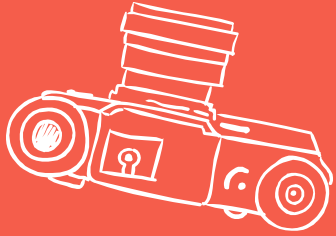




# Algoritmos e Estrutura de Dados Avançado

---

## Árvore Binária de Busca



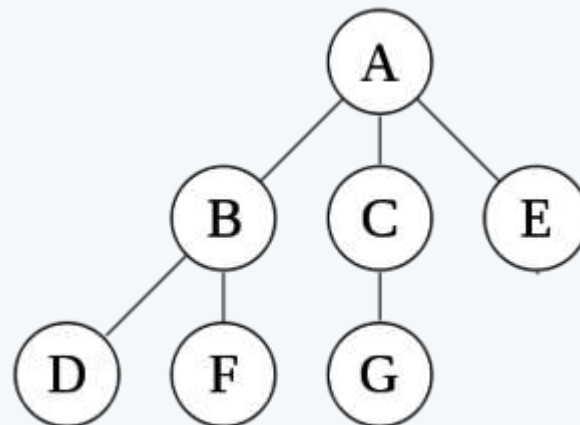
# 1. Árvores



# Árvore

Uma árvore é uma estrutura de dados que contém um conjunto de nós e arestas, interligando estes nós, onde:

- Nós (ou vértices) – são os elementos representados na árvore
- Arestas – são as conexões entre os nós
- Um dos nós é especialmente designado como o nó raiz



Ativar o Windows  
Acesse as configurações do  
ativar o Windows.



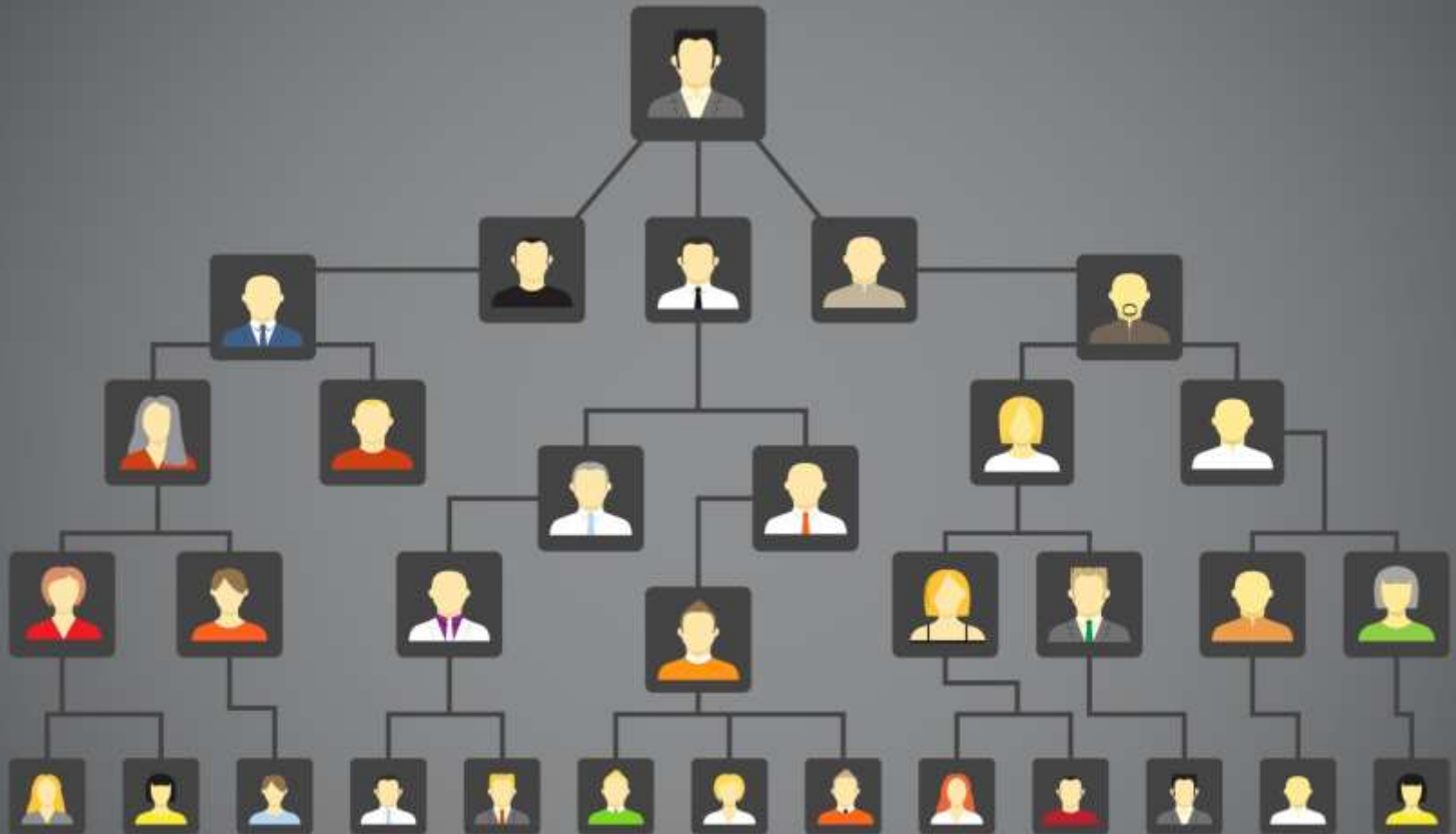
# Aplicações

Árvores são especialmente adequadas para representar estruturas hierárquicas:

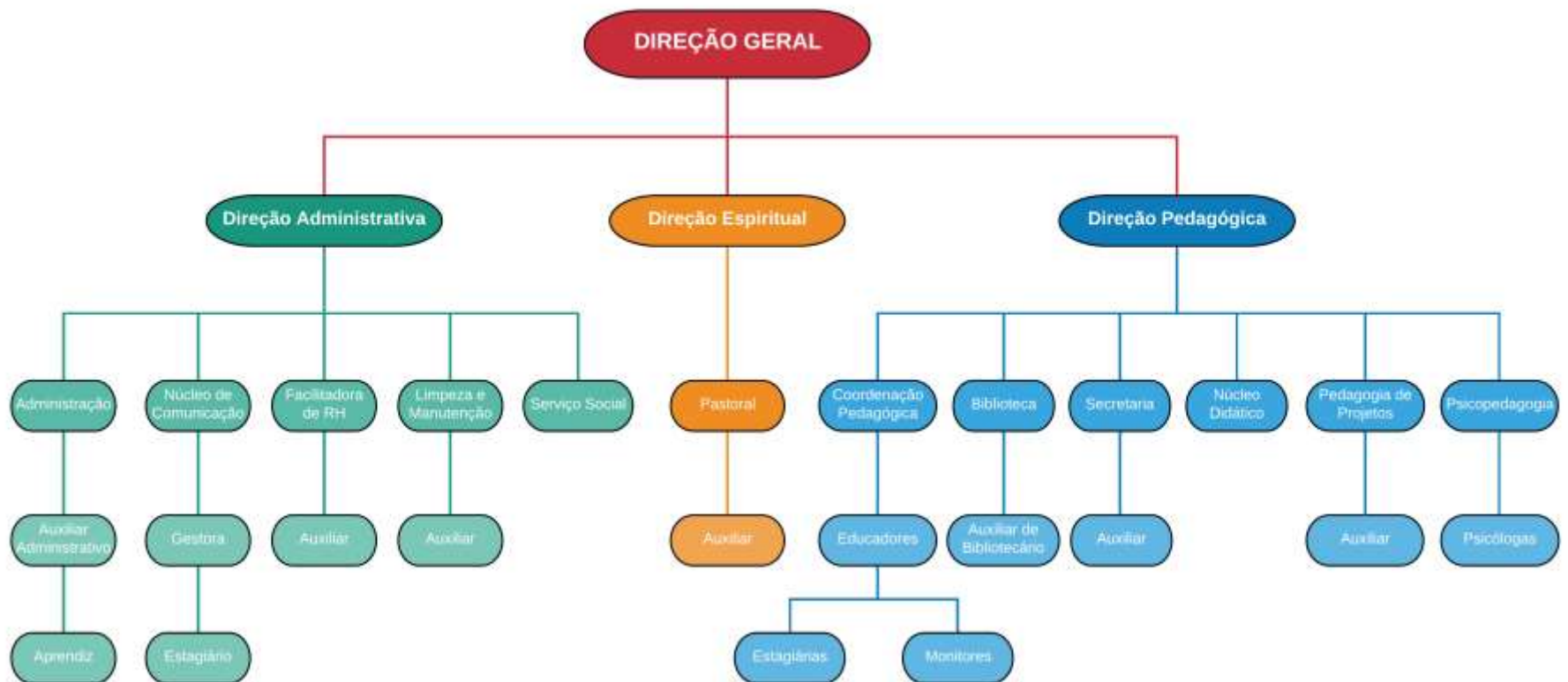
- Relações de descendência (pai, filho, etc)
- Diagrama hierárquico de uma organização
- Campeonatos de modalidade desportiva
- Taxonomia (classificação das espécies)

Ativar o Windows  
Acesse as configurações do  
ativar o Windows.

# Árvore genealógica




















# Organograma





# Tabela da copa

| OITAVAS |   |   | QUARTAS |   |       | SEMI |      |   | FINAL |   |  |
|---------|---|---|---------|---|-------|------|------|---|-------|---|--|
| 1A      |    | 2 | URU.    | 0 | FRA.  | 1    | FRA. | 1 | FRA.  | 4 |  |
| 2B      |    | 1 |         |   |       |      |      |   |       |   |  |
| 1C      |    | 4 | FRA.    | 2 | BEL.  | 0    | BEL. | 0 | BEL.  | 0 |  |
| 2D      |    | 3 |         |   |       |      |      |   |       |   |  |
| 1E      |    | 2 | BRA.    | 1 | BEL.  | 2    | BEL. | 2 | BEL.  | 2 |  |
| 2F      |    | 0 |         |   |       |      |      |   |       |   |  |
| 1G      |    | 3 | BEL.    | 2 | ING.  | 0    | ING. | 0 | ING.  | 0 |  |
| 2H      |    | 2 |         |   |       |      |      |   |       |   |  |
| 1B      |   | 1 | RUS.    | 2 | CRO.* | 2    | CRO. | 2 | CRO.  | 2 |  |
| 2A      |  | 1 |         |   |       |      |      |   |       |   |  |
| 1D      |  | 1 | CRO.*   | 2 | SUE.  | 0    | CRO. | 2 | CRO.  | 2 |  |
| 2C      |  | 1 |         |   |       |      |      |   |       |   |  |
| 1F      |  | 1 | SUE.    | 0 | ING.  | 2    | ING. | 1 | ING.  | 1 |  |
| 2E      |  | 0 |         |   |       |      |      |   |       |   |  |
| 1H      |  | 1 | COL.    | 1 | ING.* | 1    | ING. | 1 | ING.  | 1 |  |
| 2G      |  | 1 |         |   |       |      |      |   |       |   |  |



#FOXNARUSSIA

TODOS OS JOGOS AO VIVO

VENCEU NOS PÊNALTIS\*



#FOXNARUSSIA

TODOS OS JOGOS AO VIVO

VENCEU NOS PÊNALTIS\*

Ativar o Windows  
Acesse as configurações do  
ativar o Windows.

