UFSC-CTC-INE

INE5384 - Estruturas de Dados

Métodos de Pesquisa de Dados

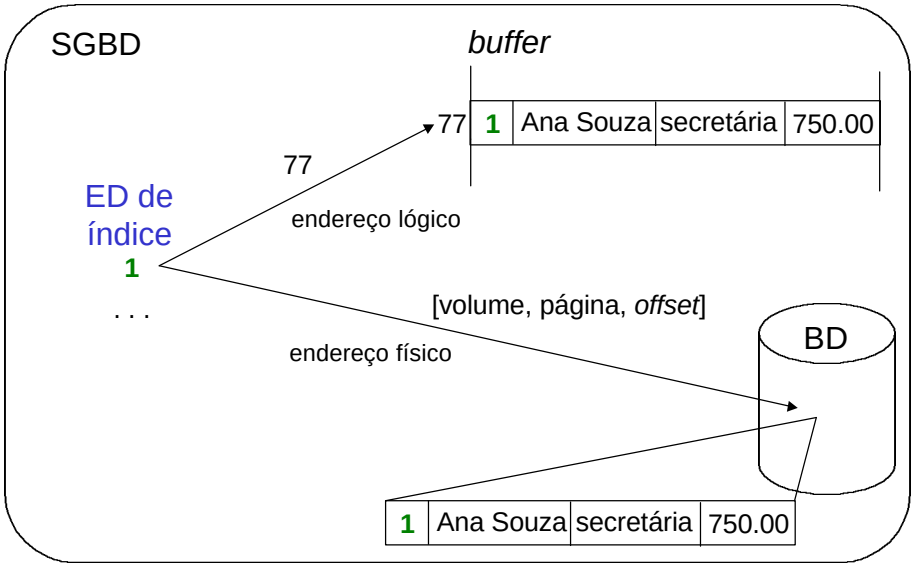
Prof. Ronaldo S. Mello

2002/2

Métodos de Pesquisa de Dados

- Encontrar um dado em um conjunto de dados de forma eficiente
- Baseia-se na noção de uma chave (índice) de pesquisa
- Aplicação típica: SGBD
  - busca de dados em disco
  - busca de dados no buffer do SGBD

Exemplo: SGBD



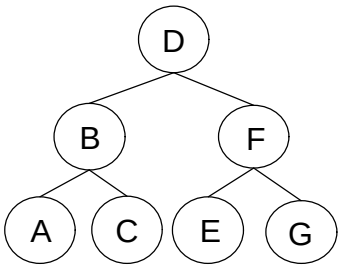
Métodos de Pesquisa de Dados

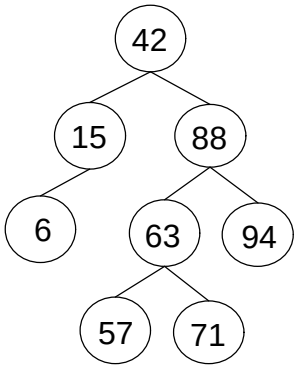
- Árvores Binárias de Pesquisa (ABP)
- Árvores-B
- *Hashing*

Árvore Binária de Pesquisa

- Uma Árvore Binária de Pesquisa (ABP) é uma árvore binária que mantém um conjunto finito de chaves únicas, tal que:
  - todos os nodos da subárvore à esquerda possuem valor de chave menor que o valor de chave do nodo raiz;
  - todos os nodos da subárvore à direita possuem valor de chave maior que o valor de chave do nodo raiz;
- Algumas implementações de ABP permitem repetição de chave

Exemplo de ABP





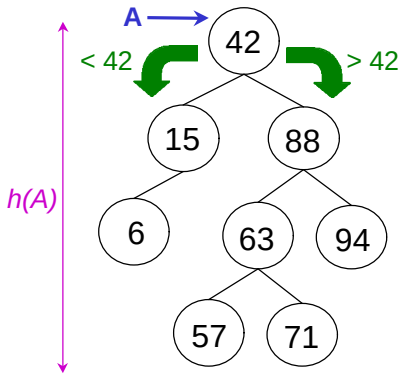
Vantagem de uma ABP

- Complexidade de busca de um objeto em uma árvore binária → *O(n)*
- Complexidade de busca de um objeto em uma ABP → *O(h(A))*

P3: $h(A) = \lfloor log_2\ n \rfloor$

Logo:

$O(h(A)) = O(log_2\ n)$

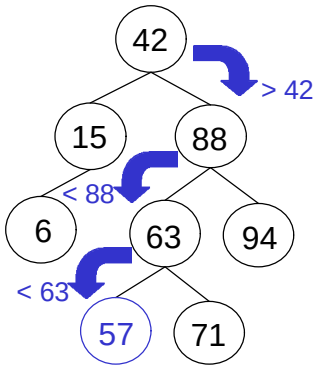


Operações em uma ABP

- Pesquisa(chave)
- Incluir(chave, dado)
- Excluir(chave)

Pesquisa em uma ABP

- Navegação na subárvore ESQ ou DIR conforme o valor de chave desejado
- Exemplo:
  - chave = 57



