AED2 - Aula 0508

Árvores rubro-negras

Árvores rubro-negras são árvores binárias de busca balanceadas,

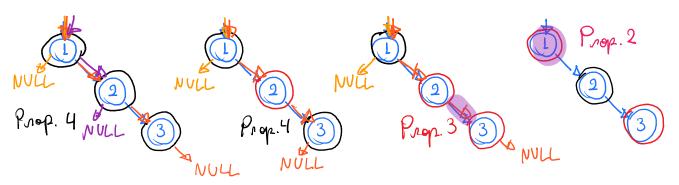
• que usam rotações mas não usam fator de balanceamento.

Definição:

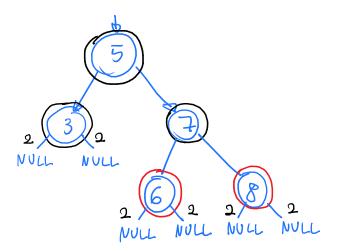
- 1. Cada nó é vermelho ou preto.
- 2. Raiz é sempre preta.
- 3. Dois nós vermelhos não podem ser adjacentes,
 - o u seja, um nó vermelho só pode ter filhos pretos.
- 4. Todo caminho da raiz até um apontador NULL (caminho raiz-NULL)
 - o tem o mesmo número de nós pretos.
 - Pense nesses caminhos como buscas mal sucedidas.

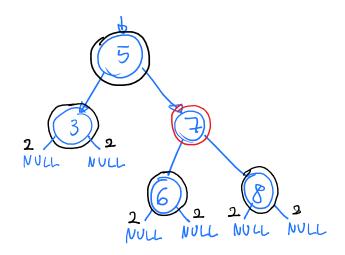
Para ganhar intuição de que essas propriedades levam a uma árvore balanceada

• note que uma lista com três nós já não pode ser uma árvore rubro-negra.



Exemplos de árvores rubro-negras:





NULL

Altura máxima de árvores rubro-negras

to # de arrator

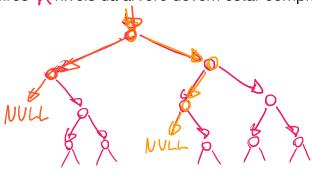
Lembre que a altura de urna árvore é igual

- o ao comprimento do maior caminho raiz-NULL.
- Assim, queremos limitar superiormente o comprimento de qualquer caminho.

Observe que, se todo caminho raiz-NULL de uma árvore tem pelo menos Knós,

• então os primeiros K níveis da árvore devem estar completos.

a = 0 b
centra-positivo
~ b = 0 ~ a



K=4

Caso contrário, haveria um caminho da raiz até o nó ausente, resultando

- o em um caminho raiz-NULL de comprimento menor que K-1
- lembrando que comprimento de um caminho Pé |V(P) | -↓

Supondo que os primeiros K níveis da árvore estão completos,

- o número de nós
 M desta árvore é maior ou igual que
 - o número de nós numa árvore binária completa de altura K-L
 - ou seja, $\eta \geqslant 2^{k} 1 = 2^{e} + 2^{1} + 2^{2} + ... + 2^{k-1}$
- Portanto, 2 k ≤ n+1 = N K ≤ lg(n+1)

Note que, numa árvore rubro-negra, pela propriedade 4 da definição,

- todo caminho raiz-NULL tem um mesmo número de nós pretos.
 - o Digamos que este número é 🤘
- Assim, só os nós pretos já preenchem os primeiros K níveis da árvore.

Desta observação, temos um limitante superior para o número de nós pretos

- No entanto, queremos um limitante para o número total de nós
 - o em qualquer caminho, já que isso limita a altura da árvore.

Pelas propriedades 1 e 3 da definição, todo caminho tem no máximo

- um nó vermelho para cada nó preto,
 - o já que os caminhos começam na raiz, que é preta,
- e não são permitidos dois nós vermelhos consecutivos.

Assim, o número total de nós em qualquer caminho raiz-NULL da árvore é

• no máximo 2 vezes o número de nós pretos $= 2 \times 4 2 \lg (n+1)$

Como a altura da árvore é igual ao comprimento do maior caminho raiz-NULL,

- e o comprimento de um caminho é menor que o seu número de nós,
 - o resultado segue.