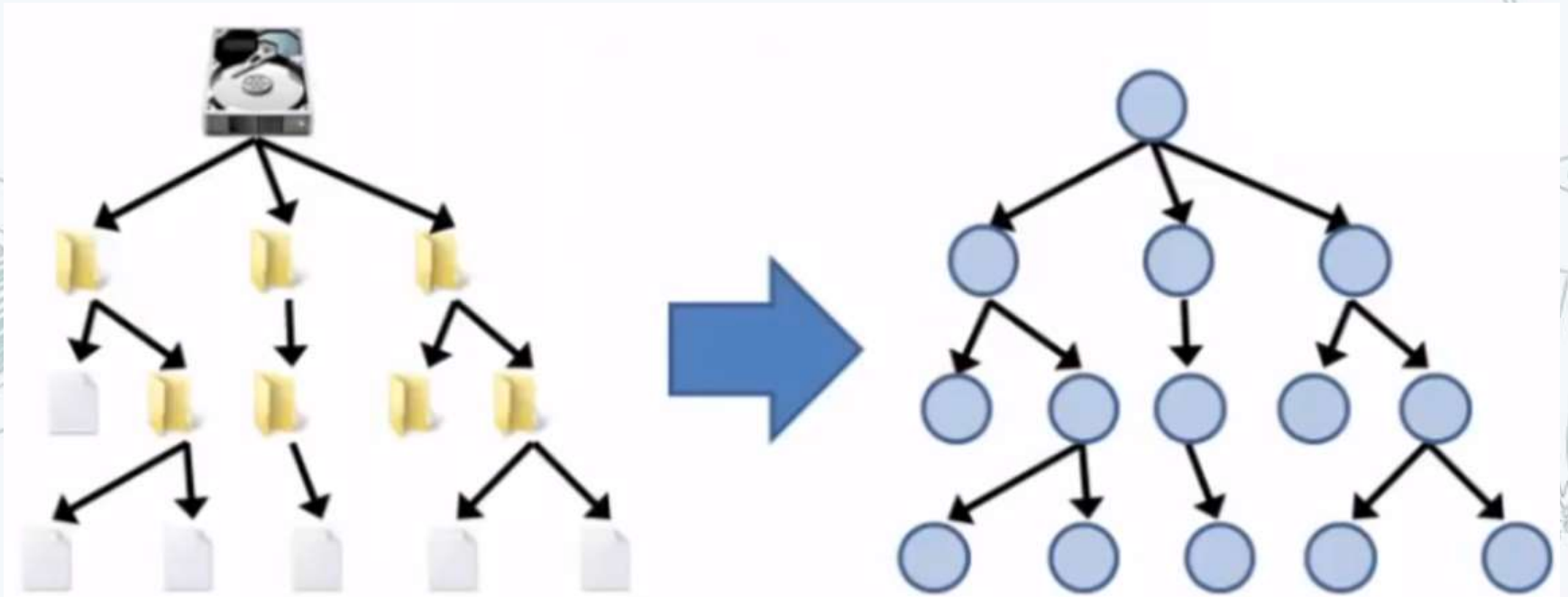
# Classificação biológica





# Em computação

- Estrutura de diretórios (pastas e arquivos)
- Busca de dados
- Representação de espaço de soluções (jogo de xadrez)



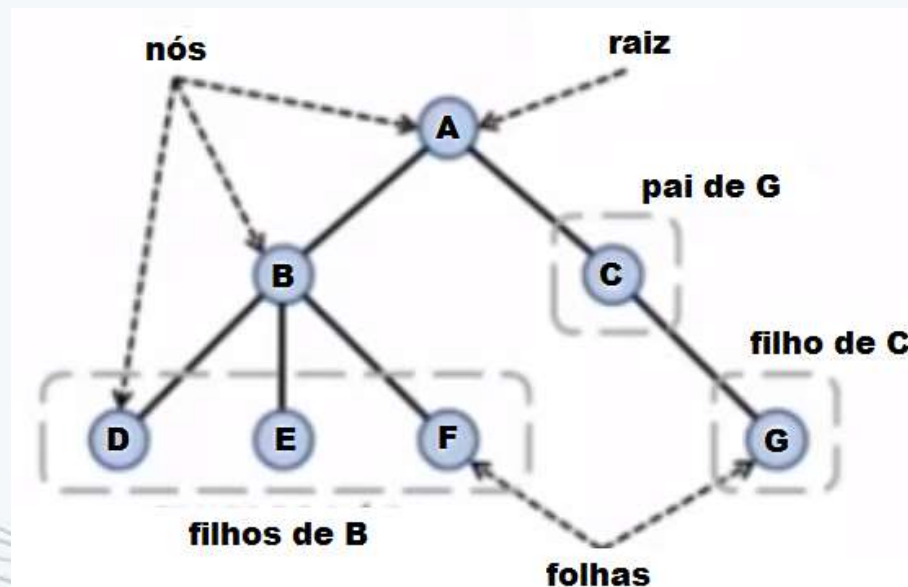


2.

# Propriedades

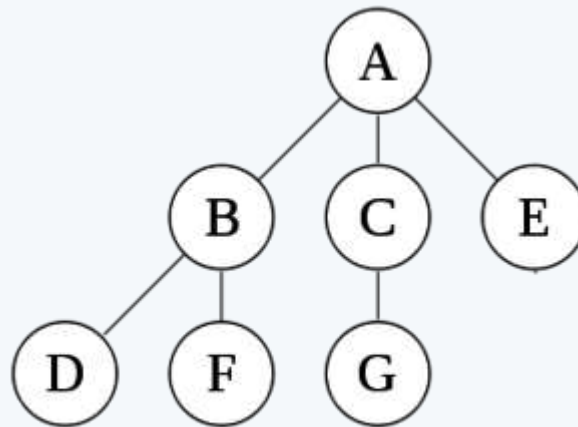
# Propriedades de uma árvore

- Pai – é o nó antecessor imediato
- Filho – é um nó sucessor imediato
- Raiz – é o nó que não possui pai
- Nó interno – nó que possui pelo menos um filho
- Folha (nó terminal) – qualquer nó que não possui filho



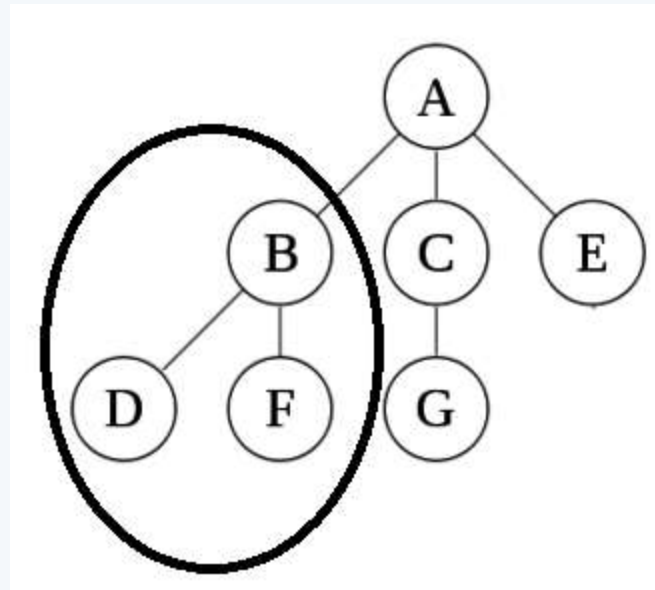
# Caminho em uma árvore

- Em uma árvore existe sempre uma aresta ligando o vértice anterior com o seguinte
- Através dos vértices, existe exatamente um caminho entre a raiz e cada um dos nós da árvore



# Sub-árvore

- Dado um nó interno, cada filho seu é a raiz de uma nova sub-árvore
- Qualquer nó interno é a raiz de uma sub-árvore consistindo dele e dos nós abaixo dele

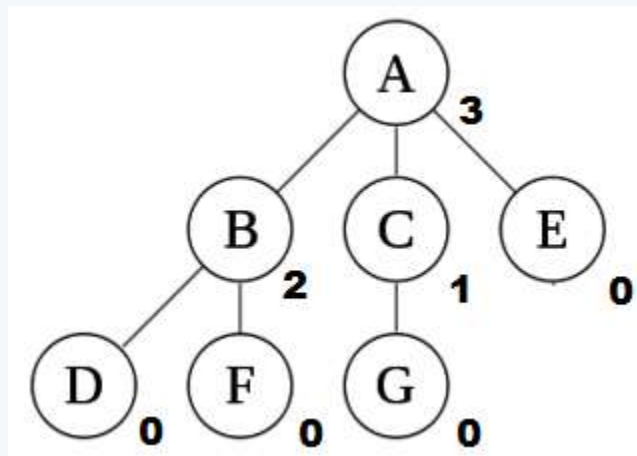


Ativar o Windows  
Acesse as configurações do  
ativar o Windows.



# Grau de um nó

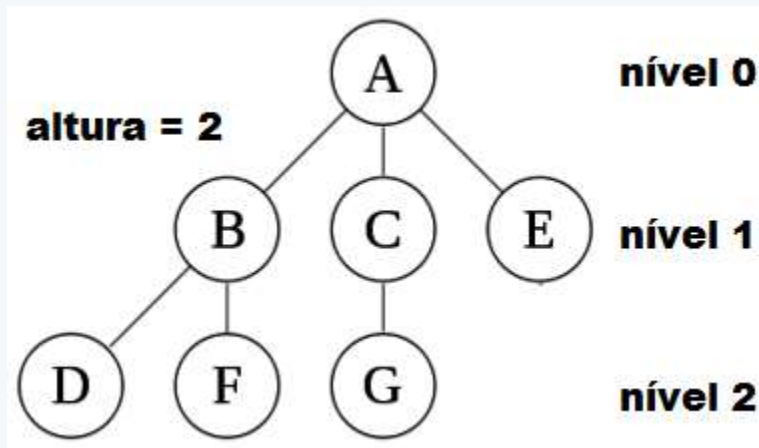
- É o número de sub-árvores (ou filhos) de um determinado nó



Ativar o Windows  
Acesse as configurações do  
ativar o Windows.

