

Onde estamos ...

1 Árvores

Capítulo 06 – Árvores

Pontos fundamentais a serem cobertos:
Pontos fundamentais

a serem cobertos:

- 1 Contexto e motivação
- 2 Definição
- 3 Implementações
- 4 Exercícios

Definição

- Uma árvore é uma estrutura hierárquica composta por nós e ligações entre eles
- Pode ser vista como um grafo acíclico
- Cada nó possui somente um pai e zero ou mais filhos

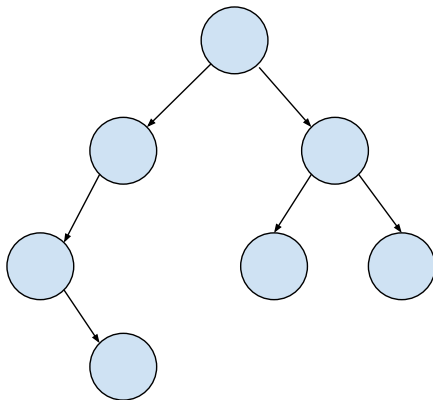


Figura 1: Exemplo de uma árvore

Árvore Binária de Busca - Definição

Árvore onde cada nó possui até 2 filhos. O filho da esquerda só pode conter chaves menores do que a do pai, enquanto que o filho da direita só comporta chaves maiores do que a do pai.

Árvore Binária de Busca

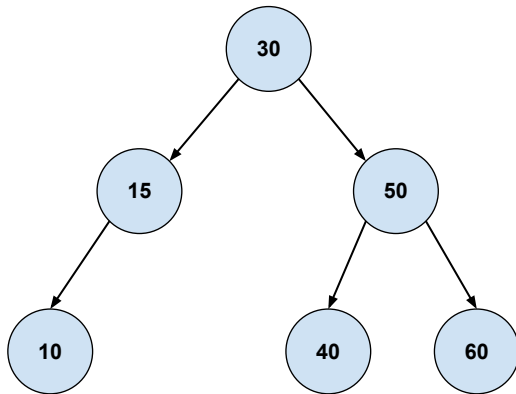
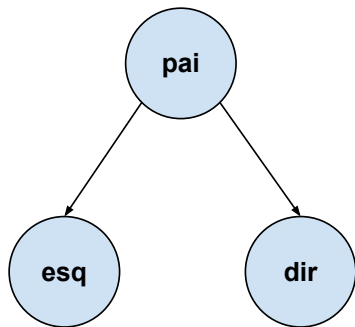


Figura 2: Exemplo de árvore binária de busca

Árvore Binária de Busca



```
struct ArvoreBinaria {  
    struct ArvoreBinaria* esq;  
    struct ArvoreBinaria* dir;
```

```
    // contém a chave e os dados satélite  
    Tipoltem valor;  
};
```

Figura 3: Estrutura básica / nó

Operações Básicas

- Inserção
- Busca
- Remoção

Usos Comuns

- Dicionários / vetores associativos
- Filas de prioridades

Quando a árvore está balanceada todas as três operações podem ser implementadas com complexidade computacional igual a $O(\log n)$.

No pior caso (desbalanceamento) estas operações possuem complexidade $O(n)$ [?].

Árvore Binária de Busca - Inserção

```
INSERÇÃO(ARVORE, ITEM) {  
    SE ITEM->CHAVE = ARVORE->CHAVE  
        ARVORE->ITEM = ITEM  
        return  
  
    SE ITEM->CHAVE < ARVORE->CHAVE  
        SE ARVORE->ESQ = NULO ENTÃO  
            ARVORE->ESQ = ARVORE(ITEM)  
        SENÃO  
            INSERÇÃO(ARVORE->ESQ, ITEM)  
    SENÃO  
        SE ARVORE->DIR = NULO ENTÃO  
            ARVORE->DIR = ARVORE(ITEM)  
        SENÃO  
            INSERÇÃO(ARVORE->DIR, ITEM)  
}
```

