MC-202 — Unidade 16
Árvores Balanceadas

Rafael C. S. Schouery
rafael@ic.unicamp.br

Universidade Estadual de Campinas

1º semestre/2017

Eficiência da busca, inserção e remoção

Qual é o tempo da busca, inserção e remoção em ABBs?

Eficiência da busca, inserção e remoção

Qual é o tempo da busca, inserção e remoção em ABBs?

- depende da altura da árvore...

Eficiência da busca, inserção e remoção

Qual é o tempo da busca, inserção e remoção em ABBs?

- depende da altura da árvore...

Ex: 31 nós

Eficiência da busca, inserção e remoção

Qual é o tempo da busca, inserção e remoção em ABBs?

- depende da altura da árvore...

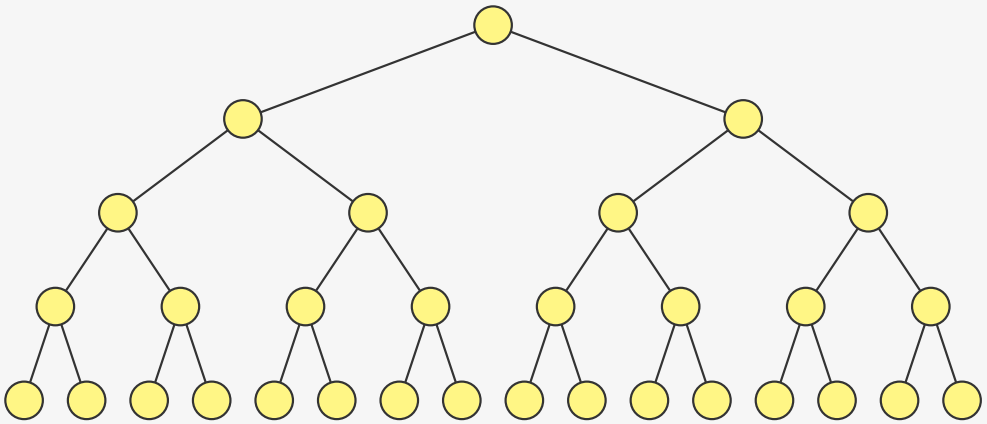
Ex: 31 nós

Eficiência da busca, inserção e remoção

Qual é o tempo da busca, inserção e remoção em ABBs?

- depende da altura da árvore...

Ex: 31 nós



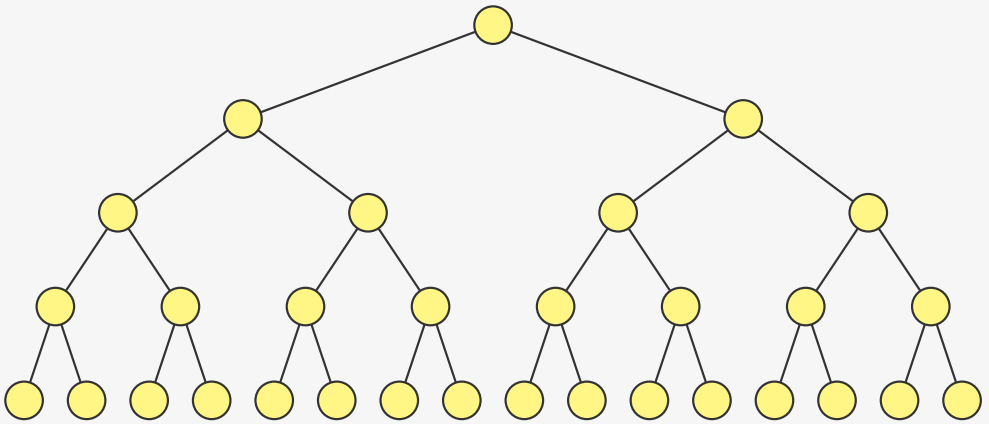
Melhor árvore: $\lceil \lg n + 1 \rceil$

Eficiência da busca, inserção e remoção

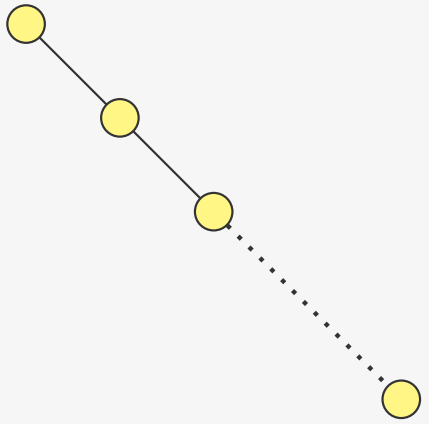
Qual é o tempo da busca, inserção e remoção em ABBs?

- depende da altura da árvore...

Ex: 31 nós



Melhor árvore: $\lceil \lg n + 1 \rceil$



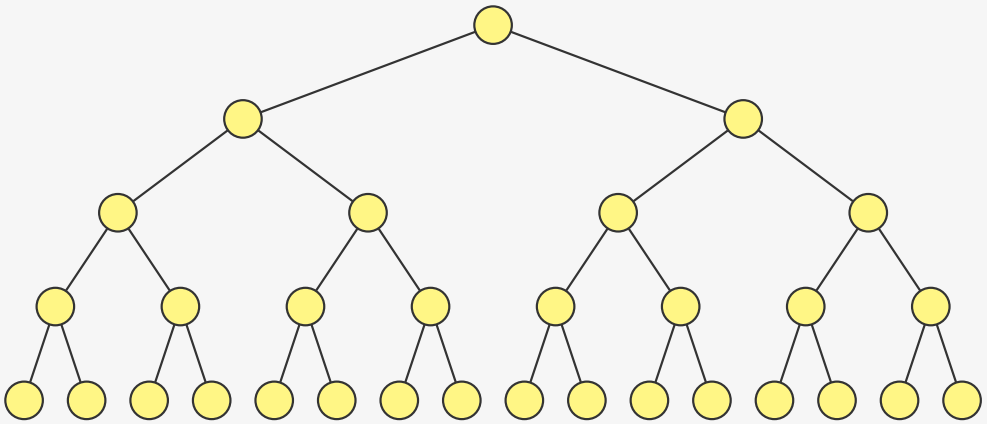
Pior árvore: n

Eficiência da busca, inserção e remoção

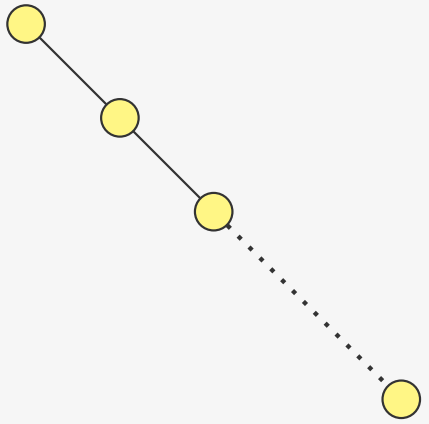
Qual é o tempo da busca, inserção e remoção em ABBs?

- depende da altura da árvore...

Ex: 31 nós



Melhor árvore: $\lceil \lg n + 1 \rceil$



Pior árvore: n

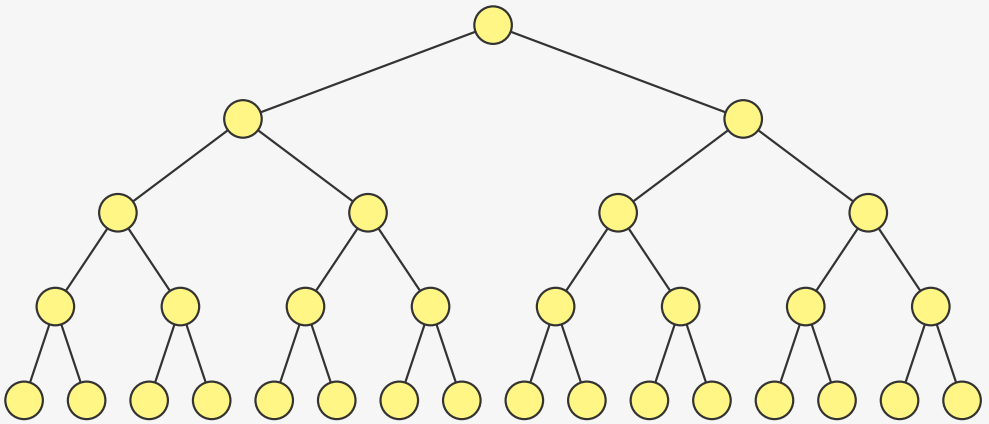
Para ter a pior árvore basta inserir em ordem crescente...

Eficiência da busca, inserção e remoção

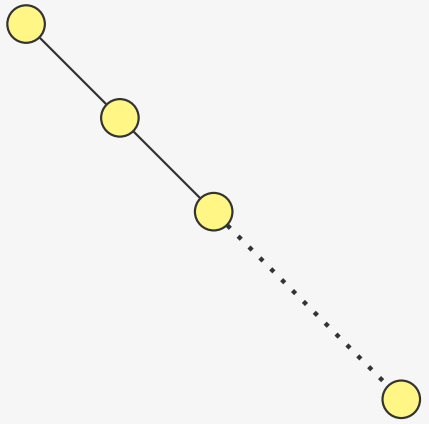
Qual é o tempo da busca, inserção e remoção em ABBs?

- depende da altura da árvore...

Ex: 31 nós



Melhor árvore: $\lceil \lg n + 1 \rceil$



Pior árvore: n

Para ter a pior árvore basta inserir em ordem crescente...

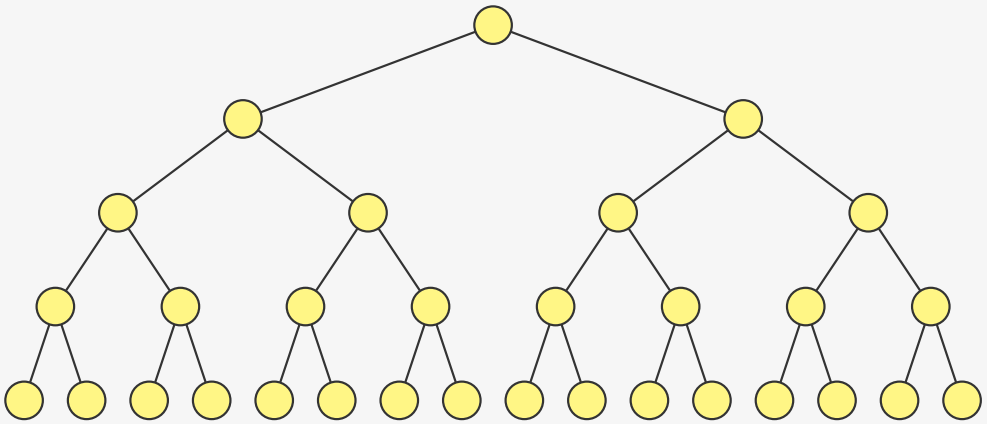
Veremos uma árvore balanceada

Eficiência da busca, inserção e remoção

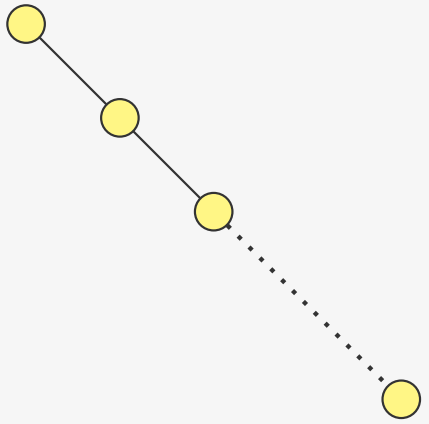
Qual é o tempo da busca, inserção e remoção em ABBs?

- depende da altura da árvore...

Ex: 31 nós



Melhor árvore: $\lceil \lg n + 1 \rceil$



Pior árvore: n

Para ter a pior árvore basta inserir em ordem crescente...

Veremos uma árvore balanceada

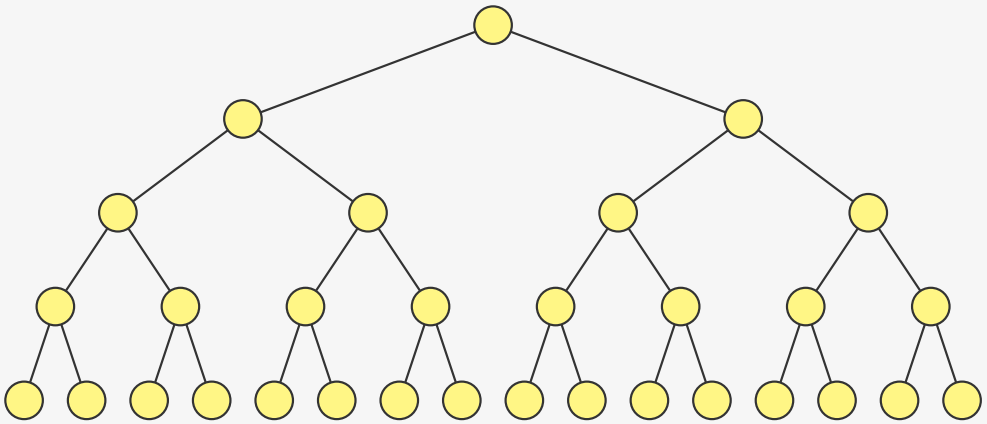
- Não é a melhor árvore possível, mas é “quase”

Eficiência da busca, inserção e remoção

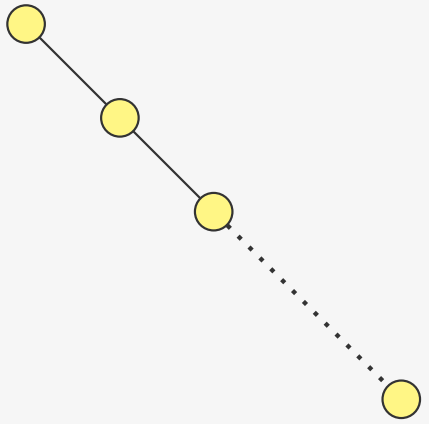
Qual é o tempo da busca, inserção e remoção em ABBs?

- depende da altura da árvore...