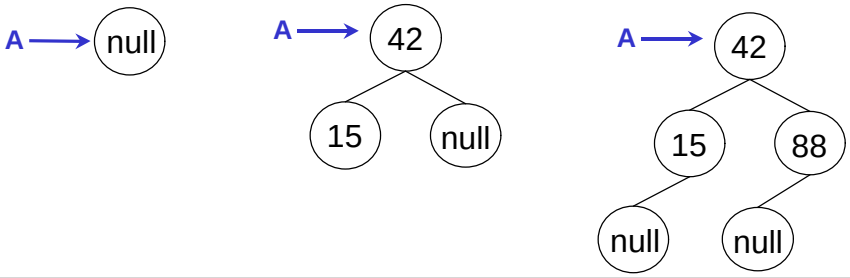
Implementação

```
Classe ÁrvoreBináriaPesquisa
Subclasse de ÁrvoreBinária
início
   chave Object;

   construtor ÁrvoreBináriaPesquisa (chave object, dado object);
   início
        this.dado ← dado;
        this.chave ← chave;
        esq ← NULL;
        dir ← NULL;
   fim;
fim;
```

Observação

- Assume-se a existência de nodos folha "vazios" na ABP:
  - nodo folha vazio: chave = NULL
  - simplifica o controle de exclusão de nodos
  - pode ser reaproveitado em uma inclusão
  - pode ser implementado um coletor de lixo



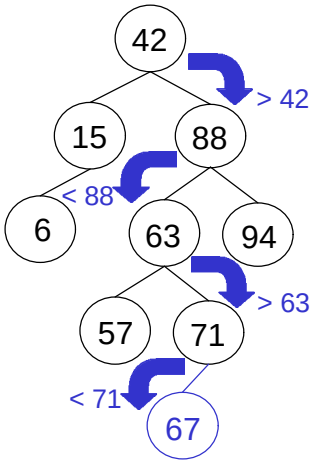
Pesquisa

```
Método pesquisa(ch object) retorna object;
início

   se this = NULL ou chave = NULL então
                              Exceção ChaveInexistente();
   se chave = ch então retorna dado;
   se ch < chave então retorna esq.pesquisa(ch)
   senão                      retorna dir.pesquisa(ch);
fim;
```

Inclusão em uma ABP

- Navegação na subárvore ESQ ou DIR até encontrar a posição correta para inclusão
- Exemplo:
  - chave = 67
- Complexidade:
  - *O(log$_2$ n)*



Inclusão

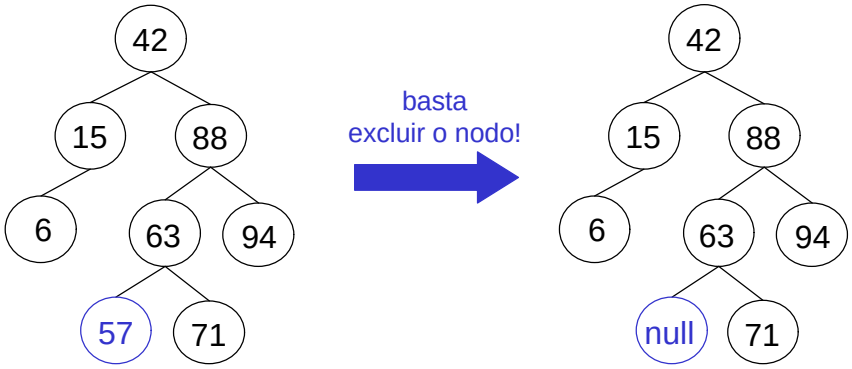
```
Método inclui(ch object, d object);
início
   se chave = NULL então
   início
         chave ← ch;
         dado ← d;
   fim
   senão se ch < chave então
                  se esq = NULL então
                           esq ← ÁrvoreBináriaPesquisa(ch, d);
                  senão  esq.inclui(ch, d);
            senão se ch > chave então
                           se dir = NULL então
                                    dir ← ÁrvoreBináriaPesquisa(ch, d);
                           senão  dir.inclui(ch, d);
                     senão Exceção ChaveExistente(); /* ch = chave */
fim;
```

Exclusão em uma ABP

- Problema: manter a ABP correta após a exclusão de um nodo
- Dois casos podem ocorrer:
  - o nodo a ser retirado é folha
  - o nodo a ser retirado não é folha

Exclusão em uma ABP

- Caso 1: nodo é folha
- Exemplo:
  - chave = 57



Exclusão em uma ABP

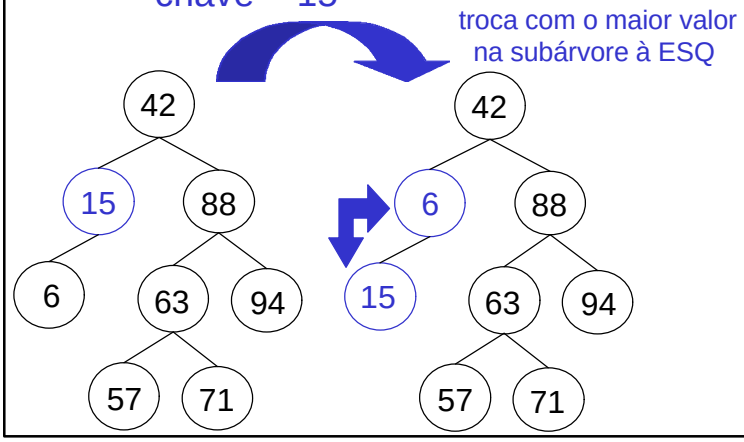
- Caso 2: nodo não é folha
- Estratégia (recursiva):
  - trocar o valor do nodo a ser removido com:
    - o valor do nodo que tenha a maior chave da sua subárvore à esquerda (nodo imediatamente anterior)

