Newton

Quem se prepara, não para.

Estrutura de Dados

Professora: Michelle Hanne Soares de Andrade
michelle.andrade@newtonpaiva.br

Árvore

Newton
Quem se prepara, não para.

Dado um conjunto de vértices e arestas, um caminho e uma lista de vértices distintos na qual cada vértice na lista e conectado ao próximo por uma aresta.

- **Árvore (livre):** Um conjunto de vértices (nodos) e arestas que satisfaz a seguinte condição: existe exatamente um caminho conectando qualquer par de nodos.
  - Se houver algum par de nodos para o qual existe mais de um caminho ou nenhum caminho temos um grafo.
- **Floresta:** Um conjunto de arvores disjuntas.

Árvore

Newton
Quem se prepara, não para.

Em computação: Em geral, árvores referem-se a estruturas que possuem um nodo designado como raiz. Nestas árvores, cada nodo é a raiz de uma sub árvore.

Fonte: *Secoes 5.1 e 7.2 do livro "Estruturas de Dados Usando C" de Tenembaum, Langsam e Augenstein*

Árvore

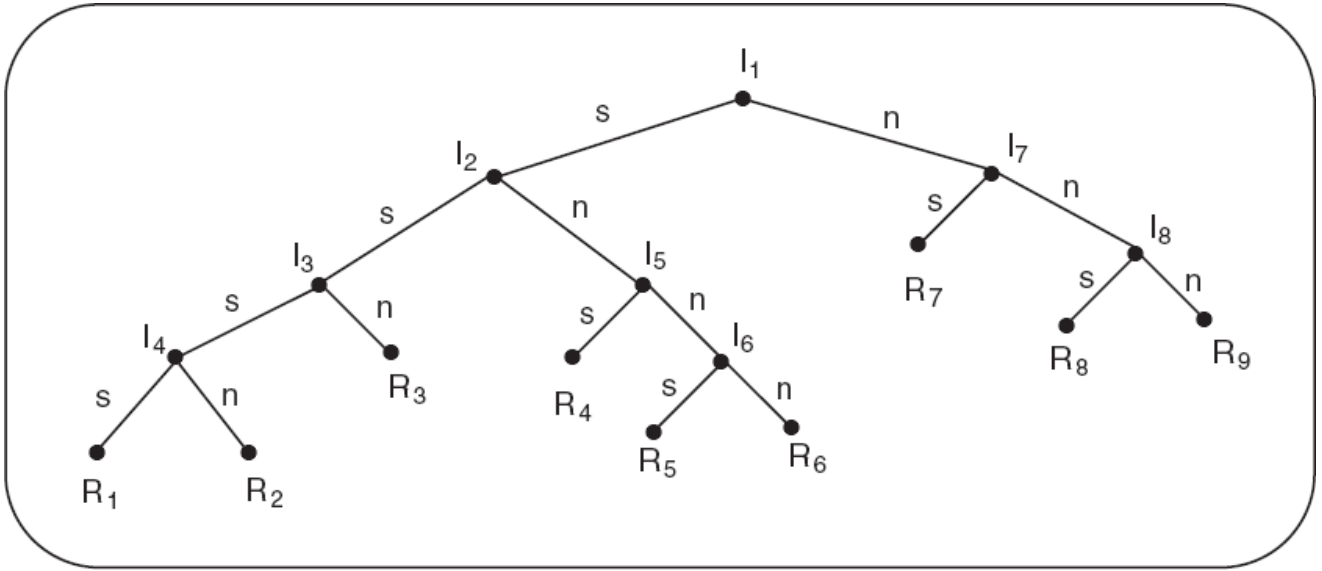
Newton
Quem se prepara, não para.

Desenho da árvore:

- **Raiz no topo**:
  - existe a noção de um nodo estar acima (mais próximo da raiz) ou abaixo dele (mais longe da raiz)
  - **PAI**: todo nodo, exceto a raiz tem um único pai, que é o nodo logo acima dele
  - **FILHOS**: são os nodos logo abaixo de um determinado nodo
  - **FOLHAS** ou nodos terminais: nodos que não possuem filhos
  - **Nodos INTERNOS** ou não terminais: que possuem filhos

Fonte: *Secoes 5.1 e 7.2 do livro "Estruturas de Dados Usando C" de Tenembaum, Langsam e Augenstein*

Árvore



Estrutura de árvore de decisão.