Ex: 31 nós



Melhor árvore: $\lceil \lg n + 1 \rceil$



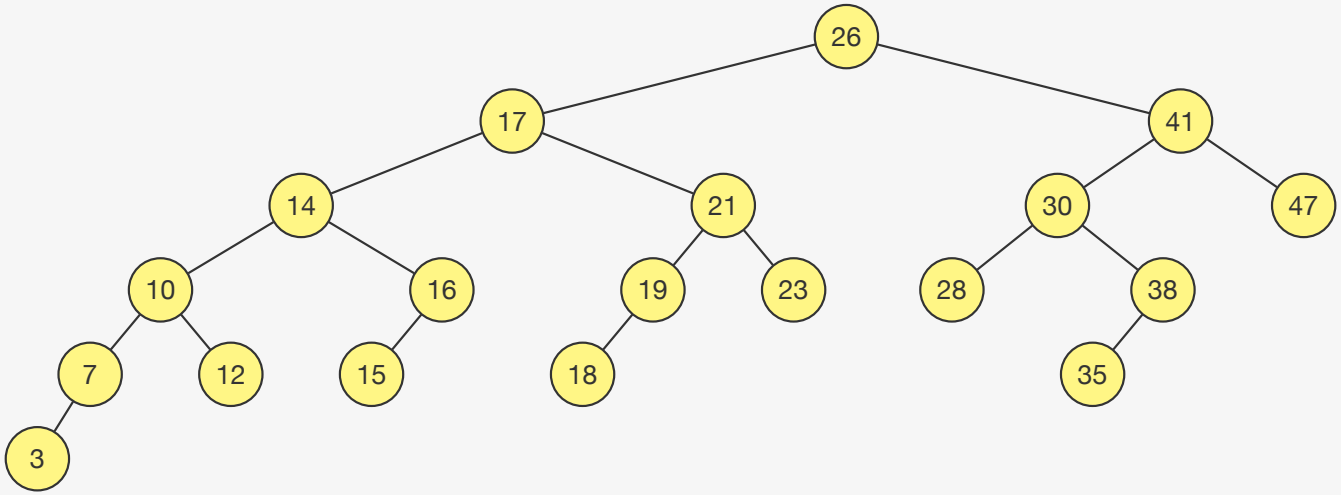
Pior árvore: n

Para ter a pior árvore basta inserir em ordem crescente...

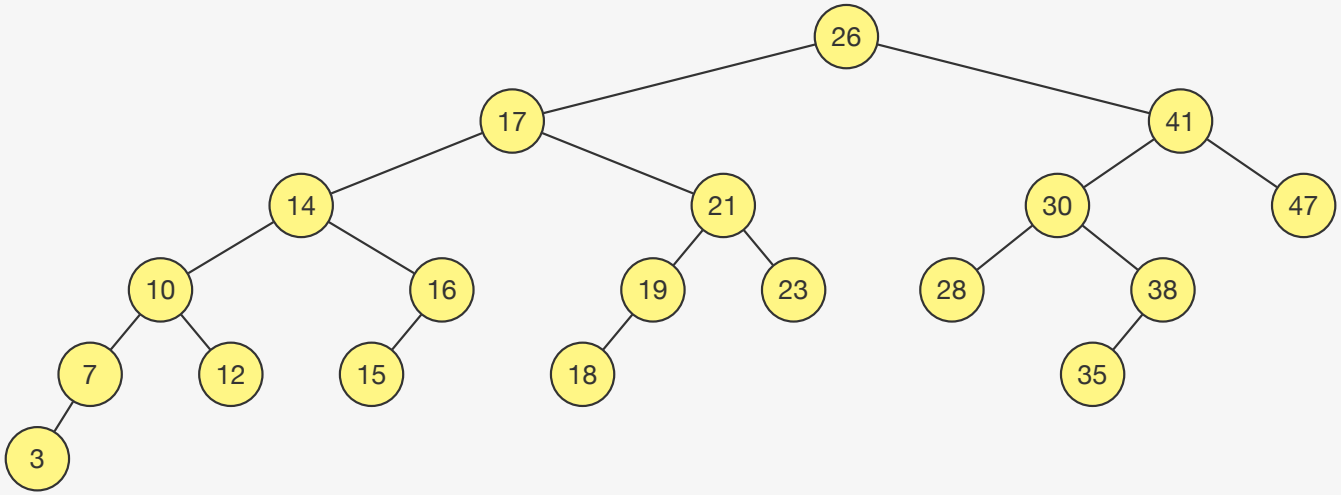
Veremos uma árvore balanceada

- Não é a melhor árvore possível, mas é “quase”
- Operações em $\mathrm{O}(\lg n)$

Árvores Rubro-Negras Esquerdistas

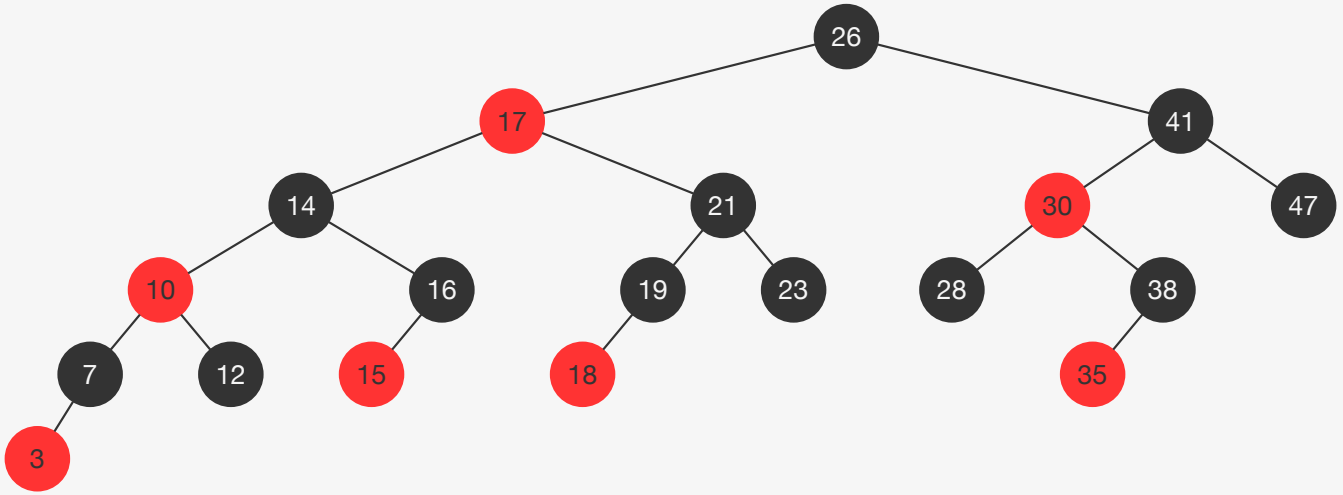


Árvores Rubro-Negras Esquerdistas



Uma árvore **rubro**-**negra** esquerdista é uma ABB tal que:

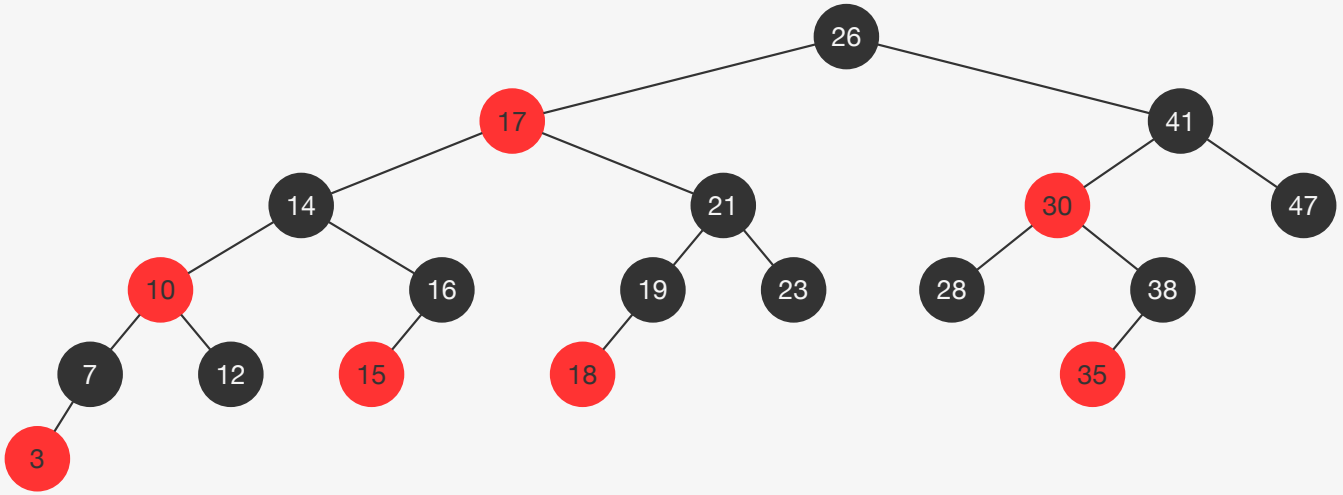
Árvores Rubro-Negras Esquerdistas



Uma árvore **rubro**-**negra** esquerdista é uma ABB tal que:

1. Todo nó é ou **vermelho** ou **preto**

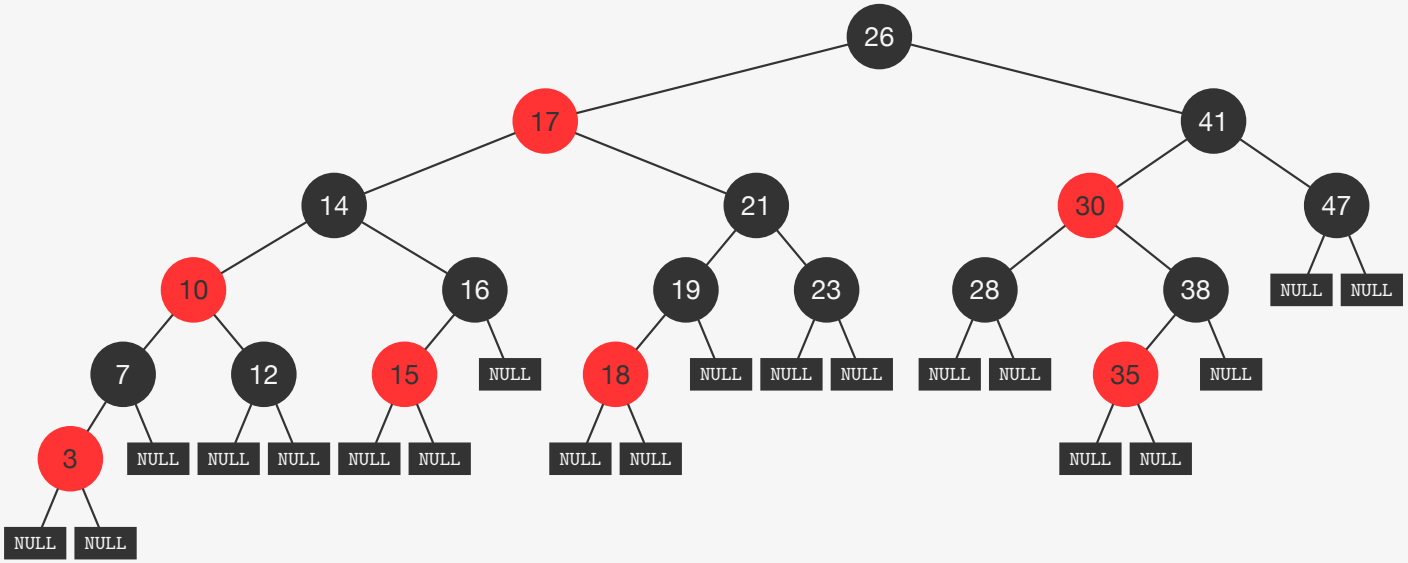
Árvores Rubro-Negras Esquerdistas



Uma árvore **rubro**-**negra** esquerdista é uma ABB tal que:

1. Todo nó é ou **vermelho** ou **preto**
2. A raiz é **preta**

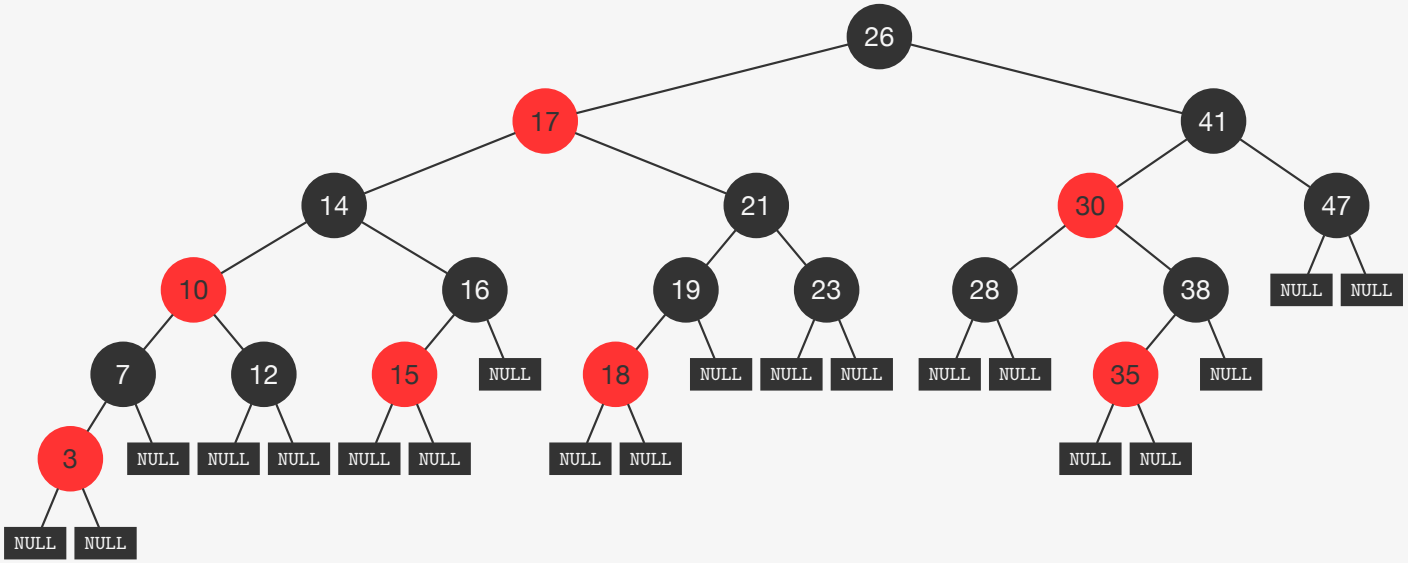
Árvores Rubro-Negras Esquerdistas



Uma árvore **rubro**-**negra** esquerdista é uma ABB tal que:

