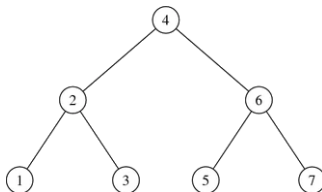


# Algoritmos e Estrutura de Dados

## Árvores Binárias de Busca

prof. Frederico Santos de Oliveira

Universidade Federal de Mato Grosso  
Faculdade de Engenharia



# Roteiro

- 1 Objetivos
- 2 Introdução
- 3 Implementação
  - Busca
  - Inserção
  - Remoção
- 4 Conclusão

# Objetivos

Esta aula tem como objetivos:

- 1 Apresentar os conceitos básicos sobre árvores binárias de busca (ABBs);
- 2 Exemplificar os algoritmos de manipulação de ABBs por meio de pseudo-códigos.

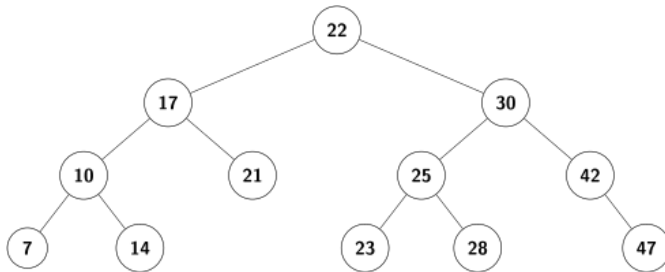
# Árvore Binária de Busca

## Introdução

- Um Árvore Binária de Busca (ABB) é um tipo especial de árvore binária, ou seja, possui as mesmas propriedades de uma árvore binária.
- No entanto, na ABB o item armazenado no nodo determina a sua posição na árvore.
- Em uma ABB, temos a seguinte regra para posicionamento dos valores na árvore, para cada nodo **pai**:
  - Todos os valores da subárvore **esquerda** são **menores** do que o nodo pai.
  - Todos os valores da subárvore **direita** são **maiores** do que o nodo pai.

# Árvore Binária de Busca

## Exemplo



# Árvore Binária de Busca

## Operações

A seguir, são apresentados os seguintes algoritmos sobre ABBs:

- 1 Busca
- 2 Inserção
- 3 Remoção

# Árvore Binária de Busca

## Busca

A seguir, são apresentados duas versões do algoritmo de busca:

- 1 Uma versão **recursiva**
- 2 Uma versão **iterativa**

# Árvore Binária de Busca

## Busca Recursiva

- Algoritmos recursivos possuem pelo menos um **caso base** e um **caso recursivo**.
- O caso base resolve o problema, ou seja, encontra ou não o elemento na árvore:
  - Retorna NULL quando o elemento não foi encontrado.
  - Retorna um ponteiro para o elemento procurado, caso tenha sido encontrado.
- Existem dois casos recursivos:
  - O primeiro ocorre quando o elemento procurado é menor que o valor armazenado no nodo raiz .
    - Nesse caso, busca-se **recursivamente** na subárvore da esquerda.
  - O segundo caso ocorre quando o valor procurado é maior que o valor armazenado no nodo raiz.
    - Nesse caso, busca-se **recursivamente** na subárvore da direita.



# Árvore Binária de Busca

## Função Buscar - Versão Recursiva

---

### Algoritmo 1: Busca

---

**Entrada:** Ponteiro para a raiz  $r$  e o item  $x$  a ser procurado.

**Saída:** Retorna o nodo que contém o item  $x$  ou NULL caso não encontrado.

```
1 início
2   se  $(r = NULL)$  ou  $(x = r.item)$  então
3     retorna  $r$ 
4   senão se  $(x > r.item)$  então
5     retorna Buscar( $r.dir$ )
6   senão
7     retorna Buscar( $r.esq$ )
```

---

# Árvore Binária de Busca

## Busca Iterativa

- O algoritmo iterativo utiliza um laço de repetição (**enquanto**) para caminhar na árvore.
- Para isso, utiliza um ponteiro auxiliar, chamado **atual**:
  - Percorre a árvore a fim de encontrar o elemento procurado.
- O laço termina em duas situações:
  - Quando (**atual = NULL**), o que indica que o elemento não foi encontrado.
  - Ou quando (**atual.item = x**), ou seja, encontrou o elemento procurado.
- Por fim, retorn NULL ou o ponteiro para o elemento encontrado.