Fonte: Nicoletti, 2018

Árvore

Newton
Quem se prepara, não para.

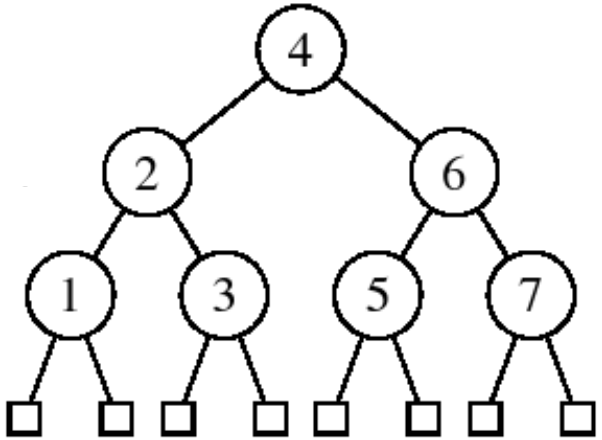
Definição :

- Seja G = (V,E) um grafo.
- **(1)** G é *acíclico* se G não contém ciclos.
- **(2)** G é uma *árvore* se G for um grafo acíclico conexo.
- **(3)** G é uma *floresta* se G for acíclico, independentemente de ser conexo ou não.

Fonte: Nicoletti, 2018

Árvores

Árvores Ordenadas: árvores nas quais a ordem dos filhos é significativa
Árvores n-aria: árvores nas quais todos os nodos internos obrigatoriamente tem "n" filhos. Ex: arvore binaria.



Nível de um nó:

Nível da raiz = 0
Nível de outros nós = nível do pai + 1

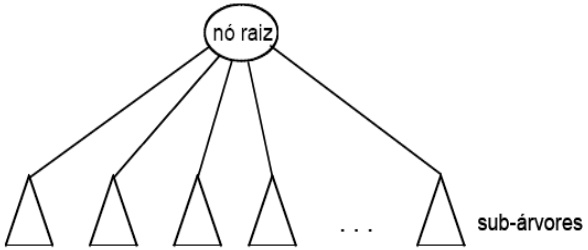
Altura da arvore:

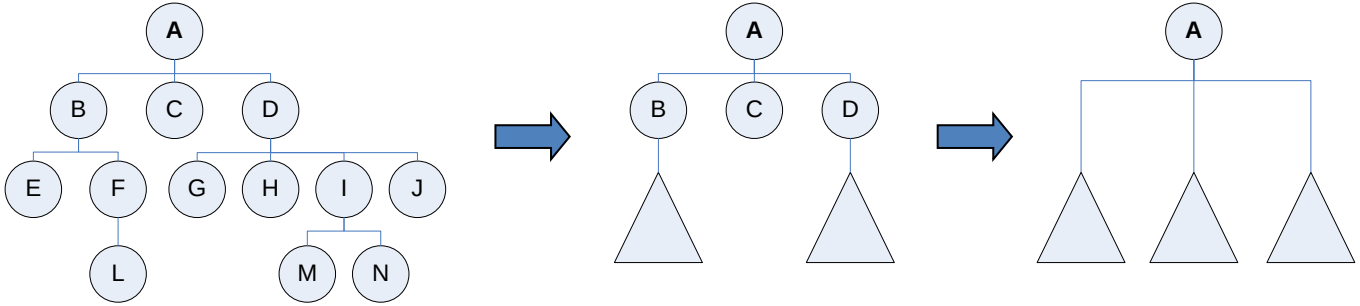
Nível máximo de um nodo (interno ou externo) da arvore.

