Material Completo

Árvores Binárias
- Árvore Binária Completa
- Árvore Binária Balanceada
- Classe nó em Java
- Classe Árvore Binária em Java

Inserção em Árvores Binárias
- Inserção em Java com retorno de Referência
- Inserção em Java com Passagem de Pai
- Análise de Complexidade

Algorítmo de Pesquisa
- Funcionamento Básico
- Algorítmo de Pesquisa em Java
- Análise de Complexidade
- Caminhamento
- Alguns caminhamentos
  - Caminhamento Central ou Em Ordem
  - Caminhar pós fixado ou pós ordem
  - Caminhar pré-fixado ou pré ordem
  - Método para conseguir a altura de uma árvore

Remoção em Árvores Binárias
- Funcionamento Básico da Remoção
- Código em Java

Árvores Binárias

Árvore Binária Completa

- Cada nó é uma folha ou possui exatamente dois filhos.
- Todos os nós folhas possuem uma altura h
- O número de nós internos é (2^h) - 1
- O número de nós folhas é 2^h

- O número total de nós é ((2^h)-1) + (2^h) = 2^(h+1) - 1

> 💡 A RAÍZ SEMPRE ESTÁ NO h=0

Árvore Binária Balanceada

- A diferença de altura das subárvores da esquerda e da direita será sempre +-1

Classe nó em Java

```
class No{
    int elemento;
    No esq;
    No dir;
    No ( int elemento ) {
        this(elemento, null, null);
    }
    No(int elemento, No esq, No dir){
        this.elemento = elemento;
        this.esq = esq;
        this.dir = dir
    }
}
```

Classe Árvore Binária em Java

```
class ArvoreBinaria{
    No raiz;
    ArvoreBinaria() { raiz = null;}
```

```
    void inserir(int x) {}
    void inserirPai(int x){}
    boolean pesquisar(int x) {}
    void caminharCentral(){}
    void caminharPre(){}
    void caminharPos(){}
    void remover(int x){}
}
```

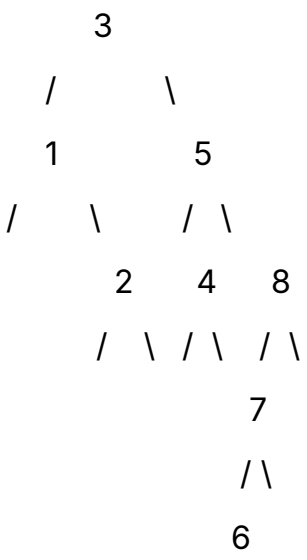
Inserção em Árvores Binárias

1 - Se a raíz estiver vazia, insere-se o elemento nela

2 - Senão, se o novo elemento for **menor** que o da raíz, chama recursivamente a inserção para a subárvore da **esquerda**

3 - Senão, se o novo elemento for **maior** que o da raíz, chama recursivamente a inserção para a subárvore da **direita**

4 - Senão, se o novo elemento for **igual** a raíz, não adicionar a árvore



Inserção em Java com retorno de Referência

```
void inserir ( int x ) throws Exception{
    raiz = inserir(x,raiz);
```

```
}

No inserir(int x, No i) throws Exception{
    if (i==null){
        i = new No(x);
    } else if(x < i.elemento ) {
        i.esq = inserir(x,i.esq);
    } else if(x > i.elemento) {
        i.dir = inserir(x,i.dir);
    } else{
        throw new Exception("Erro!");
    }
    return i;
}
```

> Nessa função, o algoritmo sempre no final das contas retornará o endereço do primeiro nó

Inserção em Java com Passagem de Pai

```
void inserirPai(int x) throws Exception {
    if(raiz == null){
        raiz = new No(x);
    } else if (x < raiz.elemento){
            inserirPai(x,raiz.esq,raiz);
    } else if(x > raiz.elemento){
            inserirPai(x,raiz.dir,raiz);
    } else{
        throw new Exception("Erro!");
    }
}

void inserirPai(int x, No i, No pai) throws Exception{
    if(i == null){
        if(x < pai.elemento)  {
```

```
            pai.esq = new No(x);
        } else{
            pai.dir = new No(x);
        }
    } else if (x < i.elemento) {
        inserirPai(x, i.esq, i);
    } else if(x > i.elemento){
        inserirPai(x, i.dir, i);
    }else{
        throw new Exception("Erro!");
    }
}
```