# Árvore Binária de Busca

## Função Buscar - Versão Iterativa

---

### Algoritmo 2: Busca

---

**Entrada:** Ponteiro para a raiz  $r$  e o item  $x$  a ser procurado.

**Saída:** Retorna o nodo que contém o item  $x$  ou NULL caso não encontrado.

```
1 início
2   indica se ( $r = NULL$ ) então
3     retorna NULL
4   senão
5     atual  $\leftarrow r$ 
6     enquanto ( $atual \neq NULL$ ) AND ( $atual.item \neq x$ ) faça
7       se ( $x > atual.item$ ) então
8         atual  $\leftarrow atual.dir$ 
9       senão
10        atual  $\leftarrow atual.esq$ 
11    retorna atual
```

---

Adaptado de (BACKES, 2016)

# Árvore Binária de Busca

- As operações de inserção e remoção de nodos na ABB devem ser realizadas respeitando as regras de posicionamento dos nodos.
- No caso médio, as operações de busca, inserção e remoção possuem tempo de execução  $O(\log(n))$  em que  $n$  é a quantidade de nodos na árvore.
- No entanto, em seu pior caso, as operações de busca, inserção e remoção possuem tempo de execução  $O(n)$

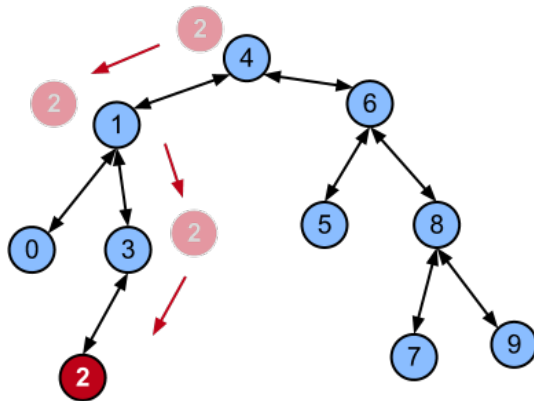
# Árvore Binária de Busca

## Inserção

- Inserir um novo nodo em uma ABB é uma tarefa bastante simples.
- Basta alocar espaço para o novo nodo e procurar a sua posição na ABB.
- A seguir, os passos necessários:
  - 1 Primeiro, compare o valor a ser inserido com a **raiz**.
  - 2 Se o valor for menor do que a raiz, vá para a subárvore da **esquerda**.
  - 3 Caso contrário, vá para a subárvore da **direita**.
  - 4 Aplique o método recursivamente até chegar a um **nodo folha**.

# Árvore Binária de Busca

## Inserção



# Árvore Binária de Busca

## Inserção

A seguir, são apresentados duas versões do algoritmo de inserção:

- 1 Uma versão **recursiva**
- 2 Uma versão **iterativa**

# Árvore Binária de Busca

## Inserção Recursiva

- Algoritmos recursivos possuem pelo menos um **caso base** e um **caso recursivo**.
- O caso base resolve o problema, ou seja, insere um elemento na árvore:
  - Neste algoritmo, o caso base ocorre quando a raiz da árvore (ou subárvore) é igual a NULL.
  - Nesse caso, cria-se um novo nodo, e este será a raiz da árvore (ou subárvore).
- Existem dois casos recursivos:
  - O primeiro ocorre quando o elemento a ser inserido é menor que o valor armazenado no nodo raiz .
    - Nesse caso, insere-se **recursivamente** na subárvore da esquerda.
  - O segundo caso ocorre quando o valor a ser inserido é maior que o valor armazenado no nodo raiz.
    - Nesse caso, insere-se **recursivamente** na subárvore da direita.



# Árvore Binária de Busca

## Inserção - Versão Recursiva

### Algoritmo 3: InserirÁrvore

**Entrada:** Ponteiro  $r$  para raiz, item  $x$  a ser inserido na árvore.

```
1 início
2   se  $(r=NULL)$  então
3     novo  $\leftarrow$  ALOCA_NODO()
4     novo.item  $\leftarrow x$ 
5     novo.esq  $\leftarrow NULL$ 
6     novo.dir  $\leftarrow NULL$ 
7      $r \leftarrow$  novo
8   senão se  $(x < r.item)$  então
9     InserirÁrvore( $r.esq, x$ )
10  senão
11    InserirÁrvore( $r.dir, x$ )
```