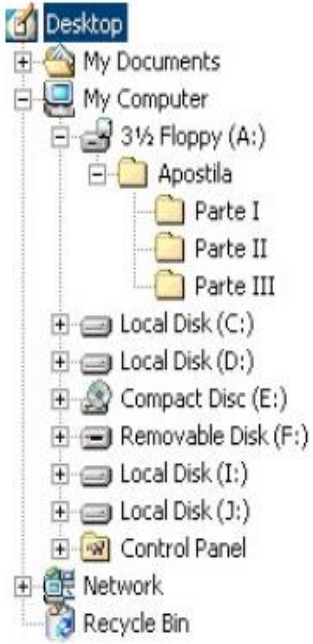
Árvores

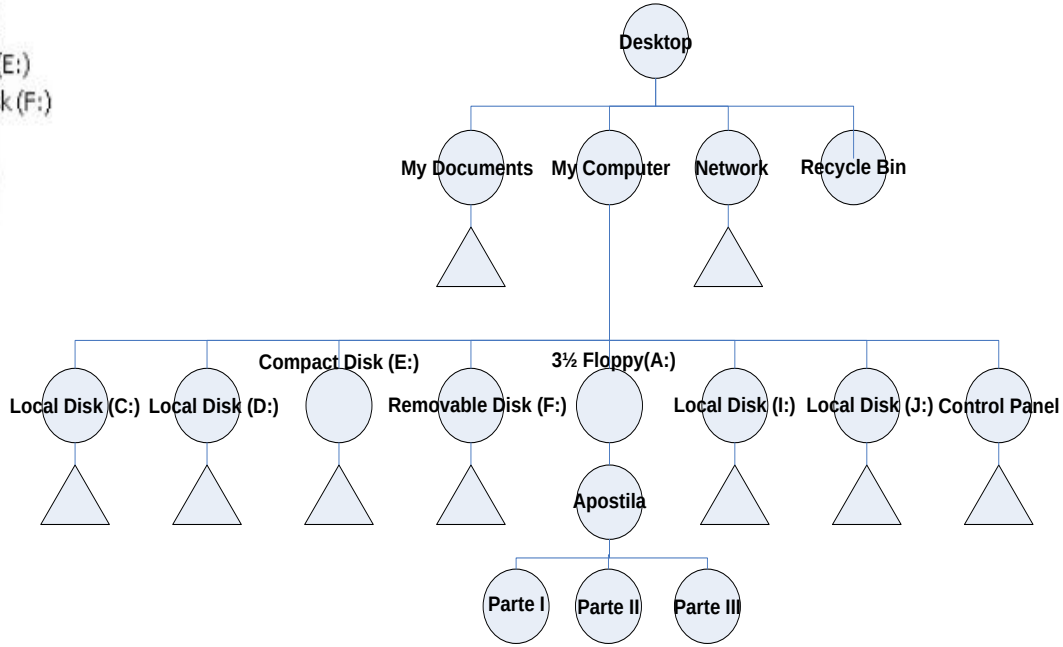
Newton
Quem se prepara, não para.

- **Árvore** é uma estrutura de dado adequada para representar hierarquias;
- Definida por um conjunto de **nós**;
- Desse conjunto, há um nó ***R*** denominado de raiz, que contém zero ou mais *sub-árvores*, cujas raízes são ligadas diretamente a ***R***.
- Esses nós raízes das sub-árvores são ditos **filhos** do nó pai, ***R***;
- Nós com filhos são comumente chamados de ***nós internos***;
- Nós que ***não*** têm filhos são chamados de ***nós externos*** (***folhas***);


nó raiz
. . .
sub-árvores

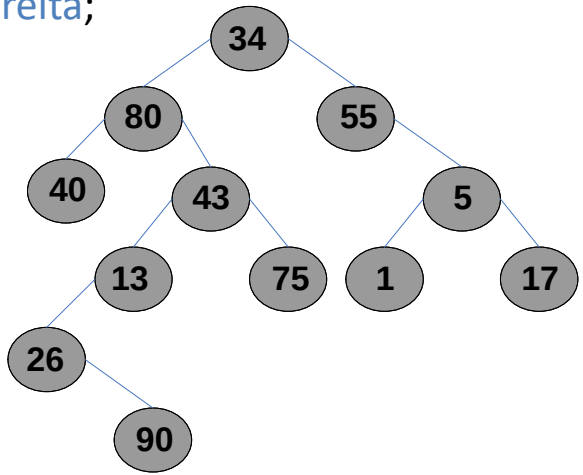

A
B
C
D
E
F
G
H
I
J
L
M
N
A
B
C
D
A


Desktop
My Documents
My Computer
3½ Floppy (A:)
Apostila
Parte I
Parte II
Parte III
Local Disk (C:)
Local Disk (D:)
Compact Disc (E:)
Removable Disk (F:)
Local Disk (I:)
Local Disk (J:)
Control Panel
Network
Recycle Bin


Desktop
My Documents
My Computer
Network
Recycle Bin
Local Disk (C:)
Local Disk (D:)
Compact Disk (E:)
Removable Disk (F:)
3½ Floppy(A:)
Local Disk (I:)
Local Disk (J:)
Control Panel
Apostila
Parte I
Parte II
Parte III

