Estruturas de dados II

Árvores AVL

André Tavares da Silva

andre.silva@udesc.br

UDESC

Árvores binárias

- Balanceamento de árvores
  - Árvore degenerada
    - Ocorre quando cada nível da árvore apresenta um único nó
    - Neste caso a árvore representa uma estrutura linear
    - Uma busca nesta árvore corresponde a complexidade $O(n)$
  - Árvore cheia ou completa
    - Uma árvore cheia possui um balanceamento perfeito
    - Todos os nós possuem dois filhos, exceto os nós folhas
    - Uma busca nesta árvore corresponde a complexidade $O(\log n)$

UDESC

Árvores binárias

- Benefícios de uma árvore balanceada
  - Minimizam o número de comparações efetuadas no pior caso

- Para garantir esta propriedade, é necessário
  - Reconstruir a árvore em seu estado ideal a cada alteração de estrutura
    - Operações de inclusão, exclusão ou alteração (quando envolver a chave)
  - Para cada operação acima, pode ser necessário um balanceamento
    - Balanceamento consiste em reequilibrar a árvore em suas dimensões
    - Isto é, largura e altura

UDESC

Árvores AVL

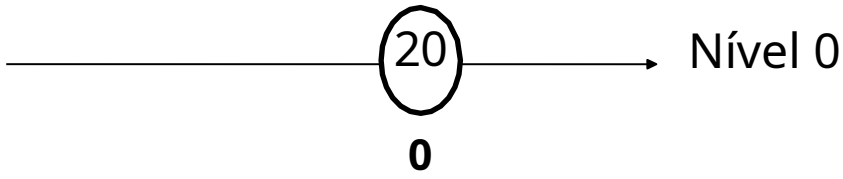
- Árvore AVL é uma árvore binária altamente balanceada
  - Introduzida por Adelson-Velskii e Landis, 1962
  - A cada inserção ou exclusão, é executada uma rotina de balanceamento
  - Visam manter a altura da subárvores à esquerda e à direita equilibradas

  - Formalmente
    - Alturas das subárvores à esq. e dir. diferem-se, no máximo, em uma unidade
    - Para este cálculo, usa-se o fator de balanceamento (FB)