Árvore Binária de Busca - Inserção

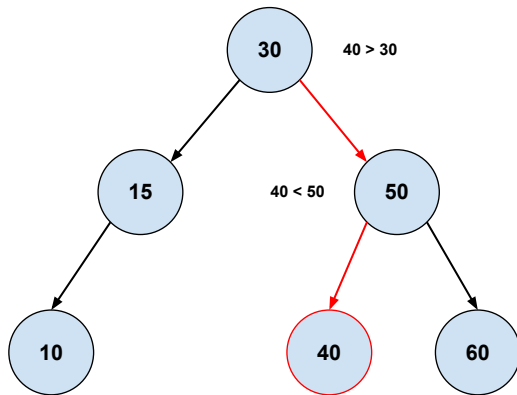


Figura 4: Exemplo de inserção da chave 40

Árvore Binária de Busca - Inserção

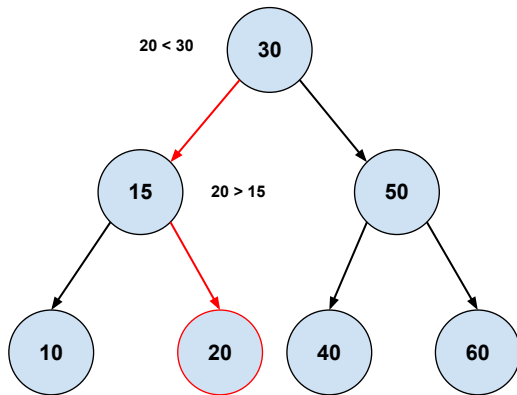


Figura 5: Exemplo de inserção da chave 20

Árvore Binária de Busca - Busca

```
BUSCA(ARVORE, CHAVE) {  
    SE ARVORE = NULO  
        return NULO  
  
    SE ARVORE->CHAVE = CHAVE  
        return ARVORE  
  
    SE CHAVE < ARVORE->CHAVE  
        return BUSCA(ARVORE->ESQ, CHAVE)  
    SENÃO  
        return BUSCA(ARVORE->DIR, CHAVE)  
}
```

Árvore Binária de Busca - Remoção

A remoção de um nó se enquadra em um dos seguintes casos:

- ① Remoção de um nó folha (nenhum filho)
- ② Remoção de um nó com somente um filho
- ③ Remoção de um nó com dois filhos

O tratamento de cada caso foi apresentado em sala de aula.

Onde estamos ...

1 Árvores

Uma árvore binária de busca balanceada garante operações de busca, inserção e remoção com complexidade $O(\log n)$, onde n é o número de nós, o que a torna atrativa para diversas aplicações.

Determinadas sequências de inserções ou remoções podem fazer com que uma ABB fique desbalanceada, tornando suas operações $O(n)$.

Cálculo da Altura

```
ALTURA(ARVORE) {  
    SE ARVORE = NULO  
        return -1  
  
    A1 = ALTURA(ARVORE->DIR)  
    A2 = ALTURA(ARVORE->ESQ)  
  
    return maior(A1, A2) + 1  
}
```

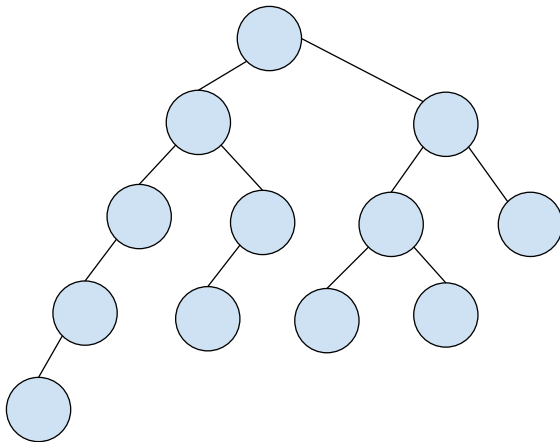


Figura 6: Exercício: determine a altura de cada subárvore.