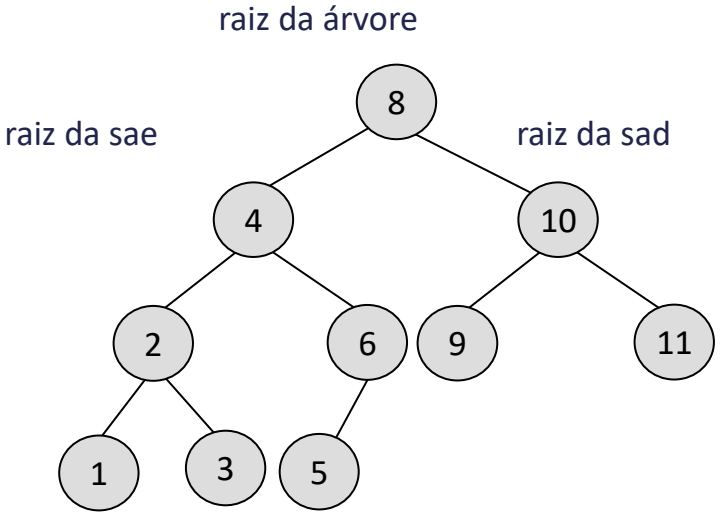
Árvores Binária

- Uma árvore binária é definida como um conjunto finito de nós que:
  - ou está vazio (árvore vazia);
  - ou consiste de um nó raiz mais os elementos de duas árvores binárias distintas – sub-árvore esquerda e sub-árvore direita;
- Cada nó tem, no máximo, duas sub-árvores;



Árvores de Pesquisa

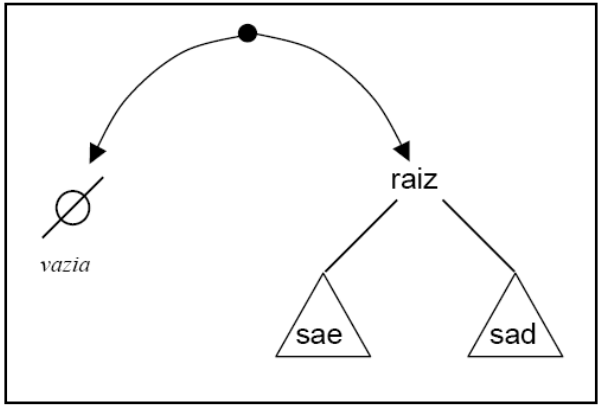
- A árvore de pesquisa é uma estrutura de dados muito eficiente para armazenar informação.
- Particularmente adequada quando existe necessidade de considerar todos ou alguma combinação de:
  1. Acesso direto e seqüencial eficientes.
  2. Facilidade de inserção e retirada de registros.
  3. Boa taxa de utilização de memória.
  4. Utilização de memória primária e secundária.



Árvores de Pesquisa

Temos a relação invariante

1. Todos os registros com chaves menores estão na subárvore à esquerda.
2. Todos os registros com chaves maiores estão na subárvore à direita.



SAD: Subarvore à direita
SAE: Subarvore à esquerda

Percursos em Árvores Binárias

Newton
Quem se prepara, não para.

- **Muitas operações** em árvores binárias envolvem o percurso de todas as suas sub-árvores, executando alguma ação de tratamento em cada nó;

- É comum percorrer uma árvore em uma das seguintes ordens:

    - **Pré-Ordem**: tratar ***raiz***, percorrer ***sae***, percorrer ***sad***;

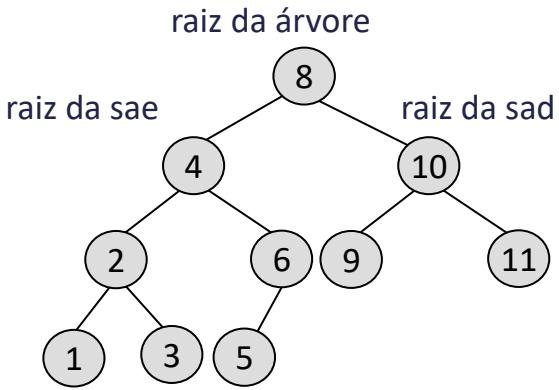
    - **In-Ordem (ordem simétrica)**: percorrer ***sae***, tratar ***raiz***, percorrer ***sad***;

    - **Pós-Ordem**: percorrer ***sae***, percorrer ***sad***, tratar ***raiz***.

