervalo {0, 1, ..., m-1}
  - Esta coleção é dita ser **universal** se, para cada para de chaves **k,l ∈ H**, o número de funções hash **H** para as quais *h(k)=h(l)* é no máximo **H**
  - Ou seja, as chaces de colisão não é maior que **1/m**

Hash Universal

- Teorema 11.3
  - Suponha que uma função hash *h* seja escolhida a partir de uma coleção universal de funções hash e seja usada para efetuar o hash de *n* chaves em uma tabela **T** de tamanho *m*, usando encadeamento para resolver as colisões. Se a chave *k* não está na tabela, estão o comprimento máximo esperado para $n_{h(k)}$ da lista para a qual a chave *k* efetua o hash é no máximo α. Se a chave *k* está na tabela, então o comprimento esperado $n_{h(k)}$ da lista que contém a chave *k* é no máximo $1 + \alpha$.

Hash Universal

- Corolário 11.4:
  - Usando-se o hash universal e a resolução de colisões pelo encadeamento em uma tabela com *m* posições, demora o tempo esperado Θ(n) para tratar qualquer sequência de *n* operações INSERT, SEACH e DELETE contendo O(m) operações INSERT.

Projeto de classe universal de funções hash

- Inicialmente escolhe-se um número primo **p** grande o suficiente para que todas chave **k** possível esteja no intervalo [**0, p-1**]
- Seja Z*p = {0, 1, ..., p-1} e Zp = {1, ..., p-1}, onde **p > m**.
- É possível definir a função hash $h_{a,b}$ para qualquer **a ∈ Z*p** e **b ∈ Zp** sando a transformação linear:

$$h_{a,b}(k)=((ak+b) \bmod p) \bmod m$$

- *Cada função hash* $h_{a,b}(k)$ mapeia Zp para Zm

Endereçamento Aberto

- No endereçamento aberto, todos os elementos estão armazenados na própria tabela hash
  - Para se inserir um elemento com endereçamento aberto, ***sondamos*** a tabela até encontrarmos uma posição vazia.
  - Para determinar quais as posições a serem sondadas, estende-se a função hash:
    - H: U x{0,1,...,m-1}⇨{0,1,...,m-1}
    - Assim, para uma chave **k**, a sequência de sondagem é:
      - ⟨ h(k,0), h(k,1), ..., h(k, m-1) ⟩

HASH-INSERT(T,k)

1. i ⟵0
2. Repeat j⟵h(k,i)
3. if T[j] = None
4. T[j]⟵k
5. return j
6. else i ⟵ i+1
7. Until i = m
8. Error “Hash table overflow”

HASH-SEARCH(T,k)

1. i ⟵0
2. Repeat j⟵h(k,i)
3. if T[j] = K
4. return j
5. i ⟵ i+1
6. Until T[j] = None or i=m
7. Return None

Sondagem Linaer

- Dada uma função de hash comum *h'*, referida como **função hassh auxiliar**, a sondagem linear é:
  - ***h(k,i)=(h'(k)+i) mod m, i=0,1,...,m-1***
- Esta implementação é facil!|
  - Contudo há o aparecimento do **agrupamento primário**
    - **São construídas longas sequências ocupadas!**

Sondagem Quadrática

- A função hash é da forma:
  - $h(k,i) = (h'(k)+c_1 i+c_2 i^2) \bmod m$
    - Onde C_1 e C_2 são constantes auxiliares diferentes de zero.

Análise do Endereçamento Aberto

- Teorema 11.6:
  - Dada uma tabela hash de endereço aberto com fator de carga $\alpha = n/m < 1$, o número esperado de sondagens em uma pesquisa mal sucedida é no máximo $1/(1-\alpha)$, supondo-se hash uniforme.
- Corolário 11.7:
  - A inserção de um elemento em uma tabela hash de endereço aberto com fator de carga α exige no máximo $1/(1-\alpha)$ sondagens em média, supondo-se hash uniforme

Análise do Endereçamento Aberto

- Teorema 11.8:
  - Dada uma tabela hash de endereçamento aberto com fator de carga $\alpha < 1$, o número de sondagens em uma pesquisa bem sucedida é no máximo

$$(1/\alpha)\ln(1/(1-\alpha))$$

supondo-se o hash uniforme e considerando-se que cada chave na tabela tem igual probabilidade de ser pesquisada.

Exercícios Práticos

Exercício 1: Considera uma Tabela hash com m=1000 e a função hash $h(k) = \lfloor m(kA \bmod 1) \rfloor$ para A=(Sqrt(5)-1)/2. Implemente uma função que calcule os hash para uma chave k

Exercício 2: Implemente em Python uma tabela hash de endereçamento aberto . Seja a tabela com 11 posições, e seja a função de hash primária h'(k)=k mod m. Demostre a inserção das chaves 5, 28, 19, 15, 20, 33, 12, 17 e 10.