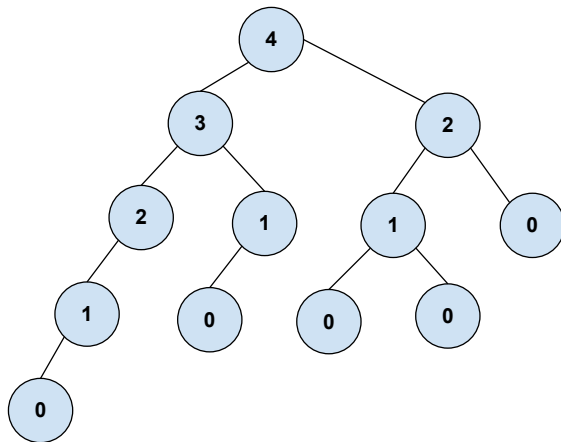


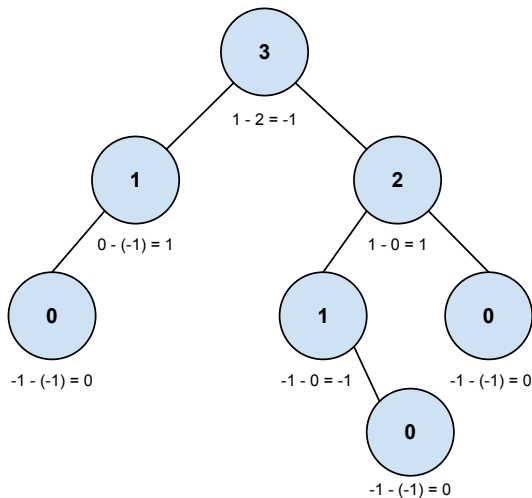
Figura 7: Resposta do exercício.

Cálculo do Fator de Balanceamento

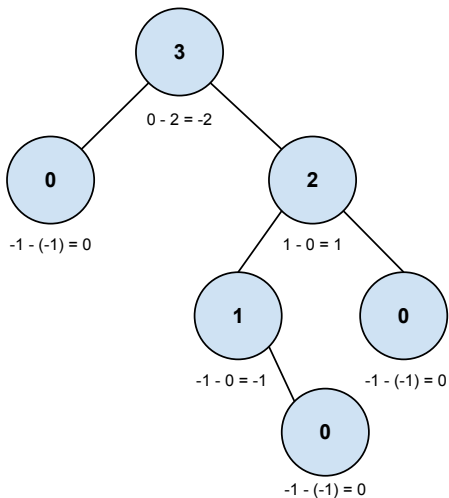
```
FB(ARVORE) {  
    A1 = ALTURA(ARVORE->ESQ)  
    A2 = ALTURA(ARVORE->DIR)  
    return A1 - A2  
}
```

- Uma ABB está balanceada quando cada nó possui um FB igual a -1, 0 ou 1
- Uma inserção ou remoção pode tornar uma árvore desbalanceada, necessitando de rotações para o seu balanceamento

Exemplo de ABB Balanceada



Exemplo de ABB Desbalanceada

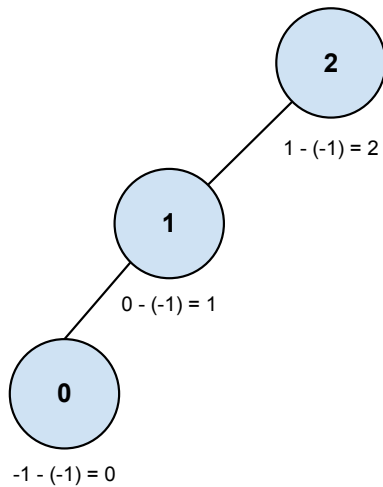


Operação de rotação

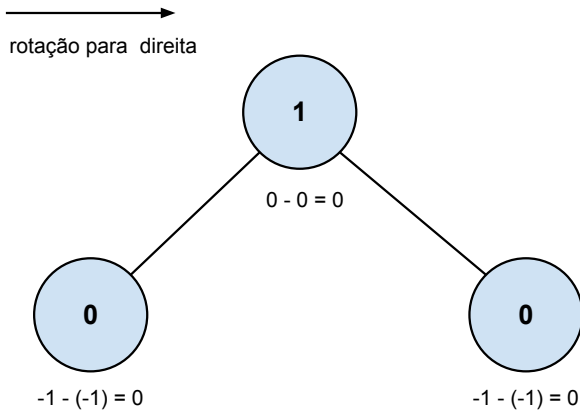
```
ROTACAO_DIREITA(RAIZ) {  
    PIVO      = RAIZ->ESQ  
    RAIZ->ESQ = PIVO->DIR  
    PIVO->DIR = RAIZ  
    RAIZ      = PIVO  
}
```

```
ROTACAO_ESQUERDA(RAIZ) {  
    PIVO      = RAIZ->DIR  
    RAIZ->DIR = PIVO->ESQ  
    PIVO->ESQ = RAIZ  
    RAIZ      = PIVO  
}
```

