

# Árvores: Introdução

Estruturas de Dados e Algoritmos – Ciência da Computação



Prof. Daniel Saad Nogueira  
Nunes

IFB – Instituto Federal de Brasília,  
Campus Taguatinga



# Sumário

---

## 1 Introdução



# Árvores

---

- Árvores são EDs utilizadas para resolver muitos problemas.
- Podemos ter vários tipos diferentes de árvore.
- São de natureza recursiva e hierárquica.



# Árvores

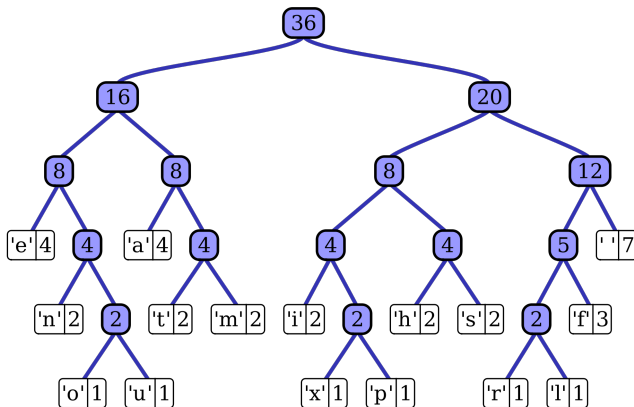
---

## Árvores de Huffman

- Utilizadas em compressão de dados.
- Organizam os símbolos mais frequentes próximo da raiz.
- Códigos menores para os símbolos mais frequentes.



# Árvores de Huffman





# Árvores

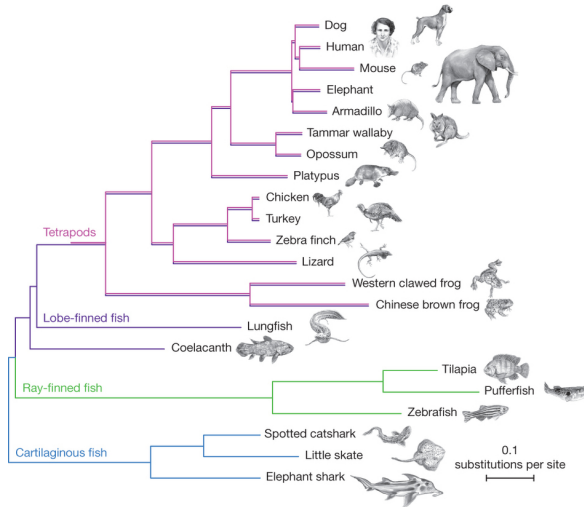
---

## Árvores Filogenéticas

- Utilizadas em Biologia Computacional.
- Estimam a distância evolutiva de organismos.



# Árvores Filogenéticas





# Árvores

---

## Árvores de Segmentos


- Utilizadas em problemas diversos.
- Armazenam alguma propriedade sobre um dado intervalo  $[l, r]$ .





# Árvores de Segmentos

---



`figuras/segment-tree.png`



# Árvores

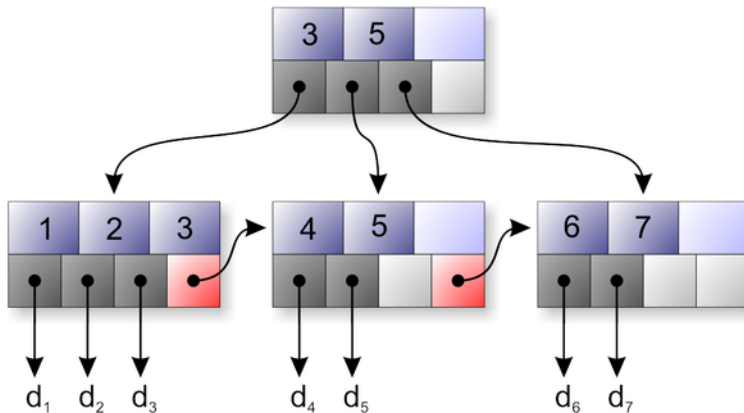
---

## Árvores B e B+

- Utilizadas em Sistemas de Arquivos.
- Convertem endereço de bloco de arquivo para endereço de disco.