# Árvore Binária de Busca

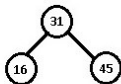
## Inserção



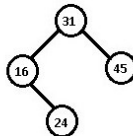
**Insert 31**



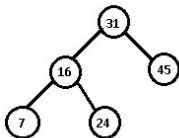
**Insert 16**



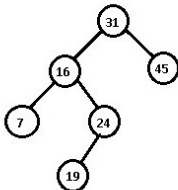
**Insert 45**



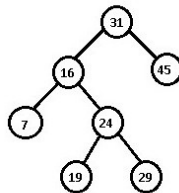
**Insert 24**



**Insert 7**



**Insert 19**



**Insert 29**

# Árvore Binária de Busca

## Inserção - Versão Iterativa

- Na versão iterativa, primeiramente, cria-se o novo nodo que será inserido na árvore.
- Em seguida, deve-se encontrar a posição em que o novo nodo será inserido na árvore.
  - Para isso, utiliza-se um laço de repetição (**enquanto**) para encontrar a posição em que será inserido o novo elemento.
  - Existem dois ponteiros auxiliares:
    - Atual → percorre a árvore a fim de encontrar a posição em que será inserido o novo nodo.
    - Anterior → é o nodo anterior ao nodo **atual**, ou seja, o seu pai.
  - O laço termina quando (**atual = NULL**).
  - Por fim, o novo nodo é inserido como filho do nodo **anterior**.

# Árvore Binária de Busca

## Inserção - Versão Iterativa

---

### Algoritmo 4: InserirÁrvore

---

**Entrada:** Ponteiro  $r$  para raiz, item  $x$  a ser inserido na árvore.

```
1 início
2   novo ← ALOCA_NODO()
3   novo.item ← x
4   novo.esq ← NULL
5   novo.dir ← NULL
6   se ( $r=NULL$ ) então
7      $r$  ← novo
8   senão
9     atual ← r
10    anterior ← NULL
11    enquanto ( $atual \neq NULL$ ) faça
12      anterior ← atual
13      se ( $novo.item < atual.item$ ) então
14         $atual$  ←  $atual.esq$ 
15      senão
16         $atual$  ←  $atual.dir$ 
17    se ( $novo.item < anterior.item$ ) então
18       $anterior.esq$  ← novo
19    senão
20       $anterior.dir$  ← novo
```

---

Adaptado de (BACKES, 2016)

# Árvore Binária de Busca

## Remoção

- Para remover um item de uma ABB precisamos procurar o nodo a ser removido, que pode ser um nodo **folha** ou um nó **interno**.
  - Caso seja um nodo interno, pode possuir um ou dois filhos.
  - Um dos filhos substitui o pai e é preciso reorganizar a árvore para que ela continue sendo uma ABB.
- Para realizar a remoção de um elemento, será necessário utilizar uma função auxiliar, denominada **RemoverAtual**.
- Essa função auxiliar recebe como parâmetro o endereço de um nodo da árvore (**atual**) a ser removido e retorna qual deverá ser o seu nodo **substituto**, que será o seu antecessor na árvore.

# Árvore Binária de Busca

## Função RemoverAtual

---

### Algoritmo 5: RemoverAtual

---

**Entrada:** Ponteiro **atual** para o nodo a ser removido.

**Saída:** Retorna o nodo substituto do nodo **atual**.

```
1 início
2   // Verifica se atual possui apenas um filho.
3   se (atual.esq = NULL) então
4       nodo2 ← atual.dir
5       DESALOCA_NODO(atual)
6       retorna nodo2
7   // Busca o antecessor: nodo mais à direita
8   // da sub-árvore da esquerda.
9   nodo1 ← atual
10  nodo2 ← atual.esq
11  enquanto (nodo2.dir ≠ NULL) faça
12      nodo1 ← nodo2 // Pai do antecessor.
13      nodo2 ← nodo2.dir // Antecessor.
14  // Se o antecessor não for o filho à esq do nodo atual.
15  se (nodo1 ≠ atual) então
16      // Antecessor herda os filhos do nodo atual.
17      nodo1.dir ← nodo2.esq
18      nodo2.esq ← atual.esq
19  nodo2.dir ← atual.dir // Antecessor herda o filho da dir
20  DESALOCA_NODO(atual)
21  retorna nodo2
```

---

Adaptado de (BACKES, 2016)

# Árvore Binária de Busca

## Função RemoverAtual

- Primeiro, a função verifica se o nodo é um nodo folha, ou possua um único filho, o filho à direita. O tratamento nos dois casos é o mesmo:
  - Copia o filho da direita para um nó auxiliar (**nodo2**), libera o nodo **atual**, e retorna o filho da direita (**nodo2**), que será o substituto do nodo **atual** na árvore.
  - No caso de **atual** ser um nodo folha, o filho à direita será NULL, portanto a função retorna NULL.
  - Essa etapa é realizada nas linhas 2 até 6.
- Caso **atual** possua os dois filhos, busca o nodo antecessor (nodo mais à direita da subárvore da esquerda de **atual**). Isso é realizado pelo laço **enquanto**, linha 7 até 13.
- Ao fim desta etapa, laço **enquanto**, o **nodo2** aponta para o antecessor, e **nodo1** aponta para o pai do antecessor.