    OU
    - o valor do nodo que tenha a menor chave da sua subárvore à direita (nodo imediatamente superior)
  - ir à subárvore onde foi feita a troca (esq ou dir) e remover o nodo

    (em algum momento ele será folha!)
- Complexidade: $O(log_2\ n)$ (percorre um único caminho na ABP)

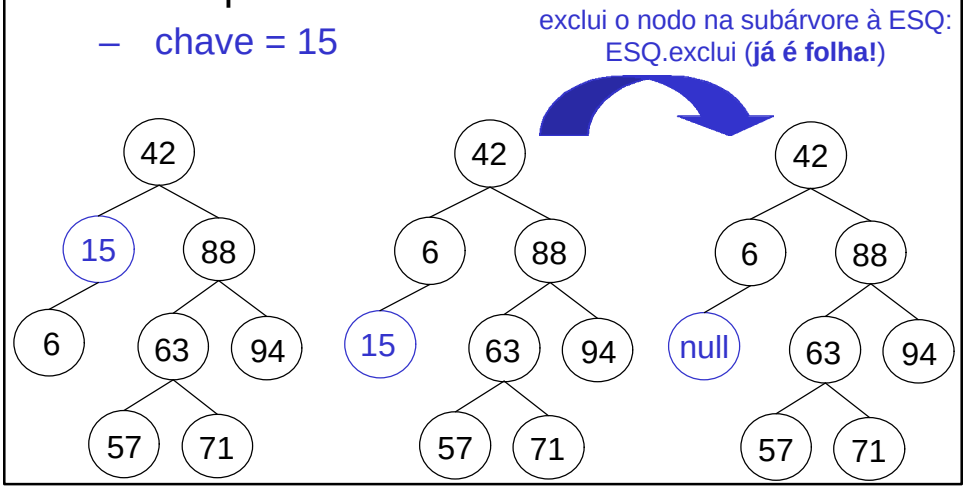
Exclusão em uma ABP

- Caso 2: nodo não é folha
- Exemplo 1:
  - chave = 15



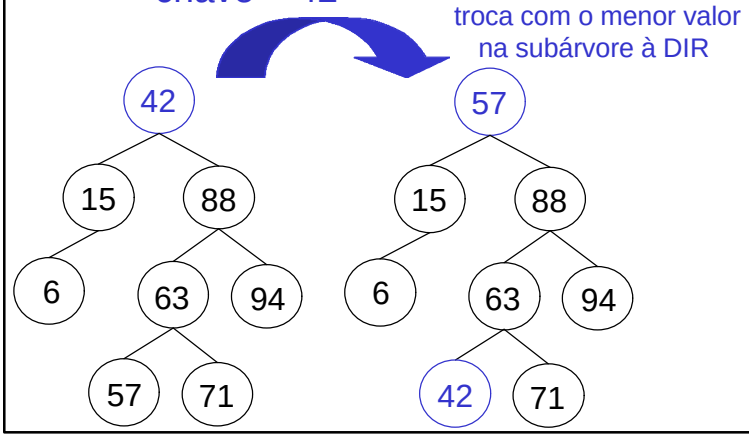
Exclusão em uma ABP

- Caso 2: nodo não é folha
- Exemplo 1:
  - chave = 15



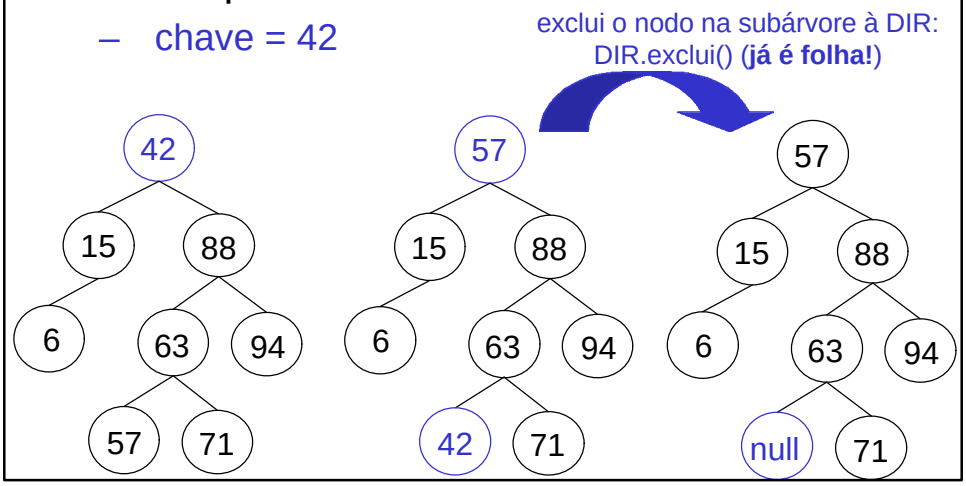
Exclusão em uma ABP

- Caso 2: nodo não é folha
- Exemplo 2:
  - chave = 42



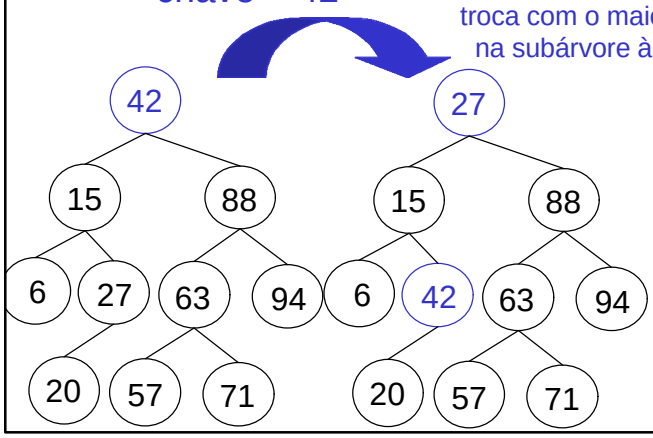
Exclusão em uma ABP

- Caso 2: nodo não é folha
- Exemplo 2:
  - chave = 42



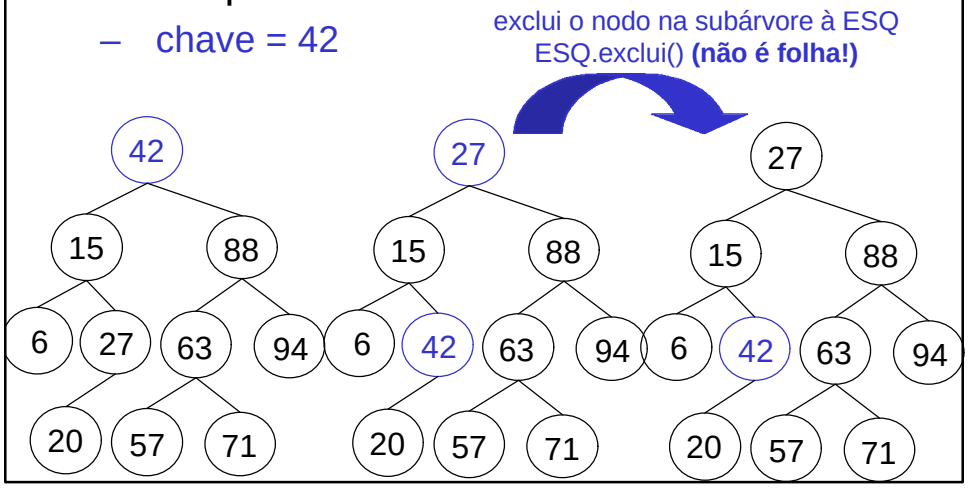
Exclusão em uma ABP

- Caso 2: nodo não é folha
- Exemplo 3:
  - chave = 42



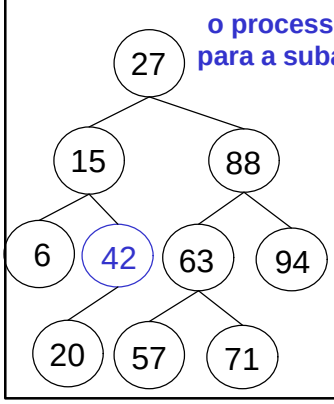
Exclusão em uma ABP

- Caso 2: nodo não é folha
- Exemplo 3:
  - chave = 42



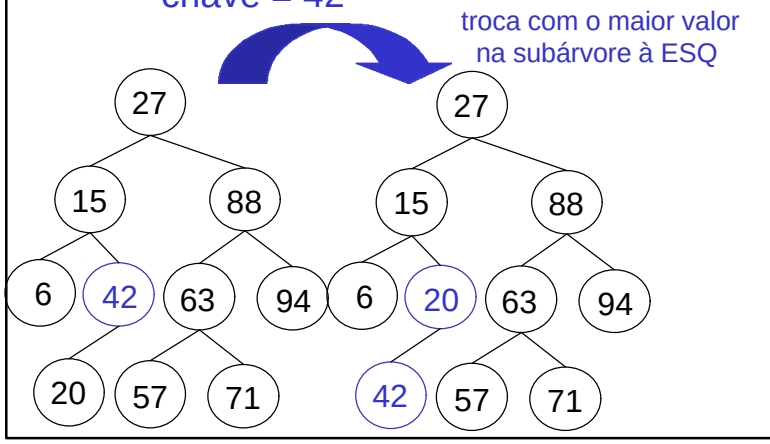
Exclusão em uma ABP

- Caso 2: nodo não é folha
- Exemplo 3:
  - chave = 42



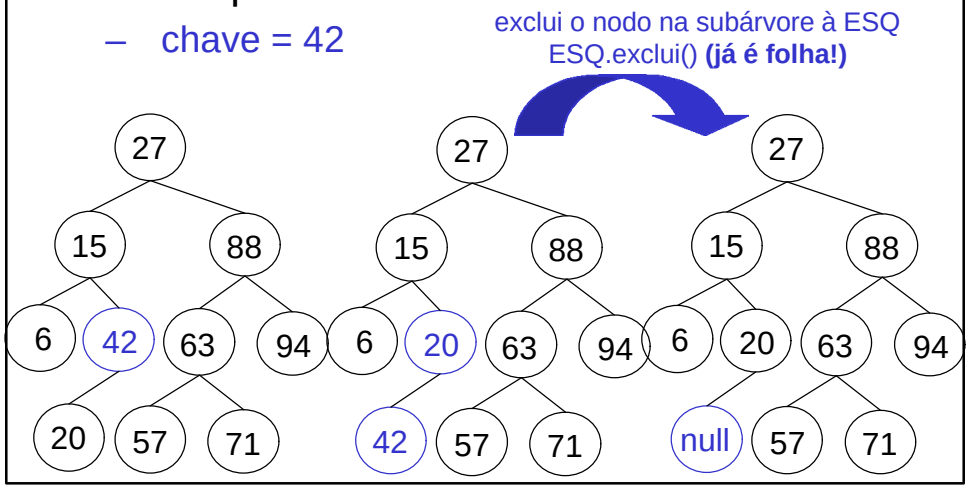
Exclusão em uma ABP

- Caso 2: nodo não é folha
- Exemplo 3:
  - chave = 42



Exclusão em uma ABP

- Caso 2: nodo não é folha