1. Todo nó é ou **vermelho** ou **preto**
2. A raiz é **preta**
3. As folhas são `NULL` e tem cor **preta**

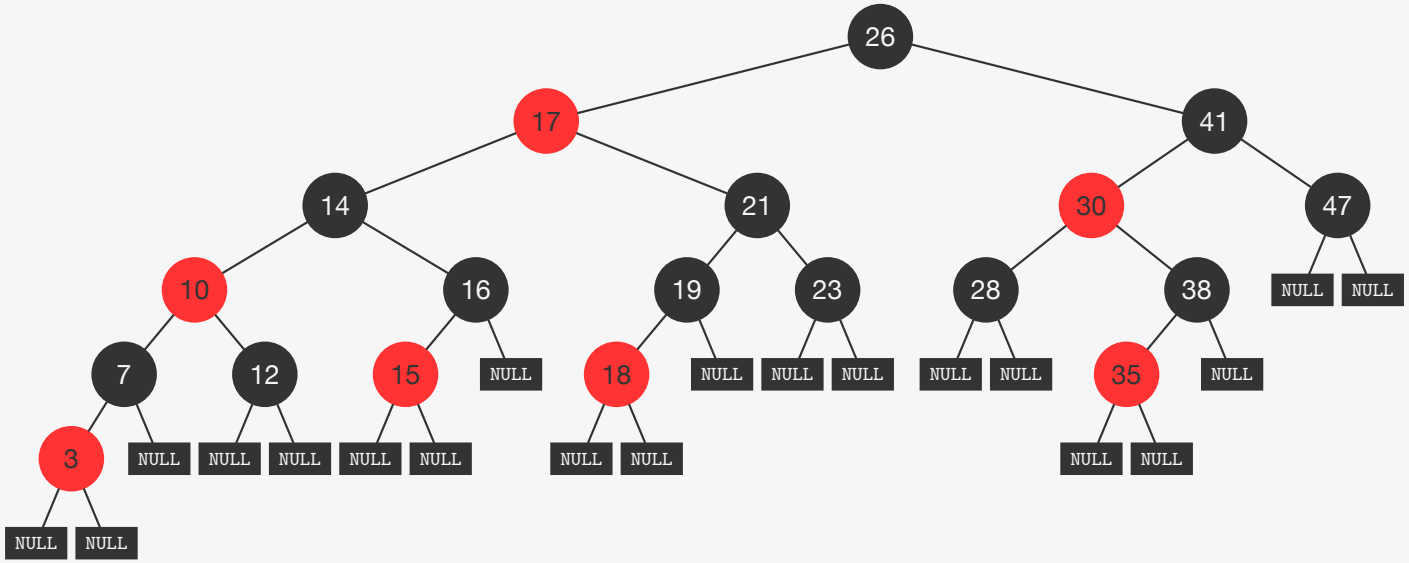
Árvores Rubro-Negras Esquerdistas



Uma árvore **rubro**-**negra** esquerdista é uma ABB tal que:

1. Todo nó é ou **vermelho** ou **preto**
2. A raiz é **preta**
3. As folhas são `NULL` e tem cor **preta**
4. Se um nó é **vermelho**, seus dois filhos são **pretos**

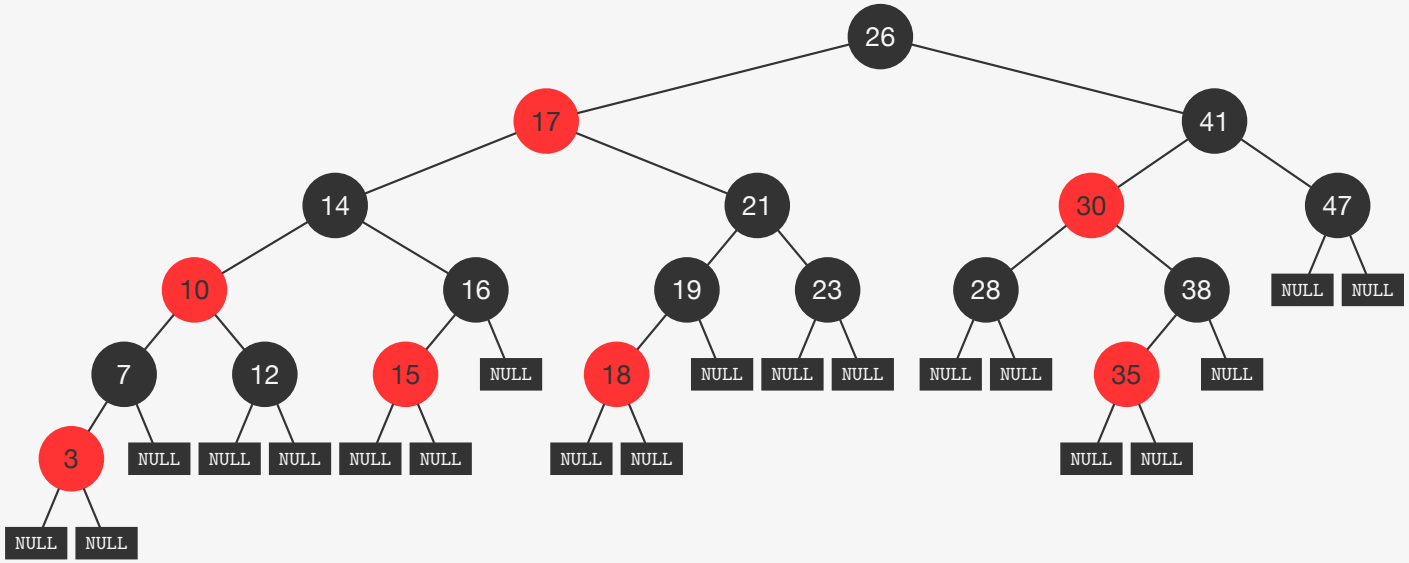
Árvores Rubro-Negras Esquerdistas



Uma árvore **rubro**-**negra** esquerdista é uma ABB tal que:

1. Todo nó é ou **vermelho** ou **preto**
2. A raiz é **preta**
3. As folhas são `NULL` e tem cor **preta**
4. Se um nó é **vermelho**, seus dois filhos são **pretos**
   - ele é o filho esquerdo do seu pai (por isso, *esquerdista*)

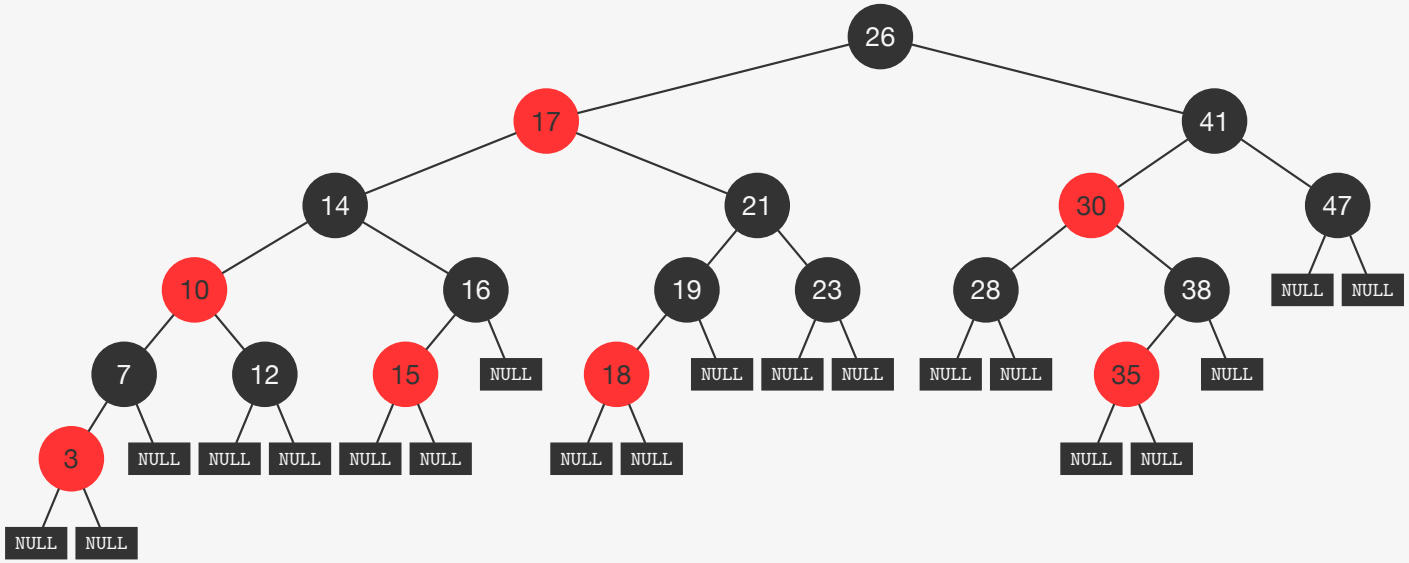
Árvores Rubro-Negras Esquerdistas



Uma árvore **rubro**-**negra** esquerdista é uma ABB tal que:

1. Todo nó é ou **vermelho** ou **preto**
2. A raiz é **preta**
3. As folhas são `NULL` e tem cor **preta**
4. Se um nó é **vermelho**, seus dois filhos são **pretos**
   - ele é o filho esquerdo do seu pai (por isso, *esquerdista*)
5. Em cada nó, todo caminho dele para uma de suas folhas descendentes tem a mesma quantidade de nós **pretos**

Árvores Rubro-Negras Esquerdistas



Uma árvore **rubro**-**negra** esquerdista é uma ABB tal que:

1. Todo nó é ou **vermelho** ou **preto**
2. A raiz é **preta**
3. As folhas são `NULL` e tem cor **preta**
4. Se um nó é **vermelho**, seus dois filhos são **pretos**
   - ele é o filho esquerdo do seu pai (por isso, *esquerdista*)
5. Em cada nó, todo caminho dele para uma de suas folhas descendentes tem a mesma quantidade de nós **pretos**
   - É a altura-**negra** do nó

Altura da Árvore Rubro-Negra Esquerdista

Seja bh a altura-**negra** da árvore

Altura da Árvore Rubro-Negra Esquerdista

Seja bh a altura-**negra** da árvore

A árvore tem pelo menos $2^{bh} - 1$ nós internos:

Altura da Árvore Rubro-Negra Esquerdista

Seja bh a altura-**negra** da árvore

A árvore tem pelo menos $2^{bh} - 1$ nós internos:

- Se $bh = 0$, a árvore é apenas uma folha `NULL`

Altura da Árvore Rubro-Negra Esquerdista

Seja bh a altura-**negra** da árvore

A árvore tem pelo menos $2^{bh} - 1$ nós internos:

- Se $bh = 0$, a árvore é apenas uma folha `NULL`
  - tem exatamente $2^{bh} - 1 = 0$ nós internos

Altura da Árvore Rubro-Negra Esquerdista

Seja bh a altura-**negra** da árvore

A árvore tem pelo menos $2^{bh} - 1$ nós internos:

- Se $bh = 0$, a árvore é apenas uma folha `NULL`
  - tem exatamente $2^{bh} - 1 = 0$ nós internos
- Se $bh > 0$, seus filhos tem altura-**negra**

Altura da Árvore Rubro-Negra Esquerdista

Seja bh a altura-**negra** da árvore

A árvore tem pelo menos $2^{bh} - 1$ nós internos:

- Se $bh = 0$, a árvore é apenas uma folha `NULL`
  - tem exatamente $2^{bh} - 1 = 0$ nós internos
- Se $bh > 0$, seus filhos tem altura-**negra**
  - bh se for um filho **vermelho**

Altura da Árvore Rubro-Negra Esquerdista

Seja bh a altura-**negra** da árvore

A árvore tem pelo menos $2^{bh} - 1$ nós internos:

- Se $bh = 0$, a árvore é apenas uma folha `NULL`
  - tem exatamente $2^{bh} - 1 = 0$ nós internos
- Se $bh > 0$, seus filhos tem altura-**negra**
  - bh se for um filho **vermelho**
  - $bh - 1$ se for um filho **preto**

Altura da Árvore Rubro-Negra Esquerdista

Seja bh a altura-**negra** da árvore

A árvore tem pelo menos $2^{bh} - 1$ nós internos:

- Se $bh = 0$, a árvore é apenas uma folha `NULL`
  - tem exatamente $2^{bh} - 1 = 0$ nós internos
- Se $bh > 0$, seus filhos tem altura-**negra**
  - bh se for um filho **vermelho**
  - $bh - 1$ se for um filho **preto**
- cada filho tem pelo menos $2^{bh-1} - 1$ nós internos

Altura da Árvore Rubro-Negra Esquerdista

Seja bh a altura-**negra** da árvore

A árvore tem pelo menos $2^{bh} - 1$ nós internos:

- Se $bh = 0$, a árvore é apenas uma folha `NULL`
  - tem exatamente $2^{bh} - 1 = 0$ nós internos
- Se $bh > 0$, seus filhos tem altura-**negra**
  - bh se for um filho **vermelho**
  - $bh - 1$ se for um filho **preto**
- cada filho tem pelo menos $2^{bh-1} - 1$ nós internos
- a árvore tem pelo menos $2(2^{bh-1} - 1) + 1$ nós internos

Altura da Árvore Rubro-Negra Esquerdista

Seja bh a altura-**negra** da árvore

A árvore tem pelo menos $2^{bh} - 1$ nós internos:

- Se $bh = 0$, a árvore é apenas uma folha `NULL`
  - tem exatamente $2^{bh} - 1 = 0$ nós internos
- Se $bh > 0$, seus filhos tem altura-**negra**
  - bh se for um filho **vermelho**
  - $bh - 1$ se for um filho **preto**
- cada filho tem pelo menos $2^{bh-1} - 1$ nós internos
- a árvore tem pelo menos $2(2^{bh-1} - 1) + 1$ nós internos
  - ou seja, tem pelo menos $2^{bh} - 1$ nós internos

Altura da Árvore Rubro-Negra Esquerdista

Seja bh a altura-**negra** da árvore

A árvore tem pelo menos $2^{bh} - 1$ nós internos:

- Se $bh = 0$, a árvore é apenas uma folha `NULL`
  - tem exatamente $2^{bh} - 1 = 0$ nós internos
- Se $bh > 0$, seus filhos tem altura-**negra**
  - bh se for um filho **vermelho**
  - $bh - 1$ se for um filho **preto**
- cada filho tem pelo menos $2^{bh-1} - 1$ nós internos
- a árvore tem pelo menos $2(2^{bh-1} - 1) + 1$ nós internos
  - ou seja, tem pelo menos $2^{bh} - 1$ nós internos

A altura-**negra** bh é pelo menos metade da altura h da árvore

Altura da Árvore Rubro-Negra Esquerdista

Seja bh a altura-**negra** da árvore

A árvore tem pelo menos $2^{bh} - 1$ nós internos:

- Se $bh = 0$, a árvore é apenas uma folha `NULL`
  - tem exatamente $2^{bh} - 1 = 0$ nós internos
- Se $bh > 0$, seus filhos tem altura-**negra**
  - bh se for um filho **vermelho**
  - $bh - 1$ se for um filho **preto**
- cada filho tem pelo menos $2^{bh-1} - 1$ nós internos
- a árvore tem pelo menos $2(2^{bh-1} - 1) + 1$ nós internos
  - ou seja, tem pelo menos $2^{bh} - 1$ nós internos