# Altura e nível

- Altura ou profundidade é o comprimento do caminho mais longo da raiz até uma das suas folhas
- Nível é o número de nós no caminho entre um elemento e a raiz



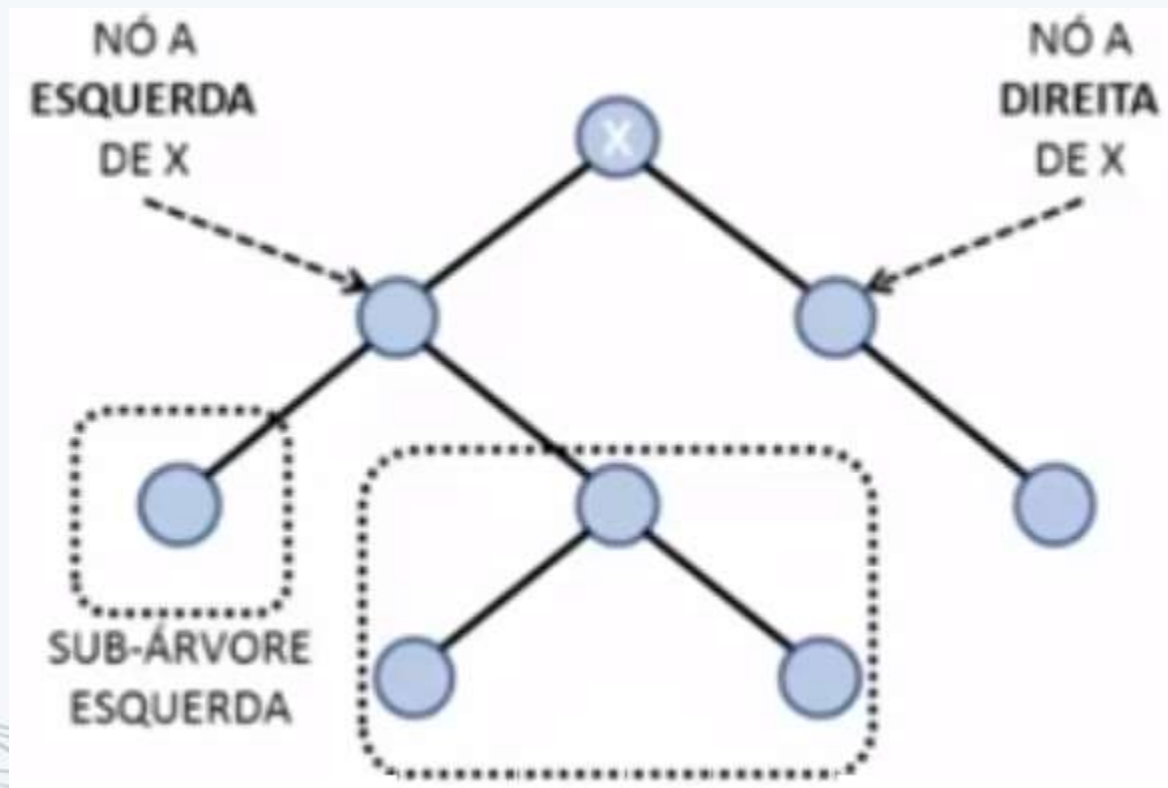
3.

# Árvore binária de busca



# Árvore binária

- É um tipo especial de árvore onde cada nó pode possuir até dois filhos
- O grau de cada nó (número de filhos) pode ser 0, 1 ou 2



Ativar o Windows  
Acesse as configurações do  
ativar o Windows.

# Árvore binária de busca

- É um tipo de árvore binária onde cada nó possui um valor (chave) associado a ele, e esse valor determina a posição do nó na árvore
- Nesta árvore não pode haver valores repetidos

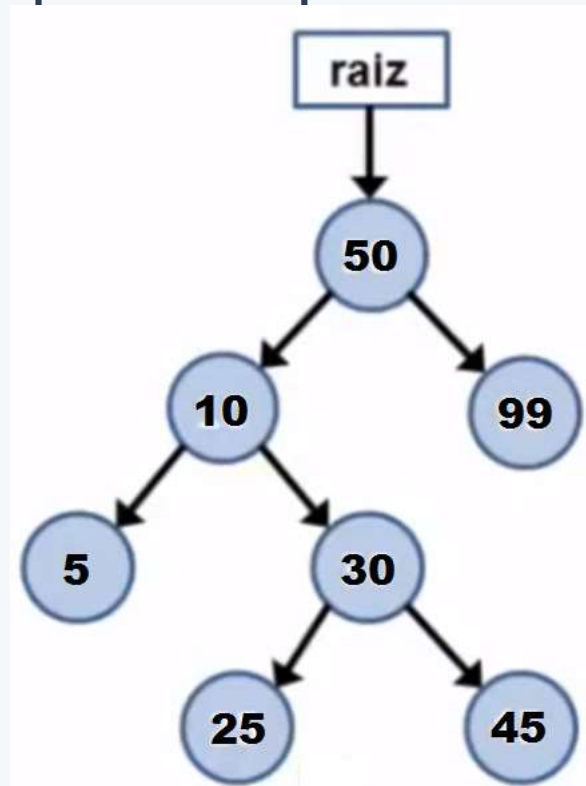
Ativar o Windows  
Acesse as configurações do Windows para ativar o Windows.



# Posicionamento dos valores

Para cada nó pai:

- Todos os valores da sub-árvore esquerda são menores do que o nó pai
- Todos os valores da sub-árvore direita são maiores do que o nó pai.



Ativar o Windows  
Acesse as configurações do  
ativar o Windows.



# Ajustes

- A inserção e a remoção de nós da árvore deve ser realizada respeitando suas propriedades

Ativar o Windows  
Acesse as configurações do  
ativar o Windows.

# 4. Implementação





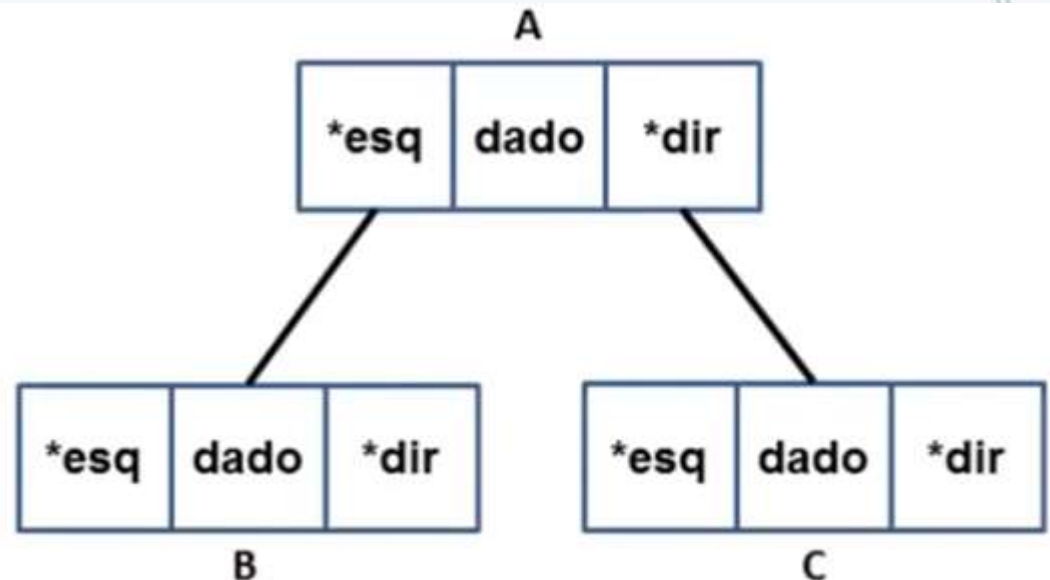
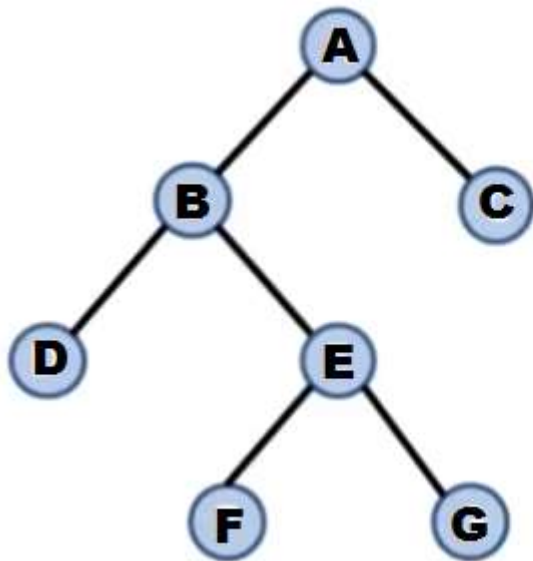
# Operações

- Criação da árvore
- Inserção de um elemento
- Remoção de um elemento
- Acesso a um elemento
- Destruição da árvore

Ativar o Windows  
Acesse as configurações do Windows para  
ativar o Windows.

# Alocação dinâmica (lista encadeada)

- Cada nó da árvore é tratado como uma estrutura alocada dinamicamente à medida em que os dados são inseridos.
- Esta estrutura deve conter os dados específicos mais dois ponteiros que apontam para o filho da esquerda e para o filho da direita