FB = **altura da subárvore esquerda** – **altura da subárvore direita**

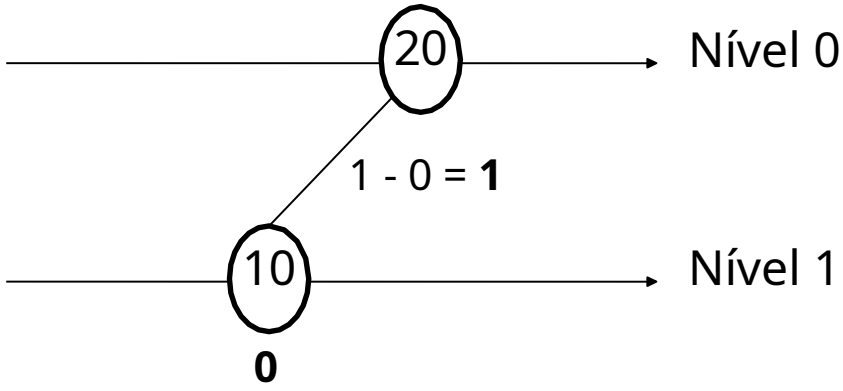
Árvores AVL

- Exemplo
  - 20



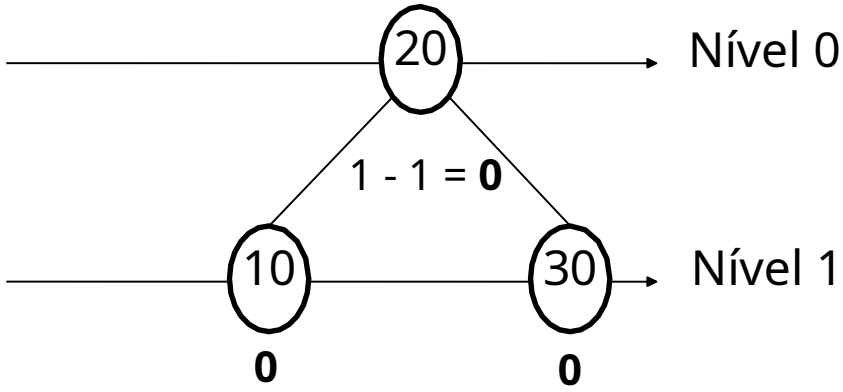
Árvores AVL

- Exemplo
  - 20
  - 10



Árvores AVL

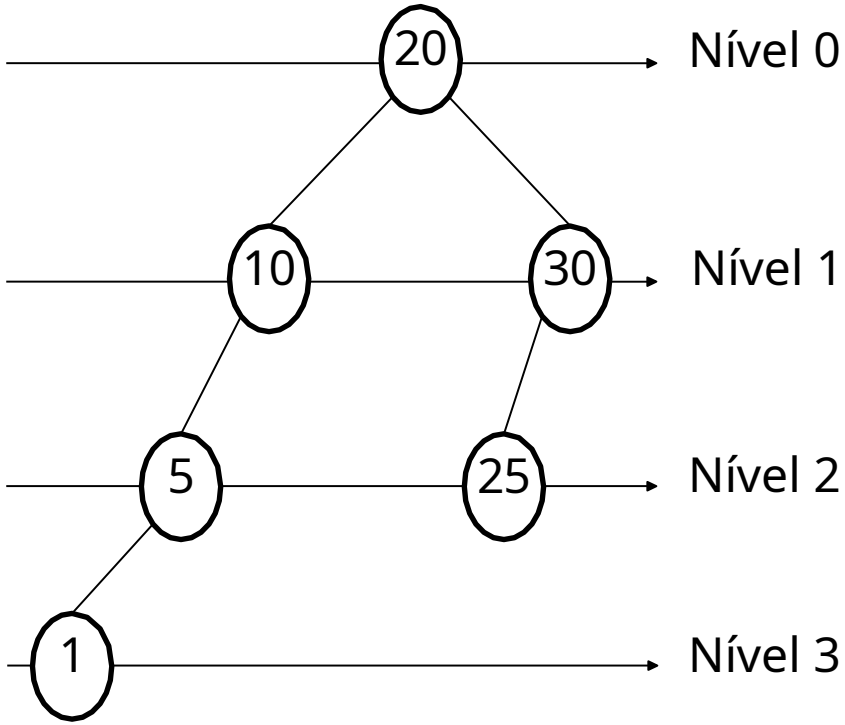
- Exemplo
  - 20
  - 10
  - 30



UDESC

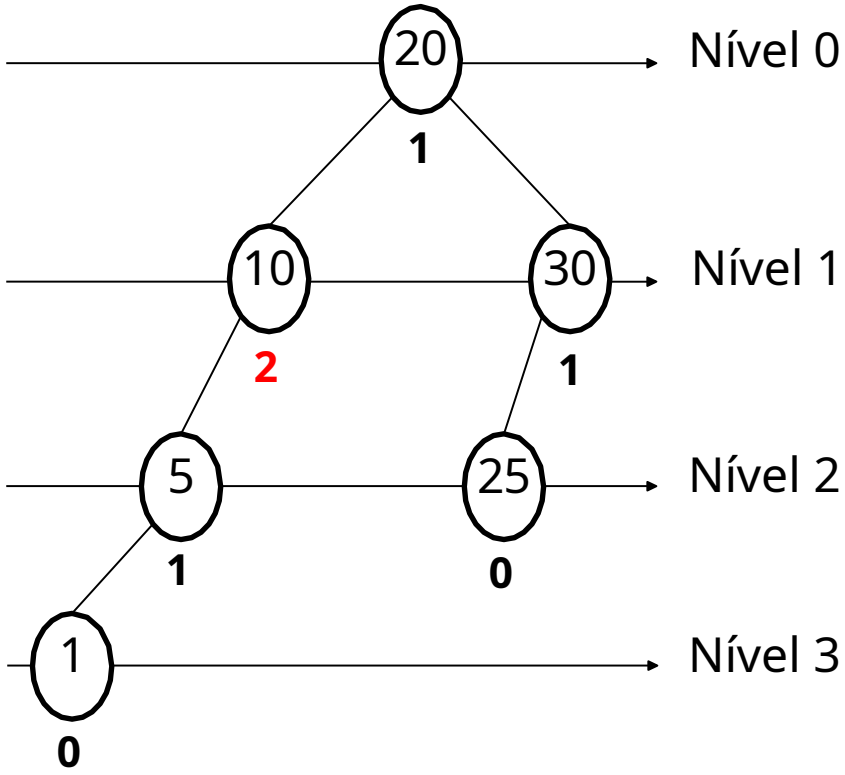
Árvores AVL

- Exemplo
  - 20
  - 10
  - 30
  - 5
  - 25
  - 1



Árvores AVL

- Exemplo
  - 20
  - 10
  - 30
  - 5
  - 25
  - 1



Operações de rotação

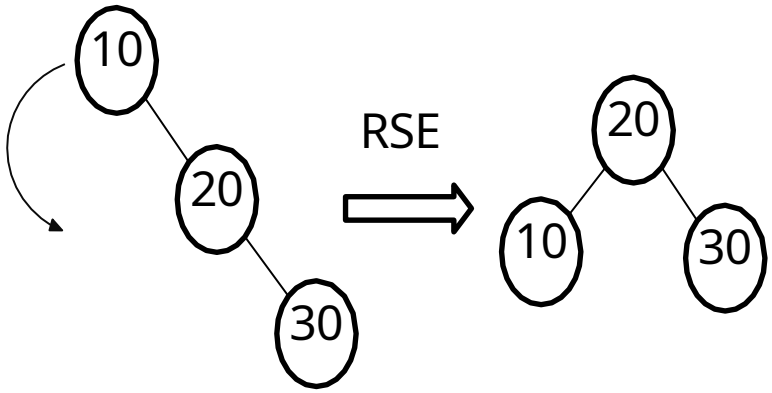
- Quando a árvore está desbalanceada, é possível realizar as seguintes operações para balanceá-la novamente
  - Rotação simples
    - Rotação simples à esquerda (RSE) ou rotação LL
    - Rotação simples à direita (RSD) ou rotação RR
  - Rotação dupla
    - Rotação dupla à esquerda (RDE) ou rotação RL