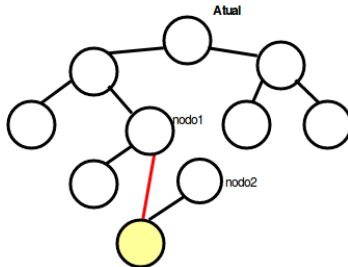
# Árvore Binária de Busca

## Função RemoverAtual

- Ao executar o comando da linha 17:

$$nodo1.dir \leftarrow nodo2.esq,$$

nodo1 herda o filho da esquerda de nodo2, conforme a figura abaixo.



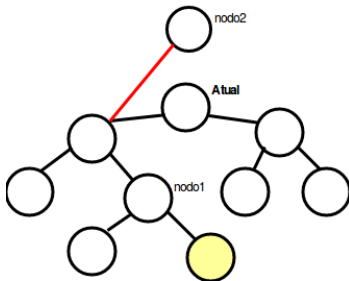
# Árvore Binária de Busca

## Função RemoverAtual

- O próximo comando, linha 18:

$$nodo2.esq \leftarrow atual.esq$$

faz com que nodo2 herde o filho da esquerda do nodo a ser removido, o nodo raiz, conforme a figura abaixo.



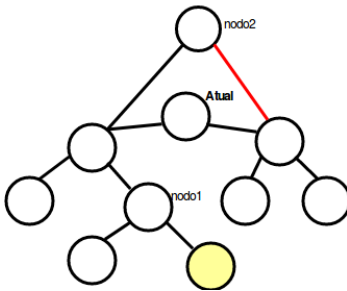
# Árvore Binária de Busca

## Função RemoverAtual

- O último comando desta etapa, linha 19:

$nodo2.dir \leftarrow atual.dir$

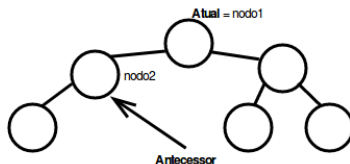
faz com que nodo2 herde o filho da direita do nodo a ser removido, o nodo raiz, conforme a figura abaixo.



# Árvore Binária de Busca

## Função RemoverAtual

- Observe que as linhas 17 e 18 não são executadas caso o pai do nodo antecessor seja igual ao nodo **atual**, ou seja executa apenas se (**nodo1**  $\neq$  **atual**), pois se (**nodo1** = **atual**) a subárvore da esquerda não será alterada.
- Isso ocorre, quando tem-se a situação mostrada na figura abaixo.



- O comando da linha 19 é executado independente de  $nodo1 \neq atual$  ( o nodo antecessor ser o filho da esquerda do nodo atual).

# Árvore Binária de Busca

## Função RemoverAtual

- Ao fim desta etapa, do laço **enquanto**, o **nodo2** aponta para o antecessor, e **nodo1** aponta para o pai do antecessor.

# Árvore Binária de Busca

## Função RemoverÁrvore

- Para remover um elemento, tem-se três casos a serem tratados:
  - ① Caso o nodo a ser removido seja a raiz.
    - Nesse caso, busca seu antecessor e este passa a ser a nova raiz.
  - ② Caso contrário, verifica se o nodo a ser removido está à direita de seu pai.
    - Busca o antecessor e este passa a ser o filho à direita do pai do nodo a ser removido.
  - ③ Ou se está à esquerda de seu pai.
    - Busca o antecessor e este passa a ser o filho à esquerda do pai do nodo a ser removido.