Análise de Complexidade

- Melhor caso: Θ(1) comparações
- Pior caso: Θ(n) comparações
- Caso médio: Θ(lg(n)) comparações

Algorítmo de Pesquisa

Funcionamento Básico

1- Se a raíz estiver vazia **retornar pesquisa negativa**

2- Senão, se o elemento procurado for **igual** ao da raíz, retornar **pesquisa positiva**

3- Senão, se o elemento procurado for **menor** que o da raíz , chamar o método de pesquisa para a subárvore da **esquerda**

4- Senão, se o elemento procurado for **maior**, chamar o método de pesquisa para a subárvore da **direita**

Algorítmo de Pesquisa em Java

```
boolean pesquisa(int x){
    return pesquisa(x, raiz);
}

boolean pesquisa(int x, No i){
    boolean resp;
    if(i == null){
        resp null;
    }else if (x == i.elemento){
        resp = true;
    }else if ( x < i.elemento){
        resp = pesquisar(x,i.esq);
    }else{
        resp = pesquisar(x,i.dir);
    }
    return resp;
}
```

Análise de Complexidade

- Melhor caso: Θ(1) comparações - Acontece na raíz
- Pior caso: Θ(n) comparações - Acontece quando o elemento está na folha(elementos em ordem)
- Caso médio: Θ(lg(n)) comparações - Acontece quando o elemento está na folha

Caminhamento

- Θ(n) visitas

Alguns caminhamentos

Caminhamento Central ou Em Ordem

- Não é coincidência ele caminhar em ordem crescente

```
void caminharCentral(No i){
    if(i != null){
        caminharCentral(i.esq);
        System.out.println(i.elemento + " ");
        caminharCentral(i.dir);
    }
}
```

Caminhar pós fixado ou pós ordem

```
void caminharPos(No i){
    if(i!= null){
        caminharPos(i.esq);
        caminharPos(i.dir);
        System.out.println(i.elemento + "");
    }
}
```

Caminhar pré-fixado ou pré ordem

```
void caminharPre(No i){
    if(i!=null){
        System.out.println(i.elemento + "");
        caminharPre(i.esq);
        caminharPre(i.dir);
    }
}
```

Método para conseguir a altura de uma árvore

```
int getAltura(No i, int altura){
    if(i == null){
```

```
            altura--;
        } else{
            int alturaEsq = getAltura(i.esq, altura+1);
            int alturaDir = getAltura(i.dir, altura+1);
            int altura = (alturaEsq > alturaDir) ? alturaEsq:altu
        }
        return altura;
    }
```

Remoção em Árvores Binárias

Funcionamento Básico da Remoção

1- Se o elemento estiver em uma folha, removê-la

2- Senão, se o elemento estiver em um nó intermediário com um filho, fazer com que seu pai aponte para o seu filho

3- Senão, se o elemento estiver em um nó intermediário com dois filhos, substituí-lo pelo maior nó da subárvore a esquerda **ou** substituí-lo pelo menor nó da subárvore a direita

Código em Java

```
void remover(int x) throws Exception{
 raiz = remover(x,raiz);
}

No remover(int x, No i) throws Exception{
if(i == null) {throw new Exception("Erro!");
}else if(x<i.elemento){ i.esq = remover(x,i.esq)
}else if(x>i.elemento){i.dir = remover(x,i.dir;
}else if(i.dir == null){ i = i.esq;
```

```
}else if(i.esq == null){ i= i.dir;
}else{ i.esq = maiorEsq(i,i.esq);}
return i;

No maiorEsq(No i, No j){
if(j.dir == null){i.elemento = j.elemento; j = j.esq}
else{j.dir = maiorEsq(j,j.dir);}
return j;
}
```