

Árvores

Árvores de Busca Binária em Java

Prof. Leandro Colevati

Introdução

- Operações Básicas (Devem partir da raiz):
 - Criação da árvore
 - Inserção
 - Busca
 - Remoção
 - Atravessamento
 - Prefixo
 - Infixo
 - Posfixo

Alocação Dinâmica

- Considere a definição do tipo **Árvore** abaixo:
 - Ponteiro raiz → NULL

Alocação Dinâmica

- Considere a definição do tipo **Árvore** abaixo:

```
class No {  
    tipo    dado;  
    No     esquerda; //Ponteiro  
    No     direita; //Ponteiro  
  
}
```



Alocação Dinâmica

■ Inserindo um elemento :

```
private void insertLeaf(No no, No raizSubArvore) {
    se (raiz == nulo) {
        raiz = no;
    } senão {
        se (no.dado < raizSubArvore.dado) {
            se (raizSubArvore.esquerda == nulo) {
                raizSubArvore.esquerda = no;
            } senao {
                insertLeaf (no, raizSubArvore.esquerda);
            }
        }
        se (no.dado >= raizSubArvore.dado) {
            se (raizSubArvore.direita == nulo) {
                raizSubArvore.direita = no;
            } senao {
                inserLeaf(no, raizSubArvore.direita);
            }
        }
    }
}
```

```
void insert(int dado) {
    No no = new No();
    no.dado = dado;
    no.esquerda = nulo;
    no.direita = nulo;
    insertLeaf (no, raiz);
}
```

Alocação Dinâmica

■ Buscando um elemento:

```
private No nodeSearch(No raizSubArvore, tipo valor) {
    se (raiz == nulo) {
        exceção("Árvore vazia");
    } senão se (raizSubArvore.dado > valor) {
        retorne nodeSearch(raizSubArvore.esquerda, valor);
    } senão se (raizSubArvore.dado < valor) {
        retorne nodeSearch(raizSubArvore.direita, valor);
    } senao {
        retorne raizSubArvore;
    }
}
```

```
private int nodeLevel(No raizSubArvore, tipo valor) {
    se (raiz == nulo) {
        exceção("Árvore vazia");
    } senão se (raizSubArvore.dado > valor) {
        retorne 1 + nodeLevel(raizSubArvore.esquerda, valor);
    } senão se (raizSubArvore.dado < valor) {
        retorne 1 + nodeLevel(raizSubArvore.direita, valor);
    } senao {
        retorne 0;
    }
}
```

```
public void search(int valor) {
    try {
        No no = nodeSearch(raiz, valor);
        int level = nodeLevel(raiz, valor);
        imprime("Dado "+no.dado+" nível "+level);
    } catch(Exception e) {
        Excecao("Valor não existente");
    }
}
```

Alocação Dinâmica

■ Removendo um elemento (3 casos):

```
private void removeChild(No raizSubArvore, int valor) {
    No pai = nodeParent(null, raiz, valor);
    No no = nodeSearch(raizSubArvore, valor);
    se (no.esquerda != null && no.direita != null) {
        No noTroca = no.esquerda;
        enquanto (noTroca.direita != null) {
            noTroca = noTroca.direita;
        }
        pai = nodeParent(nulo, raiz, noTroca.dado);
        no.dado = noTroca.dado;
        noTroca.dado = valor;
        removeOneOrZeroLeaf(pai, noTroca);
    } senao {
        removeOneOrZeroLeaf(pai, no);
    }
}
```

```
private No nodeParent(No parent, No raizSubArvore, int valor) {
    se (raiz == null) {
        Excecao("Árvore Vazia");
    } senao se (raizSubArvore.dado > valor) {
        retorne nodeParent(raizSubArvore, raizSubArvore.esquerda, valor);
    } senao se (raizSubArvore.dado < valor) {
        retorne nodeParent(raizSubArvore, raizSubArvore.direita, valor);
    } senao {
        se (parent == null) {
            retorne raiz;
        } senao {
            retorne parent;
        }
    }
}
```

```
private void removeOneOrZeroLeaf(No pai, No no) {
    se (no.esquerda == null && no.direita == null) {
        change(pai, no, null);
    } senao se (no.esquerda == null) {
        change(pai, no, no.direita);
    } senao se (no.direita == null) {
        change(pai, no, no.esquerda);
    }
}
```

```
private void change(No pai, No no, No novoNo) {
    se (pai.esquerda != nulo && pai.esquerda.dado == no.dado) {
        pai.esquerda = novoNo;
    } senao se (pai.direita.dado == no.dado) {
        pai.direita = novoNo;
    }
}
```

```
void remove (int valor) {
    try {
        removeChild (raiz, valor);
    } catch(Exception e) {
        Excecao("Valor não existente");
    }
}
```

