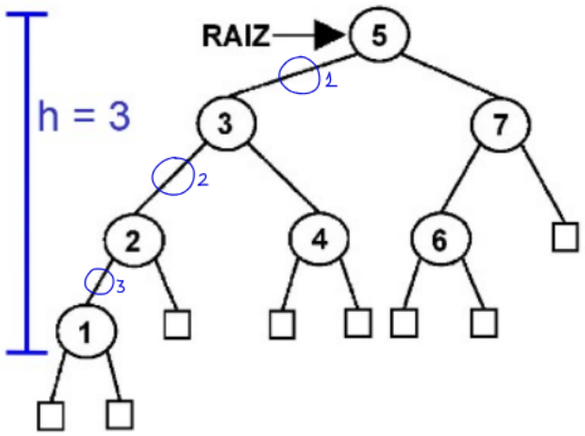
NOTEBOOK

_____

_____

Árvore Binaria

Custo de inserção, remoção e pesquisa pode ser O(lg(N)).



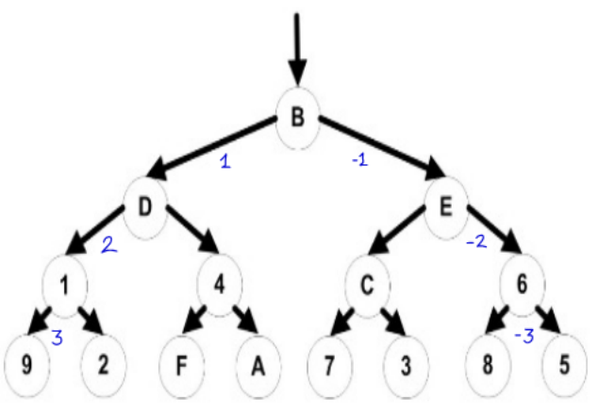
Altura: maior distância entre um nó e a raiz.
OBS: conte pelas arestas (ligações dos nós).

Para uma árvore binaria COMPLETA:

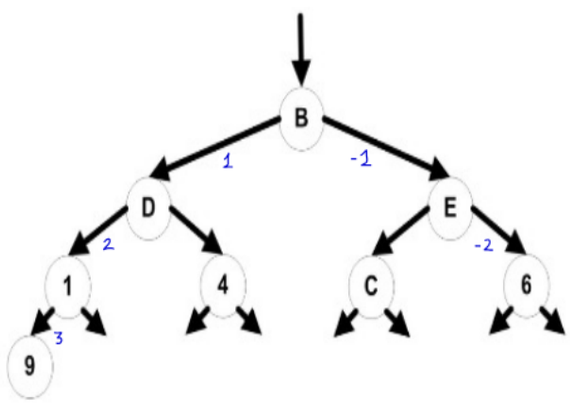
- todos os nós folhas possuem a mesma altura H.
- O número de nós internos é $2^H - 1$.
- O número de nós folhas é 2^H.

Árvore Balanceada

Árvore em que para TODOS os nós, a diferença entre a altura de suas árvores da esquerda e da direita será sempre de 0 ou +- 1.



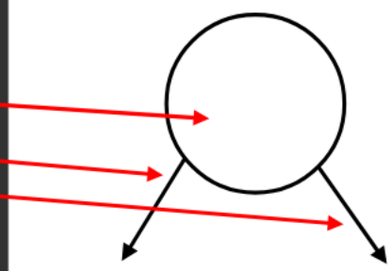
Diferença de : 3 - 3 = 0



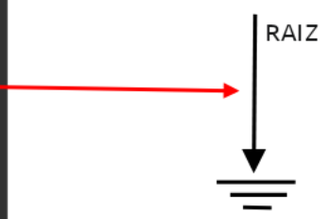
Diferença de : 3 - 2 = 1

Código fonte

```
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;

    }
}
```



```
class ArvoreBinaria{

    No raiz;

    ArvoreBinaria(){
        raiz = null;
    }

}
```



Inserção em Java

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

No inserir(int x, No i) throws Exception {

    if(i == null) { // se for null a raiz inserimos nela
        i = new No(x);
    } else if(x > i.elemento) { // se o elemento é maior, vamos para a direita
        i.dir = inserir(x, i.dir);
    } else if(x < i.elemento) { // se o elemento é menor, vamos para esquerda
        i.esq = inserir(x, i.esq);
    } else { //  se o elemento  É algo diferente de > || < , erro!
        throw new Exception("erro!");

    }

    return i;
}
```

Melhor caso: O(1) comparações, e acontece quando, por exemplo inserirmos na raiz.

Pior caso: O(N) comparações, e acontece quando, por exemplo inserirmos os elementos na ordem crescente ou decrescente.

Pesquisar em Java

```java
boolean pesquisar(int x) throws Exception {
    return pesquisar(x, raiz);
}

boolean pesquisar(int x, No i) throws Exception {

    boolean resp = false;
    if(i == null) { // se for null, arvore vazia
        resp = false;
    }else if( i.elemento == x){ // se for igual, achamos o numero desejado
      resp = true;
    } else if(x > i.elemento) { // se o elemento é maior, vamos para a direita
        i.dir = pesquisar(x, i.dir);
    } else if(x < i.elemento) { // se o elemento é menor, vamos para esquerda
        i.esq = pesquisar(x, i.esq);
    }

    return resp;
}
```

Complexidade a mesma de inserção

Caminhamentos

- Caminhamento central

Caminhar para esquerda
Mostrar elemento
Caminhar para a direita

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

- Caminhar em pre-ordem

Mostrar elemento
Caminhar para esquerda
Caminhar para a direita

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

- Caminhar pós-ordem

Caminhar para esquerda
Caminhar para a direita
Mostrar elemento

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