    - Rotação dupla à direita (RDD) ou rotação LR

- Quando o FB resultar em um valor diferente de 0, -1 ou 1, deverá ser realizada uma das operações de balanceamento

Operações de rotação

- Rotação simples à esquerda (RSE) ou rotação LL
  - Neste caso a subárvore está mais alta (profunda) no lado direito
  - Situação para realização de uma RSE
    - FB do nó desbalanceado for negativo e;
    - Nó à direita também apresentar FB negativo

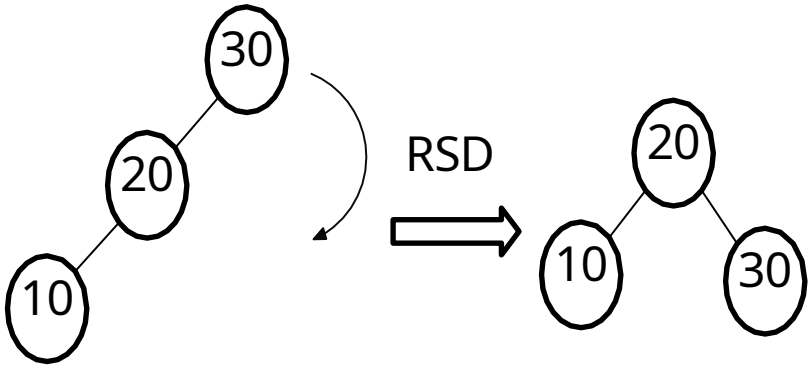
- Exemplo: RSE(10,20)
  - Nó 10 passa a ser filho de 20
  - Nó 20 passa a ser pai de 10 e 30



Operações de rotação

- Rotação simples à direita (RSD) ou rotação RR
  - Neste caso a subárvore está mais alta (profunda) no lado esquerdo
  - Situação para realização de uma RSD
    - FB do nó desbalanceado for positivo e;
    - Nó à esquerda também apresentar FB positivo