A altura-**negra** bh é pelo menos metade da altura h da árvore

- Não existe nó **vermelho** com filho **vermelho**

Altura da Árvore Rubro-Negra Esquerdista

Seja bh a altura-**negra** da árvore

A árvore tem pelo menos $2^{bh} - 1$ nós internos:

- Se $bh = 0$, a árvore é apenas uma folha `NULL`
  - tem exatamente $2^{bh} - 1 = 0$ nós internos
- Se $bh > 0$, seus filhos tem altura-**negra**
  - bh se for um filho **vermelho**
  - $bh - 1$ se for um filho **preto**
- cada filho tem pelo menos $2^{bh-1} - 1$ nós internos
- a árvore tem pelo menos $2(2^{bh-1} - 1) + 1$ nós internos
  - ou seja, tem pelo menos $2^{bh} - 1$ nós internos

A altura-**negra** bh é pelo menos metade da altura h da árvore

- Não existe nó **vermelho** com filho **vermelho**
- O número de nós internos n é $n \geq 2^{bh} - 1 \geq 2^{h/2} - 1$

Altura da Árvore Rubro-Negra Esquerdista

Seja bh a altura-**negra** da árvore

A árvore tem pelo menos $2^{bh} - 1$ nós internos:

- Se $bh = 0$, a árvore é apenas uma folha `NULL`
  - tem exatamente $2^{bh} - 1 = 0$ nós internos
- Se $bh > 0$, seus filhos tem altura-**negra**
  - bh se for um filho **vermelho**
  - $bh - 1$ se for um filho **preto**
- cada filho tem pelo menos $2^{bh-1} - 1$ nós internos
- a árvore tem pelo menos $2(2^{bh-1} - 1) + 1$ nós internos
  - ou seja, tem pelo menos $2^{bh} - 1$ nós internos

A altura-**negra** bh é pelo menos metade da altura h da árvore

- Não existe nó **vermelho** com filho **vermelho**
- O número de nós internos n é $n \geq 2^{bh} - 1 \geq 2^{h/2} - 1$
- Ou seja, $h \leq 2 \lg(n + 1) = \mathrm{O}(\lg n)$

Alterando a Struct e testando a cor

```
1 enum Cor {VERMELHO, PRETO};
2
3 typedef struct No {
4     int chave;
5     enum Cor cor;
6     struct No *esq, *dir;
7 } No;
```

Alterando a Struct e testando a cor

```
1 enum Cor {VERMELHO, PRETO};
2
3 typedef struct No {
4     int chave;
5     enum Cor cor;
6     struct No *esq, *dir;
7 } No;
```

```
1 int ehVermelho(No *x) {
2   if (x == NULL)
3     return 0;
4   return x->cor == VERMELHO;
5 }
```

Alterando a Struct e testando a cor

```
1 enum Cor {VERMELHO, PRETO};
2
3 typedef struct No {
4     int chave;
5     enum Cor cor;
6     struct No *esq, *dir;
7 } No;
```

```
1 int ehVermelho(No *x) {
2   if (x == NULL)
3     return 0;
4   return x->cor == VERMELHO;
5 }
```

```
1 int ehPreto(No *x) {
2   if (x == NULL)
3     return 1;
4   return x->cor == PRETO;
5 }
```

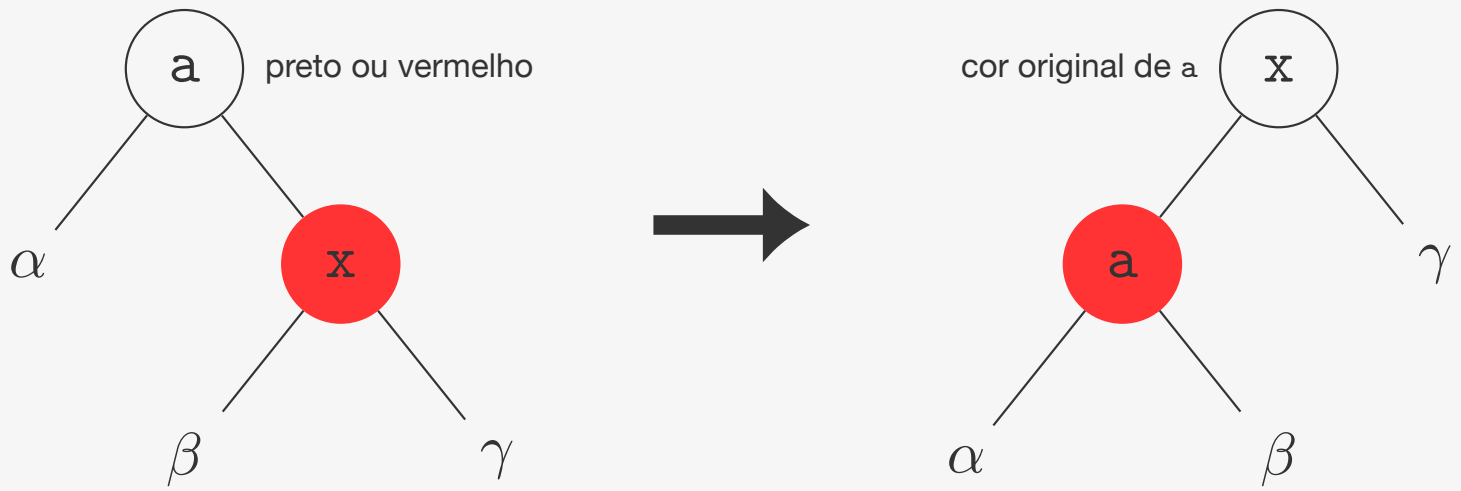
Alterando a Struct e testando a cor

```
1 enum Cor {VERMELHO, PRETO};
2
3 typedef struct No {
4     int chave;
5     enum Cor cor;
6     struct No *esq, *dir;
7 } No;
```

```
1 int ehVermelho(No *x) {
2   if (x == NULL)
3     return 0;
4   return x->cor == VERMELHO;
5 }
```

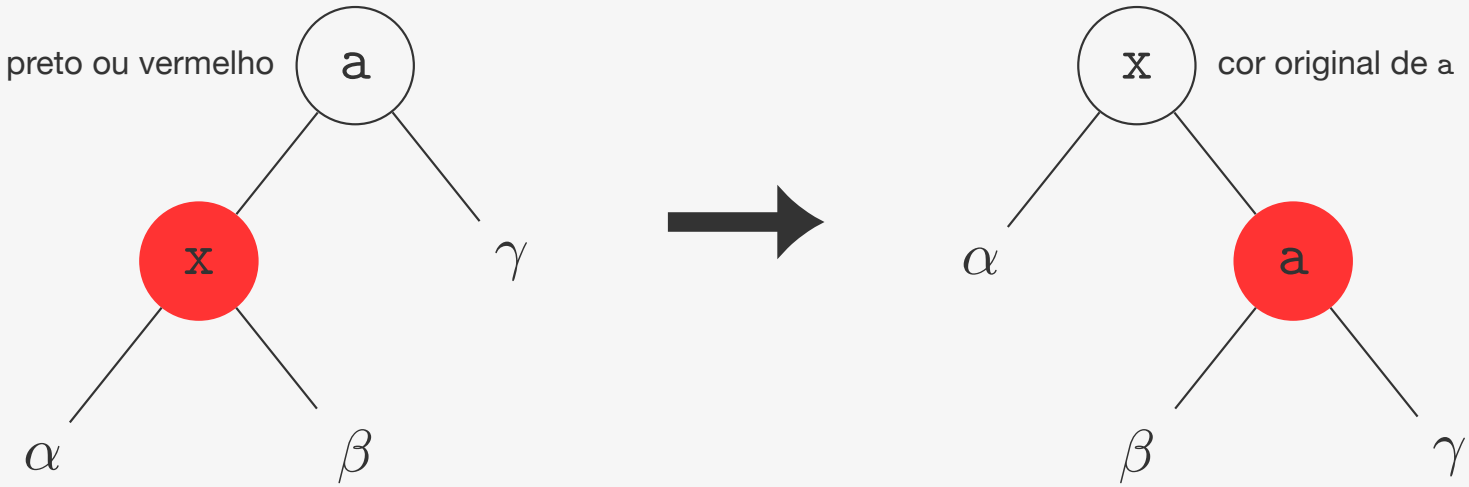
```
1 int ehPreto(No *x) {
2   if (x == NULL)
3     return 1;
4   return x->cor == PRETO;
5 }
```

Rotação para a esquerda



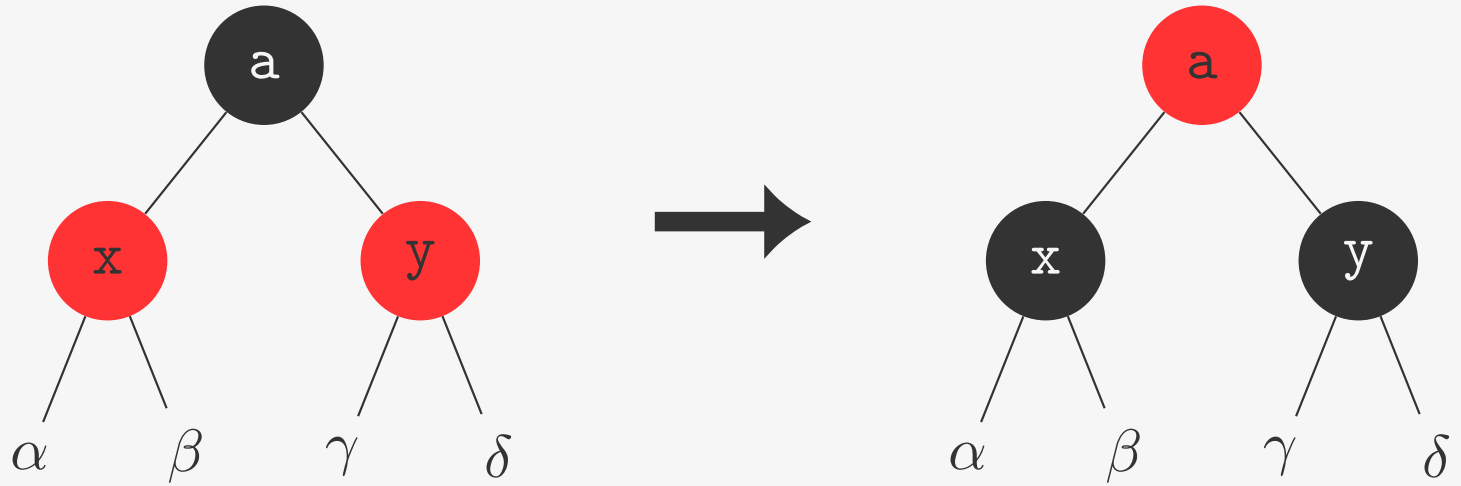

```
1 No * rotaciona_para_esquerda(No *arvore) {
2   No *x = arvore->dir;
3   arvore->dir = x->esq;
4   x->esq = arvore;
5   x->cor = arvore->cor;
6   arvore->cor = VERMELHO;
7   return x;
8 }
```

Rotação para a direita




```
1 No * rotaciona_para_direita(No *arvore) {
2   No *x = arvore->esq;
3   arvore->esq = x->dir;
4   x->dir = arvore;
5   x->cor = arvore->cor;
6   arvore->cor = VERMELHO;
7   return x;
8 }
```

Subindo a cor




```
1 void sobe_vermelho(No *arvore) {
2   arvore->cor = VERMELHO;
3   arvore->esq->cor = PRETO;
4   arvore->dir->cor = PRETO;
5 }
```

Inserindo

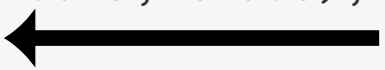
Inserimos como em uma ABB, mas precisamos manter as propriedades da árvore **rubro-negra** esquerdista

```
1  No* inserir_rec(No *arvore, int chave) {
2    No *novo;
3    if (arvore == NULL) {
4      novo = malloc(sizeof(No));
5      novo->esq = novo->dir = NULL;
6      novo->chave = chave;
7      novo->cor = VERMELHO;
8      return novo;
9    }
10   if (chave < arvore->chave)
11     arvore->esq = inserir_rec(arvore->esq, chave);
12   else
13     arvore->dir = inserir_rec(arvore->dir, chave);
14   /* corrige a árvore */
15   ...
16   return arvore;
17 }
18
19 void inserir(No **arvore, int chave) {
20   *arvore = inserir_rec(*arvore, chave);
21   (*arvore)->cor = PRETO;
22 }
```

Inserindo

Inserimos como em uma ABB, mas precisamos manter as propriedades da árvore **rubro**-**negra** esquerdista

```
1  No* inserir_rec(No *arvore, int chave) {
2    No *novo;
3    if (arvore == NULL) {
4      novo = malloc(sizeof(No));
5      novo->esq = novo->dir = NULL;
6      novo->chave = chave;
7      novo->cor = VERMELHO;
8      return novo;
9    }
10   if (chave < arvore->chave)
11     arvore->esq = inserir_rec(arvore->esq, chave);
12   else
13     arvore->dir = inserir_rec(arvore->dir, chave);
14   /* corrige a árvore */
15   ...
16   return arvore;
17 }
18
19 void inserir(No **arvore, int chave) {
20   *arvore = inserir_rec(*arvore, chave);
21   (*arvore)->cor = PRETO;
22 }
```



Mantém a raiz preta

Inserção - Caso 1

Inserção - Caso 1

- Nó é **preto**

Inserção - Caso 1

- Nó é **preto**
  - não sabemos a cor do seu pai

Inserção - Caso 1

- Nó é **preto**
  - não sabemos a cor do seu pai
  - nem se ele é o filho esquerdo ou direito

Inserção - Caso 1

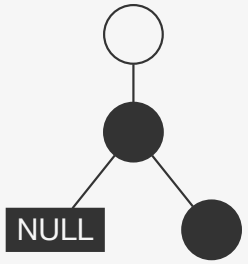
- Nó é **preto**
  - não sabemos a cor do seu pai
  - nem se ele é o filho esquerdo ou direito
- Filho direito é **preto** (tem que ser - por que?)