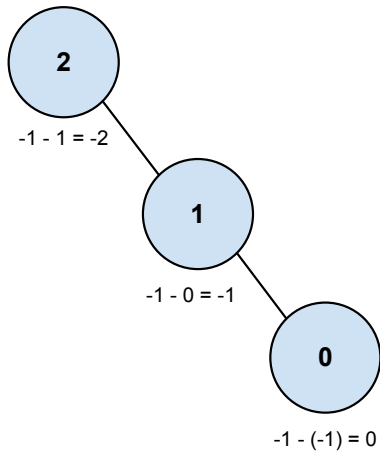

Rotação para Direita



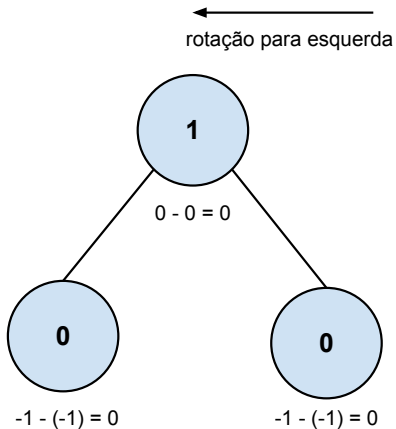
Rotação para Direita



Rotação para Esquerda



Rotação para Esquerda



- **AVL** desenvolvida por G. M. **Adelson-Velskii** and E. M. **Landis**
- Garante o balanceamento da árvore ao realizar rotações após cada inserção ou remoção na ABB

Balanceamento - Inserção

```
BALANCEAMENTO(RAIZ) {  
    SE FB(RAIZ) = -2 ENTÃO  
        SE FB(RAIZ->DIR) = -1 ENTÃO  
            ROTACAO_ESQUERDA(RAIZ)  
        SENÃO  
            ROTACAO_DIREITA(RAIZ->DIR)  
            ROTACAO_ESQUERDA(RAIZ)  
    SENÃO SE FB(RAIZ) = 2 ENTÃO  
        SE FB(RAIZ->ESQ) = 1 ENTÃO  
            ROTACAO_DIREITA(RAIZ)  
        SENÃO  
            ROTACAO_ESQUERDA(RAIZ->DIR)  
            ROTACAO_DIREITA(RAIZ)  
}
```

Balanceamento - Inserção

- Para que a árvore tenha um bom desempenho, é essencial que o balanceamento seja calculado eficientemente, isto é, sem a necessidade de percorrer toda a árvore após cada modificação
- Manter a árvore estritamente balanceada após cada modificação tem seu preço (desempenho). Árvores AVL são utilizadas normalmente onde o número de consultas é muito maior do que o número de inserções e remoções e quando a localidade de informação não é importante

Árvore de Espalhamento

- Reestrutura a árvore em cada operação de inserção, busca ou remoção por meio de operações de rotação
- Nome original: *splay tree* [?]. Não confundir com a Árvore N-Ária de Espalhamento (ANE) criada por professores da UDESC

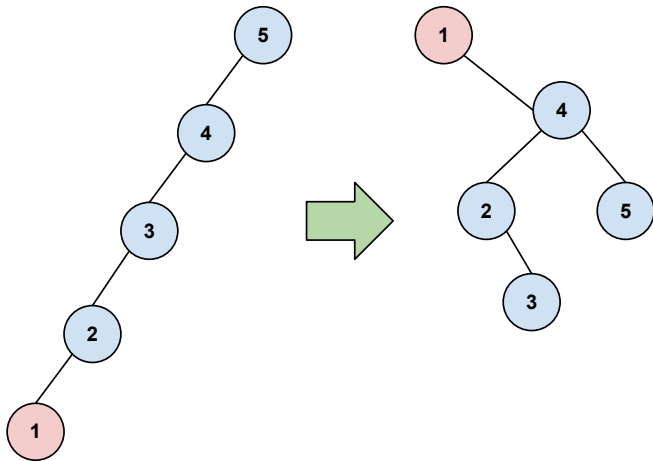
Árvore de Espalhamento

- Evita a repetição de casos ruins [$O(n)$] devido ao seu rebalanceamento natural
- Não realiza o cálculo de fatores de balanceamento, simplificando sua implementação
- Pior caso para uma operação se mantém $O(n)$, mas, ao considerar uma cadeia de operações, *garante* uma complexidade amortizada de $O(\log n)$ para suas operações básicas

Árvore de Espalhamento

- Se baseia na operação de espalhamento, que utiliza rotações para mover uma determinada chave até a raiz
- A sua complexidade $O(\log n)$ em uma análise amortizada é garantida pelas rotações efetuadas, o que a difere do uso simples de heurísticas como o *mover para a raiz*