- Exemplo 3:
  - chave = 42



Exclusão

```
Método exclui(ch object);
início
   aux ÁrvoreBináriaPesquisa;

   se this = NULL ou chave = NULL então Exceção ObjetoInexistente();
   se ch < chave então esq.exclui(ch);
   se ch > chave então dir.exclui(ch);
   /* ch = chave */
   se esq <> NULL e esq.chave <> NULL então
    início
    /* troca nodo com o maior a ESQ */          aux ← nodo.esq.maior();
                                                trocaValores(this, aux);
                                                esq.exclui(ch);
    fim
   senão se dir <> NULL e dir.chave <> NULL então
           início
    /* troca nodo com o menor a DIR */          aux ← nodo.dir.menor();
                                                trocaValores(this, aux);
                                                dir.exclui(ch);
           fim
           senão    início  /* achou e ele é folha! */
                            chave ← NULL;   dado ← NULL;
                            esq ← NULL;     dir ← NULL;
                    fim                                          fim; /* Método exclui(); */
```

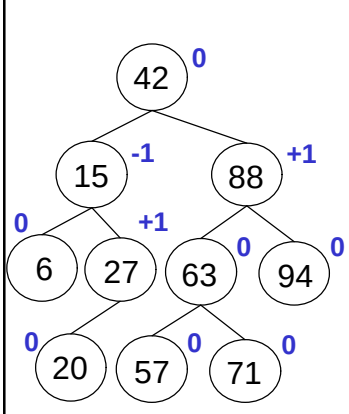
Problema com ABP

- O "desbalanceamento" progressivo de uma ABP tende a tornar linear a complexidade de pesquisa:
  - $O(\log_2 n) \Rightarrow O(n)$
- *Exemplo:*
  - *ordem de inclusão: 1,13, 24, 27, 56*
  - *complexidade da pesquisa: O(n)*
- Uma alternativa de solução: **Árvore AVL**

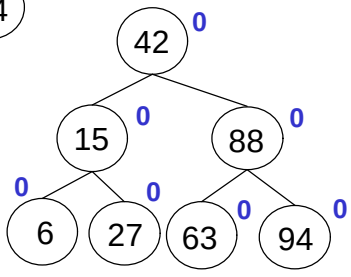
Árvore AVL

- Uma árvore AVL *A* é uma ABP tal que:
  $\forall$ *subárvore A'* $\in$ *A* (*h(A'.esq) – h(A'.dir)* $\hat{I}$ *[-1,1]*)
- Uma árvore AVL é uma ABP balanceada!
- AVL (Adelson, Velskii e Landis)

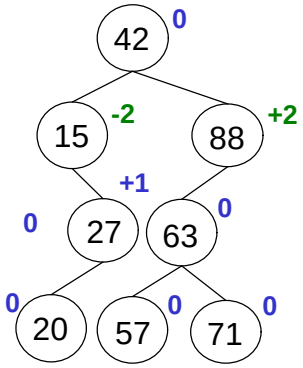
Exemplos



Árvore AVL



Árvore AVL Perfeita!



Não é árvore AVL!

Implementação

```
Classe ÁrvoreAVL
Subclasse de ÁrvoreBináriaPesquisa;
início
   fatorB inteiro;

   construtor ÁrvoreAVL (chave object, dado object);
   início
         this.dado ← dado;
         this.chave ← chave;
         esq ← NULL;
         dir ← NULL;
         fatorB ← 0;
   fim;
fim;
```

Operação de Rotação

- Como manter uma árvore AVL sempre balanceada após uma inclusão ou exclusão?
  - através de uma operação de Rotação
- Características da operação:
  - preserva a ordem das chaves
  - basta uma execução da operação de rotação para tornar a árvore novamente AVL

Operação de Rotação

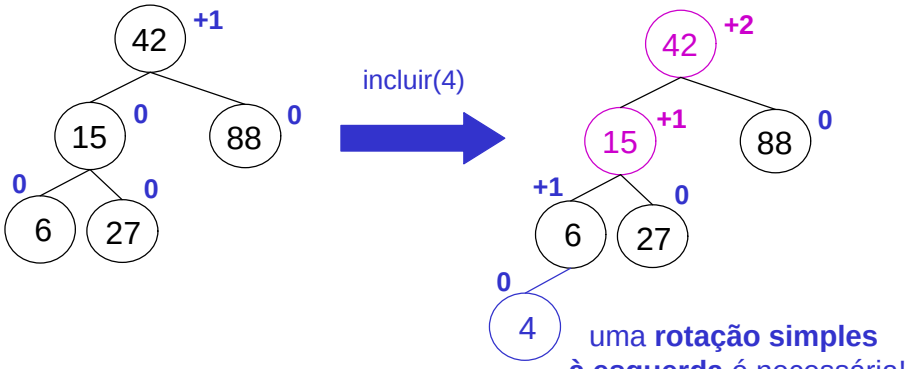
- Tipos de rotação:
  - Rotação simples:
    - à esquerda
    - à direita
  - Rotação dupla:
    - à esquerda
    - à direita