Inserção - Caso 1

- Nó é **preto**
  - não sabemos a cor do seu pai
  - nem se ele é o filho esquerdo ou direito
- Filho direito é **preto** (tem que ser - por que?)
- Inserimos no filho esquerdo

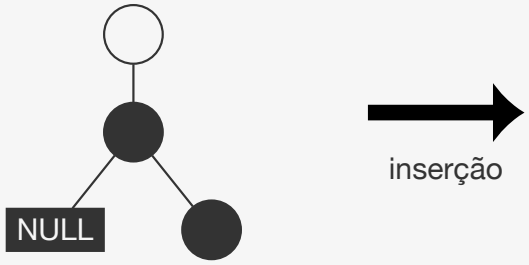
Inserção - Caso 1

- Nó é **preto**
  - não sabemos a cor do seu pai
  - nem se ele é o filho esquerdo ou direito
- Filho direito é **preto** (tem que ser - por que?)
- Inserimos no filho esquerdo



Inserção - Caso 1

- Nó é **preto**
  - não sabemos a cor do seu pai
  - nem se ele é o filho esquerdo ou direito
- Filho direito é **preto** (tem que ser - por que?)
- Inserimos no filho esquerdo

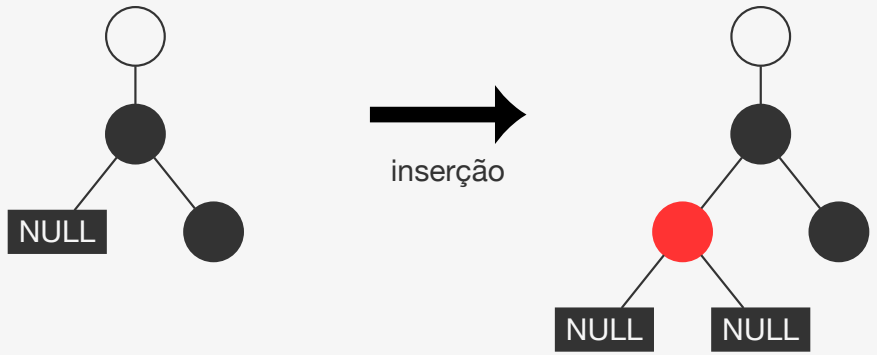


Inserção - Caso 1

- Nó é **preto**
  - não sabemos a cor do seu pai
  - nem se ele é o filho esquerdo ou direito
- Filho direito é **preto** (tem que ser - por que?)
- Inserimos no filho esquerdo

Inserção - Caso 1

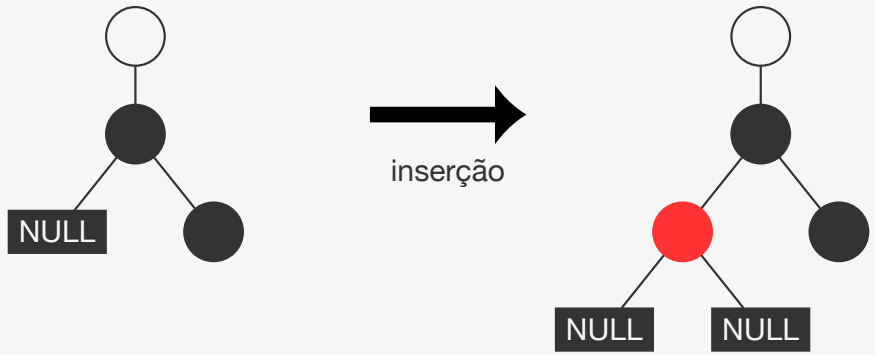
- Nó é **preto**
  - não sabemos a cor do seu pai
  - nem se ele é o filho esquerdo ou direito
- Filho direito é **preto** (tem que ser - por que?)
- Inserimos no filho esquerdo




```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Inserção - Caso 1

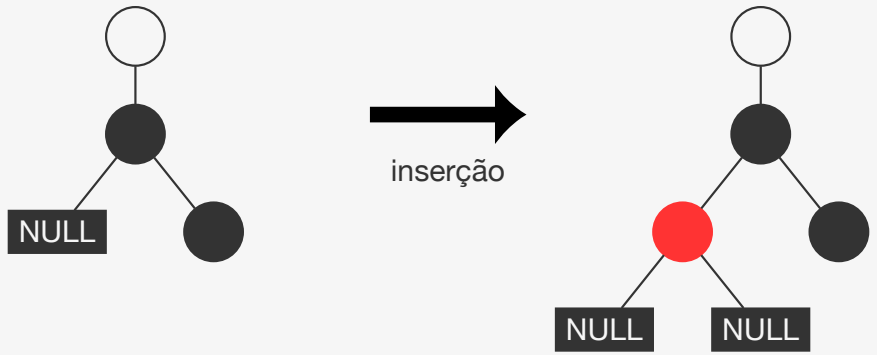
- Nó é **preto**
  - não sabemos a cor do seu pai
  - nem se ele é o filho esquerdo ou direito
- Filho direito é **preto** (tem que ser - por que?)
- Inserimos no filho esquerdo




```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Inserção - Caso 1

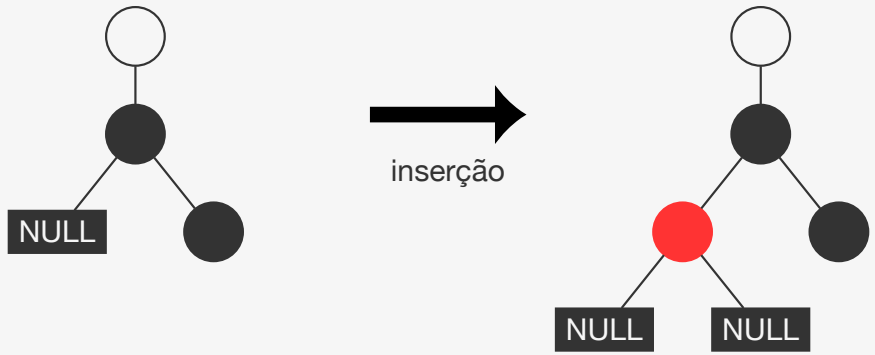
- Nó é **preto**
  - não sabemos a cor do seu pai
  - nem se ele é o filho esquerdo ou direito
- Filho direito é **preto** (tem que ser - por que?)
- Inserimos no filho esquerdo




```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Inserção - Caso 1

- Nó é **preto**
  - não sabemos a cor do seu pai
  - nem se ele é o filho esquerdo ou direito
- Filho direito é **preto** (tem que ser - por que?)
- Inserimos no filho esquerdo




```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Inserção - Caso 2

```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Inserção - Caso 2

- Nó é **preto**
  - não sabemos a cor do seu pai
  - nem se ele é o filho esquerdo ou direito

```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Inserção - Caso 2

- Nó é **preto**
  - não sabemos a cor do seu pai
  - nem se ele é o filho esquerdo ou direito
- Filho esquerdo é **preto**

```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

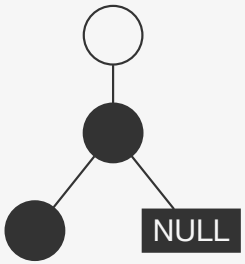
Inserção - Caso 2

- Nó é **preto**
  - não sabemos a cor do seu pai
  - nem se ele é o filho esquerdo ou direito
- Filho esquerdo é **preto**
- Inserimos no filho direito

```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Inserção - Caso 2

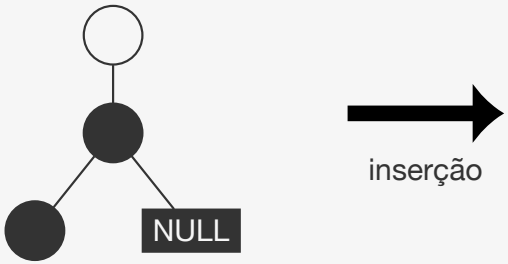
- Nó é **preto**
  - não sabemos a cor do seu pai
  - nem se ele é o filho esquerdo ou direito
- Filho esquerdo é **preto**
- Inserimos no filho direito




```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Inserção - Caso 2

- Nó é **preto**
  - não sabemos a cor do seu pai
  - nem se ele é o filho esquerdo ou direito
- Filho esquerdo é **preto**
- Inserimos no filho direito




```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Inserção - Caso 2

- Nó é **preto**
  - não sabemos a cor do seu pai
  - nem se ele é o filho esquerdo ou direito
- Filho esquerdo é **preto**
- Inserimos no filho direito

```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Inserção - Caso 2

- Nó é **preto**
  - não sabemos a cor do seu pai
  - nem se ele é o filho esquerdo ou direito
- Filho esquerdo é **preto**
- Inserimos no filho direito

```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

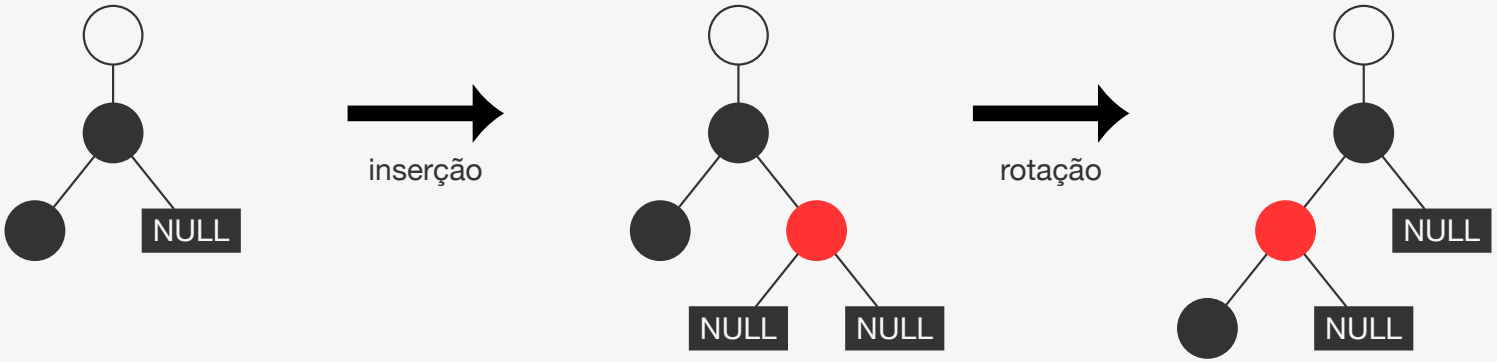
Inserção - Caso 2

- Nó é **preto**
  - não sabemos a cor do seu pai
  - nem se ele é o filho esquerdo ou direito
- Filho esquerdo é **preto**
- Inserimos no filho direito

```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Inserção - Caso 2

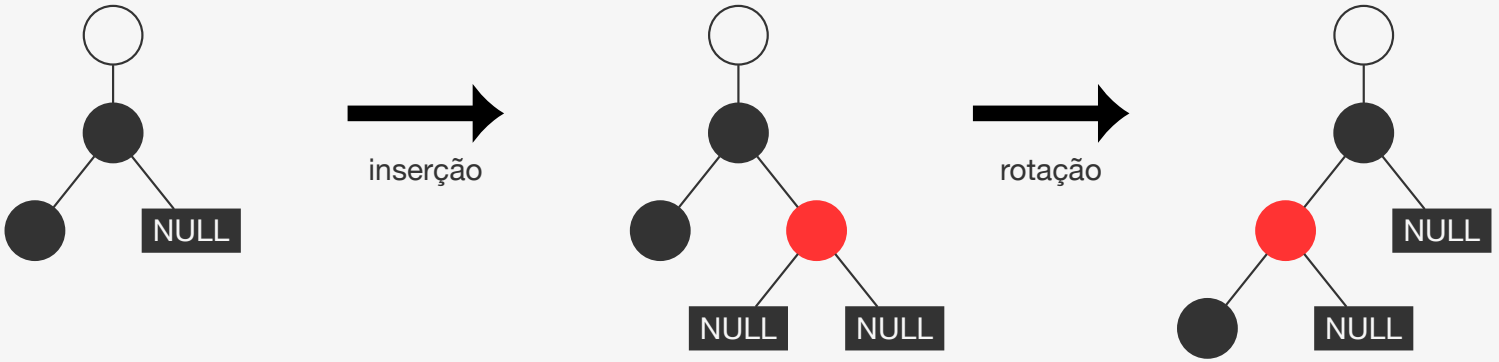
- Nó é **preto**
  - não sabemos a cor do seu pai
  - nem se ele é o filho esquerdo ou direito
- Filho esquerdo é **preto**
- Inserimos no filho direito




```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Inserção - Caso 2

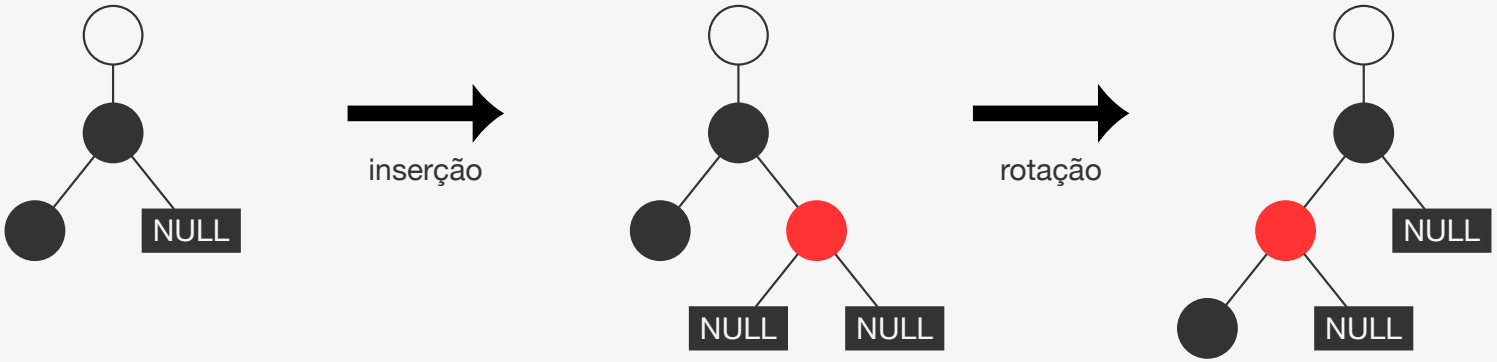
- Nó é **preto**
  - não sabemos a cor do seu pai
  - nem se ele é o filho esquerdo ou direito
- Filho esquerdo é **preto**
- Inserimos no filho direito




```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Inserção - Caso 2

- Nó é **preto**
  - não sabemos a cor do seu pai
  - nem se ele é o filho esquerdo ou direito
- Filho esquerdo é **preto**
- Inserimos no filho direito




```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Inserção - Caso 3

```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Inserção - Caso 3

- Nó é **preto**
  - não sabemos a cor do seu pai
  - nem se ele é o filho esquerdo ou direito

```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Inserção - Caso 3

- Nó é **preto**
  - não sabemos a cor do seu pai
  - nem se ele é o filho esquerdo ou direito
- Filho esquerdo é **vermelho**

```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

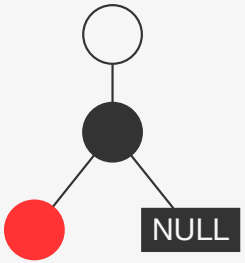
Inserção - Caso 3

- Nó é **preto**
  - não sabemos a cor do seu pai
  - nem se ele é o filho esquerdo ou direito
- Filho esquerdo é **vermelho**
- Inserimos no filho direito