Exemplo - Espalhamento pela chave 1



Espalhamento Move a chave desejada para a raiz por uma sequência bem definida de operações de rotação

Busca Busca uma chave na árvore

Inserção Insere uma nova chave na árvore

Remoção Remove uma chave da árvore

- Uma árvore de espalhamento é uma árvore binária de busca válida, logo operações como os percursos (pré-em-pós) são idênticas as operações em uma ABB
- As operações de inserção, busca e remoção podem ser definidas com base na operação de espalhamento

Árvore de Espalhamento - Busca

```
BUSCA(RAIZ, CHAVE) {  
    return ESPALHAMENTO(RAIZ, CHAVE)  
}
```

Árvore de Espalhamento - Inserção

```
INSERE(RAIZ, CHAVE) {  
    INSERE_ABB(RAIZ, CHAVE)  
    return ESPALHAMENTO(RAIZ, CHAVE)  
}
```

Árvore de Espalhamento - Remoção

```
REMOVE(RAIZ, CHAVE) {  
    RAIZ = ESPALHAMENTO(RAIZ, CHAVE)  
  
    SE RAIZ->DIR ENTÃO  
        AUX = ESPALHAMENTO(RAIZ->DIR, CHAVE)  
        AUX->ESQ = RAIZ->ESQ  
    SENÃO  
        AUX = RAIZ->ESQ  
  
    return AUX  
}
```


Estratégias de Espalhamento

Duas estratégias:

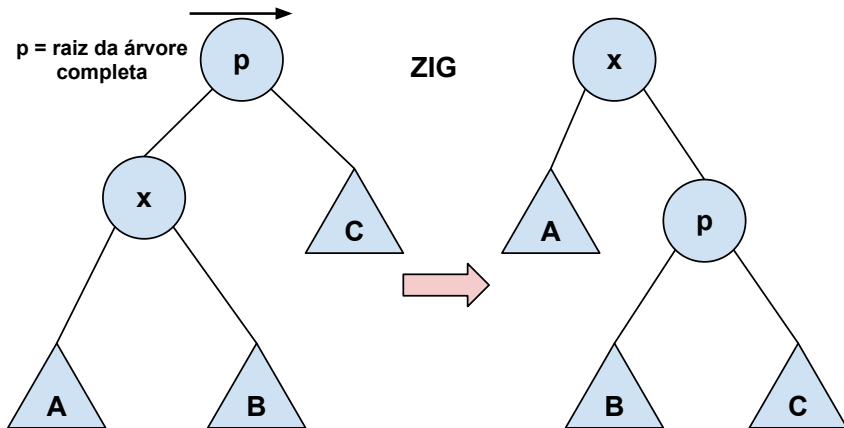
Bottom-Up Parte do nó acessado e o movimenta para a raiz da árvore por meio de rotações

Top-Down Parte do nó raiz, rotacionando e *removendo do caminho* os nós entre a raiz e o nó desejado, armazenando-os em duas árvores auxiliares, remontando a árvore completa na sua etapa final.

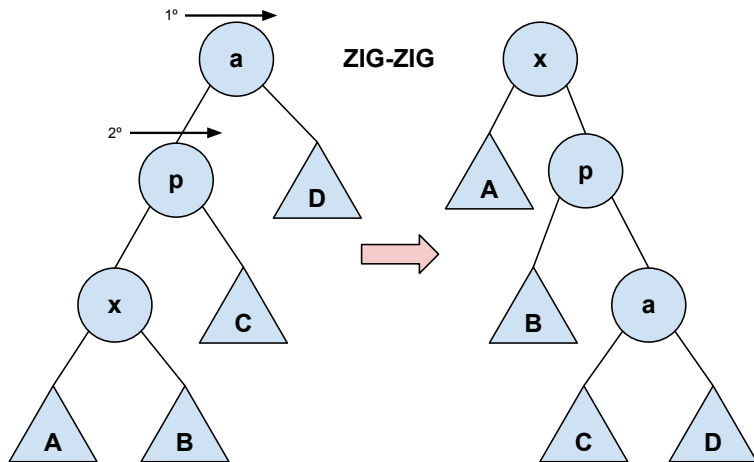
Espalhamento Bottom-Up

- Na estratégia Bottom-Up, a operação de espalhamento realiza rotações subindo gradativamente de níveis, a partir da chave desejada
- Enquanto a chave não estiver na raiz, deve-se verificar qual o caso aplicável (ZIG, ZIG-ZIG ou ZIG-ZAG) e realizar as rotações necessárias