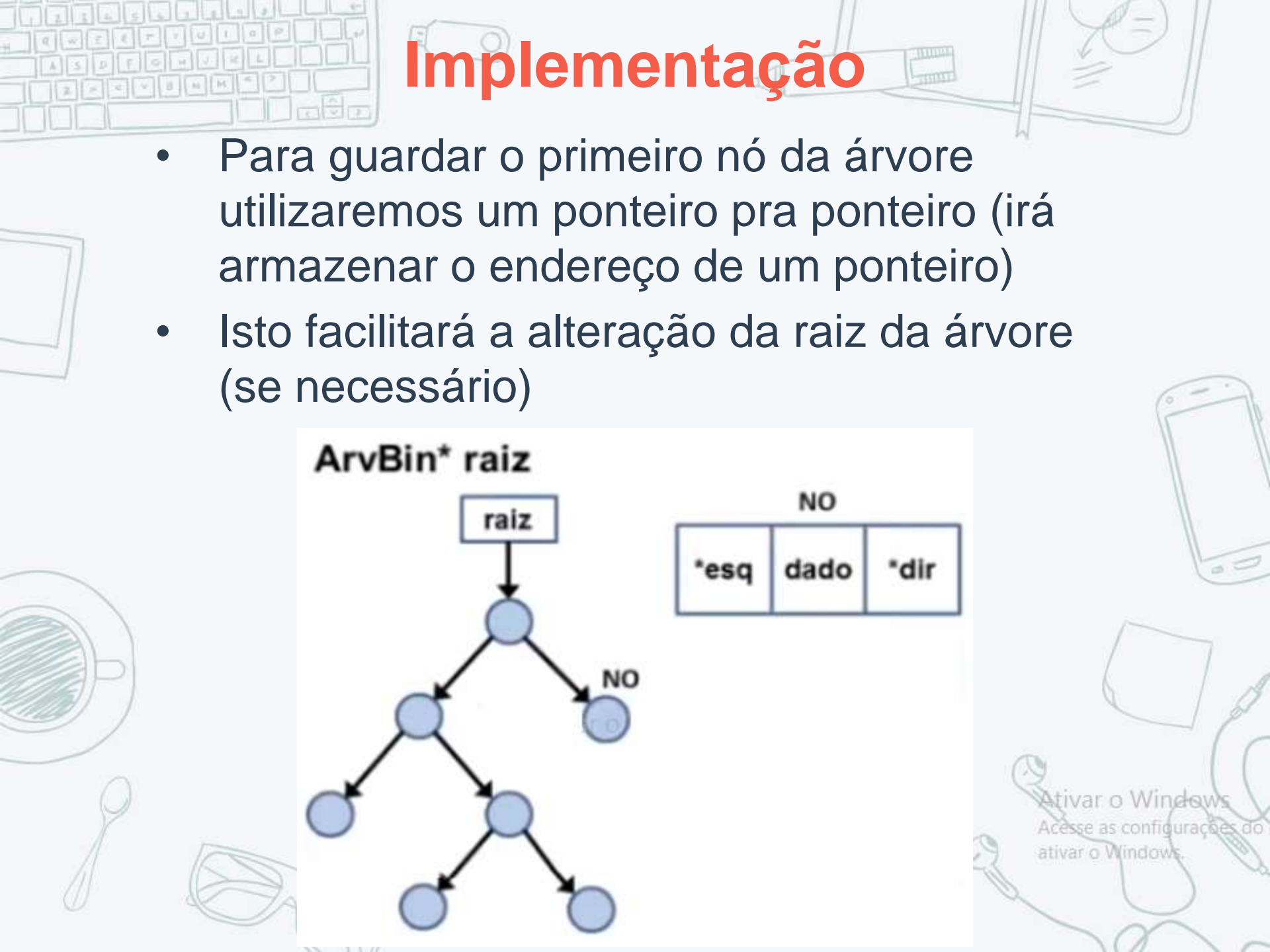
# Implementação

- Para guardar o primeiro nó da árvore utilizaremos um ponteiro pra ponteiro (irá armazenar o endereço de um ponteiro)
- Isto facilitará a alteração da raiz da árvore (se necessário)

The diagram illustrates the implementation of a binary tree using pointers. It shows a root pointer **ArvBin\* raiz** pointing to a node. A node structure is shown as a table with columns **\*esq**, **dados**, and **\*dir**. The tree structure has a root node with a left child and a right child labeled **NO**. The left child has its own left and right children.

```
graph TD; raiz[raiz] --> N1(( )); N1 --> N2(( )); N1 --> NO1((NO)); N2 --> N3(( )); N2 --> N4(( )); N4 --> N5(( )); N4 --> N6(( ));
```

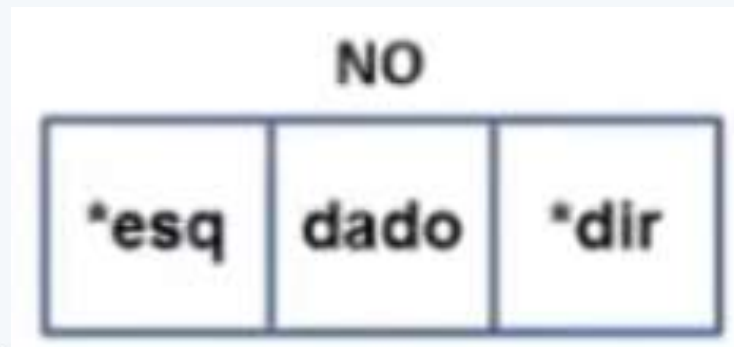
- # Implementação
- Para guardar o primeiro nó da árvore utilizaremos um ponteiro pra ponteiro (irá armazenar o endereço de um ponteiro)
  - Isto facilitará a alteração da raiz da árvore (se necessário)
- 
- O diagrama ilustra a implementação de uma árvore binária de busca (ArvBin\*). No topo, há um ponteiro rotulado "ArvBin\* raiz" que aponta para um nó "raiz". Este nó é o ponto de partida para a árvore. A árvore em si é composta por nós circulares azuis. O nó "raiz" tem dois filhos: um à esquerda e um à direita. O filho à direita é rotulado "NO", indicando um nó nulo. O filho à esquerda também tem dois filhos: um à esquerda e um à direita. O filho à esquerda deste nó tem dois filhos: um à esquerda e um à direita. O filho à direita deste nó tem dois filhos: um à esquerda e um à direita. A estrutura da árvore é a seguinte:
- ```
graph TD; raiz[raiz] --> n1(( )); n1 --> n2(( )); n1 --> n3((NO)); n2 --> n4(( )); n2 --> n5(( )); n4 --> n6(( )); n4 --> n7(( )); n5 --> n8(( )); n5 --> n9(( ));
```
- À direita do diagrama, há um retângulo que representa o formato de um nó. Ele é dividido em três partes: a primeira parte é rotulada "NO", a segunda parte é rotulada "\*esq" e a terceira parte é rotulada "\*dir".
- | NO   |      |      |
|------|------|------|
| *esq | dado | *dir |



# Estrutura

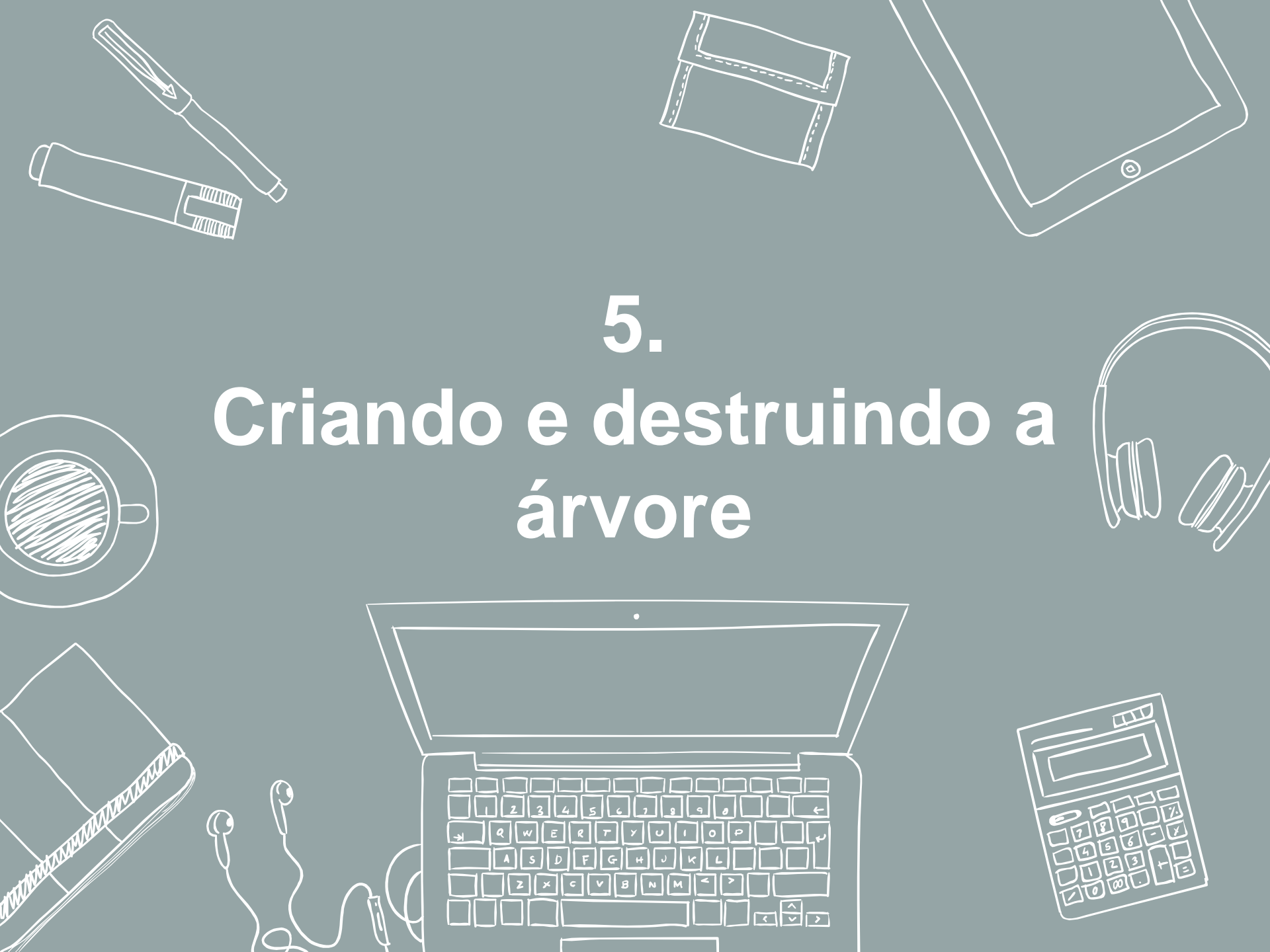
- No exemplo que iremos implementar, cada nó será tratado como uma estrutura:

```
struct NO{  
    int info;  
    struct NO *esq;  
    struct NO *dir;  
};  
typedef struct NO* ArvBin;
```



Ativar o Windows  
Acesse as configurações do  
ativar o Windows.

# 5. Criando e destruindo a árvore



# Criação da árvore

- A criação da árvore consiste na declaração da raiz e na atribuição do valor NULL à raiz

**ArvBin\* raiz**



Ativar o Windows  
Acesse as configurações do  
ativar o Windows.

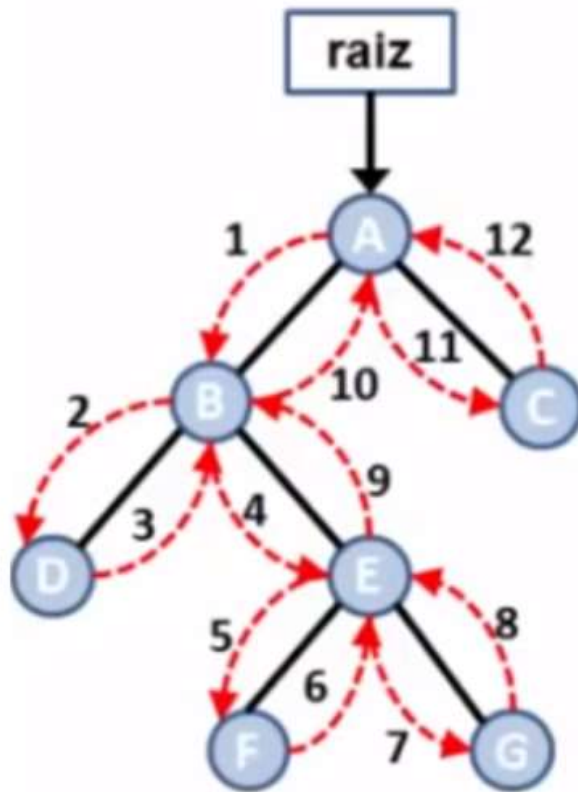
# Criação da árvore

```
ArvBin* cria_ArvBin(){  
    ArvBin* raiz = (ArvBin*) malloc(sizeof(ArvBin));  
    if(raiz != NULL)  
        *raiz = NULL;  
    return raiz;  
}
```