```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Inserção - Caso 3

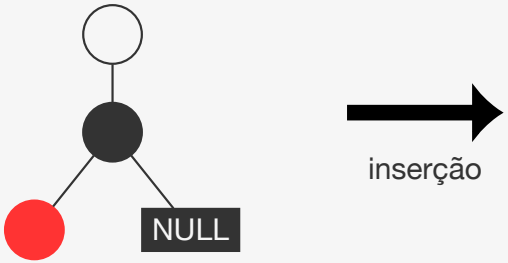
- Nó é **preto**
  - não sabemos a cor do seu pai
  - nem se ele é o filho esquerdo ou direito
- Filho esquerdo é **vermelho**
- Inserimos no filho direito




```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Inserção - Caso 3

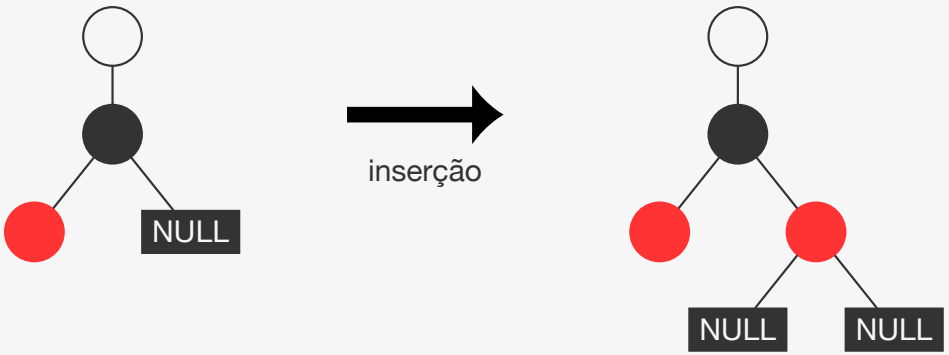
- Nó é **preto**
  - não sabemos a cor do seu pai
  - nem se ele é o filho esquerdo ou direito
- Filho esquerdo é **vermelho**
- Inserimos no filho direito




```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Inserção - Caso 3

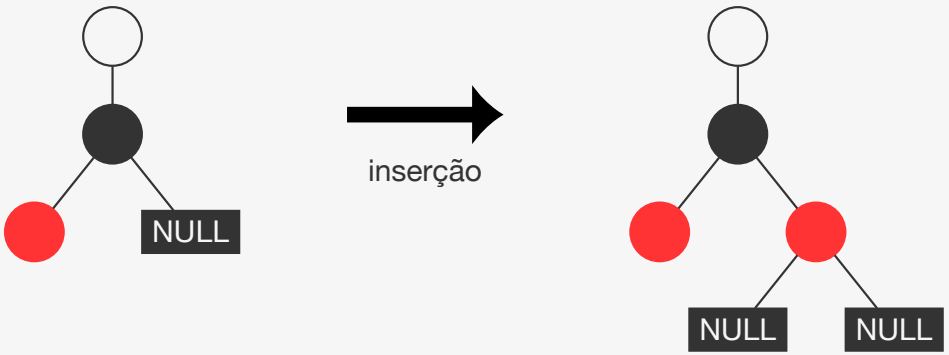
- Nó é **preto**
  - não sabemos a cor do seu pai
  - nem se ele é o filho esquerdo ou direito
- Filho esquerdo é **vermelho**
- Inserimos no filho direito




```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Inserção - Caso 3

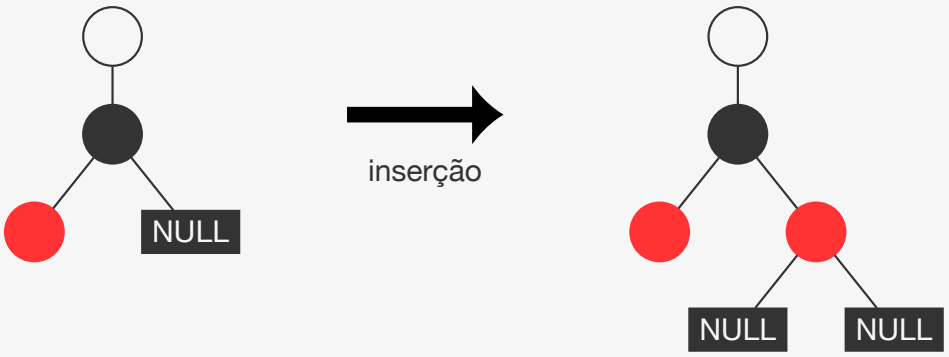
- Nó é **preto**
  - não sabemos a cor do seu pai
  - nem se ele é o filho esquerdo ou direito
- Filho esquerdo é **vermelho**
- Inserimos no filho direito




```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Inserção - Caso 3

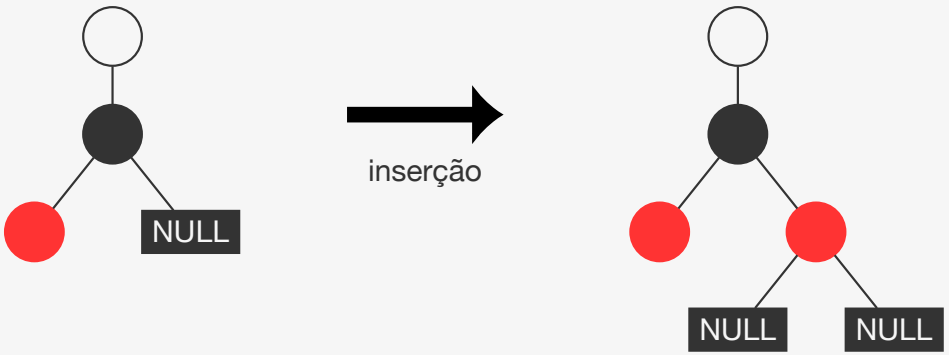
- Nó é **preto**
  - não sabemos a cor do seu pai
  - nem se ele é o filho esquerdo ou direito
- Filho esquerdo é **vermelho**
- Inserimos no filho direito




```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Inserção - Caso 3

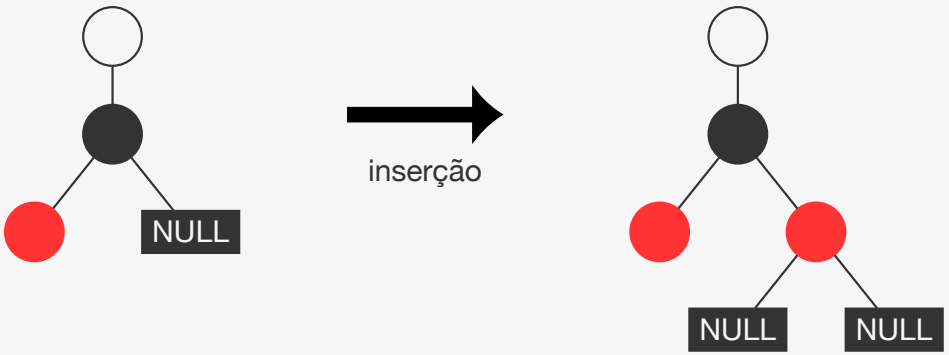
- Nó é **preto**
  - não sabemos a cor do seu pai
  - nem se ele é o filho esquerdo ou direito
- Filho esquerdo é **vermelho**
- Inserimos no filho direito




```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Inserção - Caso 3

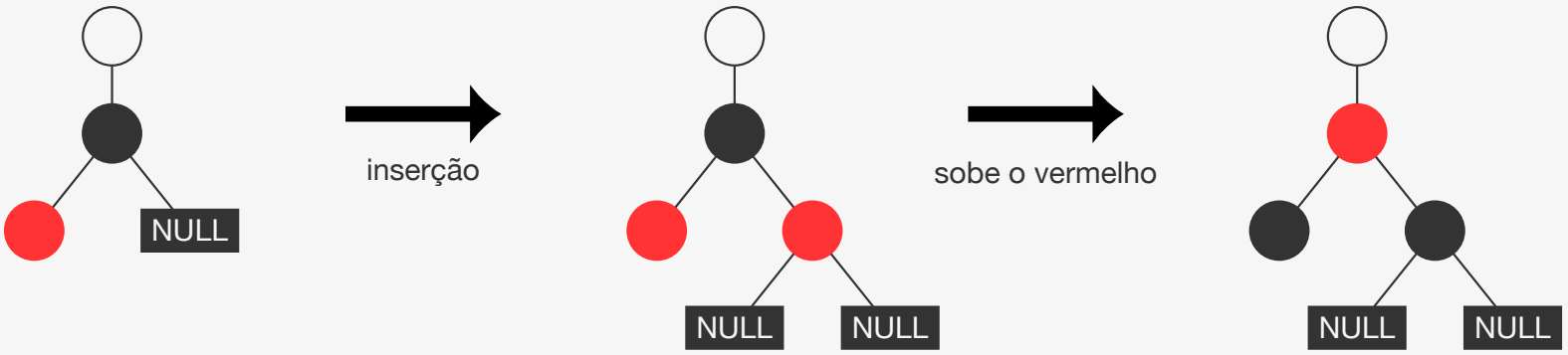
- Nó é **preto**
  - não sabemos a cor do seu pai
  - nem se ele é o filho esquerdo ou direito
- Filho esquerdo é **vermelho**
- Inserimos no filho direito




```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Inserção - Caso 3

- Nó é **preto**
  - não sabemos a cor do seu pai
  - nem se ele é o filho esquerdo ou direito
- Filho esquerdo é **vermelho**
- Inserimos no filho direito




```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Inserção - Caso 4

```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Inserção - Caso 4

- Nó é **vermelho**

```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Inserção - Caso 4

- Nó é **vermelho**
  - seu pai é **preto** (ele não é a raiz - por que?)

```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Inserção - Caso 4

- Nó é **vermelho**
  - seu pai é **preto** (ele não é a raiz - por que?)
  - é o filho esquerdo

```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

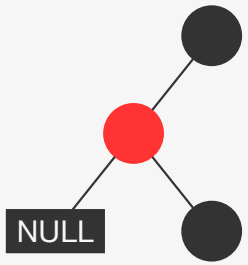
Inserção - Caso 4

- Nó é **vermelho**
  - seu pai é **preto** (ele não é a raiz - por que?)
  - é o filho esquerdo
- Inserimos no filho esquerdo

```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Inserção - Caso 4

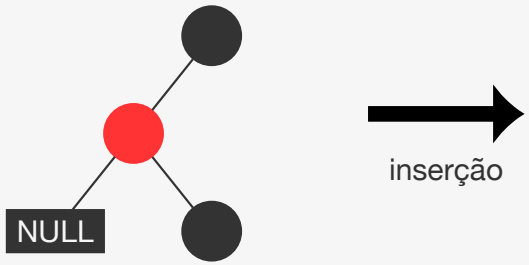
- Nó é **vermelho**
  - seu pai é **preto** (ele não é a raiz - por que?)
  - é o filho esquerdo
- Inserimos no filho esquerdo




```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Inserção - Caso 4

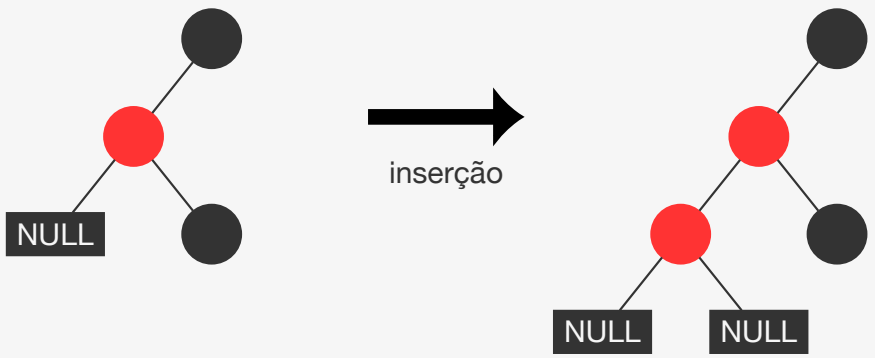
- Nó é **vermelho**
  - seu pai é **preto** (ele não é a raiz - por que?)
  - é o filho esquerdo
- Inserimos no filho esquerdo




```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Inserção - Caso 4

- Nó é **vermelho**
  - seu pai é **preto** (ele não é a raiz - por que?)
  - é o filho esquerdo
- Inserimos no filho esquerdo




```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Inserção - Caso 4

- Nó é **vermelho**
  - seu pai é **preto** (ele não é a raiz - por que?)
  - é o filho esquerdo
- Inserimos no filho esquerdo

```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Inserção - Caso 4

- Nó é **vermelho**
  - seu pai é **preto** (ele não é a raiz - por que?)
  - é o filho esquerdo
- Inserimos no filho esquerdo

```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Inserção - Caso 4

- Nó é **vermelho**
  - seu pai é **preto** (ele não é a raiz - por que?)
  - é o filho esquerdo
- Inserimos no filho esquerdo

```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Inserção - Caso 5

```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Inserção - Caso 5

- o nó é **vermelho**
  - seu pai é **preto** (ele não é a raiz)
  - é o filho esquerdo

```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

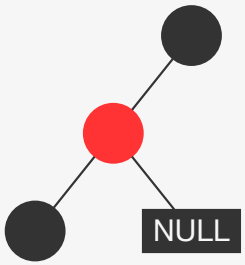
Inserção - Caso 5

- o nó é **vermelho**
  - seu pai é **preto** (ele não é a raiz)
  - é o filho esquerdo
- inserimos no filho direito

```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Inserção - Caso 5

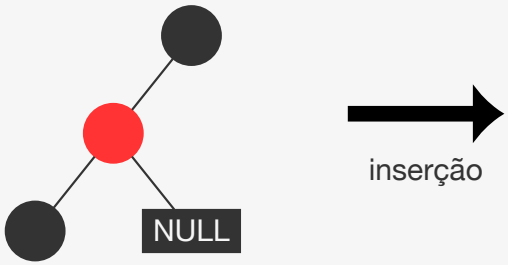
- o nó é **vermelho**
  - seu pai é **preto** (ele não é a raiz)
  - é o filho esquerdo
- inserimos no filho direito




```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Inserção - Caso 5

- o nó é **vermelho**
  - seu pai é **preto** (ele não é a raiz)
  - é o filho esquerdo
- inserimos no filho direito




```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Inserção - Caso 5

- o nó é **vermelho**
  - seu pai é **preto** (ele não é a raiz)
  - é o filho esquerdo
- inserimos no filho direito

```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Inserção - Caso 5

- o nó é **vermelho**
  - seu pai é **preto** (ele não é a raiz)
  - é o filho esquerdo
- inserimos no filho direito