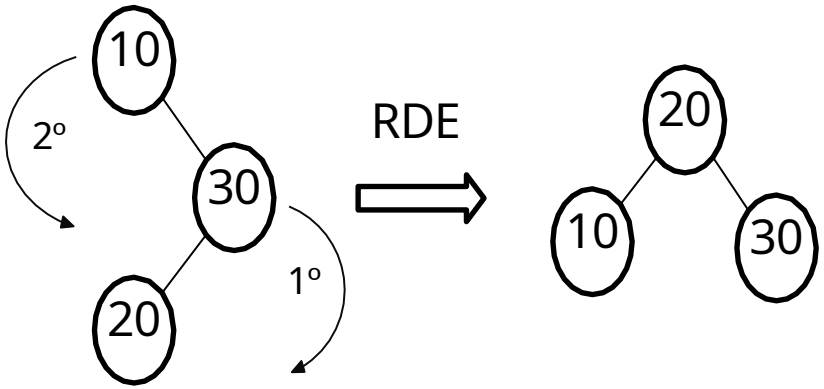
- Exemplo: RSD(30,20)
  - Nó 30 passa a ser filho de 20
  - Nó 20 passa a ser pai de 10 e 30



Operações de rotação

- Rotação dupla à esquerda (RDE) ou rotação RL
  - Neste caso a subárvore está mais alta (profunda) no lado direito
  - Situação para realização de uma RDE
    - FB do nó desbalanceado for negativo e;
    - Nó a direita apresentar FB positivo

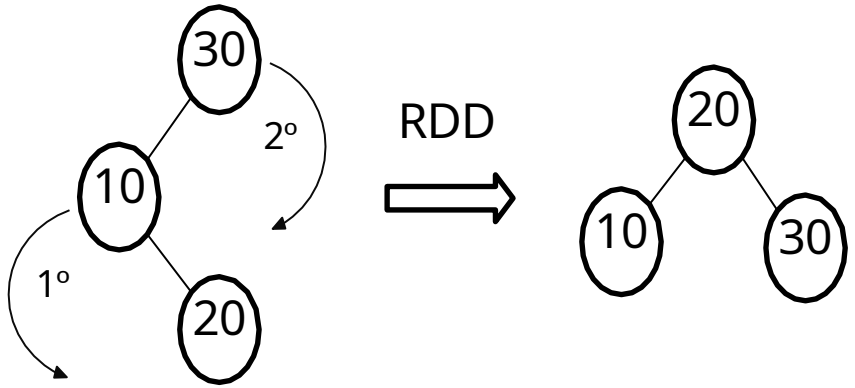
- Exemplo: RDE(10,30)
  - RSD(30,20)
  - RSE(10,20)



Operações de rotação

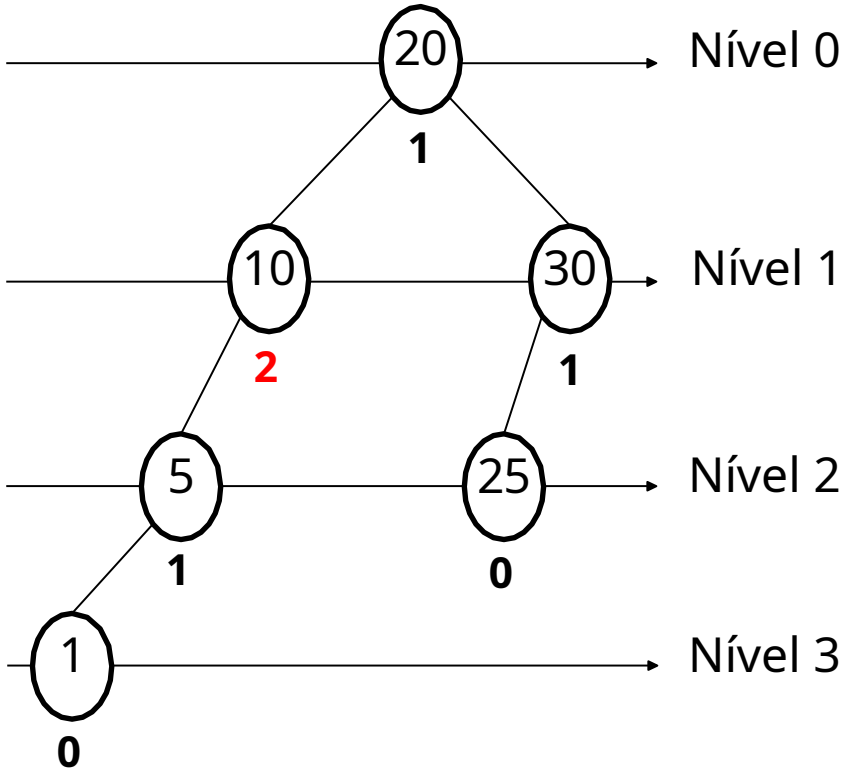
- Rotação dupla à direita (RDD) ou rotação LR
  - Neste caso a subárvore está mais alta (profunda) no lado esquerdo
  - Situação para realização de uma RDD
    - FB do nó desbalanceado for positivo e;
    - Nó a direita apresentar FB negativo