Rotação Simples

- Executada toda vez que uma (sub)árvore fica desbalanceada com um *fatorB*:
  - positivo e sua subárvore ESQ também tem um *fatorB* positivo (Rotação Simples à Esquerda - RotaçãoE)

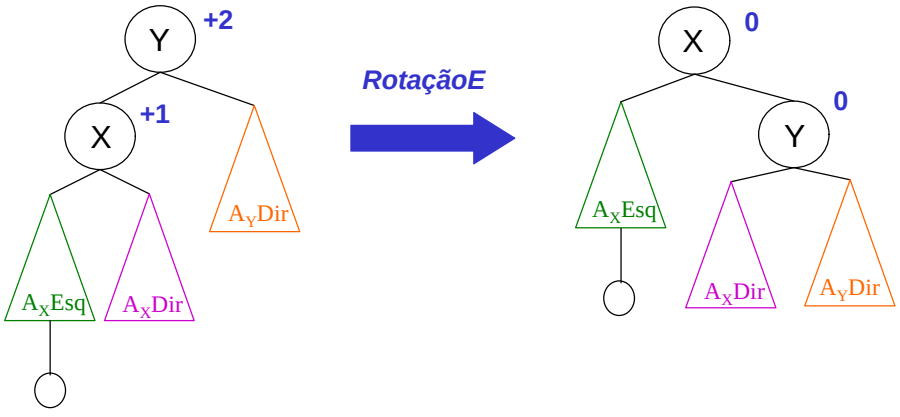
  OU
  - negativo e sua subárvore DIR também tem um *fatorB* negativo (Rotação Simples à Direita - RotaçãoD)

Exemplo

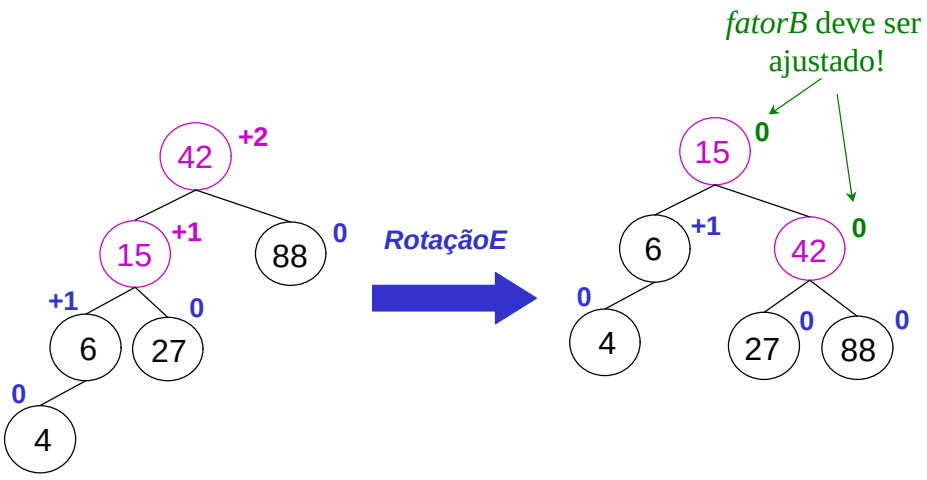


Rotação Simples à Esquerda

- Princípio de funcionamento



Exemplo



Implementação

```
Método RotaçãoE();
início
    ÁrvoreAVL arvoreAux;

    se this = NULL ou chave = NULL então Exceção ÁrvoreVazia();
    arvoreAux ← dir;
    dir ← esq;
    esq ← dir.esq;
    dir.esq ← dir.dir;
    dir.dir ← arvoreAux;
    /* troca de chave e de dado entre this e this.dir */
    trocaValores(this,dir);
    dir.ajustaFatorB();
    ajustaFatorB();
fim;
```

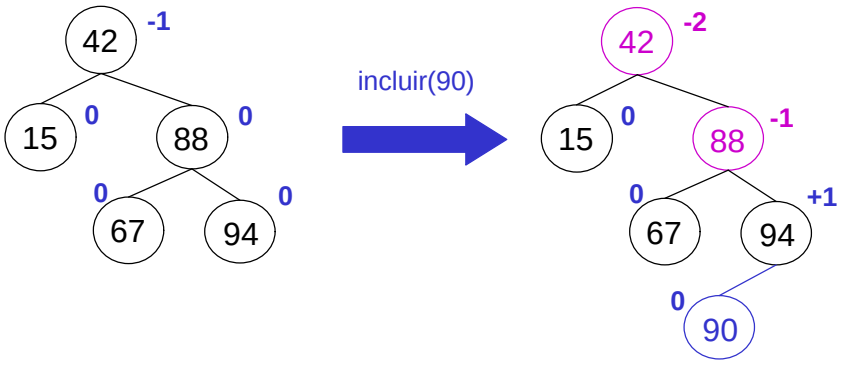
Implementação

```
Método RotaçãoE();
início
    ÁrvoreAVL arvoreAux;

    se this = NULL ou chave = NULL então Exceção ÁrvoreVazia();
    arvoreAux ← dir;
    dir ← esq;
    esq ← dir.esq;
    dir.esq ← dir.dir;
    dir.dir ← arvoreAux;
    /* troca de chave e de dado entre this e this.dir */
    trocaValores(this,dir);
    dir.ajustaFatorB();
    ajustaFatorB();
fim;
```