# Destruição da árvore

- Envolve percorrer os nós da árvore de modo a liberar a memória alocada para cada eles.



|      |                                   |
|------|-----------------------------------|
| 1    | visita B                          |
| 2    | visita D                          |
| ✗ 3  | libera D, volta para B            |
| 4    | visita E                          |
| 5    | visita F                          |
| ✗ 6  | libera F, volta para E            |
| 7    | visita G                          |
| ✗ 8  | libera G, volta para E            |
| ✗ 9  | libera E, volta para B            |
| ✗ 10 | libera B, volta para A            |
| 11   | visita C                          |
| ✗ 12 | libera C, volta para A e libera A |

# **Destruição da árvore**

```
void libera_NO(struct NO* no){  
    if(no == NULL) return;  
    libera_NO(no->esq);  
    libera_NO(no->dir);  
    free(no);  
}
```

```
void libera_ArvBin(ArvBin* raiz){  
    if(raiz == NULL)  
        return;  
    libera_NO(*raiz);  
    free(raiz);  
}
```

# 6. Inserção



# Inserção

Para inserir um valor  $x$  na árvore binária de busca, deve-se comparar  $x$  com a raiz

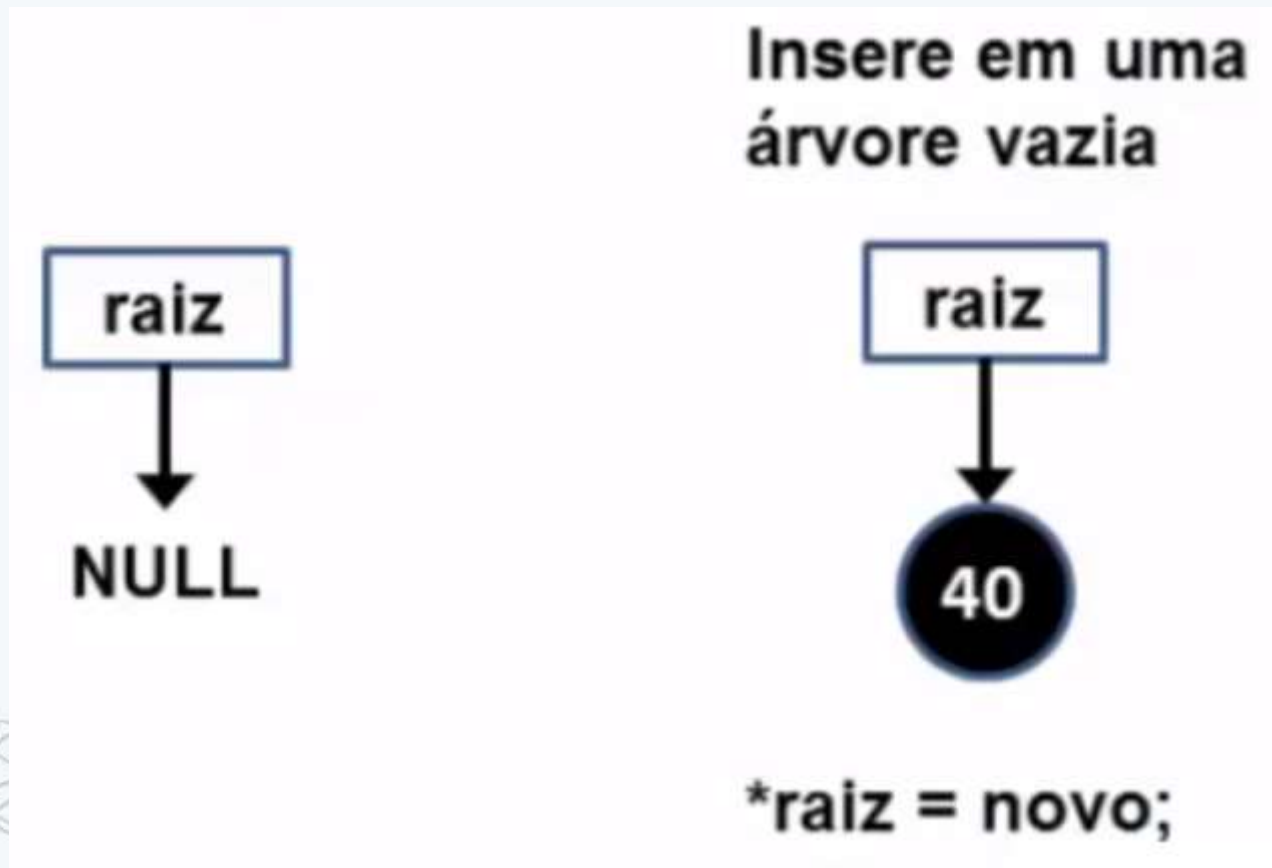
- Se for menor, vai para a sub-árvore da esquerda
- Se for maior, vai para a sub-árvore da direita

Este procedimento deve ser repetido, até não haver mais nó a ser comparado.

# Inserção inicial

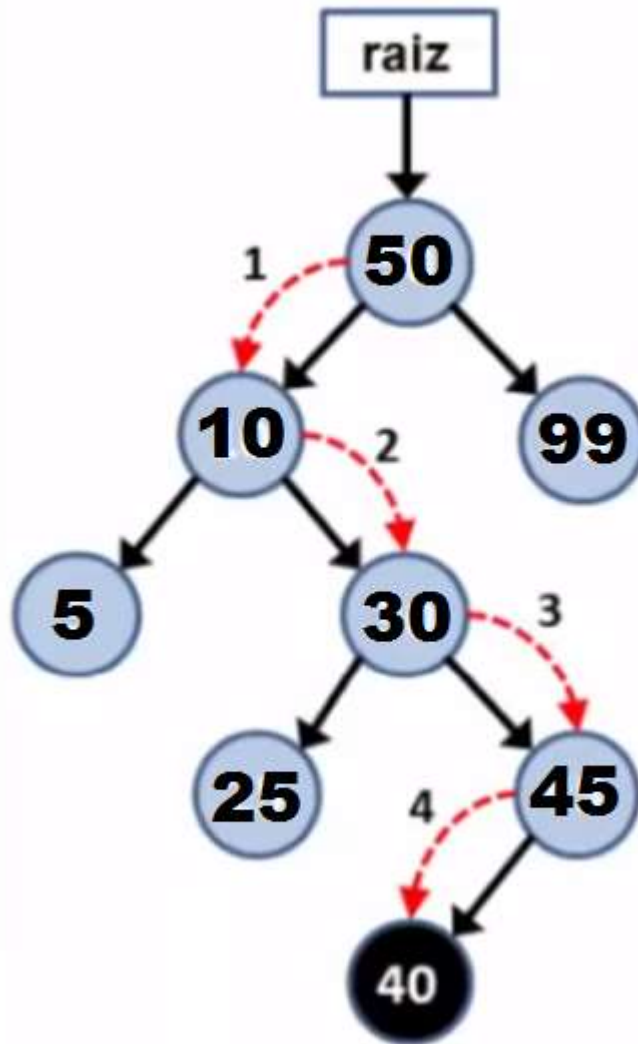
Existe o caso onde a inserção é feita em uma árvore vazia.

Neste caso, a inserção é fazer com que a raiz aponte para o nó inserido.



# Inserção

Insere como um nó folha



1 valor é menor do que 50:  
visita filho da esquerda

2 valor é maior do que 10:  
visita filho da direita

3 valor é maior do que 30:  
visita filho da direita

4 valor é menor do que 45:  
visita filho da esquerda

Não existe filho da esquerda.  
Valor passa a ser o filho da  
esquerda de 45



# Inserção

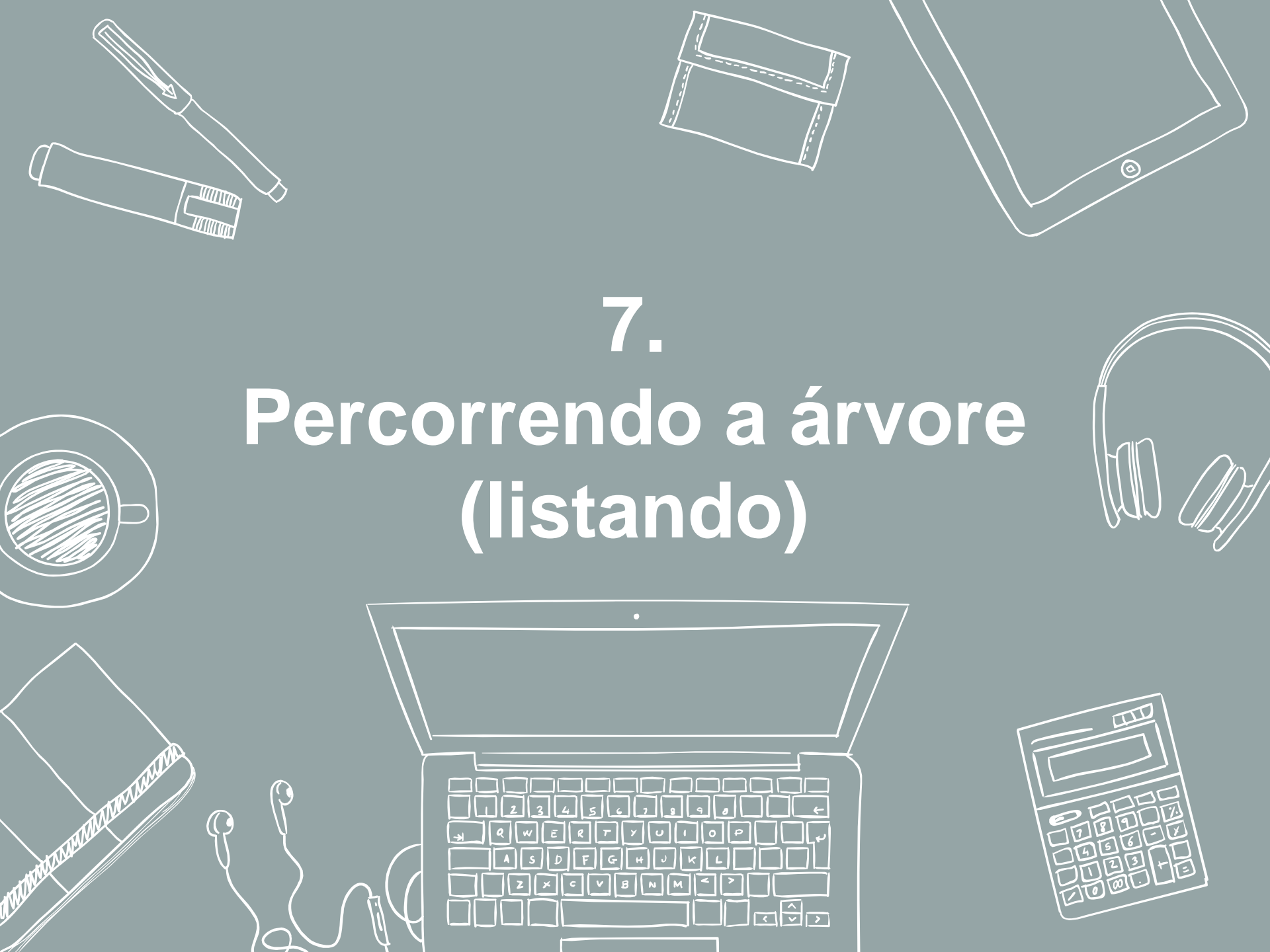
```
int insere_ArvBin(ArvBin* raiz, int valor){  
    if(raiz == NULL) return 0;  
    struct NO* novo;  
    novo = (struct NO*) malloc(sizeof(struct NO));  
    if(novo == NULL) return 0;  
    novo->info = valor;  
    novo->dir = NULL;  
    novo->esq = NULL;  
    if(*raiz == NULL) {  
        *raiz = novo;  
        return 1;  
    }  
}
```