# Árvores B e B+





# Árvores de Sufixos

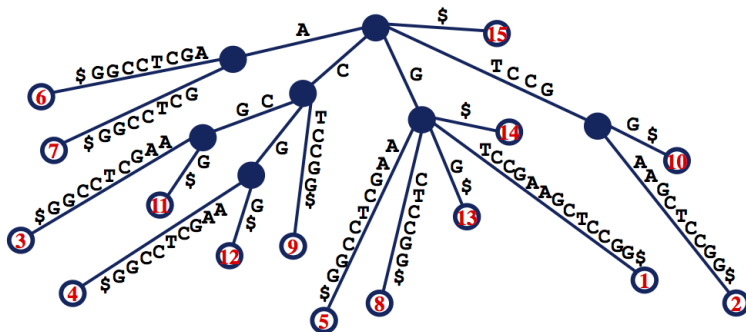
---

## Árvores de Sufixos

- Utilizadas em processamento de palavras.
- Codificam todos os sufixos de um texto de maneira compacta.
- Resolvem vários problemas sobre strings.



# Árvores de Sufixos





# Árvores

---

- Podemos citar várias outras:
  - ▶  $K^2$ -Tree: representação compacta de relações binárias.
  - ▶ Árvore-Rubro-Negra: árvore ordenada.
  - ▶ Fenwick Tree: calcula eficientemente soma de prefixos.
  - ▶ Tries: utilizadas em casamento de múltiplos padrões.
- Já deu para entender a infinidade de problemas que podemos resolver com árvores, certo?



# Árvores

---

- No nosso curso, estudaremos uma das famílias mais básicas de árvores.
- As árvores binárias.
- Estamos trabalhando só com a ponta do *iceberg*.
- No entanto, servirá de alicerce ao estudar estruturas mais complexas posteriormente.



# Sumário

---

## 2 Árvores Binárias





# Sumário

---

## 2 Árvores Binárias

- Terminologia
- Estrutura



# Terminologia

---

- Antes de elaborar qualquer algoritmo sobre árvore binárias, precisamos entender a sua representação.



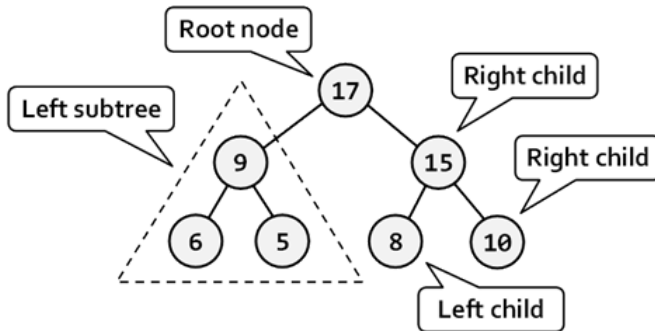
# Terminologia

---

- **Raiz:** corresponde ao topo de uma árvore.
- **Pai:** nó que precede imediatamente um segundo em um caminho partindo da raiz.
- **Filho:** Nó que ocorre imediatamente após o outro em um caminho partindo da raiz.
- **Filho da esquerda:** se  $x$  é pai de  $y$  e  $y$  ocorre imediatamente após  $x$  ao seguir para esquerda, então  $y$  é o filho da esquerda de  $x$ .
- **Filho da direita:** se  $x$  é pai de  $y$  e  $y$  ocorre imediatamente após  $x$  ao seguir para direita, então  $y$  é o filho da direita de  $x$ .
- **Folha:** nó que não possui nenhum descendente.



