

# SIN 211 - Algoritmos e Estruturas de Dados (Árvores)

Prof<sup>o</sup>: Joelson Antônio dos Santos

Universidade Federal de Viçosa  
Instituto de Ciências Exatas e Tecnológicas  
Campus de Rio Paranaíba - MG

*joelsonn.santos@gmail.com*  
*Sala: BBT 233*

6 de junho de 2018

# Aula de Hoje

## 1 Recursividade

- Breve revisão

## 2 Árvores

- Conceitos
- Árvores Binárias (parte 1)

# Recursividade

- Divide um determinado **problema** que não pode ser resolvido diretamente, em partes menores **subproblemas** de mesma natureza que podem ser resolvidos facilmente.
- Os subproblemas também podem ser divididos em problemas menores.
- Um subproblema torna-se **trivial**, quando é possível resolvê-lo sem uma nova divisão.

# Recursividade

- Um processo recursivo consiste em duas partes:
  - O caso trivial, conhecido como **passo base**.
  - Um método geral que reduz o problema a um ou mais problemas menores (subproblemas) de mesma natureza, conhecido como **passo recursivo**.
- **Exemplo:**  $n$ -ésimo número da sequência de *Fibonacci*.

# Recursividade - Fibonacci

- A sequência de Fibonacci consiste em uma lei cujo o  $n$ -ésimo elemento é composto pela soma dos dois elementos anteriores da sequência.
  - 1, 1, 2, 3, 5, 8, 13,  $\dots$
- Uma implementação da Fibonacci deve retornar elemento da sequência função de o  $n$ -ésimo.

# Recursividade - Fibonacci


- Logo, o  $n$ -ésimo elemento é dado por:

$$fib(n) = \begin{cases} 1 & n = 1 \\ 1 & n = 2 \\ fib(n-1) \times fib(n-2) & n > 2 \end{cases} \quad (1)$$

- Como ficaria a função recursiva?

# Recursividade

- Implementação:

```
int fibonaccii(int n){  
    if((n==1) || (n==2)){  
        return 1;  
    }else{  
        return fibonaccii(n-1) + fibonaccii(n-2);  
    }  
}
```

- O que dizer sobre *fib(5)*? (**Explicação no quadro**);

- **Vantagem:**

- Implementação relativamente fácil;

- **Desvantagem:**

- Pode ser mais lenta e consumir mais recursos que utilizar funções não recursivas.



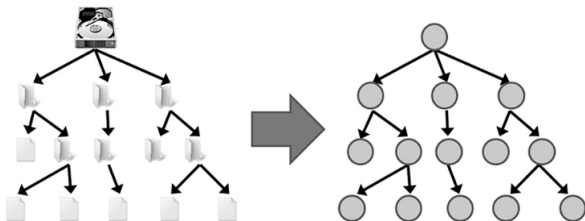
# Árvores

# Relembrando Listas

- As listas encadeadas usualmente são mais flexíveis do matrizes (vetores), mas são lineares e de difícil representação hierárquica de objetos.
  - **Ex:** Listas encadeadas são eficientes para inserção e remoção, mas ineficientes para busca.
- As operações de inserção, remoção e busca são eficientes para **listas estáticas**?

# Árvores

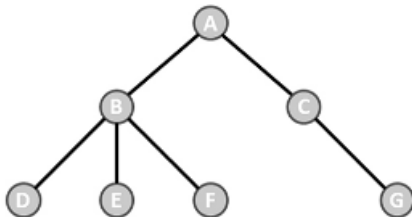
- Utilizadas para representar dados de natureza hierárquica.



- Exemplos:** Taxonomia, busca de dados armazenados no computador, diagrama hierárquico de uma organização, etc;

# Árvores

- Tipo especial de grafo **conexo** e **acíclico**.
- Possui uma representação **não linear**.
- Árvores são definidas como um conjunto de **nós** (vértices) e **arcos** (arestas);
- Diferente de árvores naturais, árvores computacionais são representadas de cima para baixo, onde a raiz fica na parte superior e as formas na parte inferior.



