**INSTITUTO FEDERAL DE ALAGOAS – IFAL / CAMPUS MACEIÓ**
**Curso Bacharelado em Sistemas de Informação**
**Disciplina de Estruturas de Dados**

*Prof. MSc. Ricardo Nunes*
*Ricardo (arroba) ifal.edu.br*

# Árvores

## Objetivos

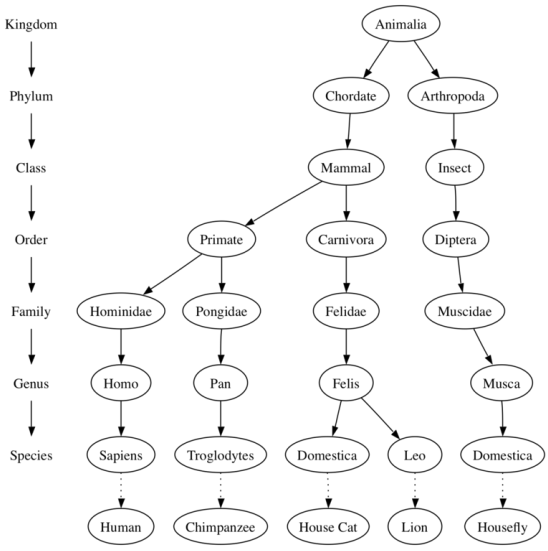
- Compreender o que é a estrutura de dados árvore e como utilizá-la.
- Entender como árvores podem ser utilizadas para implementar a estrutura de dados mapa
- Implementar árvores utilizando listas
- Implementar árvores utilizando classes e referências

### Árvores

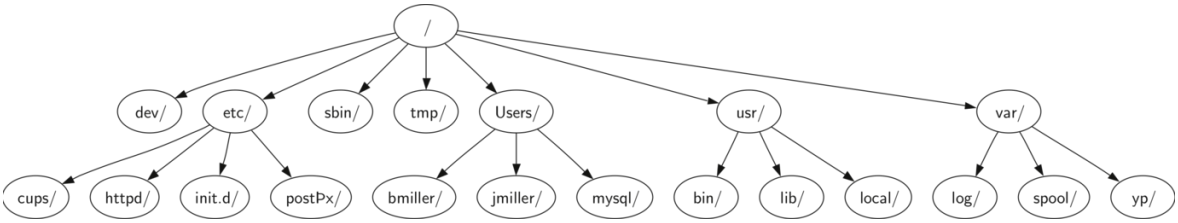
- Tipo de dado não linear
- A estrutura de dados árvore tem um *raiz (root)*, *galhos (branches)* e *folhas (leaves)*
- A raiz é representada no topo

### Exemplo Árvore e suas propriedades

- Árvores são hierárquicas: são estruturadas em camadas, do geral (raiz) ao específicos (folhas)
- Os filhos de um nó são independentes dos filhos de outro nó
- Cada nó folha é único