# Terminologia





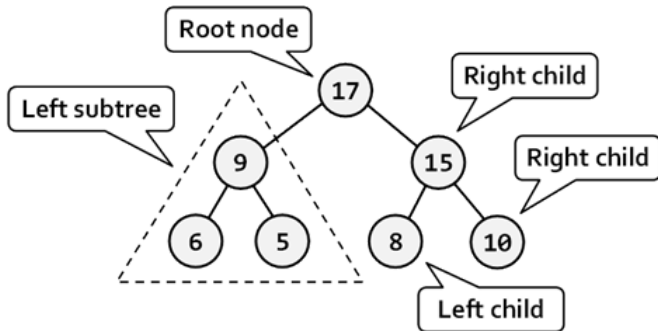
# Terminologia

---

- **Altura:** a altura do nó  $x$  corresponde a maior distância de  $x$  a uma folha.
- **Grau:** quantidade de descendentes de um nó.
- **Nível:** conjunto de nós que estão na mesma altura em relação a raiz.



# Terminologia





# Árvore Binária

---

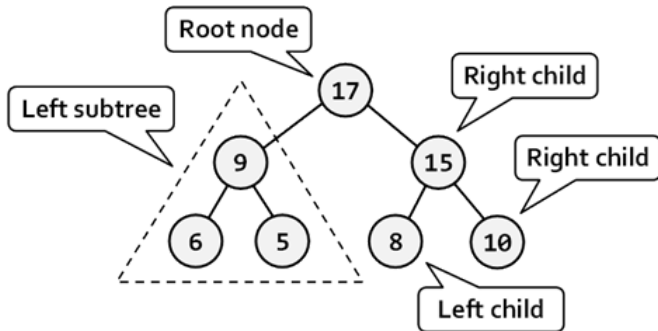
## Definição (Árvore Binária)

Uma árvore binária é composta de:

- Um potencial nó denominado de raiz.
- Caso a raiz exista:
  - ▶ Uma subárvore da esquerda.
  - ▶ Uma subárvore da direita.



# Terminologia







# Árvores Binárias

---

- Repare que a definição de uma árvore binária atua sobre ela própria.
- É uma definição recursiva!
- É natural que os algoritmos que atuem em árvores também sejam recursivos.

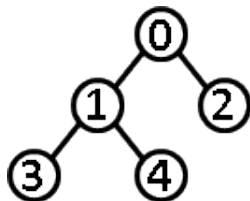


- Uma árvore binária pode ser:
  - ▶ Completa ou incompleta.
    - Uma árvore binária é completa quando todos os níveis dela estão preenchidos exceto pelo último, no qual os nós devem se encontrar mais a esquerda possível.
    - Pode ser representada através de um simples vetor.
  - ▶ Cheia ou não cheia.
    - Uma árvore binária é cheia se todo nó possui grau 0 ou 2.
  - ▶ Perfeita ou imperfeita.
    - Uma árvore binária é perfeita quando é cheia e todas as folhas possuem o mesmo nível.

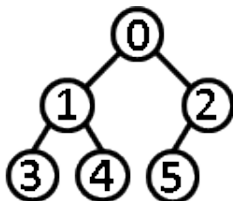


## Terminologia

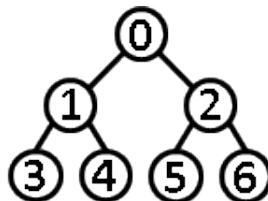
---



**full  
binary tree**



**complete  
binary tree**

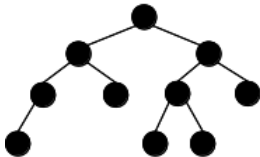


**perfect  
binary tree**

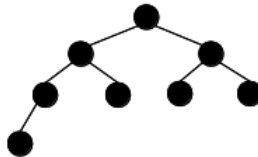


# Terminologia

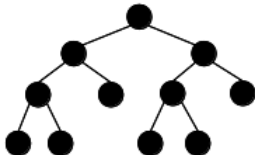
Neither complete nor full



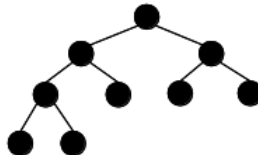
Complete but not full



Full but not complete



Complete and full





# Sumário

---

## 2 Árvores Binárias

- Terminologia
- Estrutura



# Estrutura

---

- Como visto, uma árvore binária possui, uma subárvore da esquerda e da direita.
- Podemos utilizar uma definição recursiva para representar essa estrutura computacionalmente.
- Como C não suporta tipos recursivos, emulamos essa característica através de ponteiros.