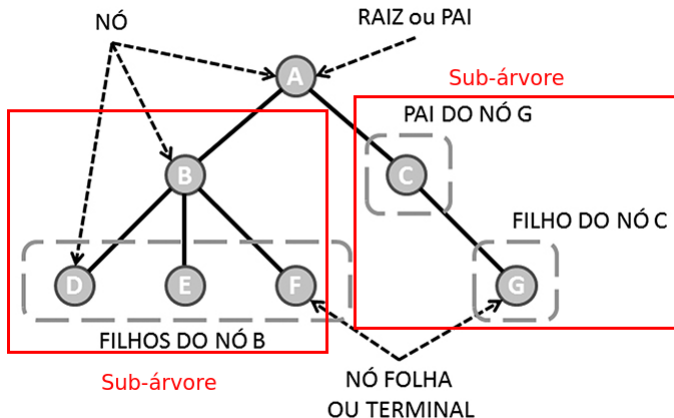
# Árvores - Notação

- Uma árvore  $A$  é uma estrutura hierárquica composta por  $n \geq 0$  nós.
  - Se  $n = 0$  dizemos que a árvore está vazia.
  - Senão:
    - Existe um nó especial denominado **raiz**.
    - Os demais nós de  $A$  são organizados em  $A_1, A_2, \dots, A_k$ , estruturas de árvores disjuntas, denominadas de sub-árvores de  $A$ .

# Árvores - Terminologia

- **Raiz:** é o nó localizado na parte mais alta da árvore, o único que não possui nó pai.
- **Pai:** também chamado ancestral, é o nó antecessor imediato de outro nó.
- **Filho:** é o nó sucessor imediato de um outro nó.
- **Nó folha:** ou nó terminal, é qualquer nó que não possui filhos.
- **Nó interno:** ou não terminal, é qualquer nó que possui ao menos um filho.
- **Caminho:** é uma sequência de nós ligados por arestas.

# Árvores - Exemplo

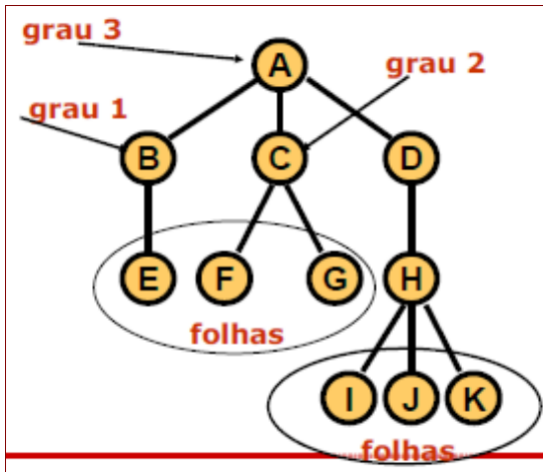


# Árvores - Terminologia

- **Grau de um nó:** é igual ao número de filhos (sub-árvores) do mesmo;
- **Folha:** nó com grau igual a zero;
- **Grau de uma árvore:** maior grau entre seus nós;

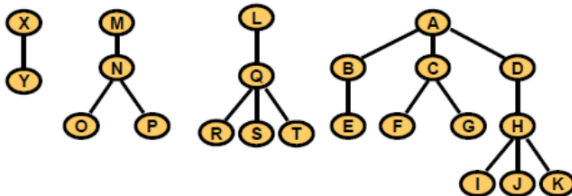


# Árvores - Exemplo



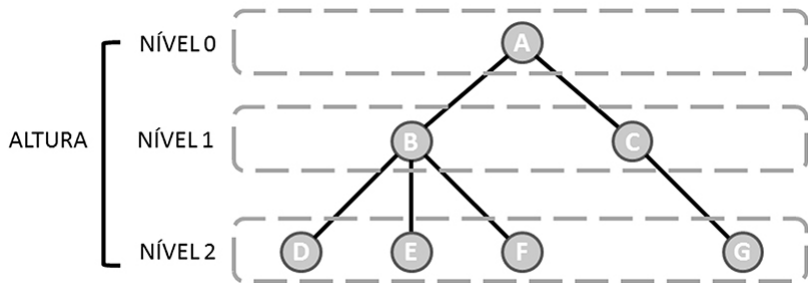
# Árvores

- **Florestas:** conjunto de zero ou mais árvores disjuntas.
- Há pouca distinção entre árvores e florestas:
  - Se excluirmos a raiz de uma árvore, teremos uma floresta.
  - Se adicionarmos um nó a uma floresta, teremos uma árvore.



# Árvores - Terminologia

- **Nível** de um nó é o número de nós entre ele e a raiz.
- **Altura ou profundidade** de uma árvore é igual ao seu **maior nível**.