Complexidade: *O(1)*

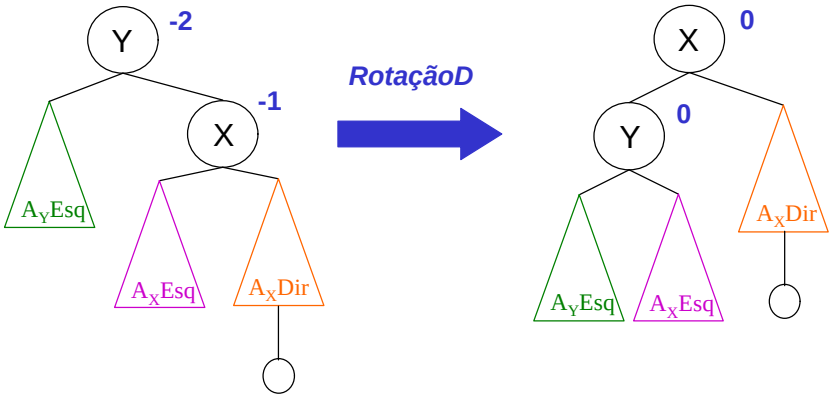
Exemplo 2



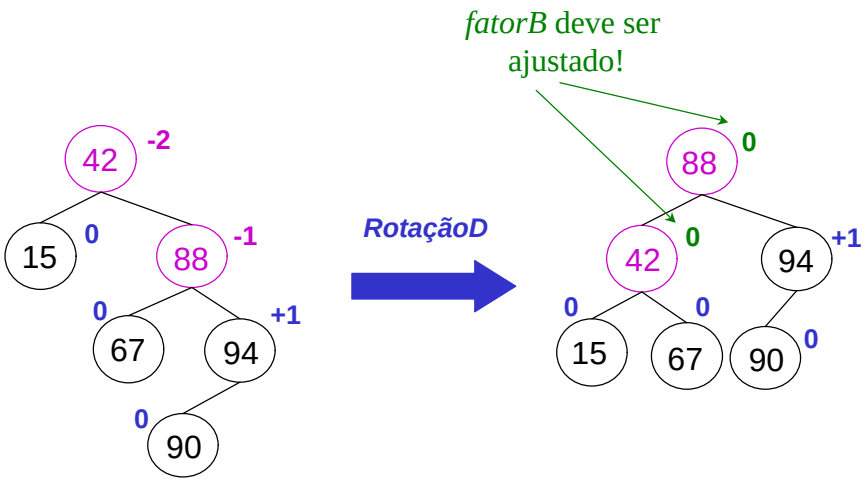
uma **rotação simples à direita** é necessária!

Rotação Simples à Direita

- Princípio de funcionamento



Exemplo 2



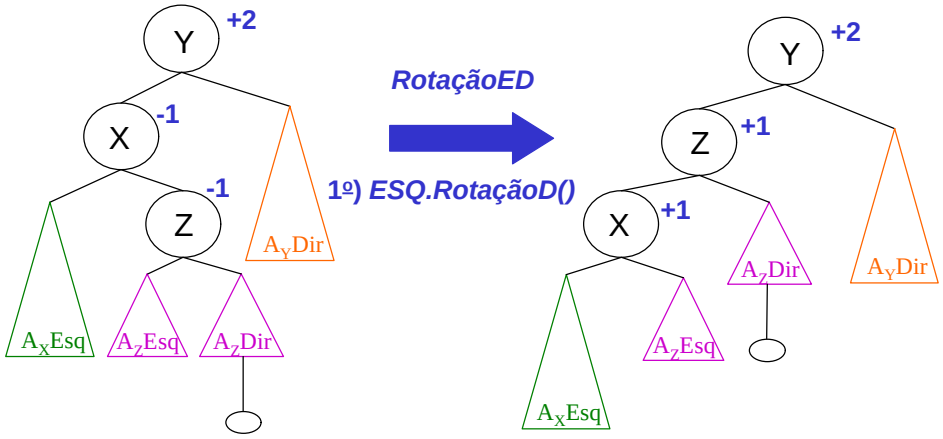
Rotação Dupla

- Executada toda vez que uma (sub)árvore fica desbalanceada com um *fatorB*:
  - positivo e sua subárvore ESQ tem um *fatorB* negativo (Rotação Dupla à Esquerda - RotaçãoED)

  OU
  - negativo e sua subárvore DIR tem um *fatorB* positivo (Rotação Dupla à Direita - RotaçãoDE)