```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

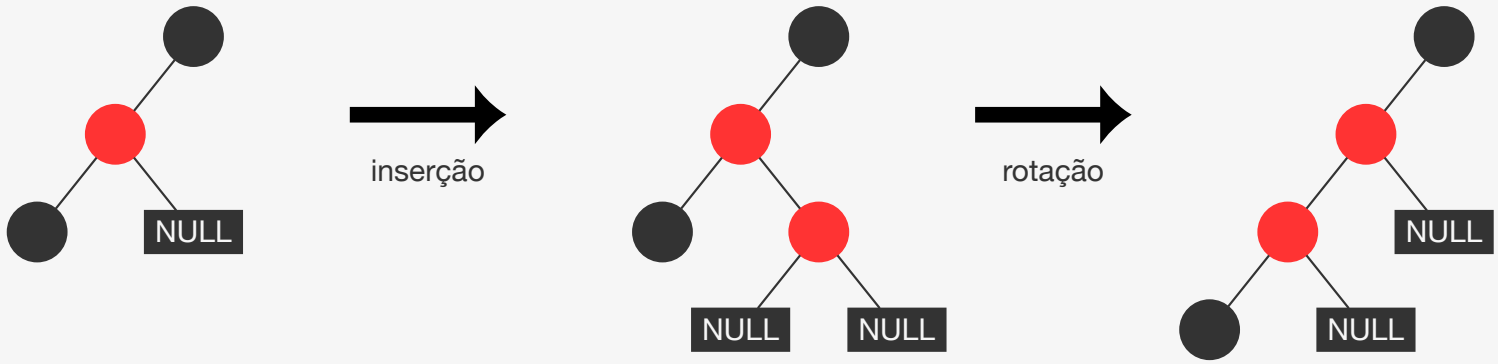
Inserção - Caso 5

- o nó é **vermelho**
  - seu pai é **preto** (ele não é a raiz)
  - é o filho esquerdo
- inserimos no filho direito

```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Inserção - Caso 5

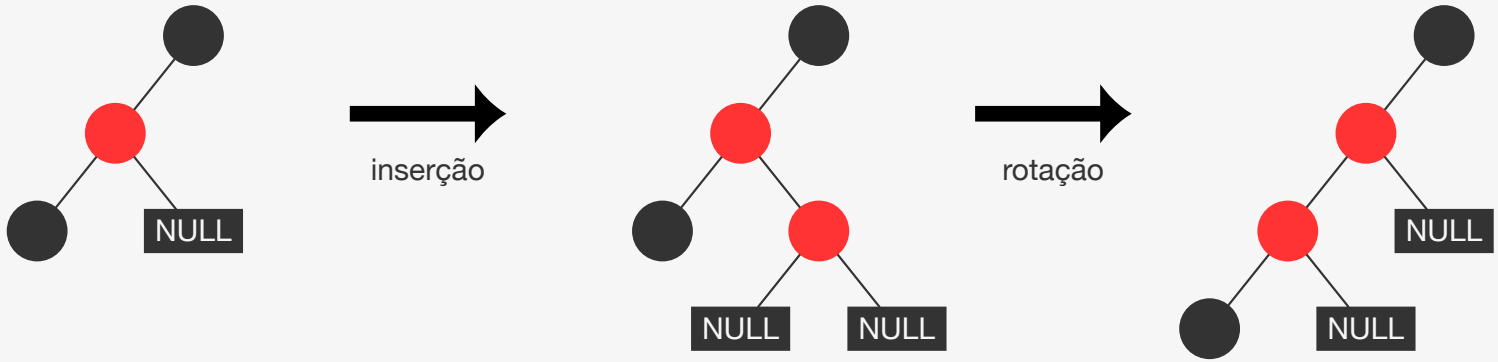
- o nó é **vermelho**
  - seu pai é **preto** (ele não é a raiz)
  - é o filho esquerdo
- inserimos no filho direito




```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Inserção - Caso 5

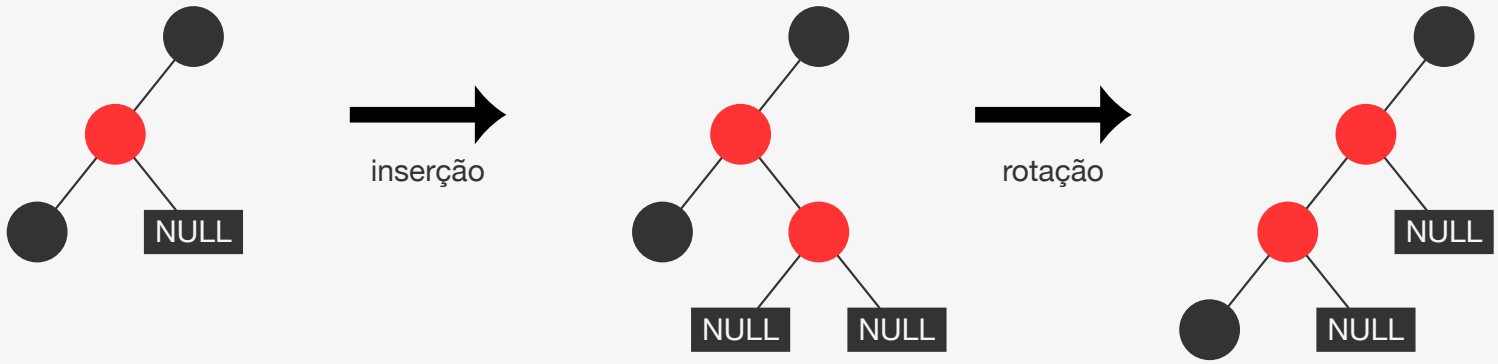
- o nó é **vermelho**
  - seu pai é **preto** (ele não é a raiz)
  - é o filho esquerdo
- inserimos no filho direito




```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Inserção - Caso 5

- o nó é **vermelho**
  - seu pai é **preto** (ele não é a raiz)
  - é o filho esquerdo
- inserimos no filho direito




```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Resolvendo problemas no pai

```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Resolvendo problemas no pai

Quais problemas sobraram para o pai resolver?

```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Resolvendo problemas no pai

Quais problemas sobraram para o pai resolver?

- Talvez o filho direito seja **vermelho** (não é esquerdista)

```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Resolvendo problemas no pai

Quais problemas sobraram para o pai resolver?

- Talvez o filho direito seja **vermelho** (não é esquerdista)
- Só pode ter acontecido porque a cor **vermelha** subiu

```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Resolvendo problemas no pai

Quais problemas sobraram para o pai resolver?

- Talvez o filho direito seja **vermelho** (não é esquerdista)
- Só pode ter acontecido porque a cor **vermelha** subiu

Se o filho esquerdo for **preto**, basta rotacionar para a esquerda

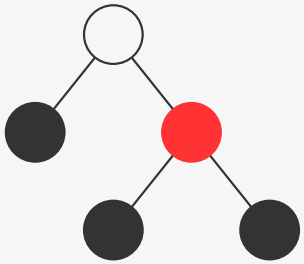
```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Resolvendo problemas no pai

Quais problemas sobraram para o pai resolver?

- Talvez o filho direito seja **vermelho** (não é esquerdista)
- Só pode ter acontecido porque a cor **vermelha** subiu

Se o filho esquerdo for **preto**, basta rotacionar para a esquerda



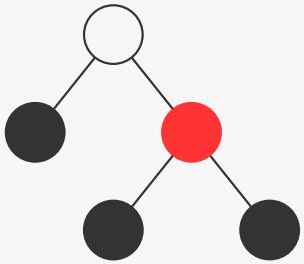
```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Resolvendo problemas no pai

Quais problemas sobraram para o pai resolver?

- Talvez o filho direito seja **vermelho** (não é esquerdista)
- Só pode ter acontecido porque a cor **vermelha** subiu

Se o filho esquerdo for **preto**, basta rotacionar para a esquerda



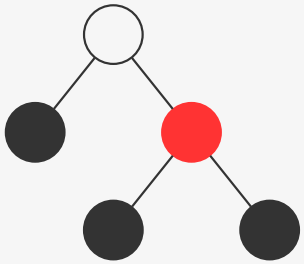
```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Resolvendo problemas no pai

Quais problemas sobraram para o pai resolver?

- Talvez o filho direito seja **vermelho** (não é esquerdista)
- Só pode ter acontecido porque a cor **vermelha** subiu

Se o filho esquerdo for **preto**, basta rotacionar para a esquerda



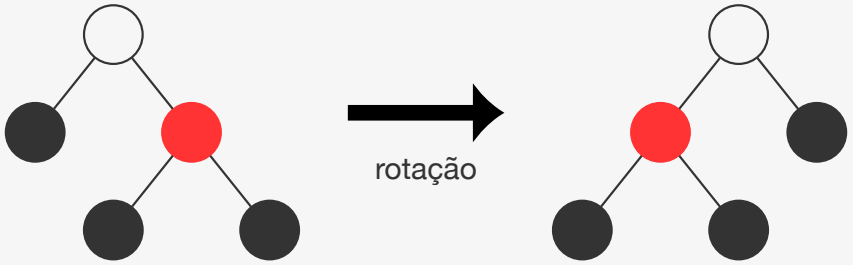
```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Resolvendo problemas no pai

Quais problemas sobraram para o pai resolver?

- Talvez o filho direito seja **vermelho** (não é esquerdista)
- Só pode ter acontecido porque a cor **vermelha** subiu

Se o filho esquerdo for **preto**, basta rotacionar para a esquerda




```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Resolvendo problemas no pai

Quais problemas sobraram para o pai resolver?

- Talvez o filho direito seja **vermelho** (não é esquerdista)
- Só pode ter acontecido porque a cor **vermelha** subiu

Se o filho esquerdo for **preto**, basta rotacionar para a esquerda

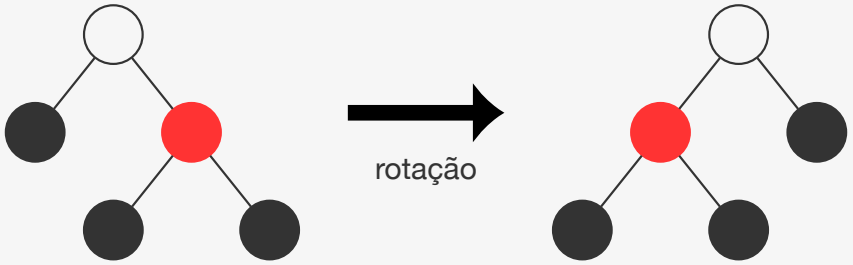
```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Resolvendo problemas no pai

Quais problemas sobraram para o pai resolver?

- Talvez o filho direito seja **vermelho** (não é esquerdista)
- Só pode ter acontecido porque a cor **vermelha** subiu

Se o filho esquerdo for **preto**, basta rotacionar para a esquerda




```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Resolvendo problemas no pai

Quais problemas sobraram para o pai resolver?

- Talvez o filho direito seja **vermelho** (não é esquerdista)
- Só pode ter acontecido porque a cor **vermelha** subiu

```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Resolvendo problemas no pai

Quais problemas sobraram para o pai resolver?

- Talvez o filho direito seja **vermelho** (não é esquerdista)
- Só pode ter acontecido porque a cor **vermelha** subiu

Se o filho esquerdo for **vermelho**, basta subir a cor

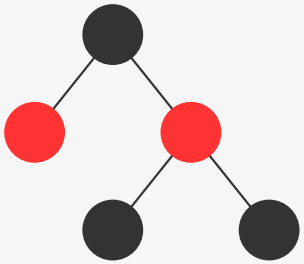
```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Resolvendo problemas no pai

Quais problemas sobraram para o pai resolver?

- Talvez o filho direito seja **vermelho** (não é esquerdista)
- Só pode ter acontecido porque a cor **vermelha** subiu

Se o filho esquerdo for **vermelho**, basta subir a cor



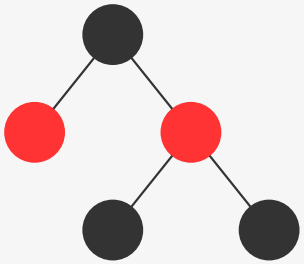
```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Resolvendo problemas no pai

Quais problemas sobraram para o pai resolver?

- Talvez o filho direito seja **vermelho** (não é esquerdista)
- Só pode ter acontecido porque a cor **vermelha** subiu

Se o filho esquerdo for **vermelho**, basta subir a cor



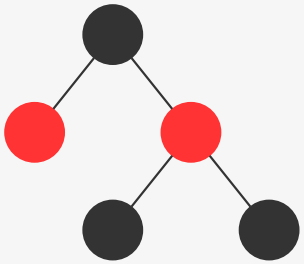
```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Resolvendo problemas no pai

Quais problemas sobraram para o pai resolver?

- Talvez o filho direito seja **vermelho** (não é esquerdista)
- Só pode ter acontecido porque a cor **vermelha** subiu

Se o filho esquerdo for **vermelho**, basta subir a cor



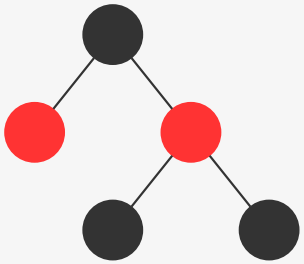
```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Resolvendo problemas no pai

Quais problemas sobraram para o pai resolver?

- Talvez o filho direito seja **vermelho** (não é esquerdista)
- Só pode ter acontecido porque a cor **vermelha** subiu

Se o filho esquerdo for **vermelho**, basta subir a cor



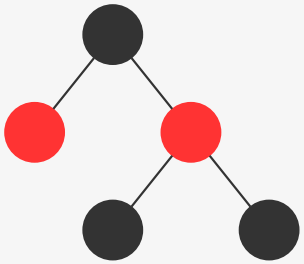
```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Resolvendo problemas no pai

Quais problemas sobraram para o pai resolver?

- Talvez o filho direito seja **vermelho** (não é esquerdista)
- Só pode ter acontecido porque a cor **vermelha** subiu

Se o filho esquerdo for **vermelho**, basta subir a cor



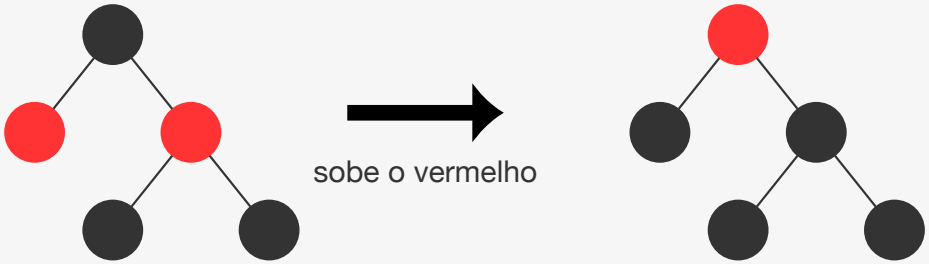
```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Resolvendo problemas no pai

Quais problemas sobraram para o pai resolver?

