Algoritmos e Programaço: Estruturas Avançadas de Dados
Trabalho Aula 4

Student: José Luís Rodes Maciel Júnior, josejunior7@edu.unisinos.br
Lecturer: Diogines D'Avila Goldoni, davilag@unisinos.br

Problema 1: Qual a complexidade de pior caso e de melhor caso para a inserção em HashTable com separate chaining?

No caso de uma hash table com separate chaining nós temos o melhor caso da inserção sendo **O(1)** onde o elemento é inserido diretamente no ínicio da bucket sem nenhuma colisão.
O pior caso aconteceria quando nós tivéssemos múltiplas colissões na mesma bucket e nós tivéssimos um sistema de prevenção de duplicatas, onde nós teríamos que percorrer toda a lista para garantir que o valor sendo inserido não exista, fazendo com que tenhamos um pior caso linear de **O(n)**.

Problema 2: Qual a complexidade de pior caso e de melhor caso para deleção em em HashTable com separate chaining?

Neste cenário nós também temos o melhor caso sendo **O(1)**, onde o item a ser deletado está na posiução inicial da bucket e será deletado sem precisar fazer nenhuma pesquisa na lista.
O pior caso também será linear, em que nós tenhamos vários itens na mesma bucket e nós tenhamos que percorrer todos os n itens da lista para poder deletar o item desejado, então a complexidade é uma **O(n)**.

Problema 3: Qual a principal desvantagem da HashFunction de divisão?

A principal desvantagem desse método é a escolha de um tamanho inadequado para a nossa hash table, o que irá acarretar em um má utilização das buckets e causando muitas colisões das chaves, como no caso se escolhermos um número com muitos múltiplos para a quantidade de buckets que nós temos, como por exemplo os números 2, 5, 10, 15, que irão causar muitas colisões e nós teremos uma utilização ruim dos nossos recursos alocados.

Problema 4: Descreva, pelo menos, 3 HashFunctions distintas e dê exemplos de suas vantagens e desvantagens.

1. Hash por Dobramento (Folding):
   Nesta função de hash nós pegamos a chave, dividimos ela em partes menores, então somamos todas estas partes e por fim pegamos o *mod m* desta soma para descobrir em qual bucket está chave será inserida.

   - Vantagem:
     - Útil para grandes itens.
     - Fácil implementação.
   - Desvantagem:
     - Não lida bem com chaves pequenas.
     - Não lida bem com tipos que não sejam ints.
     - Causa colisões em cassos de chaves repetitivas.

2. Hash de Meio-Quadrado (Mid-Square):
   Nesta função nós elevamos a chave ao quadrado, e extraímos uma quantidade x de digitos do meio deste número, então usaremos o número resultante e faremos *mod m* para descobrir o bucket.

   - Vantagem:
     - Distribui bem chaves numéricas com dígitos variados.
     - Simples de implementar para valores inteiros.
   - Desvantagem:
     - Ineficiente para chaves com muitos dígitos repetidos.
     - Não é bom para tabelas pequenas.

3. Hash por Transformação de Base (Ex.: Radix Conversion):
   Nesta função nós mudamos a base númerica em que a chave é representada. Por exemplo converter o número 154 em decimal, para 217 em octal, pegar o resultado destá conversão aplicar o *mod m* para conseguir o bucket em que está chave será inserida.

   - Vantagem:
     - Distribui melhor chaves com padrões em uma base específica.
     - Simples de implementar para tipos numéricos.
   - Desvantagem:
     - Não é boa para tipos de dados que não sejam strings ou ints.
     - Pode ter colisões dependendo o tipo da conversão.