- Exemplo: RDD(30,10)
  - RSE(10,20)
  - RSD(30,20)



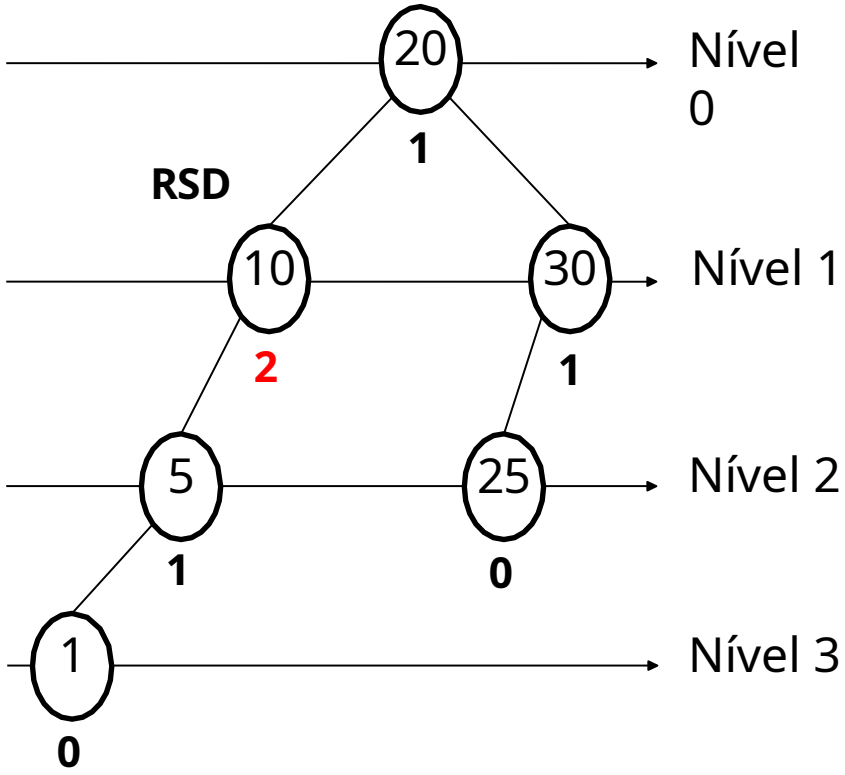
Árvores AVL

- Exemplo
  - 20
  - 10
  - 30
  - 5
  - 25
  - 1



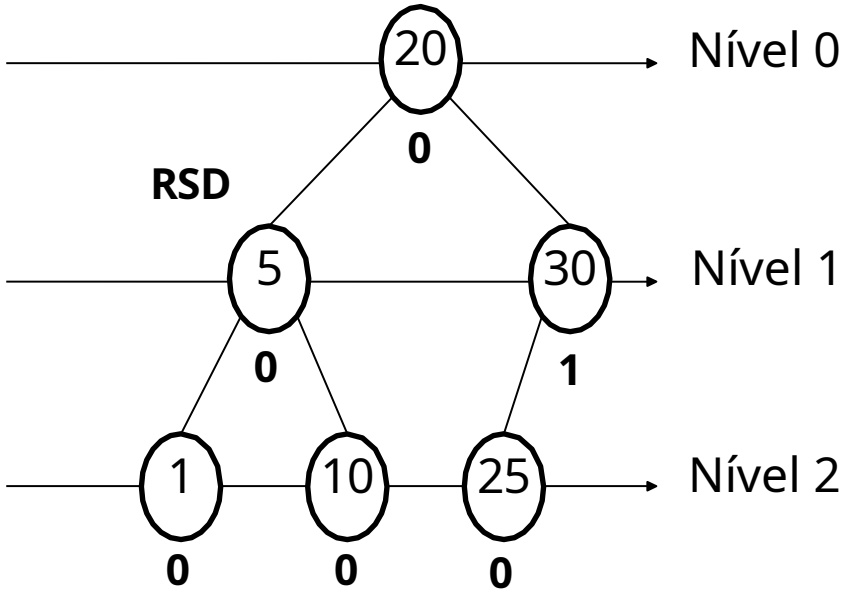
Árvores AVL

- Exemplo
  - 20
  - 10
  - 30
  - 5
  - 25
  - 1



Árvores AVL

- Exemplo
  - 20
  - 10
  - 30
  - 5
  - 25
  - 1



UDESC

Árvores AVL

- Altura do nó

```
int altura(No* no){
  int esquerda=0, direita=0;

  if (no->esquerda != NULL)
    esquerda = altura(no->esquerda) + 1;

  if (no->direita != NULL)
    direita = altura(no->direita) + 1;

  return esquerda > direita ? esquerda : direita;
}
```

UDESC

Árvores AVL

- Fator de balanceamento (FB)

```
int fb(No* no) {
  int esquerda=0, direita=0;

  if (no->esquerda != NULL)
    esquerda = altura(no->esquerda) + 1;

  if (no->direita != NULL)
    direita = altura(no->direita) + 1;

  return esquerda - direita;
}
```

UDESC

Árvores AVL

- Rotação simples à esquerda (RSE)

```
No* rse(No* no) {
  No* pai = no->pai;