- Talvez o filho direito seja **vermelho** (não é esquerdista)
- Só pode ter acontecido porque a cor **vermelha** subiu

Se o filho esquerdo for **vermelho**, basta subir a cor




```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Resolvendo problemas no pai

Quais problemas sobraram para o pai resolver?

```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Resolvendo problemas no pai

Quais problemas sobraram para o pai resolver?

- Talvez o filho esquerdo seja **vermelho**

```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Resolvendo problemas no pai

Quais problemas sobraram para o pai resolver?

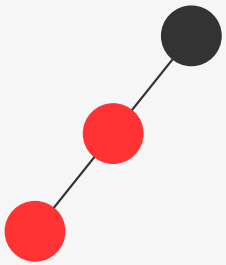
- Talvez o filho esquerdo seja **vermelho**
- E o neto mais a esquerda seja **vermelho**

```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Resolvendo problemas no pai

Quais problemas sobraram para o pai resolver?

- Talvez o filho esquerdo seja **vermelho**
- E o neto mais a esquerda seja **vermelho**

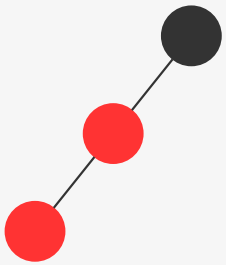


```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Resolvendo problemas no pai

Quais problemas sobraram para o pai resolver?

- Talvez o filho esquerdo seja **vermelho**
- E o neto mais a esquerda seja **vermelho**

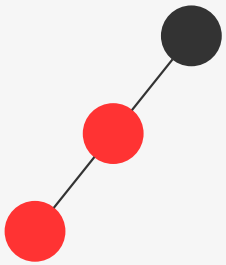


```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Resolvendo problemas no pai

Quais problemas sobraram para o pai resolver?

- Talvez o filho esquerdo seja **vermelho**
- E o neto mais a esquerda seja **vermelho**

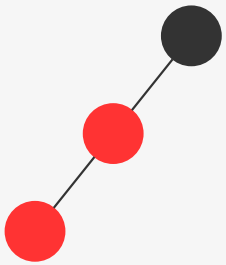


```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Resolvendo problemas no pai

Quais problemas sobraram para o pai resolver?

- Talvez o filho esquerdo seja **vermelho**
- E o neto mais a esquerda seja **vermelho**



```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Resolvendo problemas no pai

Quais problemas sobraram para o pai resolver?

- Talvez o filho esquerdo seja **vermelho**
- E o neto mais a esquerda seja **vermelho**

```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Resolvendo problemas no pai

Quais problemas sobraram para o pai resolver?

- Talvez o filho esquerdo seja **vermelho**
- E o neto mais a esquerda seja **vermelho**

```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Resolvendo problemas no pai

Quais problemas sobraram para o pai resolver?

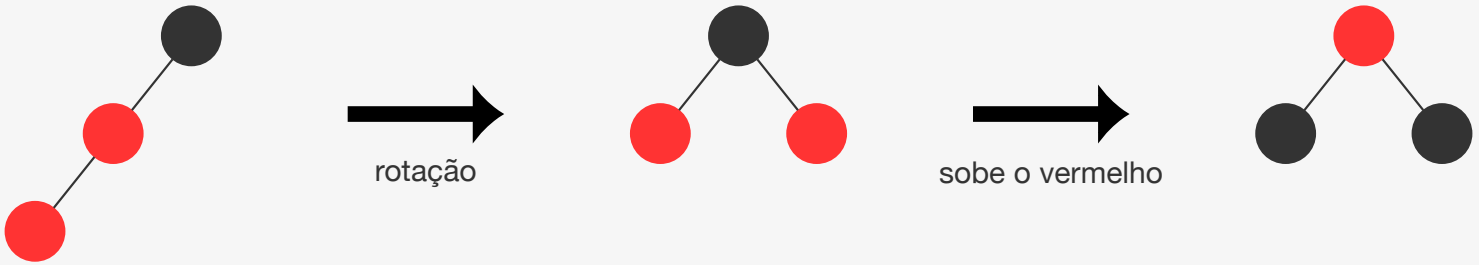
- Talvez o filho esquerdo seja **vermelho**
- E o neto mais a esquerda seja **vermelho**

```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Resolvendo problemas no pai

Quais problemas sobraram para o pai resolver?

- Talvez o filho esquerdo seja **vermelho**
- E o neto mais a esquerda seja **vermelho**




```
1   /* corrige a árvore */
2   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
3     arvore = rotaciona_para_esquerda(arvore);
4   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
5     arvore = rotaciona_para_direita(arvore);
6   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
7     sobe_vermelho(arvore);
```

Inserção - Implementação

Inserção - Implementação

```
1  No* inserir_rec(No *arvore, int chave) {
2    No *novo;
3    if (arvore == NULL) {
4      novo = malloc(sizeof(No));
5      novo->esq = novo->dir = NULL;
6      novo->chave = chave;
7      novo->cor = VERMELHO;
8      return novo;
9    }
10   if (chave < arvore->chave)
11     arvore->esq = inserir_rec(arvore->esq, chave);
12   else
13     arvore->dir = inserir_rec(arvore->dir, chave);
14   /* corrige a árvore */
15   if (ehVermelho(arvore->dir) && ehPreto(arvore->esq))
16     arvore = rotaciona_para_esquerda(arvore);
17   if (ehVermelho(arvore->esq) && ehVermelho(arvore->esq->esq))
18     arvore = rotaciona_para_direita(arvore);
19   if (ehVermelho(arvore->esq) && ehVermelho(arvore->dir))
20     sobe_vermelho(arvore);
21   return arvore;
22 }
```

Remoção

É possível fazer remoções em árvores **rubro-negras**

Remoção

É possível fazer remoções em árvores **rubro-negras**

- Mas não veremos no curso...

Remoção

É possível fazer remoções em árvores **rubro-negras**

- Mas não veremos no curso...

A ideia é basicamente a mesma:

Remoção

É possível fazer remoções em árvores **rubro-negras**

- Mas não veremos no curso...

A ideia é basicamente a mesma:

- encontrar operações que corrijam a árvore

Remoção

É possível fazer remoções em árvores **rubro-negras**

- Mas não veremos no curso...

A ideia é basicamente a mesma:

- encontrar operações que corrijam a árvore
- operações locais que mantém as propriedades globais

Remoção

É possível fazer remoções em árvores **rubro-negras**

- Mas não veremos no curso...

A ideia é basicamente a mesma:

- encontrar operações que corrijam a árvore
- operações locais que mantém as propriedades globais

Sugestão de leitura:

Remoção

É possível fazer remoções em árvores **rubro-negras**

- Mas não veremos no curso...

A ideia é basicamente a mesma:

- encontrar operações que corrijam a árvore
- operações locais que mantém as propriedades globais

Sugestão de leitura:

- Sedgewick e Wayne, Algorithms, 4th Edition, Addison-Wesley Professional, 2011.

Remoção

É possível fazer remoções em árvores **rubro-negras**

- Mas não veremos no curso...

A ideia é basicamente a mesma:

- encontrar operações que corrijam a árvore
- operações locais que mantém as propriedades globais

Sugestão de leitura:

- Sedgewick e Wayne, Algorithms, 4th Edition, Addison-Wesley Professional, 2011.
- Cormen, Leiserson, Rivest e Stein, Introduction to Algorithms, Third Edition, MIT Press, 2009.

Rubro-Negras - Conclusão

As árvores **rubro**-**negras** esquerdistas suportam as seguintes operações:

Rubro-Negras - Conclusão

As árvores **rubro**-**negras** esquerdistas suportam as seguintes operações:

- Busca

Rubro-Negras - Conclusão

As árvores **rubro**-**negras** esquerdistas suportam as seguintes operações:

- Busca
- Inserção

Rubro-Negras - Conclusão

As árvores **rubro**-**negras** esquerdistas suportam as seguintes operações:

- Busca
- Inserção
- Remoção

Rubro-Negras - Conclusão

As árvores **rubro**-**negras** esquerdistas suportam as seguintes operações:

- Busca
- Inserção
- Remoção

todas em tempo $\mathrm{O}(\lg n)$

Rubro-Negras - Conclusão

As árvores **rubro**-**negras** esquerdistas suportam as seguintes operações:

- Busca
- Inserção
- Remoção

todas em tempo $\mathrm{O}(\lg n)$

É uma variante da árvore **rubro**-**negra** com menos operações para corrigir a árvore na inserção e na remoção

Rubro-Negras - Conclusão

As árvores **rubro**-**negras** esquerdistas suportam as seguintes operações:

- Busca
- Inserção
- Remoção

todas em tempo $\mathrm{O}(\lg n)$

É uma variante da árvore **rubro**-**negra** com menos operações para corrigir a árvore na inserção e na remoção

Árvores **rubro**-**negras** são usadas como a árvore padrão no C++, no JAVA, e no kernel do Linux

Outras árvores balanceadas

Existem também outras ABBs balanceadas:

Outras árvores balanceadas

Existem também outras ABBs balanceadas:

- Uma árvore balanceada é uma árvore com altura $\mathrm{O}(\lg n)$

Outras árvores balanceadas

Existem também outras ABBs balanceadas:

- Uma árvore balanceada é uma árvore com altura $\mathrm{O}(\lg n)$

Árvores AVL:

Outras árvores balanceadas

Existem também outras ABBs balanceadas:

- Uma árvore balanceada é uma árvore com altura $\mathrm{O}(\lg n)$

Árvores AVL:

- A altura das subárvores pode variar de no máximo 1

Outras árvores balanceadas

Existem também outras ABBs balanceadas:

- Uma árvore balanceada é uma árvore com altura $\mathrm{O}(\lg n)$

Árvores AVL:

- A altura das subárvores pode variar de no máximo 1
- Tem altura $\mathrm{O}(\lg n)$

Outras árvores balanceadas

Existem também outras ABBs balanceadas:

- Uma árvore balanceada é uma árvore com altura $\mathrm{O}(\lg n)$

Árvores AVL:

- A altura das subárvores pode variar de no máximo 1
- Tem altura $\mathrm{O}(\lg n)$

ABB aleatorizada:

Outras árvores balanceadas

Existem também outras ABBs balanceadas:

- Uma árvore balanceada é uma árvore com altura $\mathrm{O}(\lg n)$

Árvores AVL:

- A altura das subárvores pode variar de no máximo 1
- Tem altura $\mathrm{O}(\lg n)$

ABB aleatorizada:

- Decide de maneira aleatória como inserir o nó

Outras árvores balanceadas

Existem também outras ABBs balanceadas:

- Uma árvore balanceada é uma árvore com altura $\mathrm{O}(\lg n)$

Árvores AVL:

- A altura das subárvores pode variar de no máximo 1
- Tem altura $\mathrm{O}(\lg n)$

ABB aleatorizada:

- Decide de maneira aleatória como inserir o nó
  - inserção normal como folha

Outras árvores balanceadas

Existem também outras ABBs balanceadas:

- Uma árvore balanceada é uma árvore com altura $\mathrm{O}(\lg n)$

Árvores AVL:

- A altura das subárvores pode variar de no máximo 1
- Tem altura $\mathrm{O}(\lg n)$

ABB aleatorizada:

- Decide de maneira aleatória como inserir o nó
  - inserção normal como folha
  - inserção na raiz - rotações trazem o nó até a raiz

Outras árvores balanceadas

Existem também outras ABBs balanceadas:

- Uma árvore balanceada é uma árvore com altura $\mathrm{O}(\lg n)$

Árvores AVL:

- A altura das subárvores pode variar de no máximo 1
- Tem altura $\mathrm{O}(\lg n)$

ABB aleatorizada:

- Decide de maneira aleatória como inserir o nó
  - inserção normal como folha
  - inserção na raiz - rotações trazem o nó até a raiz
- Altura "média" (esperada): $\mathrm{O}(\lg n)$

Outras árvores balanceadas

Existem também outras ABBs balanceadas:

- Uma árvore balanceada é uma árvore com altura $\mathrm{O}(\lg n)$

Árvores AVL:

- A altura das subárvores pode variar de no máximo 1
- Tem altura $\mathrm{O}(\lg n)$

ABB aleatorizada:

- Decide de maneira aleatória como inserir o nó
  - inserção normal como folha
  - inserção na raiz - rotações trazem o nó até a raiz
- Altura "média" (esperada): $\mathrm{O}(\lg n)$

Árvores Splay:

Outras árvores balanceadas

Existem também outras ABBs balanceadas:

- Uma árvore balanceada é uma árvore com altura $\mathrm{O}(\lg n)$

Árvores AVL:

- A altura das subárvores pode variar de no máximo 1
- Tem altura $\mathrm{O}(\lg n)$

ABB aleatorizada:

- Decide de maneira aleatória como inserir o nó
  - inserção normal como folha
  - inserção na raiz - rotações trazem o nó até a raiz
- Altura "média" (esperada): $\mathrm{O}(\lg n)$

Árvores Splay:

- Sobe os nós no caminho da busca/inserção

Outras árvores balanceadas

Existem também outras ABBs balanceadas:

- Uma árvore balanceada é uma árvore com altura $\mathrm{O}(\lg n)$

Árvores AVL:

- A altura das subárvores pode variar de no máximo 1
- Tem altura $\mathrm{O}(\lg n)$

ABB aleatorizada:

- Decide de maneira aleatória como inserir o nó
  - inserção normal como folha
  - inserção na raiz - rotações trazem o nó até a raiz
- Altura "média" (esperada): $\mathrm{O}(\lg n)$

Árvores Splay:

- Sobe os nós no caminho da busca/inserção
- Nós mais acessados ficam mais próximos da raiz

Outras árvores balanceadas

Existem também outras ABBs balanceadas:

- Uma árvore balanceada é uma árvore com altura $\mathrm{O}(\lg n)$

Árvores AVL:

- A altura das subárvores pode variar de no máximo 1
- Tem altura $\mathrm{O}(\lg n)$

ABB aleatorizada:

- Decide de maneira aleatória como inserir o nó
  - inserção normal como folha
  - inserção na raiz - rotações trazem o nó até a raiz
- Altura "média" (esperada): $\mathrm{O}(\lg n)$

Árvores Splay:

- Sobe os nós no caminho da busca/inserção
- Nós mais acessados ficam mais próximos da raiz
- Não é balanceada, mas o custo de m inserções/buscas em uma árvore Splay com n nós é $\mathrm{O}((n+m)\lg(n+m))$