

Notas de Aula - AED1 – Tabela Hash  
Prof. Jefferson T. Oliva

Vários métodos de busca realizam comparação de chaves durante o processamento

Caso os dados estejam ordenados, a busca pode ter um custo de:

- $O(n)$ : busca sequencial
- $O(\log n)$ : busca binária
- $O(\log(\log n))$ : busca por interpolação (se os dados estiverem uniformemente distribuídos)
- $O(\max(m; n/m))$ : busca sequencial indexada
- $O(\max(\log m; \log(n/m)))$ : busca binária indexada (busca sequencial indexada + busca binária)

É possível encontrar uma chave sem a necessidade de compará-la com outras ou com custo constante ( $O(1)$ )?

Sabe-se que em arranjos é possível acessar os dados de forma direta através de um índice, ou seja, com custo de  $O(1)$ .

No entanto, como não é possível saber em qual índice a chave se encontra, geralmente teríamos que fazer uma busca, cuja abordagem depende de vários fatores:

- Tamanho do arranjo
- Distribuição dos dados
- Ordenação
- Etc

## Tabela Hash

As tabelas hash (tabelas de espalhamento, tabelas de dispersão) são uma solução para este problema.

Uma tabela hash associa chaves e valores:

- Chave: uma parte da informação que compõe o item a ser inserido ou buscado
- Valor: posição onde o item deve estar no vetor

Tabelas hash têm várias aplicações, como banco de dados (exemplo, porque o download é rápido? Ao solicitar o download, a função hash serve para localizar o arquivo no banco de dados), tabela de símbolos nos compiladores, jogos, dicionários, redes de computadores (protocolo NAT), antivírus, criptografia (e.g. armazenamento de senhas, blockchain), etc.

Hashing: processamento de chave, cujo resultado é uma posição no arranjo.

A tabela hash usa uma função  $h$ , onde:

- A entrada é uma chave
- A saída é a posição (endereço) onde essa chave deve ser inserida

O endereço é usado para armazenar e recuperar registros.

Com essa função, os dados podem não ser inseridos de forma ordenada

- Por isso, o processo de aplicação de hashing é conhecido "como espalhamento"

Hash significa (Webster's New World Dictionary):

- Fazer picadinho de carne e vegetais para cozinhar
- Fazer uma bagunça

Ideia da tabela hash: particionar um conjunto de elementos (possivelmente infinito) em um número finito de classes:

- B classes (endereços), de 0 a B – 1
- Essas classes são chamadas de buckets

Conceitos relacionados (**slide 10**):

- A função h é chamada de função hash
- $h(k)$  retorna o valor hash de k. O resultado da função é usado como endereço para armazenar a informação cuja chave é k
- k pertence ao bucket  $h(k)$

A função hash é utilizada para inserir, remover ou buscar um elemento. Essa função deve ser determinística, ou seja, resultar sempre no mesmo valor para uma determinada chave.

Colisão: ocorre quando a função hash produz o mesmo endereço para chaves diferentes (**slide 11**).

Distribuição uniforme é muito difícil

Existe chance de alguns endereços serem gerados mais de uma vez e de outros nunca serem gerados

Segredos para um bom hashing:

- escolher uma boa função hash (em função dos dados)
  - Distribui uniformemente os dados, na medida do possível
  - Evita colisões
  - Fácil implementação
  - Rápido ao ser computada
- Estabelecer uma boa estratégia para tratamento de colisões

Exemplo de função hash simples e muito utilizada que produz bons resultados:

- Para chaves inteiras, calcular o resto da divisão  $k\%B$ , sendo que o resto indica a posição de armazenamento
- Para chaves do tipo string, tratar cada caractere como um valor inteiro (ASCII), somá-los e pegar o resto da divisão por B
- B deve ser primo, preferencialmente. Números primos tendem a distribuir os restos das divisões de maneira mais uniforme.

Exemplo com string:

- Seja B um arranjo de 13 elementos:
  - LOWEL = 76 79 87 69 76,
  - L+O+W+E+L = 387,
  - $h(\text{LOWEL}) = 387 \% 13 = 10$ .

