# Estrutura de Dados

Hamilton José Brumatto

Bacharelado em Ciências da Computação - UESC

31 de janeiro de 2024

## Ementa

Representação de dados. Estruturas lineares: vetor, lista, pilha e fila. Recursão. Árvores binárias. Árvores de busca. Árvores balanceadas. Algoritmos para manipulação de estruturas: inserção, remoção, busca e percurso. Ordenação de dados. Heaps. Filas com prioridades. Noções de complexidade dos algoritmos utilizados.

## Estrutura de Dados

Para cada estrutura é apresentada a definição e algoritmos para inserção, remoção e busca de dados tanto quanto algoritmos de percurso. Também é apresentada a noção de complexidade dos algoritmos.

# 1 Introdução:

## 1.1 Tipos primitivos e abstratos de dados: revisão

*Como o computador armazena os tipos de dados. Como ampliar a representação de dados.*

## 1.2 Ponteiros: revisão

*Manipulação dos dados por endereço e referência.*

## 1.3 Vetores.

*Estrutura linear (sequencial na memória) de dados.*

## 1.4 Listas dinâmicas.

*Estrutura linear (dinamicamente alocável na memória) de dados.*

## 2 Estruturas Lineares:

### 2.1 Listas.

*Inserção e remoção em qualquer posição da lista.*

### 2.2 Pilhas.

*FILO - Primeiro a ser inserido é o último a ser removido.*

### 2.3 Aplicações em Pilhas.

*Principais aplicações, como por exemplo, simulação de recursão.*

### 2.4 Filas.

*FIFO - Primeiro a ser inserido é o primeiro a ser removido.*

### 2.5 Filas com prioridade.

*O elemento a ser removido obedece hierarquia de prioridade e inserção.*

# 3 Estruturas de Árvores.

## 3.1 Árvores Gerais Enraizadas.

*Implementação geral de uma estrutura de árvore enraizada.*

## 3.2 Árvores N-árias.

*Arvores com N filhos, em especial: Árvores Binárias.*

## 3.3 Árvores Binárias de Busca.

*Árvore Binária com ordenação específica.*

## 3.4 Árvores Múltiplas de ordem M.

*Árvores com múltiplas chaves e filhos (normalmente de busca).*

## 3.5 Árvores Balanceada.

*Árvores de busca balanceadas (Binária ou Múltipla).*

## 3.6 Heap.

*Árvore Binária implementada em vetor (com prioridade).*

4 Classificação de Ordem:

*Ordenação de elementos em uma estrutura linear.*

4.1 Principais algoritmos de ordenação.

*Ordenação em tempo quadrático e $n \log n$.*

4.2 Algoritmos de ordenação restritos.

*Algoritmos de ordenação em tempo linear.*

São duas provas teóricas no semestre e um conjunto de práticas:

- Prova 1: Avaliação Teórica 1 (A1)
- Prova 2: Avaliação Teórica 2 (A2)
- Exercícios Práticos (P): São 13 ao longo do semestre.

  *Serão consideradas as 10 melhores notas em uma média simples.*

**MS: Cálculo da Média do Semestre**

$$MS = (A1 + A2 + P)/3$$

- Avaliação Final ($AF$): Se necessária

**MF: Cálculo da Média Final do Semestre**

$$MF = \begin{cases} MS, & \text{se } MS < 1,6 \text{ ou } MS \geqslant 7 \\ 0,6 \times MS + 0,4 \times AF, & \text{se } 1,6 \leqslant MS < 7 \end{cases}$$

Será aprovado quem tiver $MF \geqslant 5$

Informações adicionais: Moodle - cicuesc.pro.br/

- O plano do curso, detalhado, incluindo o cronograma de aula com temas, dias não letivos, datas de provas.
- Todo o material de curso, incluindo slides de aulas e provas anteriores
- A participação em sala e através da dedicação em exercícios é fundamental para a progressão na disciplina, tanto que não será feita uma cobrança direta de presença, esta cobrança se dá automaticamente através das avaliações previstas na disciplina (Dentro do critério de 75% obrigatórias).
- Também existem fóruns para tirar dúvidas e dar avisos sobre os temas da classe.

1 Leiserson, Charles E.; Stein, Clifford; Rivest, Ronald L.; Cormen,Thomas H. **Algoritmos - Trad. 2ª Ed. Americana**, Editora Campus, 2002.

2 Preiss, Bruno. Estruturas de Dados e Algoritmos Editora Campus, 2001.

3 Drozdek, Adam. Estruturas de Dados e Algoritmos em C++. Thomson Pioneira, 2001.

4 Skiena, Steven S. The Algorithm Design Manual Springer-Verlag, 1997. Online: `http://www2.toki.or.id/book/AlgDesignManual/`

5 Skiena, Steven S.; Revilla, Miguel A. Programming Challenge Springer, 2003

6 LaFore, R. Aprenda em 24 Horas Estruturas de Dados e Algoritmos. Editora Campus, 1999.

1 Leiserson, Charles E.; Stein, Clifford; Rivest, Ronald L.; Cormen,Thomas H. **Algoritmos - Trad. 2ª Ed. Americana**, Editora Campus, 2002.