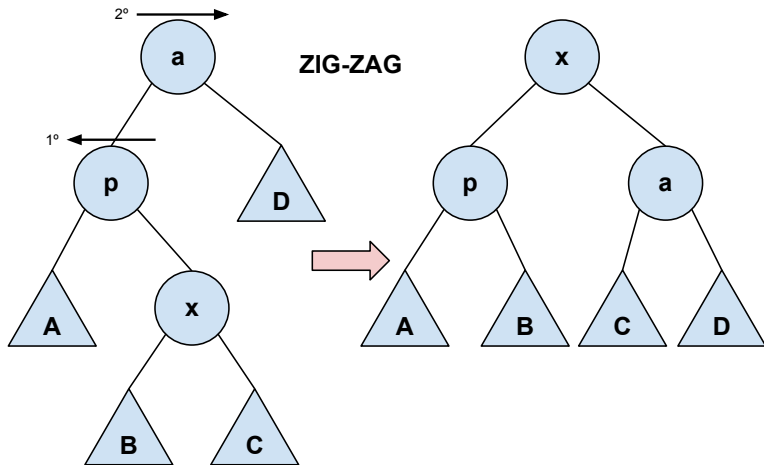
Caso 1: ZIG



Caso 2: ZIG-ZIG



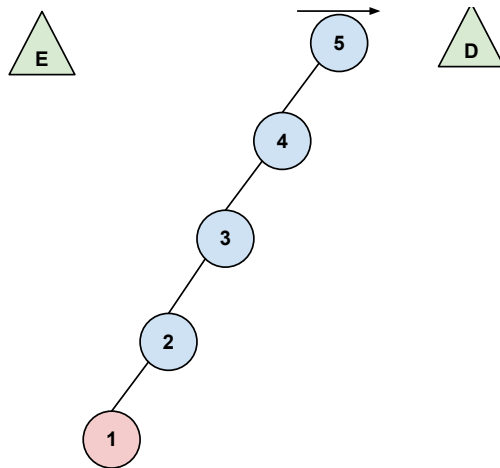
Caso 3: ZIG-ZAG



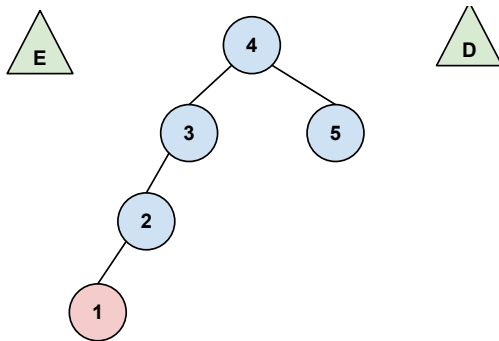
Espalhamento Top-Down

- Na estratégia Top-Down as chaves que estão no caminho da chave desejada para a raiz são rotacionadas e removidas para árvores auxiliares seguindo uma sequência de operações bem definidas
- Quando a chave desejada chega até a raiz, a árvore é remontada pelo retorno das chaves removidas

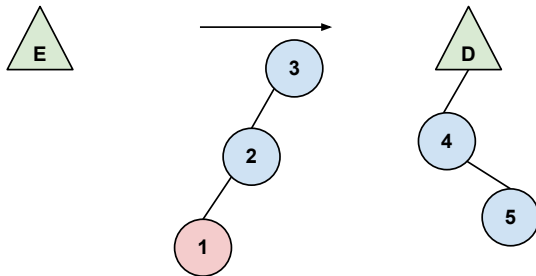
Exemplo: Top-Down 1/6



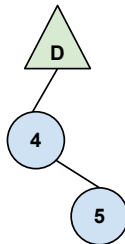
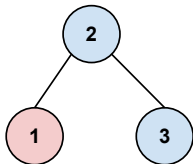
Exemplo: Top-Down 2/6



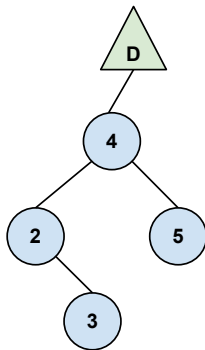
Exemplo: Top-Down 3/6



Exemplo: Top-Down 4/6



Exemplo: Top-Down 5/6



Exemplo: Top-Down 6/6