- Uma raiz  $A$  possui três sub-árvores  $\{B;E\}$ ,  $\{C;F;G\}$  e  $\{D\}$ .
- A árvore  $\{C; F; G\}$  tem o nó  $C$  como raiz. O nó  $C$  está no nível 1 quando comparada à árvore toda. Os nós  $F$  e  $G$  são sub árvores de  $C$ .
- A árvore  $\{B;E\}$  possui o nó  $B$  como raiz.
- Como poderíamos representar essa árvore?

- Qual o grau e o nível de cada nó?

# Árvores

Nó	Grau	Nível	Tipo
A	3	0	raiz
B	1	1	interno
C	2	1	interno
D	0	1	folha
E	0	2	folha
F	0	2	folha
G	0	2	folha

# Árvores

- Tipos de representação de árvore.
  - Parênteses aninhados ( $A(B(D,E,F), C(G))$ )

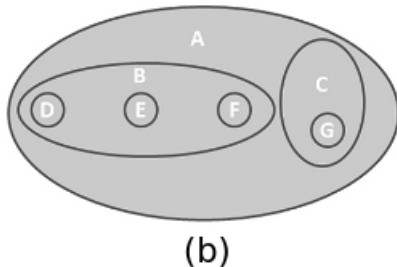
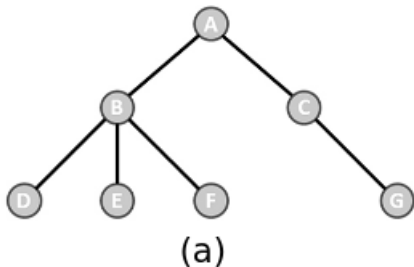


Figura: (a) grafo; (b) diagrama de Venn (ou conjuntos aninhados)

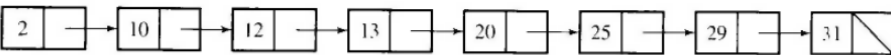
- A representação hierárquica possibilita **acelerar o processo de pesquisa** e esse é, geralmente, o principal enfoque quando trabalhamos com árvores como estruturas de dados.
- Considerando uma lista ordenada, a pesquisa tem sempre que iniciar no primeiro elemento e percorrer a lista completamente.



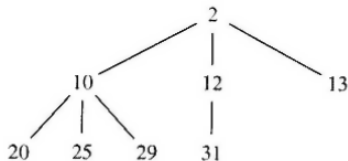
- Se os elementos são armazenados numa árvore ordenada, seguindo algum critério, o número de comparações pode ser consideravelmente reduzido mesmo para o pior caso.

# Árvores

- Quantas comparações devemos fazer para encontrarmos o elemento 31 em cada uma das estruturas:



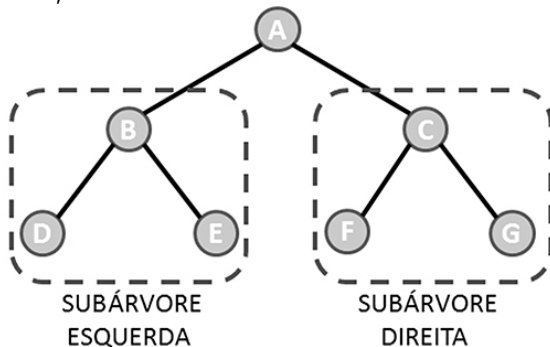
(a)



(b)

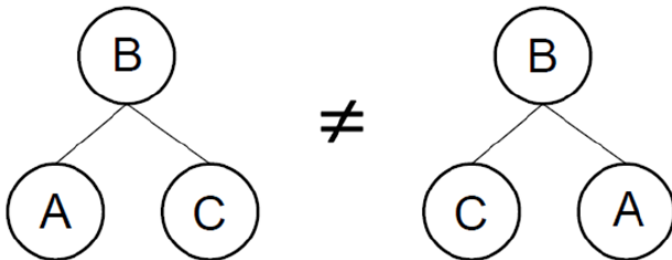
# Árvores Binárias

- Tipo especial de árvore em que cada nó possui zero, uma ou no máximo duas sub-árvores. A árvore da **direita** e a árvore da **esquerda**;
- Caso o nó não possua subárvores (filhos), este será um nó **folha**;