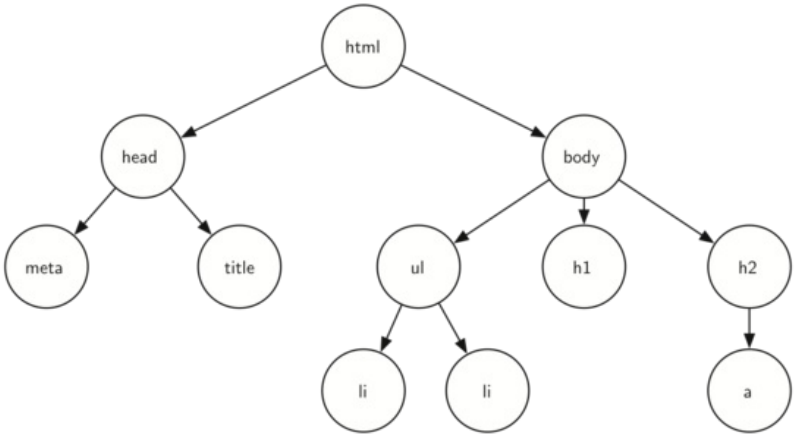

### Exemplo Árvore (2)




### Exemplo Árvore (3)

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
    <meta http-equiv="Content-Type"
          content="text/html; charset=utf-8" />
    <title>simple</title>
</head>
<body>
<h1>A simple web page</h1>
<ul>
    <li>List item one</li>
    <li>List item two</li>
</ul>
<h2><a href="http://www.cs.luther.edu">Luther CS </a><h2>
</body>
</html>
```

## Exemplo Árvore (3 - Cont.)




## Vocabulário

- Nó (Node)
  - parte fundamental de uma árvores
  - Ter um nome, chamado de **chave**
  - Opcionalmente pode ter um conteúdo **payload**
  - também chamado de vértice
- Arestas (Edge)
  - Conecta dois nós para mostrar que há uma relação entre eles
  - também chamado de arcos
- Raiz (root)
  - Único nó que não tem arestas de entrada
- Caminho (path)
  - Lista ordenada de nós conectados por suas arestas
  - Felidae → Felis → Domestica
- Filhos (childen): Conjunto de nós que têm aresta de entrada para o mesmo nó
- Pai: um nó é pai de todos os nós conectados a ele por uma aresta de saída
- Irmão: nós com o mesmo pai
- Subárvore:
- Folha: nó sem filhos
- Nível: número de arestas de uma caminho da raiz até um nó
- Altura: valor máximo do nível em uma árvore

## Definição

Uma árvore consiste de um conjunto de nós e um conjunto de arestas que conectam pares de nós.

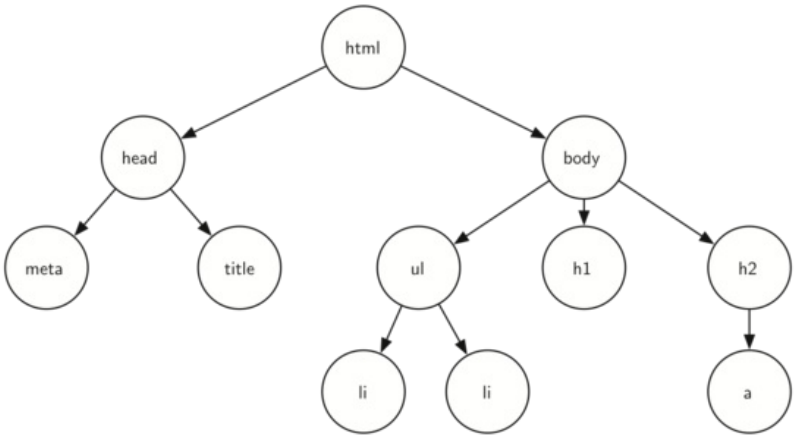
Propriedades:

- Um nó de uma árvore é designado como nó raiz
- Cada nó n, exceto o raiz, é conectado por uma aresta de exatamente um nó para outro nó p, onde p é o pai de n
- Há um caminho ligando raiz a cada nó
- Se cada nó árvore tem no máximo dois filhos, então diz-se que é uma árvore binária
- armazena elementos hierarquicamente
- cada elemento tem um pai e zero ou mais filhos
- elemento no topo é a raiz da árvore

## Representação lista de listas

Em uma lista de listas:

- o nó raiz é armazenado na primeira posição
- o segundo elemento da lista será uma lista que repesenta a subárvore esquerda
- o terceiro elemento da lista será outro lista que repsenta a subárvore direta




## Representação lista de listas (2)

```
myTree = ['a',   #root
          ['b',  #left subtree
             ['d', [], []],
              ['e', [], []] ],
           ['c',  #right subtree
              ['f', [], []],
           [] ]
         ]
```