# Inserção

```
struct NO* atual = *raiz;  
struct NO* ant = NULL;  
while(atual != NULL){  
    ant = atual;  
    if(valor == atual->info){  
        free(novo);  
        return 0;  
    }  
    if(valor > atual->info)  
        atual = atual->dir;  
    else  
        atual = atual->esq;  
}  
if(valor > ant->info)  
    ant->dir = novo;  
else  
    ant->esq = novo;  
return 1;
```

# 7.

## Percorrendo a árvore (listando)



# Percorrendo uma árvore

- Muitas operações em árvores binárias necessitam que se percorra todos os nós executando alguma ação ou tratamento em cada nó.
- Cada nó deve ser visitado um única vez
- Isso gera uma sequência linear de nós, cuja ordem depende de como a árvore foi percorrida.

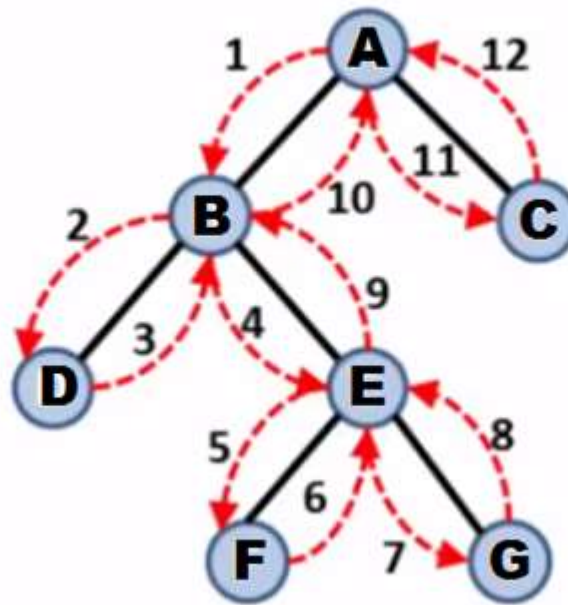
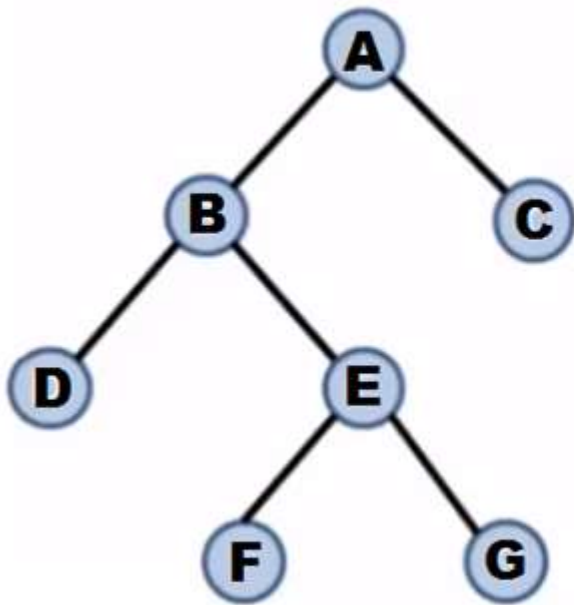
# Percorrendo uma árvore

Podemos percorrer a árvore de várias formas, das quais há 3 maneiras principais:

- Pré-ordem – visita a raiz, o filho da esquerda e o filho da direita (centro, esquerda, direita)
- Em-ordem – visita o filho da esquerda, a raiz e o filho da direita (esquerda, centro, direita)
- Pós-ordem – visita o filho da esquerda, o filho da direita e a raiz (esquerda, direita, centro)

Ativar o Windows  
Acesse as configurações do  
ativar o Windows.

# Pré-ordem



RESULTADO: ABDEFGC

|    |                         |
|----|-------------------------|
| 1  | imprime A, visita B     |
| 2  | imprime B, visita D     |
| 3  | imprime D, volta para B |
| 4  | visita E                |
| 5  | imprime E, visita F     |
| 6  | imprime F, volta para E |
| 7  | visita G                |
| 8  | imprime G, volta para E |
| 9  | volta para B            |
| 10 | volta para A            |
| 11 | visita C                |

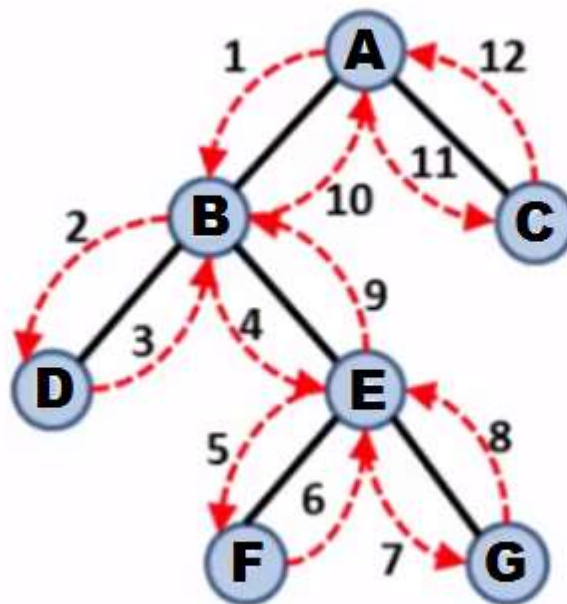
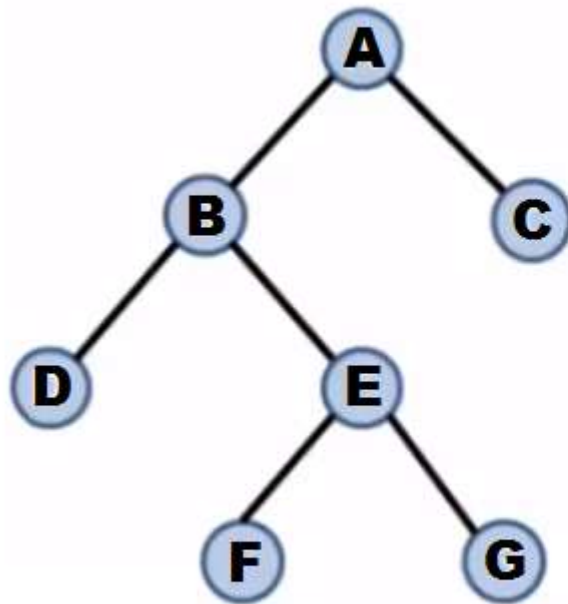


# Pré-ordem

```
void preOrdem_ArvBin(ArvBin *raiz){  
    if(raiz == NULL)  
        return;  
    if(*raiz != NULL){  
        printf("%d ",(*raiz)->info);  
        preOrdem_ArvBin(&((*raiz)->esq));  
        preOrdem_ArvBin(&((*raiz)->dir));  
    }  
}
```

Ativar o Windows  
Acesse as configurações do  
ativar o Windows.

# Em-ordem



RESULTADO: DBFEGAC

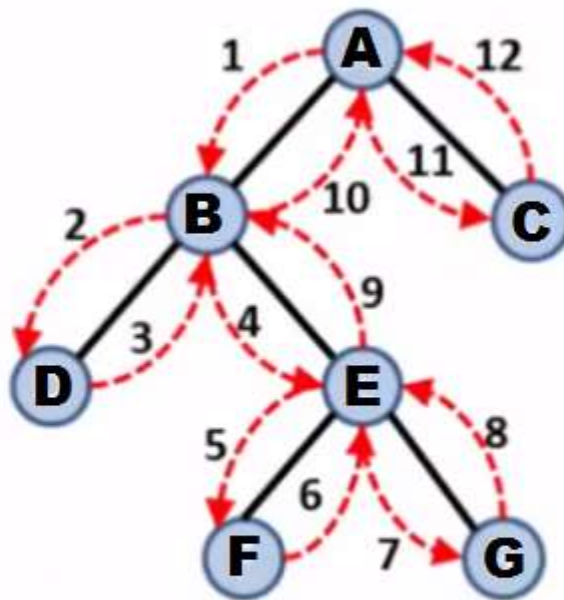
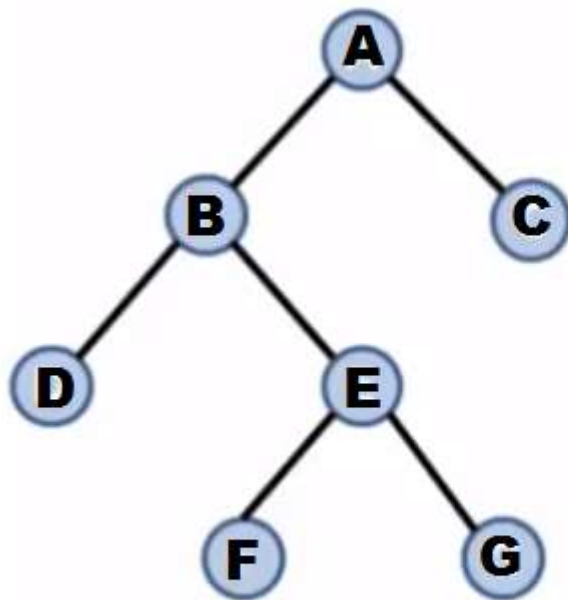
|    |                         |
|----|-------------------------|
| 1  | visita B                |
| 2  | visita D                |
| 3  | imprime D, volta para B |
| 4  | imprime B, visita E     |
| 5  | visita F                |
| 6  | imprime F, volta para E |
| 7  | imprime E, visita G     |
| 8  | imprime G, volta para E |
| 9  | volta para B            |
| 10 | volta para A            |
| 11 | imprime A, visita C     |

# Em-ordem

```
void emOrdem_ArvBin(ArvBin *raiz){  
    if(raiz == NULL)  
        return;  
    if(*raiz != NULL){  
        emOrdem_ArvBin(&((*raiz)->esq));  
        printf("%d ",(*raiz)->info);  
        emOrdem_ArvBin(&((*raiz)->dir));  
    }  
}
```

Ativar o Windows  
Acesse as configurações do  
ativar o Windows.

