

# Complexidade de Algoritmos

Prof. Diego Buchinger  
diego.buchinger@outlook.com  
diego.buchinger@udesc.br

Prof. Cristiano Damiani Vasconcellos  
cristiano.vasconcellos@udesc.br

---

# Análise de complexidade de Estruturas de Dados Fundamentais

---

# Estruturas de Dados Básicas

---

Pilha, Fila, Lista Encadeada Simples, Lista Duplamente Encadeada, Árvore Binária, Árvore Rubro Negra

Analisar os seguintes métodos:

\* OBS: Fila tem um ponteiro para o início e o fim!

Fila::enqueue( elemento );

Fila::remove( );

Fila::busca( elemento );

Fila::tamanho( );

# Estruturas de Dados Básicas

---

Pilha, Fila, Lista Encadeada Simples, Lista Duplamente Encadeada, Árvore Binária, Árvore Rubro Negra

Analisar os seguintes métodos:

\* OBS: Lista Encadeada Simples => ponteiro 1º elemento da lista

Elemento => elemento + ponteiro p/ próximo elemento

ListaSimples::adicionarNoInicio( elemento );

ListaSimples::adicionarNoFim( elemento );

ListaSimples::busca( elemento );

ListaSimples::anterior( \*elemento );

ListaSimples::proximo( \*elemento );

# Estruturas de Dados Básicas

---

Pilha, Fila, Lista Encadeada Simples, Lista Duplamente Encadeada, Árvore Binária, Árvore Rubro Negra

Analisar os seguintes métodos:

\* OBS: Lista Dup. Enc. => ponteiro 1º elemento da lista

Element => elemento + ponteiro p/ anterior e próximo elemento

ListaDup::adicionarNoInicio( elemento );

ListaDup::adicionarNoFim( elemento );

ListaDup::busca( elemento );

ListaDup::anterior( \*elemento );

ListaDup::proximo( \*elemento );

# Estruturas de Dados Básicas

---

Pilha, Fila, Lista Encadeada Simples, Lista Duplamente Encadeada, Árvore Binária, Árvore Rubro Negra

Analisar os seguintes métodos:

\* OBS: Árvore => elemento + ponteiro para filhos

Árvore::inserir( elemento );

Árvore::buscar( elemento );

Árvore::pai( \*no );

Árvore::maximo( );

\*\* E se cada nó tiver um ponteiro para o pai?

# Atividade

---

Analise o código fonte do arquivo estruturas-dados.cpp e informe as complexidades de tempo e espaço dos seguintes métodos considerando o **pior** e **melhor** caso. Ilustre um exemplo de situação onde o **pior** caso e o **melhor** caso ocorreria para cada uma das funções.

[Faça em uma folha para entregar] [no máximo **duplas**]

Pilha	ListaEncadeada	ÁrvoreVP
Empilha	Adicionar	rotateLeft
Desempilha	Anexar	Inserir
Limpar	Remover	Busca
Print	Encontrar	PreOrder
Tamanho	RemoverDuplicatas	Maximo

# Análise de Complexidade envolvendo múltiplas variáveis

---



# Strings

---

Ao analisar algoritmos envolvendo *strings* devemos nos preocupar com o seu tamanho que pode não ser constante.

Qual o pior caso? Qual o melhor caso?

**Ex1:** qual seria a complexidade para processar  $n$  nomes de pessoas deixando todos em maiúsculas?

**Ex2:**

```
int compara(string[] nomes, int n){
    int i,j,r=0;
    for(i=0; i<n; i++)
        for(j=i+1; j<n; j++)
            if( strcmp(nomes[i] nomes[j]) == 0 )
                r += 1;
    return r;
}
```

# Hash

---

Uso de vetores para armazenar elementos é bom.

- Qual a complexidade para acessar um elemento?
- Qual a complexidade para adicionar um elemento?

Limitações dos vetores: os índices só podem ser inteiros

P: Alguma sugestão para podermos usar outros valores (de qualquer tipo) como índices?

# Hash - Introdução

---

Podemos criar uma operação de conversão de um tipo qualquer para um valor inteiro positivo = **FUNÇÃO HASHING**

**(chave)  $\Rightarrow$  índice**

Como poderíamos criar uma função para converter uma string em um inteiro positivo?