# Estrutura

---

```
typedef struct tree_node{
    void* data; /* Dado da árvore */
    struct tree_node* left; /* Ponteiro para subárvore da esquerda */
    struct tree_node* right; /* Ponteiro para subárvore da direita */
}tree_node;

typedef struct arvore{
    tree_node* root; /* Raiz da Arvore */
}arvore;
```



# Exemplos

---

Figura: É uma árvore binária?





## Exemplos

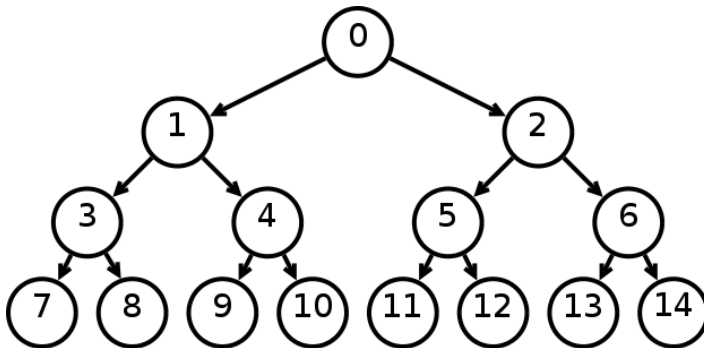
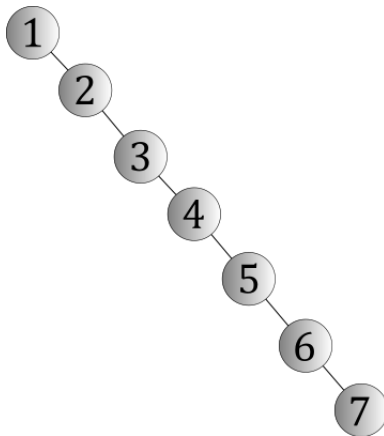


Figura: É uma árvore binária?



## Exemplos

---



**Figura:** É uma árvore binária?



# Exemplos

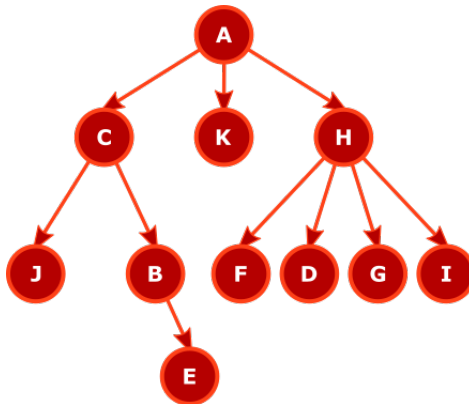


Figura: É uma árvore binária?



# Árvores Binárias

---

- Árvores representam dados de maneira hierárquica.
- Se a árvore não tiver uma certa forma, sua utilidade pode ser questionada.



## Exemplos

---

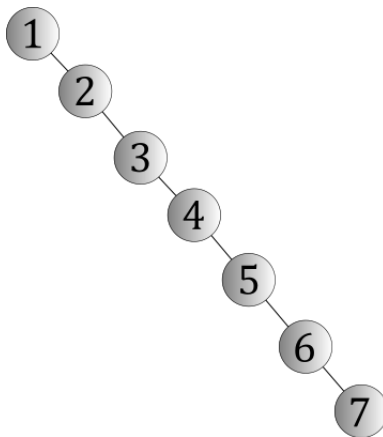


Figura: Árvore ou Lista?



# Árvores Binárias

---

- Agora que conhecemos a terminologia e definições, podemos nos concentrar em aprender algo novo e útil.



# Sumário

---

## 3 Percurso em Árvores



# Percurso em Árvores

---

- Um dos problemas fundamentais sobre esta estrutura de dados é percorrê-la.
- Qual estratégia adotar?
- Busca em largura?
- Busca em profundidade?
  - ▶ Pré-ordem?
  - ▶ Em-ordem?
  - ▶ Pós ordem?