```
myTree = ['a', ['b', ['d',[],[]], ['e',[],[]] ], ['c', ['f',[],[]], []] ]
print(myTree)
print('left subtree = ', myTree[1])
print('root = ', myTree[0])
print('right subtree = ', myTree[2])
```

## Construindo uma árvore binária

Construindo uma lista com um nó raiz, e duas sublistas vazias como filhos

```
def BinaryTree(r):
    return [r, [], []]
```

Para adicionar uma árvore á esquerda da raiz, precisa-se inserir uma nova lista na segunda posição da lista. Se a lista já tiver algo a esquerda, esse conteúdo deve ser mantido

```
def insertLeft(root,newBranch):
    t = root.pop( 1)
    if len(t) > 1:
        root.insert(1,[newBranch,t,[]])
    else:
        root.insert(1,[newBranch, [], []])
    return root
```

```
def insertRight(root,newBranch):
    t = root.pop(2)
    if len(t) > 1:
        root.insert(2,[newBranch,[],t])
    else:
        root.insert(2,[newBranch,[],[]])
    return root
```

## Construindo uma árvore binária - funções auxiliares

```
def getRootVal(root):
    return root[0]

def setRootVal(root,newVal):
    root[0] = newVal

def getLeftChild(roo):
    return roo[1]

def getRightChild(root):
    return root[2]
```

## Exemplo

```
r = BinaryTree(3)
insertLeft(r,4)
insertLeft(r,5)
insertRight(r,6)
insertRight(r,7)
l = getLeftChild(r)
print(l)

setRootVal(l,9)
print(r)
insertLeft(l,11)
print(r)
print(getRightChild(getRightChild(r)))
```

```
[5, [4, [], []], []]
[3, [9, [4, [], []], []], [7, [], [6, [], []]]]
[3, [9, [11, [4, [], []], []], []], [7, [], [6, [], []]]]
[6, [], []]
```

## Exercício

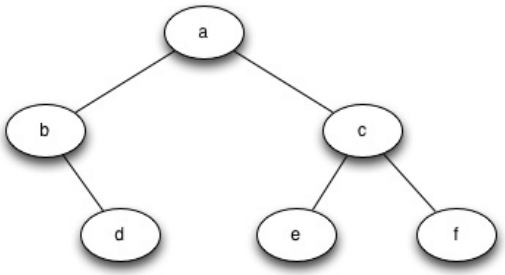
```
x = BinaryTree('a')
insertLeft(x,'b')
insertRight(x,'c')
insertRight(getRightChild(x),'d')
insertLeft(getRightChild(getRightChild(x)),'e')
```

Qual das seguintes alternativas representa a árvore para o código acima?

(A) ['a', ['b', [], []], ['c', [], ['d', [], []]]]
(B) ['a', ['c', [], ['d', ['e', [], []], []]], ['b', [], []]]
(C) ['a', ['b', [], []], ['c', [], ['d', ['e', [], []], []]]]
(D) ['a', ['b', [], ['d', ['e', [], []], []]], ['c', [], []]]

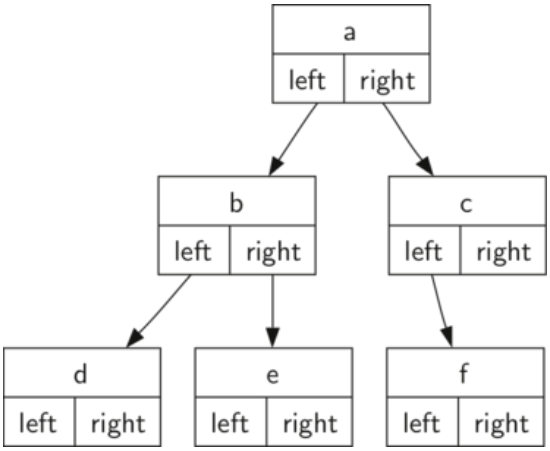
## Exercícios

Escreva uma função constroi_arvore() que retorna uma árvore utilizando lista de listas compatível com a imagem abaixo.




## Nós e Referências

Classe com os atributos: nó raiz e subárvores esquerda e direita




## class BinaryTree