- 1 – Somar o código ascii de cada letra
- 2 – Somar o código ascii de cada letra \* posição
- 3 – Somatório( código ascii de cada letra \*  $255^{\text{posição}}$  )

Testes: “Joao”, “Maria”, “ola”, “alo”, “oi”, “io”

# Hash - Introdução

---

Na conversão precisamos nos preocupar com mais um detalhe:

Qual é o tamanho do nosso vetor?

Fator de carga do Hash:  $\lambda = \frac{n^{\circ} \text{ elementos}}{\text{tamanho vetor}}$

Se tivermos um vetor de **10 posições**, qual operação podemos usar para garantir que o nosso inteiro positivo esteja entre 0 e 9?

# Hash - Introdução

---

Hash - método da divisão:

Escolhemos um valor  $m$  como divisor de modo a evitar colisões (um número que **não** seja potência de 2  $\Rightarrow \neq 2^p$ )

Um bom valor pode ser um número primo ou então  $2^p - 1$

Podemos realizar duas divisões:  $(valor \% m) \% tam$   
sendo  $m > tam$

# Hash - Introdução

---

Quando usamos uma função hashing que realiza um mapeamento único para cada índice do vetor temos um **hash perfeito**

Se os nomes só podem ter no máximo 5 letras, qual tamanho de vetor podemos utilizar para garantir um hash perfeito?

(note que um hash perfeito depende da função de hashing)

# Hash - Introdução

---

Quando usamos uma função hashing que não realiza um mapeamento único para cada índice do vetor precisamos considerar também que a função de hashing pode gerar **números iguais** para **chaves diferentes**:

Hashing (2) + vetor de 10 posições: “Maria”, “Ana”

A esse evento chamamos de **COLISÃO!**

Nem sempre é viável utilizar um hash perfeito. Logo, uma função hashing deve sempre buscar **minimizar** o número de colisões

# Hash - Colisões

---

Ok, Mas o que fazer quando ocorre uma colisão?

Abordagem 1: Não adicionamos o registro

Abordagem 2 – endereçamento aberto:

Procurar pela próxima posição vaga

❖ Sondagem Linear: avança de um em um

Método deve ser usado para inserção e busca

exemplo: busca pelo 55 (hash: 0); busca pelo 66 (hash: 1)

0	1	2	3	4	5	6	7	8	9	10
77	44	55	20	26	93	17	None	None	31	54



# Hash - Colisões

---

Ok, Mas o que fazer quando ocorre uma colisão?

## Abordagem 2 – endereçamento aberto:

Sondagem Linear pode gerar agrupamentos (*clusters*) de elementos (não interessante)

❖ Sondagem Quadrática: avança usando os quadrados

$x+1, x+4, x+9, x+16, x+25, (...)$

Método deve ser usado para inserção e busca

0	1	2	3	4	5	6	7	8	9	10
77	44	55	20	26	93	17	None	None	31	54

# Hash - Colisões

---

Ok, Mas o que fazer quando ocorre uma colisão?