# Percurso em Árvores

---

- Examinaremos agora cada uma destas abordagens.



# Sumário

---

- 3 Percurso em Árvores
  - Busca em Largura
  - Busca em profundidade



# Percurso em Árvores

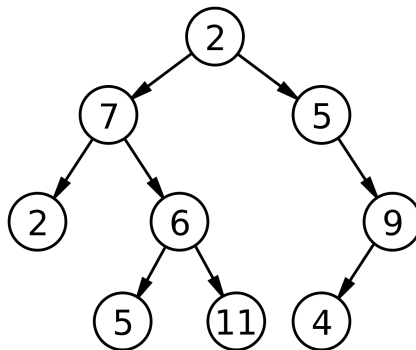
---

- Busca em largura ou amplitude.
  - ▶ Breath-first-search (BFS).
- Parte de um nó específico.
- Visita os vizinhos deste nó.
- Visita os vizinhos dos vizinhos do nó inicial.
- Visita os vizinhos dos vizinhos dos vizinhos do nó inicial.
- ...



# Percurso em Árvores

## Exemplo



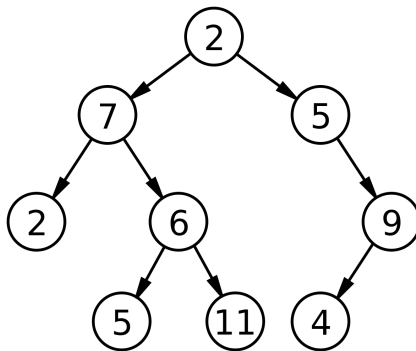
Qual a ordem dos nós a serem visitados ao partir do nó raiz?



# Percurso em Árvores

---

## Exemplo



2, 7, 5, 2, 6, 9, 5, 11, 4



# Busca em largura

---

- Como implementar uma busca em largura?



# Busca em largura

---

- Colocamos o nó inicial em uma **fila**.
- Enquanto a fila não for vazia:
  - ▶ Processo o nó que está na frente da fila.
  - ▶ Insira todos os seus vizinhos no fim da fila.
  - ▶ Retire o nó da fila.



# Busca em Largura

```
void bfs(tree_node* node){
    queue_t* queue;
    queue_initialize(&queue, construtor_no, destrutor_no);
    if(node != NULL){
        queue_push(queue, node);
    }
    while(!queue_empty(queue)){
        tree_node* no = queue_front(queue);
        processa(no);
        if(no->left != NULL){
            queue_push(queue, no->left);
        }
        if(no->right != NULL){
            queue_push(queue, no->right);
        }
        queue_pop(queue);
    }
    queue_delete(&queue);
}
```





# Sumário

---

- 3 Percurso em Árvores
  - Busca em Largura
  - Busca em profundidade



# Busca em profundidade

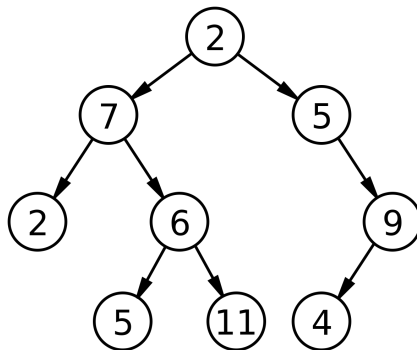
---

- A busca em profundidade (depth-first-search), difere da busca em largura pois ela busca em um ramo inteiro da árvore antes de olhar para o outro ramo.
- Basicamente um nó é visitado e a busca procede recursivamente para o vizinho imediato.



# Percurso em Árvores

## Exemplo

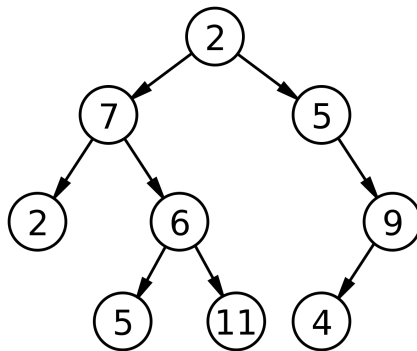


Qual a ordem dos nós a serem visitados ao partir do nó raiz?