# Árvore Binária de Busca

## Função RemoverÁrvore

---

**Algoritmo 6:** RemoverÁrvore

---

**Entrada:** Ponteiro  $r$  para raiz, item  $x$  a ser removido na árvore.

**Saída:** Retorna V ou F.

```
1 início
2   se ( $r=NULL$ ) então
3     retorna Falso
4   anterior  $\leftarrow NULL$ 
5   atual  $\leftarrow r$ 
6   enquanto ( $atual \neq NULL$ ) faça
7     se ( $valor = atual.item$ ) então
8       se ( $atual = r$ ) então
9          $r \leftarrow RemoveAtual(atual)$ 
10      senão
11        se ( $anterior.dir = atual$ ) então
12           $anterior.dir \leftarrow RemoveAtual(atual)$ 
13        senão
14           $anterior.esq \leftarrow RemoveAtual(atual)$ 
15      retorna Verdadeiro
16    senão
17      anterior  $\leftarrow atual$ 
18      se ( $x > atual.item$ ) então
19         $atual \leftarrow atual.dir$ 
20      senão
21         $atual \leftarrow atual.esq$ 
22  retorna Falso
```

---

Adaptado de (BACKES, 2016)

# Árvore Binária de Busca

## Função RemoverÁrvore

- Primeiramente, verifica se a raiz  $r$  é igual a NULL (linha 2 ).
  - Caso seja verdadeiro, retorna falso indicando um erro: a árvore está vazia (linha 3).
- Para buscar o elemento a ser removido, cria-se dois ponteiros auxiliares:
  - **atual**, ponteiro utilizado para caminhar na árvore.
  - **anterior**, pai do ponteiro **atual** .
- O laço de repetição **enquanto** termina quando **atual** é igual a NULL, o que indica que o elemento a ser removido não foi encontrado.



# Árvore Binária de Busca

## Função RemoverÁrvore

- Dentro do laço de repetição, tem-se duas operações:
  - ① Caso encontre o nodo, o remove (linha 7 até 15).
  - ② Caso não encontre, caminha na árvore a fim de encontrá-lo (linha 16 até 21).
- Ao encontrar o nodo a ser removido, tem-se três situações:
  - ① Caso o nodo a ser removido seja a raiz (**atual** =  $r$ ).
    - Chama a função RemoveAtual, o nodo antecessor de  $r$  passa a ser a nova raiz.
  - ② Caso não seja a raiz, verifica se o nodo a ser removido está à direita de seu pai.
    - Chama a função RemoveAtual, que retorna o substituto do nodo **atual**, e este passa a ser o filho à direita do pai do nodo a ser removido.
  - ③ Caso contrário, o nodo a ser removido está à esquerda de seu pai.
    - Idêntico ao anterior, no entanto o substituto passa a ser o filho à esquerda do pai do nodo a ser removido.

# Árvore Binária

## Implementação

- Para apagar uma árvore, deve-se:
  - Apagar recursivamente suas subárvores
  - Em seguida, apagar o nodo raiz.
- As chamadas recursivas param ao encontrar uma subárvore vazia, ou seja, quando (**r == NULL**)

# Árvore Binária de Busca

## Apaga Árvore

---

### Algoritmo 7: ApagarÁrvore

---

**Entrada:** Ponteiro  $r$  para raiz da árvore a ser apagada.

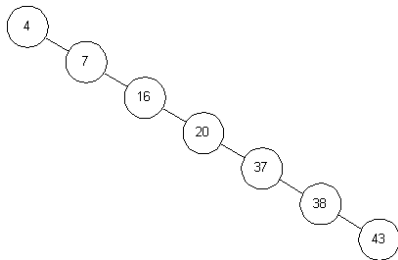
```
1 início
2   se ( $NOT(ÁrvoreVazia(r))$ ) então
3     ApagarÁrvore(r.esq)
4     ApagarÁrvore(r.dir)
5     DESALOCA_NODO(r)
```

---

Adaptado de (BACKES, 2016)

## Conclusão


- No pior caso, pode acontecer da árvore ficar **degenerada**.
  - Cada nodo possui um único filho.
  - Sua estrutura é igual a uma lista linear.
  - A altura  $h$  da árvore será:  $h = n - 1$
- Isso dependerá da ordem em que os elementos são inseridos na árvore.
- Caso isso ocorra, o custo das operações é linear ( $O(n)$ ).
- A fim de evitar seu degeneramento, pode-se realizar o seu balanceamento.
- A seguir, um exemplo de árvore degenerada.



# Referências

## Bibliografia Básica

- Bibliografia Básica

 BACKES, A. *Estrutura de Dados Descomplicada - Em Linguagem C*. 1. ed. São Paulo: Elsevier, 2016.

- Material Complementar

- Link código-fonte e listas de exercícios- Material disponível on-line

- Animação

- Link Árvore Binária de Busca
- Link Árvore Binária de Busca - Versão que utiliza sucessor como nodo substituto na remoção.

# Algoritmos e Estrutura de Dados

## Árvores Binárias de Busca

prof. Frederico Santos de Oliveira

Universidade Federal de Mato Grosso  
Faculdade de Engenharia