# Pós-ordem



RESULTADO: DFGEBCA

|    |                         |
|----|-------------------------|
| 1  | visita B                |
| 2  | visita D                |
| 3  | imprime D, volta para B |
| 4  | visita E                |
| 5  | visita F                |
| 6  | imprime F, volta para E |
| 7  | visita G                |
| 8  | imprime G, volta para E |
| 9  | imprime E, volta para B |
| 10 | imprime B, volta para A |
| 11 | visita C                |
| 12 | imprime C, volta para A |

# Pós-ordem

```
void posOrdem_ArvBin(ArvBin *raiz){  
    if(raiz == NULL)  
        return;  
    if(*raiz != NULL){  
        posOrdem_ArvBin(&((*raiz)->esq));  
        posOrdem_ArvBin(&((*raiz)->dir));  
        printf("%d ",(*raiz)->info);  
    }  
}
```

# 8. Consulta





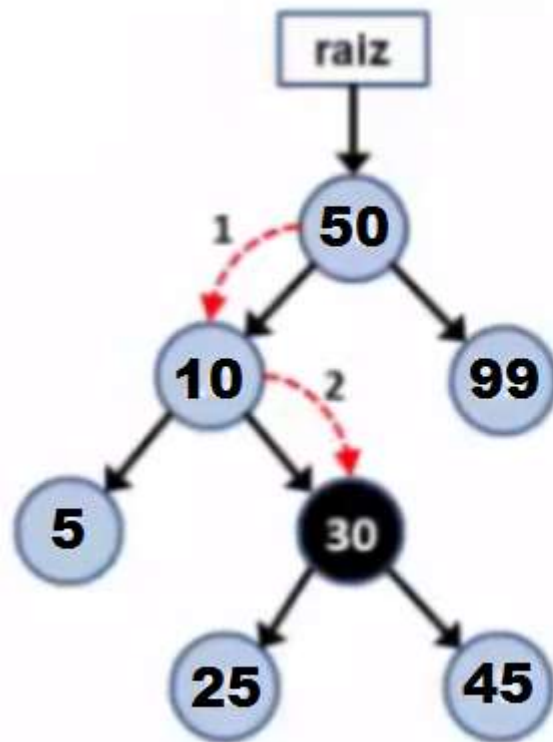
# Consulta

Para pesquisar um valor  $x$  na árvore binária de busca, compara  $x$  com a raiz

- Se for menor, vai para a sub-árvore da esquerda
- Se for maior, vai para a sub-árvore da direita

Este procedimento deve ser repetido, até não haver mais nó a ser comparado.

# Consulta

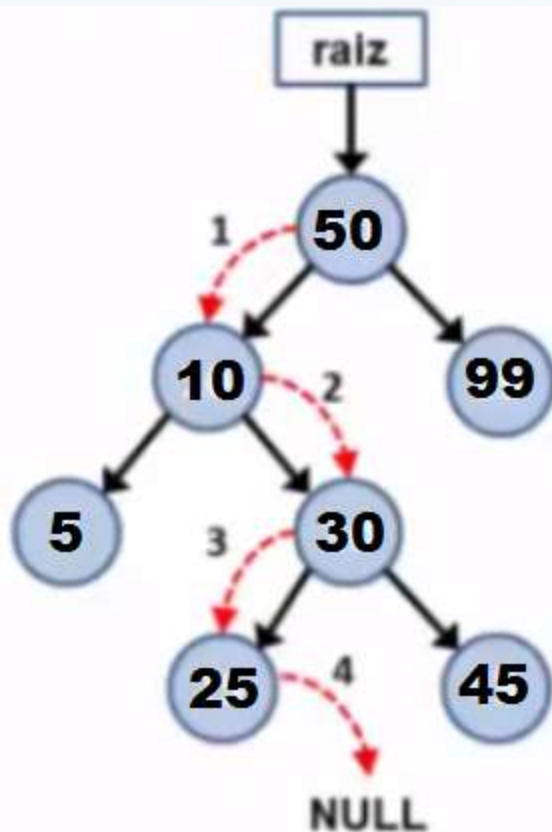


**Valor procurado: 30**

- 1 valor procurado é menor do que 50:  
visita filho da esquerda
  - 2 valor procurado é maior do que 10:  
visita filho da direita
- valor procurado é igual ao do nó:  
retornar dados do nó

Ativar o Windows  
Acesse as configurações do  
ativar o Windows.

# Consulta



**Valor procurado: 28**

- 1 valor procurado é menor do que 50:  
visita filho da esquerda
- 2 valor procurado é maior do que 10:  
visita filho da direita
- 3 valor procurado é menor do que 30:  
visita filho da esquerda
- 4 valor procurado é maior do que 25:  
visita filho da direita

Filho da direita de 25 não existe:  
a busca falhou

# Consulta

```
int consulta_ArvBin(ArvBin *raiz, int valor){  
    if(raiz == NULL)  
        return 0;  
    struct NO* atual = *raiz;  
    while(atual != NULL){  
        if(valor == atual->info){  
            return 1;  
        }  
        if(valor > atual->info)  
            atual = atual->dir;  
        else  
            atual = atual->esq;  
    }  
    return 0;  
}
```

Ativar o Windows  
Acesse as configurações do  
ativar o Windows.

# 9. Remoção





# Remoção

Existem 3 tipos de remoção:

- Nó folha (sem filhos)
- Nó com 1 filho
- Nó com 2 filhos

Ativar o Windows  
Acesse as configurações do Windows para  
ativar o Windows.



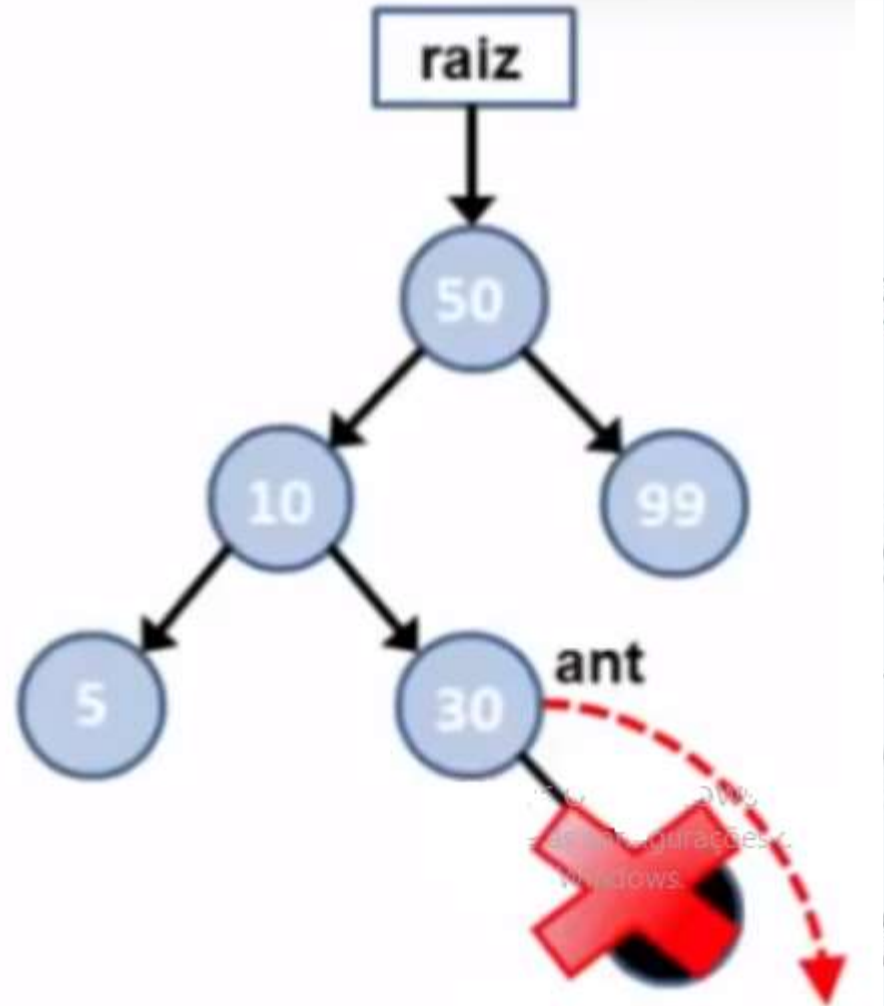
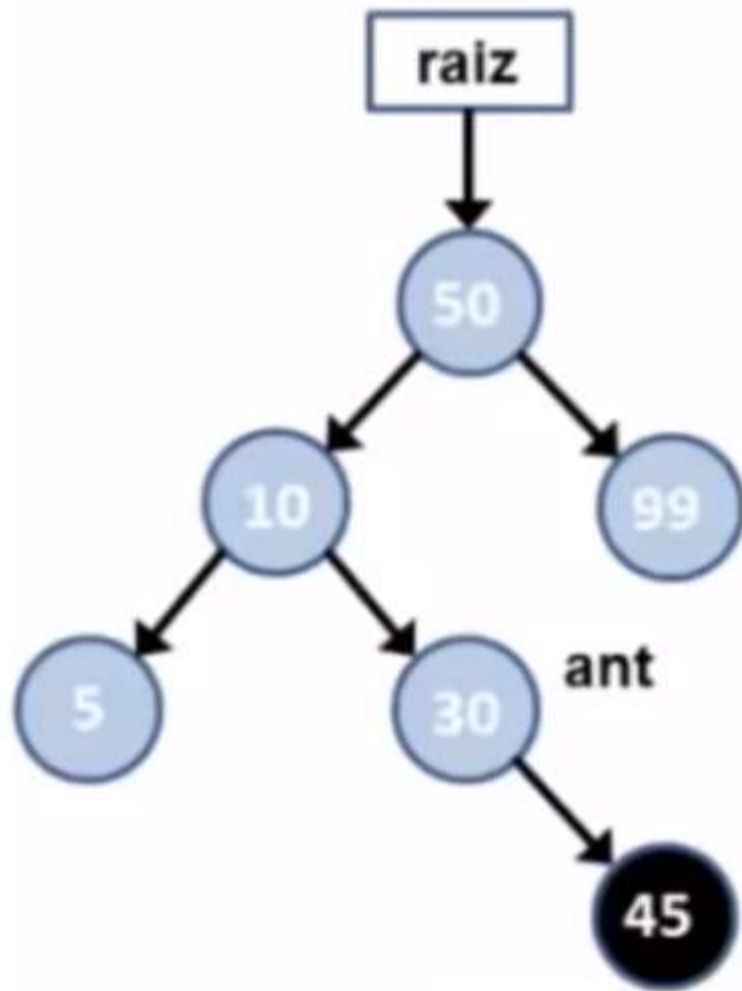


## Remoção de nó folha

Para remoção de algum nó folha, basta atribuir o valor NULL para o ponteiro (***esq*** ou ***dir***) que apontava para este nó.

Ativar o Windows  
Acesse as configurações do  
ativar o Windows.