# Percurso em Árvores

## Exemplo



2, 7, 2, 6, 5, 11, 5, 9, 4



# Busca em profundidade

---

- Como implementar uma busca em profundidade?
- Substituindo a fila da busca em largura por uma pilha.



# Busca em Profundidade

---



# Busca em profundidade

---

- Implementação recursiva é mais simples, mais elegante e até mais rápida.
- Usamos uma pilha implícita quando chamamos a função recursivamente.



# Busca em Profundidade

---

```
void dfs(tree_node* node){  
    if(node!=NULL){  
        processa(node);  
        dfs(node->left);  
        dfs(node->right);  
    }  
}
```





# Busca em profundidade

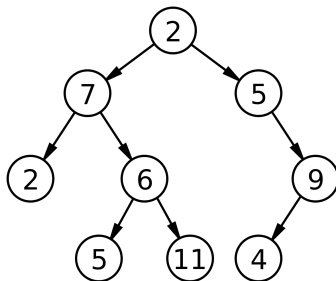
---

- A busca em profundidade possui algumas variações.
- Pré-ordem: visitamos o nó antes de proceder recursivamente aos vizinhos.
- Em-ordem: procedemos recursivamente à esquerda, visitamos o nó, e procedemos recursivamente à direita.
- Pós-ordem: procedemos recursivamente à esquerda, procedemos recursivamente à direita, visitamos o nó.



# Percurso em Árvores

## Exemplo

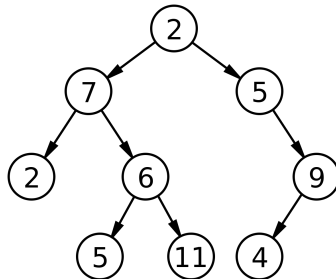


Qual a ordem dos nós a serem visitados ao partir do nó raiz em busca profundidade em pré-ordem?



# Percurso em Árvores

## Exemplo

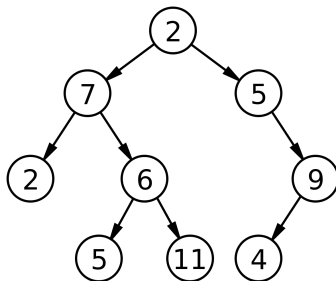


2, 7, 2, 6, 5, 11, 5, 9, 4



# Percurso em Árvores

## Exemplo

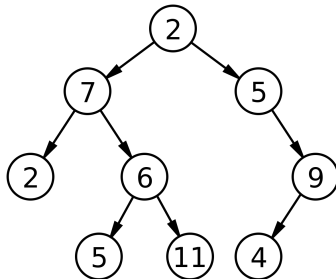


Qual a ordem dos nós a serem visitados ao partir do nó raiz em busca profundidade em-ordem?



# Percurso em Árvores

## Exemplo



2, 7, 5, 6, 11, 2, 5, 4, 9



# Busca em Profundidade

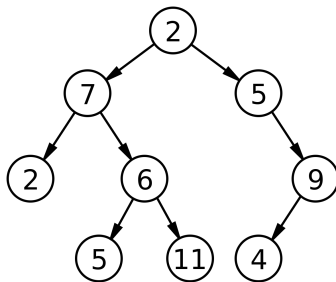
---

```
void dfs(tree_node* node){  
    if(node!=NULL){  
        dfs(node->left);  
        processa(node);  
        dfs(node->right);  
    }  
}
```



# Percurso em Árvores

## Exemplo

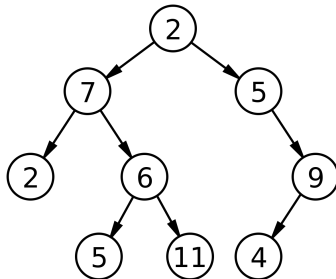


Qual a ordem dos nós a serem visitados ao partir do nó raiz em busca profundidade em pós-ordem?



# Percurso em Árvores

## Exemplo



2, 5, 11, 6, 7, 4, 9, 5, 2





# Busca em Profundidade

---