  No* direita = no->direita;

  no->direita = direita->esquerda;
  no->pai = direita;

  direita->esquerda = no;
  direita->pai = pai;

  return direita;
}
```

Árvores AVL

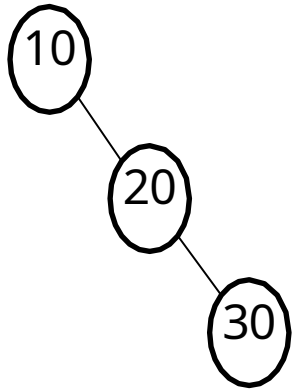
- Rotação simples à esquerda (RSE)

```
No* rse(No* no) {
  No* pai = no->pai;
  No* direita = no->direita;

  no->direita = direita->esquerda;
  no->pai = direita;

  direita->esquerda = no;
  direita->pai = pai;

  return direita;
}
```



- pai: NULL
- no: 10
  - pai: NULL
  - esquerda: NULL
  - direita: 20
- direita: 20
  - pai: 10
  - esquerda: NULL
  - direita: 30

Árvores AVL

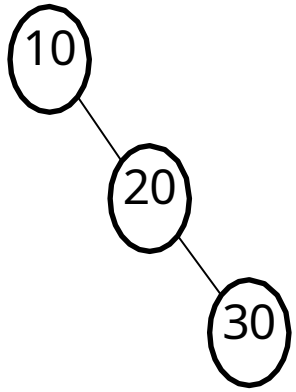
- Rotação simples à esquerda (RSE)

```
No* rse(No* no) {
  No* pai = no->pai;
  No* direita = no->direita;

  no->direita = direita->esquerda;
  no->pai = direita;

  direita->esquerda = no;
  direita->pai = pai;

  return direita;
}
```



- pai: NULL
- no: 10
  - pai: NULL
  - esquerda: NULL
  - **direita: NULL**
- direita: 20
  - pai: 10
  - esquerda: NULL
  - direita: 30