Pré-ordem

Newton
Quem se prepara, não para.

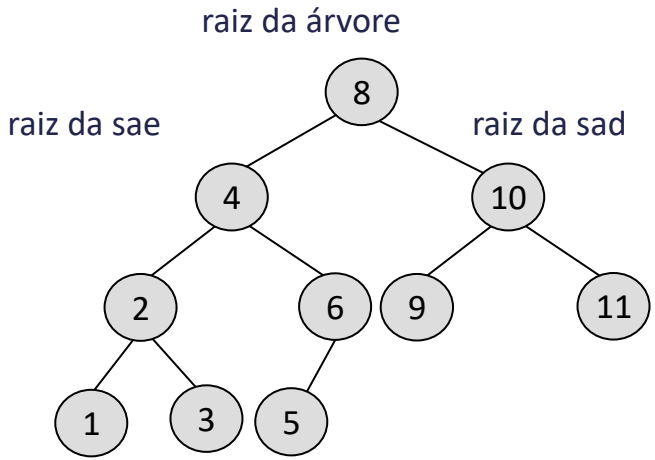
• **Imprima** os valores presentes nos nós da árvore ao lado, segundo a condição ***pré-ordem*** (tratar ***raiz***, percorrer ***sae***, percorrer ***sad***).



Resultado: 8, 4, 2, 1, 3, 6, 5, 10, 9, 11.

In-ordem

• **Imprima** os valores presentes nos nós da árvore ao lado, segundo a condição **ordem simétrica** (percorrer **sae**, tratar **raiz,** percorrer **sad**).

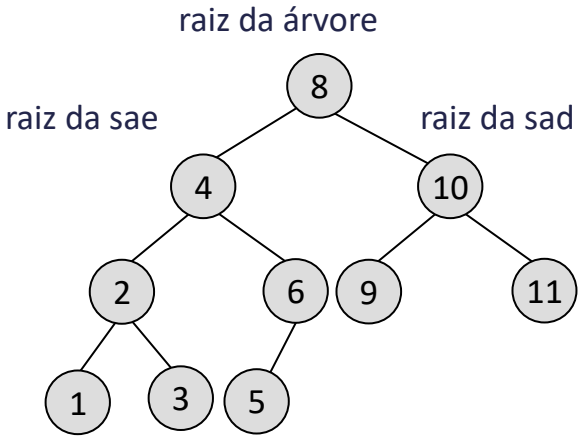


Resultado: 1, 2, 3, 4, 5, 6, 8, 9, 10, 11

Pós-ordem

Newton
Quem se prepara, não para.

- **Imprima** os valores presentes nos nós da árvore ao lado, segundo a condição **pós-ordem** (percorrer **sae**, percorrer **sad**, tratar **raiz**).



Resultado: 1, 3, 2, 5, 6, 4, 9, 11, 10, 8

Implementação do Tipo Abstrato de Dados Dicionário usando a Estrutura de Dados Árvore Binária de Pesquisa

Estrutura de dados:

- Contém as operações $inicializa$, $pesquisa$, $insere$ e $retira$.
- A operação $inicializa$ é implementada pelo construtor da classe $ArvoreBinaria$.

```
public class ArvoreBinaria {
  private static class No {
    Item reg;
    No esq, dir;
  }
  private No raiz;
```

```
  public ArvoreBinaria () {
    this.raiz = null;
  }
  public Item pesquisa (Item reg) {
    return this.pesquisa (reg, this.raiz);
  }
  public void insere (Item reg) {
    this.raiz = this.insere (reg, this.raiz);
  }
  public void retira (Item reg) {
    this.raiz = this.retira (reg, this.raiz);
  }
}
```

Método para Pesquisar na Árvore

Para encontrar um registro com uma chave reg**:**

- Compare-a com a chave que está na $raiz$.
- Se é menor, vá para a subárvore esquerda.
- Se é maior, vá para a subárvore direita.
- Repita o processo recursivamente, até que a chave procurada seja encontrada ou um nó folha é atingido.
- Se a pesquisa tiver sucesso então o registro contendo a chave passada em reg é retornado.

```
private Item pesquisa (Item reg, No p) {
  if (p == null) return null; // Registro não encontrado
  else if (reg.compara (p.reg) < 0)
         return pesquisa (reg, p.esq);
  else if (reg.compara (p.reg) > 0)
         return pesquisa (reg, p.dir);
  else return p.reg;
}
```