Alocação Dinâmica

■ Atravessamento prefixo :

```
private void prefix(No raizSubArvore) {  
    se (raiz == nulo) {  
        exceção("Árvore vazia");  
    } senao {  
        imprime(raizSubArvore.dado);  
        imprime(" ");  
        se (raizSubArvore.esquerda != nulo) {  
            prefix(raizSubArvore.esquerda);  
        }  
        se (raizSubArvore.direita != nulo) {  
            prefix(raizSubArvore.direita);  
        }  
    }  
}
```

```
void prefixSearch() {  
    prefix(raiz);  
}
```


Alocação Dinâmica

■ Atravessamento infixo :

```
private void infix(No raizSubArvore) {  
    se (raiz == nulo) {  
        exceção("Árvore vazia");  
    } senao {  
        se (raizSubArvore.esquerda != nulo) {  
            infix(raizSubArvore.esquerda);  
        }  
        imprime(raizSubArvore.dado);  
        imprime(" ");  
        se (raizSubArvore.direita != nulo) {  
            infix(raizSubArvore.direita);  
        }  
    }  
}
```

```
void infixSearch() {  
    infix(raiz);  
}
```

Alocação Dinâmica

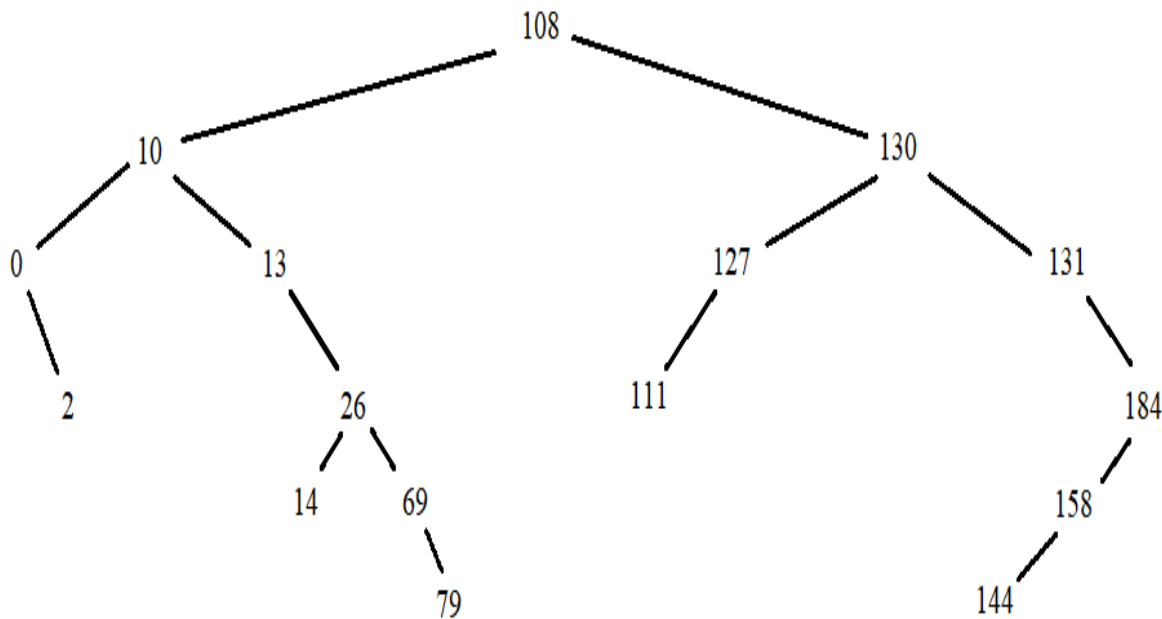
■ Atravessamento posfixo :

```
private void postfix(No raizSubArvore) {  
    se (raiz == nulo) {  
        exceção("Árvore vazia");  
    } senao {  
        se (raizSubArvore.esquerda != nulo) {  
            postfix(raizSubArvore.esquerda);  
        }  
        se (raizSubArvore.direita != nulo) {  
            postfix(raizSubArvore.direita);  
        }  
        imprime(raizSubArvore.dado);  
        imprime(" ");  
    }  
}
```

```
void postfixSearch() {  
    postfix(raiz);  
}
```

Exemplo

- Baseado no árvore de busca binária montada com:
{108, 130, 127, 10, 0, 13, 131, 184, 26, 2, 14, 158, 144, 69, 79, 111}



- Buscar o dado 158
- Fazer os atravessamentos:
 - Pré-ordem, Em ordem e Pós ordem
- Remover 144
- Remover 69
- Remover 127
- Remover 10
- Remover 108
- Fazer o atravessamento Em Ordem