2 Preiss, Bruno. Estruturas de Dados e Algoritmos Editora Campus, 2001.

3 Drozdek, Adam. Estruturas de Dados e Algoritmos em C++. Thomson Pioneira, 2001.

4 Skiena, Steven S. The Algorithm Design Manual Springer-Verlag, 1997. Online: `http://www2.toki.or.id/book/AlgDesignManual/`

5 Skiena, Steven S.; Revilla, Miguel A. Programming Challenge Springer, 2003

6 LaFore, R. Aprenda em 24 Horas Estruturas de Dados e Algoritmos. Editora Campus, 1999.

# O Aluno Aprende

**Não significa: O professor faz o aluno aprender.**

## Como devo estudar?

A construção de algoritmos representa problemas "Quebra-Cuca".
Entender a solução de um problema NÃO representa que a mesma solução irá resolver outro problema.
A experiência se adquire com a prática.

Será necessário resolver listas e listas de exercícios. Em especial, participar das avaliações práticas com originalidade (sem ficar procurando solução na internet ou junto a amigos) Resolva você!

# O Aluno Aprende

**Não significa: O professor faz o aluno aprender.**

## Como devo estudar?

A construção de algoritmos representa problemas "Quebra-Cuca". Entender a solução de um problema NÃO representa que a mesma solução irá resolver outro problema.
A experiência se adquire com a prática.

Será necessário resolver listas e listas de exercícios. Em especial, participar das avaliações práticas com originalidade (sem ficar procurando solução na internet ou junto a amigos) Resolva você!

# Mantendo tudo simples

- Projeto (IDE) x Código (C++)
- O IDE (*Integrated Development Enviroment*) é um ambiente criado para projetos grandes, com vários códigos fontes.
- Exemplos de IDE: Eclipse, Visual Studio, ..., CodeBlocks, ... "Make"
- No IDE precisamos de um projeto, e este projeto especifica todos os códigos fontes, diretivas de compilação e de edição de ligações.
- Não precisamos disto, vamos ficar no simples: C++ → compilador → executável. Ou tudo numa "coisa" só: Python

# C/C++ .. Python

- Vamos adotar o C++ como linguagem, sem a preocupação ao aspecto envolvido com a orientação a objetos.
- Tudo que for escrito no C funciona no C++
- Se sabe escrever um código usando C que funcione, então este código irá funcionar com o C++
- Vantagens:
  - E/S: não há a necessidade de indicar toda aquela string de formatação para ler ou escrever.
  - strings: são um tipo nativo é possível ler, escrever sem se preocupar com o '\0'.
  - Outras vantagens virão com o tempo (como o STL) mas não se aplica agora.
- Mas nada impede de usar outra "coisa", como Python

# Foco no que é pedido

- Cada problema tem um objetivo, um aprendizado. Uma vez conquistado o aprendizado, não precisamos repetir o mesmo exercício em todos os problemas.
  - *Faça um programa que leia um número e verifique se este número está contido entre 1 e 10. O programa deve retornar: "OK" se estiver contido, "ERRO" caso contrário, em uma linha, apenas.*
  - *Faça um programa que leia um número entre 1 e 10 e verifique se ele é par ou ímpar. O programa deve retornar: "PAR" se for par ou "Ímpar" caso contrário, em uma linha, apenas.*
- Veja no primeiro precisamos verificar os limites do número fornecido. No segundo não, pois o número fornecido, com certeza será entre 1 e 10.

# Iteração inútil

- ED, PAA, TAA, PP, ... tudo visa construção de código científico/algoritmo. Diferente de ASI, IHC, ... que visa a iteração com o usuário.
- Assim, não precisamos nos preocupar com a iteração. Se o problema pede que leia um número, não precisa pedir para o usuário um número:
  - Será fornecido ao programa, um número.
  - Com certeza será um número e não uma letra ou símbolo qualqer.
  - Ainda assim, se houver uma determinação do limite que o número deve respeitar, este número com certeza respeitará este limite.
- Somente imprima no console, aquilo que o problema pediu para gerar como saída.

# Exemplo 1:

Faça um programa que leia um número e verifique se este número está contido entre 1 e 10. O programa deve retornar: "OK" se estiver contido, "ERRO" caso contrário, em uma linha, apenas

## Solução: ed.00.intro.c01.cpp

```
#include <iostream>

using namespace std;

int main() {
  int n;
  cin >> n;
  if(n < 1 || n > 10) cout << "ERRO" << endl;
  else cout << "OK" << endl;
  return 0;
}
```

# Rodando o código:

```
g++ -Wall -std=c++20 -o ed.00.intro.c01 ed.00.intro.c01.cpp
```

- Este comando irá compilar e gerar o executável `ed.00.intro.c01`
- `-std=c++20` indica para o compilador utilizar o padrão versão 20 do C++
- `-Wall` indica para o compilador verificar todos os níveis de *Warnings* existentes
- Se o comando não gerar nenhuma saída impressa, então a compilação ocorreu de forma prevista e o executável está disponível