Procedimento para Inserir na Árvore

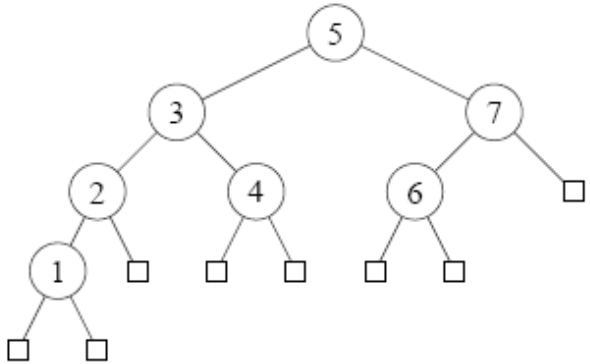
- Atingir uma referência **n**ull em um processo de pesquisa significa uma pesquisa sem sucesso.
- Caso se queira inseri-lo na árvore, a referência **n**ull atingida é justamente o ponto de inserção.

```
private No insere (Item reg, No p) {
  if (p == null) {
    p = new No (); p.reg = reg;
    p.esq = null; p.dir = null;
  }
  else if (reg.compara (p.reg) < 0)
        p.esq = insere (reg, p.esq);
  else if (reg.compara (p.reg) > 0)
        p.dir = insere (reg, p.dir);
  else System.out.println ("Erro: Registro ja existente");
  return p;
}
```

Procedimento para Retirar x da Árvore

- **Alguns comentários:**
  1. A retirada de um registro não é tão simples quanto a inserção.
  2. Se o nó que contém o registro a ser retirado possui no máximo um descendente $\Rightarrow$ a operação é simples.
  3. No caso do nó conter dois descendentes o registro a ser retirado deve ser primeiro:
     - substituído pelo registro mais à direita na subárvore esquerda;
     - ou pelo registro mais à esquerda na subárvore direita.

Exemplo da Retirada de um Registro da Árvore



Assim: para retirar o registro com chave 5 na árvore basta trocá-lo pelo registro com chave 4 ou pelo registro com chave 6, e então retirar o nó que recebeu o registro com chave 5.

Método para retirar *reg* da árvore

```
private No antecessor (No q, No r) {
  if (r.dir != null) r.dir = antecessor (q, r.dir);
  else { q.reg = r.reg; r = r.esq; }
  return r;
}
```

```
private No retira (Item reg, No p) {
  if (p == null)
        System.out.println("Erro: Registro nao encontrado");
  else if (reg.compara (p.reg) < 0)
        p.esq = retira (reg, p.esq);
  else if (reg.compara (p.reg) > 0)
        p.dir = retira (reg, p.dir);
  else {
    if (p.dir == null) p = p.esq;
    else if (p.esq == null) p = p.dir;
    else p.esq = antecessor (p, p.esq);
  }
  return p;
}
```

Análise

- O número de comparações em uma pesquisa com sucesso:

$$\begin{aligned} \text{melhor caso} &: C(n) = O(1), \\ \text{pior caso} &: C(n) = O(n), \\ \text{caso médio} &: C(n) = O(\log n). \end{aligned}$$

- O tempo de execução dos algoritmos para árvores binárias de pesquisa dependem muito do formato das árvores.

1. Para obter o pior caso basta que as chaves sejam inseridas em ordem crescente ou decrescente. Neste caso a árvore resultante é uma lista linear, cujo número médio de comparações é $(n+1)/2$.

2. Para uma **árvore de pesquisa randômica** o número esperado de comparações para recuperar um registro qualquer é cerca de $1,39 \log n$, apenas 39% pior que a árvore completamente balanceada.

Referências

- NICOLETTI, Maria do Carmo, HRUSCHKA, Estevam R. Jr.. **Fundamentos da teoria dos grafos para computação**, - 3. ed. - Rio de Janeiro : LTC, 2018.
- ZIVIANI, N. **Projeto de Algoritmos com Implementações em Java e C++** . Consultoria em Java e C++ de F.C. Botelho, Cengage Learning Brasil, ISBN 9788522108213, 2012.

http://www2.dcc.ufmg.br/livros/algoritmos-java/implementacoes-05.php
https://www.devmedia.com.br/trabalhando-com-arvores-binarias-em-java/25749