# Remoção de nó folha



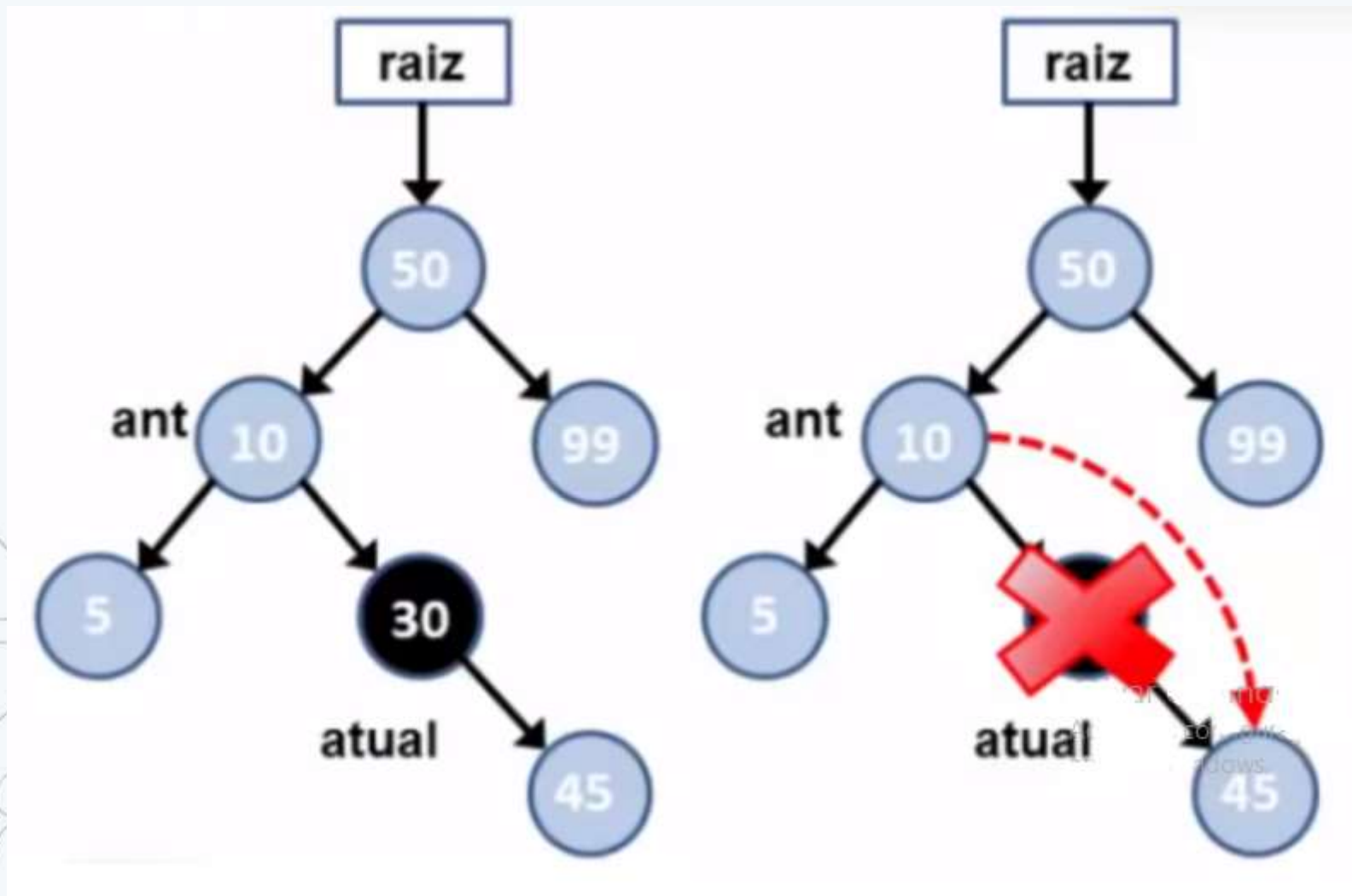


## Remoção com um filho

Para remoção de algum nó com um filho, basta atribuir o endereço deste filho para o ponteiro (***esq*** ou ***dir***) que apontava para este nó.

Ativar o Windows  
Acesse as configurações do Windows para  
ativar o Windows.

# Remoção com um filho

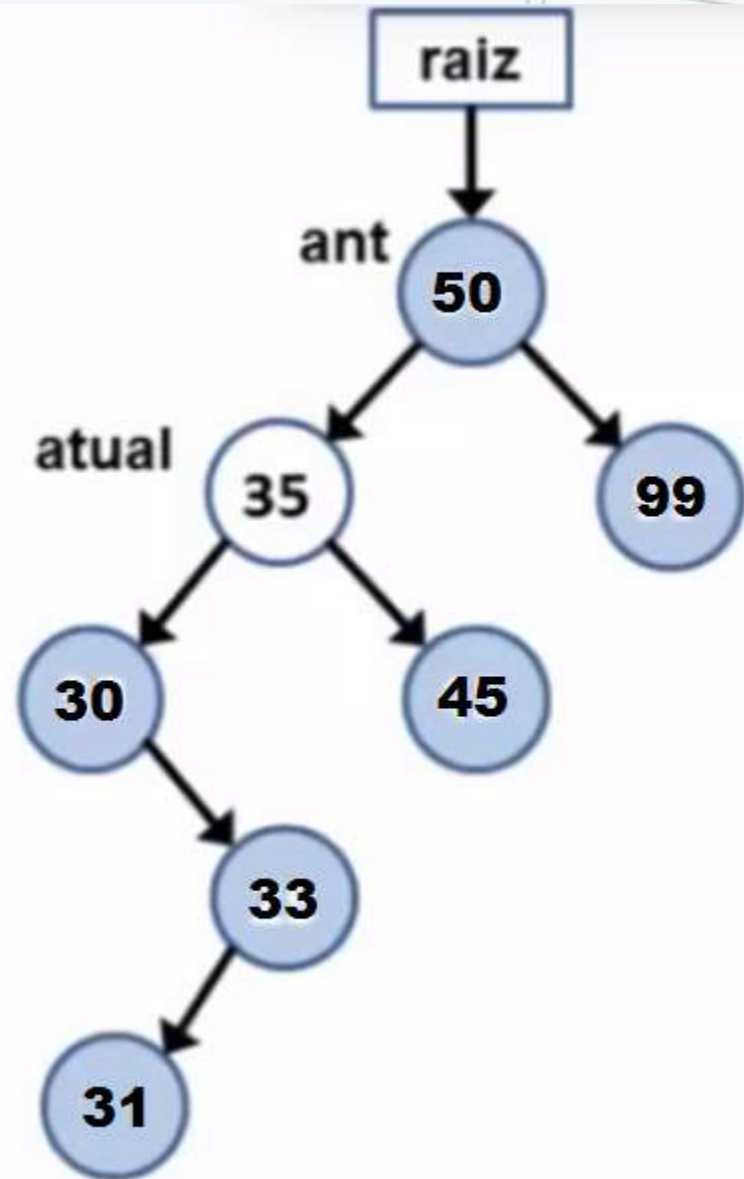
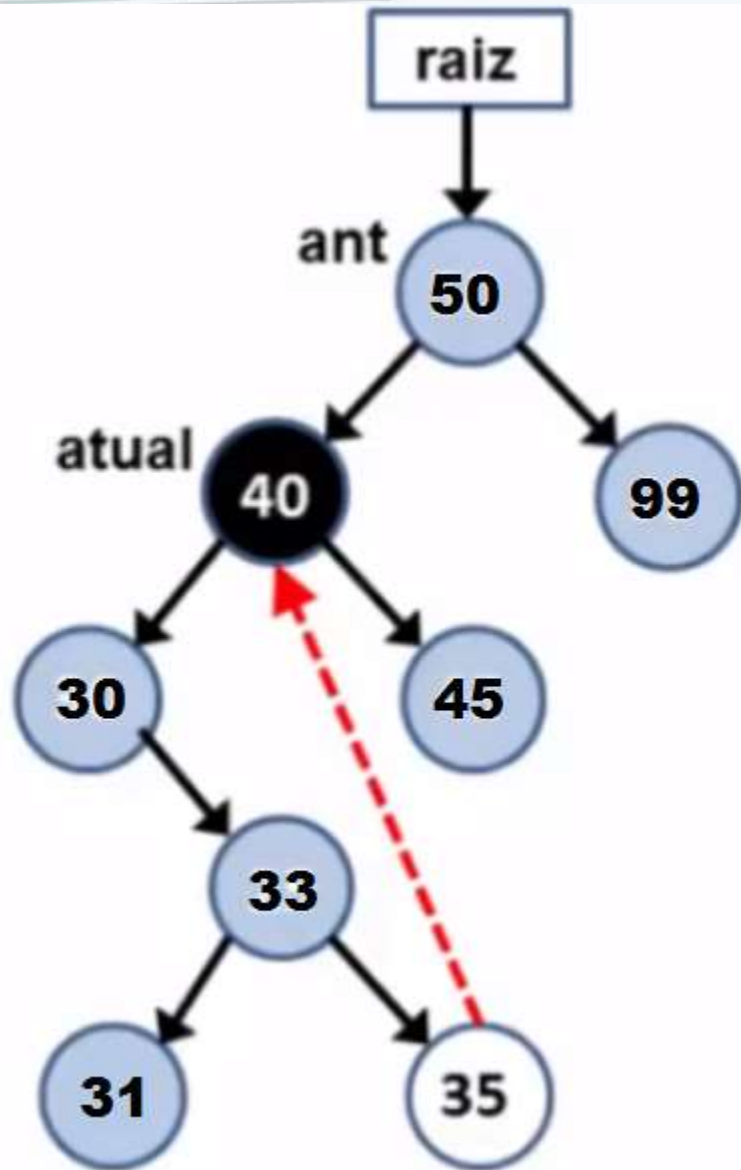


# Remoção com dois filhos

Para remoção de algum nó com dois filhos, deve-se fazer a substituição deste nó pelo nó que estiver mais à direita na sub-árvore da esquerda.

Ativar o Windows  
Acesse as configurações do  
ativar o Windows.

# Remoção com dois filhos





# Remoção

Os 3 tipos de remoção trabalham juntos.

A remoção sempre remove um elemento específico.

Cuidado:

- Não se pode remover de uma árvore sem nós
- Removendo o último nó a árvore ficará vazia.





# Referências

- ASCENCIO, Ana Fernanda Gomes; ARAÚJO, Graziela Santos de. *Estrutura de dados: algoritmos, análise de complexidade e implementações em Java e C/C++*. São Paulo: Pearson, 2010.
- BACKES, André. *Estrutura de dados descomplicada: em linguagem C*. Rio de Janeiro: Elsevier, 2016.
- TOSCANI, Laira Vieira. *Complexidade de algoritmos: análise, projeto e métodos*. 3. ed. Porto Alegre: Bookman, 2012.
- ZIVIANI, Nivio. *Projeto de algoritmos: com implementações em Pascal e C*. 2. ed. São Paulo: Thomson, 2005.
- Slides de aula dos professores André Backes, Felipe Ribeiro e Kleber Fonseca



**Muito Obrigado!**

*Alguma Pergunta?*