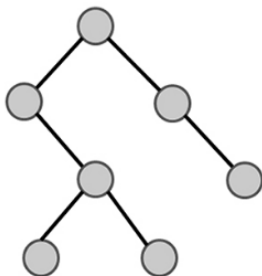
# Árvores Binárias

- Para mantermos uma árvore binária ordenada, devemos manter também cada uma de suas subárvores ordenadas.
- **Exemplo:**

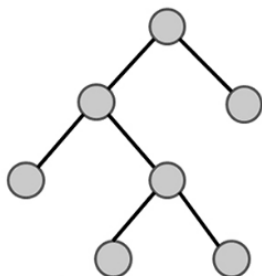


# Árvore Estritamente Binária

- É um tipo de árvore onde cada nó possui sempre nenhuma ou duas subárvores;



ÁRVORE  
BINÁRIA



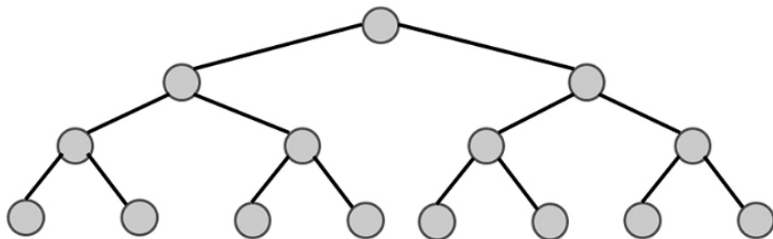
ÁRVORE ESTRITAMENTE  
BINÁRIA

# Árvore Binária Cheia

- Uma **árvore binária cheia** é uma árvore em que se um nó tem alguma sub-árvore vazia então ele está no último nível.
- É possível calcular o número de nós por nível neste tipo de árvore, assim como o número total de nós da árvore:
  - Um nível  $n$  possui exatamente  $2^n$  nós. Se um nível  $n$  possui  $m$  nós, o nível  $n + 1$  possuirá  $2n$  nós.
  - Uma árvore de altura  $H$  possui  $2^H - 1$  nós;

# Árvore Binária Cheia

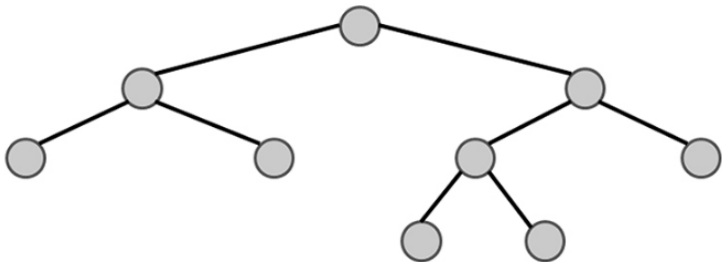
- Exemplo:



Nível (n)	0	1	2	3
Nº de nós	1	2	4	8
Potência	$2^0$	$2^1$	$2^2$	$2^3$

# Árvore Binária Completa

- **Árvore binária completa** é uma árvore em se todos os nós folhas se encontram ou no último ou no penúltimo nível da árvore.



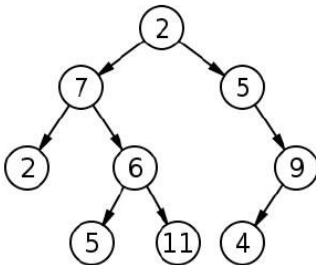


# Exercícios - Recursividade

- Dado um vetor de 3 posições preenchidos com os valores 1, 2, 3, crie uma função recursiva que imprima os valores na seguinte ordem:
  - 3, 2, 1
- Escreva uma nova função que imprima os valores na ordem 1, 2, 3.

# Exercícios - Árvores

- Uma árvore binária cheia é sempre uma árvore binária completa? Justifique.
- Dada a seguinte árvore (figura):
  - Escreva o grau, o tipo e o nível de cada nó;
  - Quantas folhas esta árvore possui?
  - Quais suas subárvores?



# Bibliografia Básica

- Estrutura de dados descomplicada em linguagem C, CAPÍTULO 11 - André Ricardo Backes, <https://www.evolution.com.br/epubreader/estrutura-de-dados-descomplicada-em-linguagem-c-1ed>
- DROZDEK, Adam. Estrutura de Dados e Algoritmos em C++. Editora Pioneira Thomson Learning, 2005. Capítulo 5.