# Rodando o código:

- Os dados que se esperam ser digitados pelo usuário estarão em um arquivo:
  `ed.00.intro.c01.1.in`, por exemplo, contém o valor: 13
- Para executar:
  `./ed.00.intro.c01 < ed.00.intro.c01.1.in`
- O resultado aparece:
  `ERRO`

# Rodando o código:

- Rodando tudo de uma vez:

```
$ for i in 1 2 3 4 5; do ./ed.00.intro.c01 < ed.00.intro.c01.$i.in; done
ERRO
OK
OK
ERRO
OK
$ for i in 1 2 3 4 5; do cat ed.00.intro.c01.$i.in; done
13
3
5
-5
1
```

# Exemplo 2:

Faça um programa que leia um número entre 1 e 10 e verifique se ele é par ou ímpar. O programa deve retornar: "PAR" se for par ou "Ímpar" caso contrário, em uma linha, apenas.

## Solução: ed.00.intro.c02.cpp

```
#include <iostream>

using namespace std;

int main() {
  int n;
  cin >> n;
  if(n % 2) cout << "IMPAR" << endl;
  else cout << "PAR" << endl;
  return 0;
}
```

# Rodando o código:

```
$ g++ -Wall -std=c++20 -o ed.00.intro.c02 ed.00.intro.c02.cpp
$ for i in 1 2 3 4 5; do ./ed.00.intro.c02 < ed.00.intro.c02.$i.in; done
PAR
IMPAR
IMPAR
PAR
IMPAR
$ for i in 1 2 3 4 5; do cat ed.00.intro.c02.$i.in; done
10
3
5
4
1
```

# Exemplo 3: Uma função:

No arquivo de nome "`intro.f01.cpp`", construa uma função cujo protótipo seja:
`int soma(int, int);`
Que receba dois valores inteiros $A$ e $B$ e retorne a soma numérica destes valores. Os valores recebidos estão no intervalo de $(-10^5 \leqslant A, B \leqslant 10^5)$

**intro.f01.cpp**

```
int soma(int x, int y) {
 return x + y;
}
```

# Exemplo 3: Resolvendo

- Neste exemplo, especificamos o nome do arquivo. Observe que em todas as práticas, o nome do arquivo será um nome específico, pois os algoritmos de testes serão preparados para aqueles arquivos específicos.
- No caso da função é necessário seguir o protótipo proposto, pois sempre será necessário uma função `main` que venha a chamar esta para testar sua funcionalidade.
- A informação do intervalo do valor inteiro serve para garantir que o número cabe em uma variável do tipo "`int`", por exemplo, se falássemos em $10^9$ teríamos de trabalhar com "`long int`"
- Para executar, são compilados separados o arquivo com a função, o arquivo com a `main` e depois juntos num executável

# Exemplo 3: Função main

ed.00.intro.c03.cpp

```
#include <iostream>

int soma(int,int);

using namespace std;

int main() {
 int a, b;
 cin >> a >> b;
 cout << soma(a, b) << endl;
 return 0;
}
```

# Exemplo 3: Rodando tudo

```
$ g++ -c -Wall -std=c++20 -o intro.f01.o intro.f01.cpp
$ g++ -c -Wall -std=c++20 -o ed.00.intro.c03.o ed.00.intro.c03.cpp
$ g++ -std=c++20 -o ed.00.intro.c03 intro.f01.o ed.00.intro.c03.o
$ for i in 1 2 3 4 5; do ./ed.00.intro.c03 < ed.00.intro.c03.$i.in; done
19923
3984
1000
0
2
$ for i in 1 2 3 4 5; do cat ed.00.intro.c03.$i.in; done
9930 9993
-4829 8813
-8925 9925
1 -1
1 1
```

# Passagem de parâmetros: Por valor

- Quando fazemos uma passagem de parâmetros por valor, na função recebemos uma cópia (o valor) do parâmetro.
- Qualquer mudança na variável da função, que contém a cópia do valor passado, não altera o valor da variável original.

```
#include <iostream>
using namespace std;
void altera(int x) {
 x = 4;
 return;
}
int main() {
 int x = 3;
 altera(x);
 cout << "x = " << x << endl;
}
```

```
x = 3
```

# Passagem de parâmetros: Por endereço

- Neste caso passamos uma cópia do endereço da variável.
- Na função, teremos um ponteiro que contém este endereço. Podemos alterar o conteúdo da variável.

```
#include <iostream>
using namespace std;
void altera(int *x) {
  *x = 4;
  return;
}
int main() {
  int x = 3;
  altera(&x);
  cout << "x = " << x << endl;
}
```

```
x = 4
```

# Passagem de parâmetros: Por referência

- Neste caso a função recebe uma referência para a variável do parâmetro.
- A variável da função é um link para a variável original. Podemos alterar o conteúdo da variável.

```
#include <iostream>
using namespace std;
void altera(int &x) {
  x = 4;
  return;
}
int main() {
  int x = 3;
  altera(x);
  cout << "x = " << x << endl;
}
```

```
x = 4
```