Árvores AVL

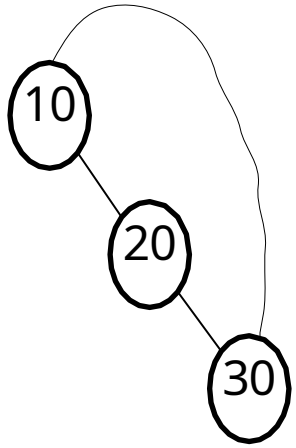
- Rotação simples à esquerda (RSE)

```
No* rse(No* no) {
  No* pai = no->pai;
  No* direita = no->direita;

  no->direita = direita->esquerda;
  no->pai = direita;

  direita->esquerda = no;
  direita->pai = pai;

  return direita;
}
```



- pai: NULL
- no: 10
  - **pai: 30**
  - esquerda: NULL
  - direita: NULL
- direita: 20
  - pai: 10
  - esquerda: NULL
  - direita: 30

Árvores AVL

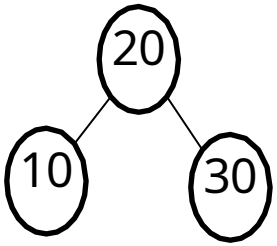
- Rotação simples à esquerda (RSE)

```
No* rse(No* no) {
  No* pai = no->pai;
  No* direita = no->direita;

  no->direita = direita->esquerda;
  no->pai = direita;

  direita->esquerda = no;
  direita->pai = pai;

  return direita;
}
```



- pai: NULL
- no: 10
  - pai: 30
  - esquerda: NULL
  - direita: NULL
- direita: 20
  - pai: 10
  - **esquerda: 10**
  - direita: 30

Árvores AVL

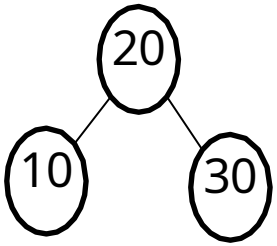
- Rotação simples à esquerda (RSE)

```
No* rse(No* no) {
  No* pai = no->pai;
  No* direita = no->direita;

  no->direita = direita->esquerda;
  no->pai = direita;

  direita->esquerda = no;
  direita->pai = pai;

  return direita;
}
```



- pai: NULL
- no: 10
  - pai: 30
  - esquerda: NULL
  - direita: NULL
- direita: 20
  - **pai: NULL**
  - esquerda: 10
  - direita: 30

Árvores AVL

- Rotação simples à direita (RSD)

```
No* rsd(No* no) {
    No* pai = no->pai;
    No* esquerda = no->esquerda;

    no->esquerda = esquerda->direita;
    no->pai = esquerda;

    esquerda->direita = no;
    esquerda->pai = pai;

    return esquerda;
}
```

Árvores AVL

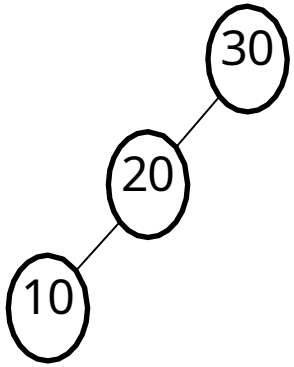
- Rotação simples à direita (RSD)

```
No* rsd(No* no) {
    No* pai = no->pai;
    No* esquerda = no->esquerda;

    no->esquerda = esquerda->direita;
    no->pai = esquerda;

    esquerda->direita = no;
    esquerda->pai = pai;

    return esquerda;
}
```



- pai: NULL
- no: 30
  - pai: NULL
  - esquerda: 20
  - direita: NULL
- esquerda: 20
  - pai: 30
  - esquerda: 10
  - direita: NULL

Árvores AVL

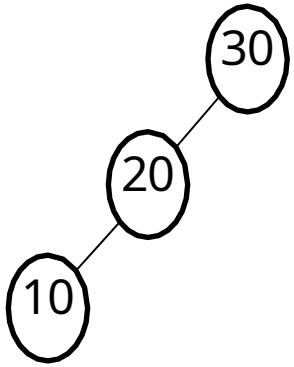
- Rotação simples à direita (RSD)

```
No* rsd(No* no) {
    No* pai = no->pai;
    No* esquerda = no->esquerda;

    no->esquerda = esquerda->direita;
    no->pai = esquerda;

    esquerda->direita = no;
    esquerda->pai = pai;

    return esquerda;
}
```