## Abordagem 2 – endereçamento aberto:

❖ Hash duplo (rehash): aplicação de uma nova função hash

*novo hash = rehash( hash antigo )*

*rehash(x) = (x + 1) % tamanho vetor*

rehash deve garantir que, eventualmente, todas as posições serão avaliadas

Método deve ser usado para inserção e busca

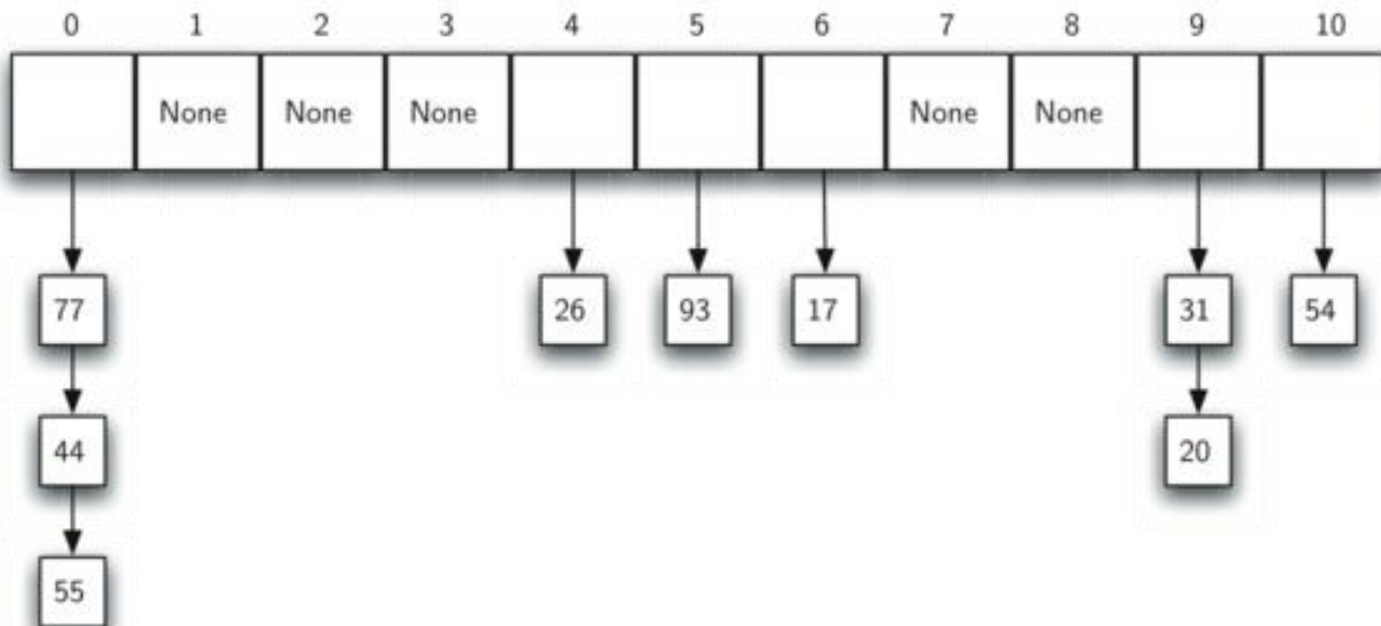
0	1	2	3	4	5	6	7	8	9	10
77	44	55	20	26	93	17	None	None	31	54

# Hash - Colisões

Ok, Mas o que fazer quando ocorre uma colisão?

## Abordagem 3 – encadeamento:

Ao invés de utilizarmos um vetor simples utilizamos um vetor de alguma estrutura de dados (lista, árvore)



# Hash – Principais funções

---

**Hash**( **int** tamanho )

**put**( chave, valor ) => boolean

**get**( chave ) => valor

**listar**( ) => (chaves, valores)

# Hash – Analise Complexidade

---

Considerando um hash de strings com capacidade  $n$  e tamanho de strings  $s$ :

- Qual a complexidade para calcular a função hash (1), (2) e (3)?
- Qual a complexidade (melhor e pior caso) para inserir e buscar um elemento usando sondagem linear? E para listar todos os elementos do hash?

# Hash – Analise Complexidade

---

Considerando um hash de strings com capacidade  $n$ , e tamanho de strings  $s$ :

- Qual a complexidade (melhor e pior caso) para inserir e buscar um elemento usando hash encadeado com lista? E para listar todos os elementos do hash?
- Qual a complexidade (melhor e pior caso) para inserir e buscar um elemento usando hash encadeado com árvore balanceada (árvore rubro-negra)?

# Atividade Extra

---

Resolva os seguintes problemas do URI online Judge e determine a complexidade para o pior caso:

- 1861 – O Hall dos Assassinos

([http://www.urionlinejudge.com.br/repository/UOJ\\_1861.html](http://www.urionlinejudge.com.br/repository/UOJ_1861.html))

**OBS: Listagem de resultados:**

**Accepted** – sua solução foi aceita;

**Wrong Answer** – o algoritmo produziu uma resposta incorreta para alguma situação;

**Time Limit Exceeded** – seu algoritmo entrou em loop infinito ou demorou muito (rever algoritmo e reduzir a sua complexidade de tempo);

**Runtime Error** – seu programa fez alguma operação indevida (divisão por zero ou acesso a memória inválida);