## Funções Hash

A função hashing tem o objetivo de mapear o endereço (posição) de uma chave.

Existem diferentes possibilidades para implementar a função de hashing

- Resto de Divisão
- Multiplicação
- Método da dobra

Resto de Divisão: a posição da chave na tabela é dada pelo resto da divisão entre a mesma pelo tamanho da tabela (**ver exemplos entre os slides 16 e 22**).

Multiplicação: a chave é multiplicada por uma constante  $0 < c < 1$  (**ver exemplo no slide 23**)

- Seleção da parte fracionária da multiplicação da chave pela constante
- Multiplicar essa parte pelo tamanho da tabela
- A parte inteira dessa última multiplicação é usada como posição

Método da dobra: a chave é interpretada como uma sequência de dígitos escrita em um papel

- Enquanto a chave for maior que o tamanho da tabela, o papel vai sendo dobrado ao meio e os dígitos que se sobrepõem são somados sem levar em consideração o "vai um"

Colisões: qualquer função hashing pode acarretar em colisões (chaves diferentes ocupam a mesma posição)

Uma função hashing é perfeita quando chaves diferentes sempre são mapeadas para posições diferentes e nunca acontece colisões. Este tipo de função se encaixa em aplicações muito específicas, geralmente quando já se conhece previamente todas as chaves que podem ser inseridas.

Uma função hashing é imperfeita quando podem acontecer colisões e é neste tipo de situação que irão se encaixar a maioria dos problemas. Isso quer dizer que será necessário utilizar alguma abordagem para fazer o tratamento de colisões.

Hash universal:

- A função hash é escolhida aleatoriamente no início de cada execução, de forma que minimize/evite tendências das chaves. Existem diferentes estratégias para gerar funções de hashing universal.
- Por exemplo,  $h(k) = (A * k + B) \% P$ 
  - P é um número primo maior do que a maior chave k
  - A é uma constante escolhida aleatoriamente de um conjunto de constantes  $\{0, 1, 2, \dots, P - 1\}$  no início da execução

→ B é uma constante escolhida aleatoriamente de um conjunto de constantes  $f\{1, 2, \dots, P - 1\}$  no início da execução

- Observação:

→ Quando falamos que a função de hashing é gerada aleatoriamente em tempo de execução, **não significa que a cada item inserido**, uma função diferente é usada. Significa que a sequência de números que será usada pela função não é conhecida previamente. Ela será criada durante a execução do programa.

→ Tipicamente, a mesma função é usada várias vezes até que ela ultrapasse um determinado número de colisões. Então, os números são gerados novamente produzindo uma nova função.

→ Ao mudar a função, todos os itens previamente mapeados tem que ser reorganizados na tabela, conforme a nova função, para que eles possam ser localizados corretamente durante a busca.

- Diz-se que h representa uma coleção de funções universal

## Implementação

Na implementação a ser realizada nessa aula não tratará colisões

O código será aprimorado na próxima aula para o tratamento de colisões

Estrutura de dados simples para tabela hash

```
typedef struct{  
    int tamanho;  
    int *buckets;  
}HashTable;
```

TAD simples para a tabela hash:

```
HashTable* gerarHT(unsigned int tam);  
  
int procurar_chave(unsigned int chave, HashTable* t);  
  
int inserir_chave(unsigned int chave, HashTable* t);  
  
int remover_chave(unsigned int chave, HashTable* t);  
  
void imprimir_tabela(HashTable* t);  
  
int liberar_tabela(HashTable* t);
```

## Códigos .c entre os slides 29 e 34

## Referências

Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C. Introduction to Algorithms. Third edition, The MIT Press, 2009.

Oliva, J. T. Tabela Hash. AE22CP - Algoritmos e Estrutura de Dados I. Notas de Aula. Engenharia de Computação. Dainf/UTFPR/Pato Branco, 2019.

Rosa, J. L. G. Métodos de Busca. SCE-181 - Introdução à Ciência da Computação II. Slides. Ciência de Computação. ICMC/USP, 2018.

Ziviani, N. Projeto de Algoritmos - com implementações em Java e C++. Thomson, 2007.