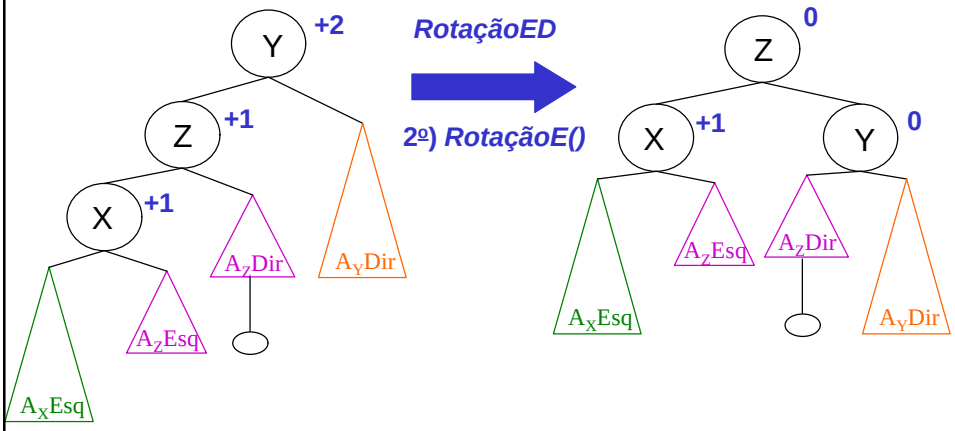
Rotação Dupla à Esquerda

- Princípio de funcionamento

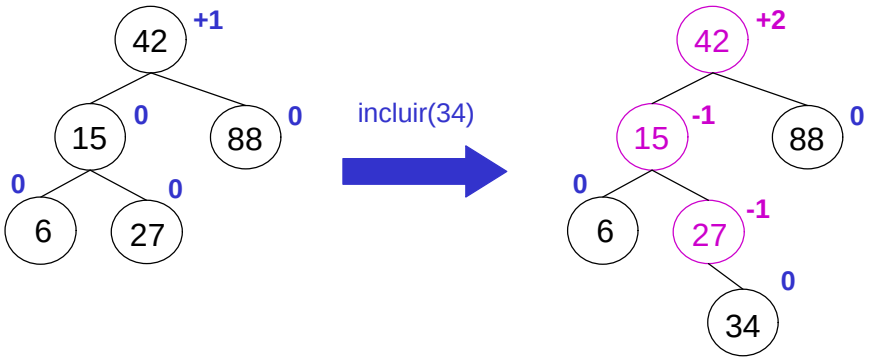


Rotação Dupla à Esquerda

- Princípio de funcionamento

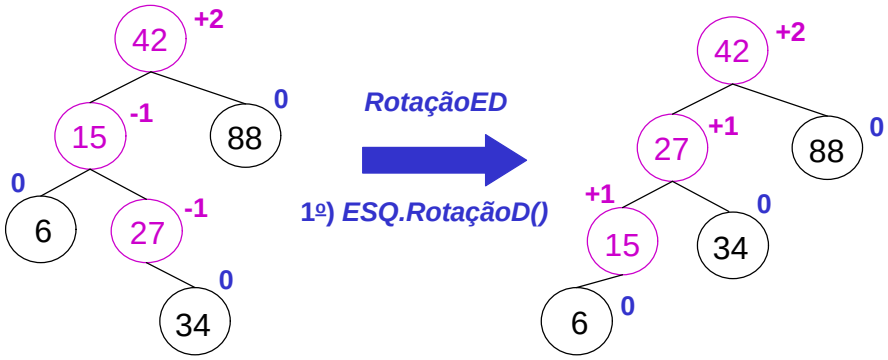


Exemplo

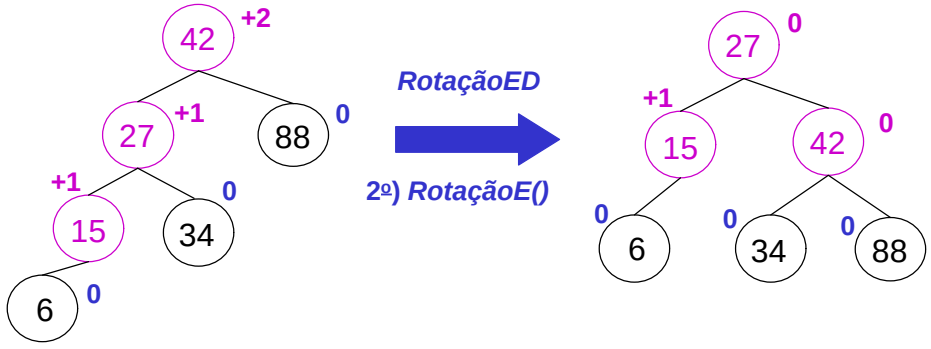


uma **rotação dupla à esquerda** é necessária!

Exemplo



Exemplo



Implementação

```
Método RotaçãoED();
início

   se this = NULL ou chave = NULL então
        Exceção ÁrvoreVazia();
   esq.RotaçãoD();
   RotaçãoE();
   fim;
```

Balanceamento

- Quando realizar o balanceamento (rotação(ões)) de uma arvore AVL?
  - Sempre que a árvore apresentar um *fatorB* fora do intervalo [-1,1]
  - O valor do *fatorB* deve ser testado sempre após uma operação de inclusão ou exclusão de nodo na árvore

Balanceamento

```
Método inclui(ch object, d object);
início
   . . .
   balanceamento();
fim;

Método exclui(ch object);
início
   . . .
   balanceamento();
fim;
```

Balanceamento

```
Método balanceamento();
início

   ajustaFatorB();
   se fatorB > 1 então
              se esq.fatorB > 0 então     rotaçãoE()
              senão                       rotaçãoED();
   senão se fatorB < -1 então
              se esq.fatorB < 0 então     rotaçãoD()
              senão                       rotaçãoDE();
fim;
```

Exercícios

- Implementar na classe ÁrvoreAVL:
  - *RotaçãoD();*
  - *ajustaFatorB();*
  - *RotaçãoDE();*
- Implementar na classe ÁrvoreBináriaPesquisa:
  - *maior() retorna ÁrvoreBináriaPesquisa;*
  - *menor() retorna ÁrvoreBináriaPesquisa;*
  - *coletorLixo();*