```
class BinaryTree:
    def __init__(self,rootObj):
        self.key = rootObj
        self.leftChild = None
        self.rightChild = None
```

- Os atributos **leftChild** e **rightChild** serão referências para outras instâncias de **BinaryTree**.
- Ao construir uma BinaryTree algum valor é armazenado na raiz.

## def insertLeft

- Para adicionar um nó filho à esquerda na própriedade leftChild de BinaryTree, uma nova BinaryTree é criada

```
def insertLeft(self,newNode):
    if self.leftChild == None:
        self.leftChild = BinaryTree(newNode)
    else:
        t = BinaryTree(newNode)
        t.leftChild = self.leftChild
        self.leftChild = t
```

Há dois casos

1. não há filho a esquerda
2. há filho, então a nova subárvore faz esse filho descer um nível

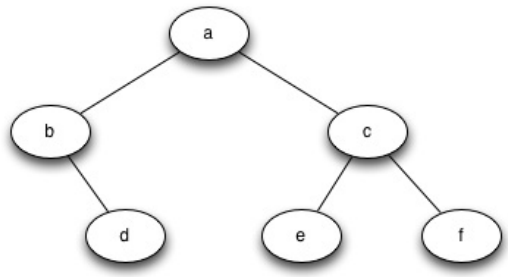
### def insertRight

```
def insertRight(self,newNode):
    if self.rightChild == None:
        self.rightChild = BinaryTree(newNode)
    else:
        t = BinaryTree(newNode)
        t.rightChild = self.rightChild
        self.rightChild = t
```

```
def getRightChild(self):
    return self.rightChild

def getLeftChild(self):
    return self.leftChild

def setRootVal(self,obj):
    self.key = obj

def getRootVal(self):
    return self.key

def __repr__(self):
    l = self.leftChild
    r = self.rightChild
    return(f'[ {self.key}, [{l}], [{r}] ] ')
```

### Exercícios

Escreva uma função constroi_arvore() que retorna uma árvore utilizando nós e referências compatível com a imagem abaixo.




## Para estudar

- Árvores
  - Seções 7.1 a 7.5 do livro [6] https://runestone.academy/ns/books/published//pythonds/Trees/toctree.html (em inglês)
  - Capítulo sobre árvores em qualquer livro sobre estruturas de dados

## Referências


1. Tradução do livro *How to Think Like a Computer Scientist: Interactive Version*, de Brad Miller e David Ranum. link: https://panda.ime.usp.br/pensepy/static/pensepy/index.html
2. Allen Downey, Jeff Elkner and Chris Meyers. *Aprenda Computação com Python 3.0*. link: https://chevitarese.files.wordpress.com/2009/09/aprendacomputaocompython3k.pdf
3. SANTOS, A. C. *Algoritmo e Estrutura de Dados I*. 2014. Disponível em http://educapes.capes.gov.br/handle/capes/176522
4. SANTOS, A. C. *Algoritmo e Estrutura de Dados II*. 2014. Disponível em 2014. Disponível em https://educapes.capes.gov.br/handle/capes/176557
5. Tradução do livro [6] *Problem Solving with Algorithms and Data Structures using Python* de Brad Miller and David Ranum.

link: https://panda.ime.usp.br/pythonds/static/pythonds_pt/index.html

6. Brad Miller and David Ranum. *Problem Solving with Algorithms and Data Structures using Python* link: https://runestone.academy/ns/books/published//pythonds/index.html
7. Caelum. Algoritmos e Estruturas Dados em Java. Disponível em https://www.caelum.com.br/download/caelum-algoritmos-estruturas-dados-java-cs14.pdf

# That's all Folks