- pai: NULL
- no: 30
  - pai: NULL
  - **esquerda: NULL**
  - direita: NULL
- esquerda: 20
  - pai: 30
  - esquerda: 10
  - direita: NULL

Árvores AVL

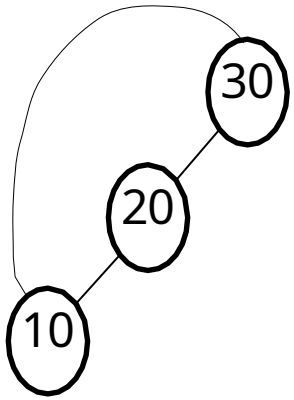
- Rotação simples à direita (RSD)

```
No* rsd(No* no) {
    No* pai = no->pai;
    No* esquerda = no->esquerda;

    no->esquerda = esquerda->direita;
    no->pai = esquerda;

    esquerda->direita = no;
    esquerda->pai = pai;

    return esquerda;
}
```



- pai: NULL
- no: 30
  - **pai: 20**
  - esquerda: NULL
  - direita: NULL
- esquerda: 20
  - pai: 30
  - esquerda: 10
  - direita: NULL

Árvores AVL

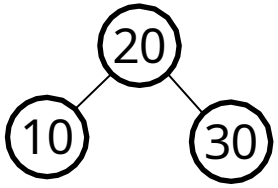
- Rotação simples à direita (RSD)

```
No* rsd(No* no) {
    No* pai = no->pai;
    No* esquerda = no->esquerda;

    no->esquerda = esquerda->direita;
    no->pai = esquerda;

    esquerda->direita = no;
    esquerda->pai = pai;

    return esquerda;
}
```



- pai: NULL
- no: 30
  - pai: 20
  - esquerda: NULL
  - direita: NULL
- esquerda: 20
  - pai: 30
  - esquerda: 10
  - **direita: 30**

Árvores AVL

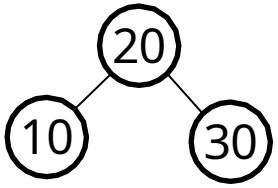
- Rotação simples à direita (RSD)

```
No* rsd(No* no) {
    No* pai = no->pai;
    No* esquerda = no->esquerda;

    no->esquerda = esquerda->direita;
    no->pai = esquerda;

    esquerda->direita = no;
    esquerda->pai = pai;

    return esquerda;
}
```



- pai: NULL
- no: 30
  - pai: 20
  - esquerda: NULL
  - direita: NULL
- esquerda: 20
  - **pai: NULL**
  - esquerda: 10
  - direita: 30

Árvores AVL

- Rotação dupla à esquerda (RDE)

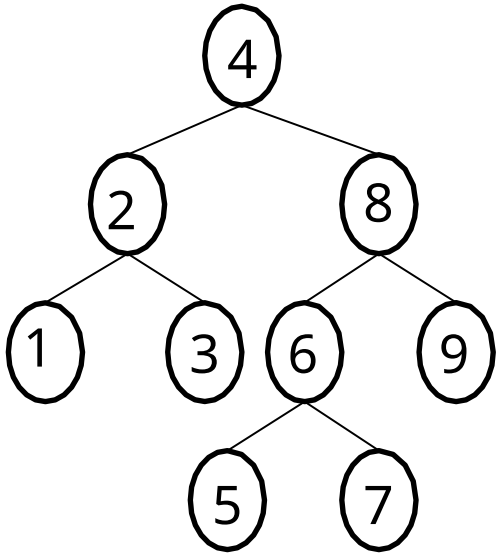
```
No* rde(No* no) {
    no->direita = rsd(no->direita);
    return rse(no);
}
```

- Rotação dupla à direita (RDD)

```
No* rdd(No* no) {
    no->esquerda = rse(no->esquerda);
    return rsd(no);
}
```

Exercícios

1. Implemente uma árvore AVL e adicione os nós de modo que a árvore apresente a respectiva topologia abaixo.



2. Implemente a operação de remoção de nós em uma árvore AVL e valide removendo o nó 6 do exercício anterior.

Estruturas de dados II

Árvores AVL

André Tavares da Silva

andre.silva@udesc.br