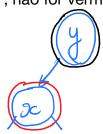
Inserção em árvores rubro-negras

A ideia geral é inserir o novo nó como uma folha vermelha,

- o após percorrer um caminho descendente na árvore,
- e usar recolorações e rotações para restabelecer as propriedades,
 - o ao percorrer o caminho no sentido ascendente.
- A seguir vamos analisar alguns casos que podem ocorrer,
 - o na volta da recursão, quando percorremos o caminho ascendente,
 - sempre considerando que o nó corrente é 💢

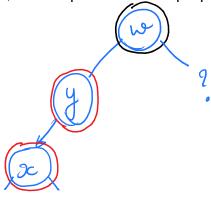


Caso 1: se , o pai de , não for vermelho,



as propriedades da definição estão mantidas e nada precisa ser feito.

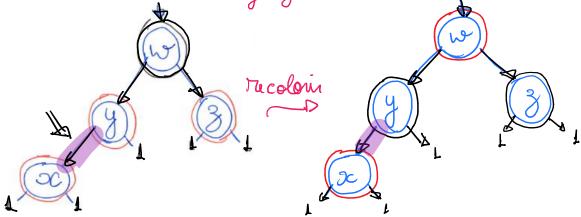
Caso 2: se ¼ é vermelho, temos que restaurar a propriedade 3.



- Sabemos que / não é a raiz, por conta da propriedade 2,
 - o e que 6, o pai de 4, é preto, por conta da propriedade 3.

Caso 2.1: tem outro filho 3 que tem cor vermelha.

Neste caso vamos recolorir y e y para preto e para vermelho.



- Note que preservamos a propriedade 4, pois
 - o todo caminho que passava por

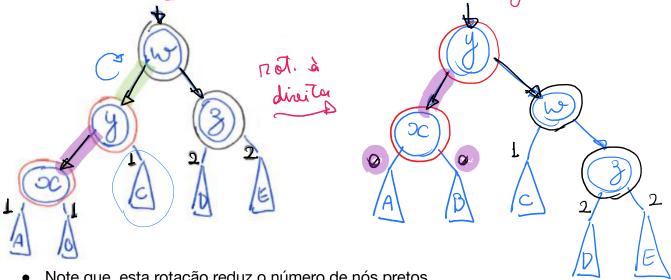
 passa por

 y ou por

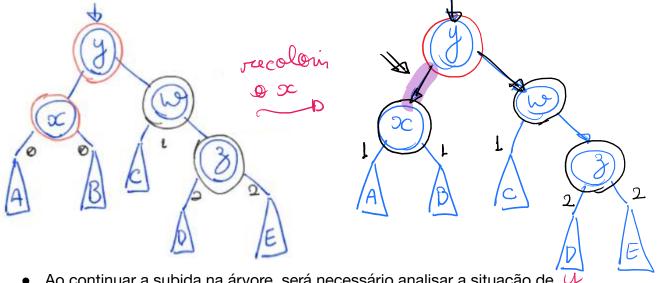
 }
 - e restauramos a propriedade 3 entre x e
- Ao continuar a subida na árvore, será necessário analisar a situação de 🔑
 - o que ficou vermelho, pois ele pode violar a propriedade 3.
- Se V for a raiz da árvore, basta mudar a cor dele para preto.
 - Quiz1: Por que isso não viola as propriedades?

Caso 2.2: P não tem outro filho vermelho.

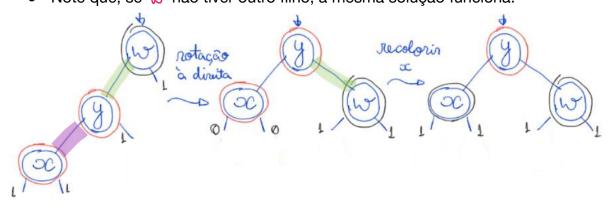
- Neste caso, pode ter outro filho preto ou n\u00e3o ter outro filho.
 - Primeiro, vamos considerar a presença de um nó preto
- Fazemos uma rotação à direita que inverte a relação entre 🔑 e



- Note que, esta rotação reduz o número de nós pretos
 - o nos caminhos que vão da raiz até os filhos de 🍼
- Então mudamos a cor de para preto, o que resolve tanto o problema
 - dos vermelhos adjacentes quanto do número de pretos nos caminhos.



- Ao continuar a subida na árvore, será necessário analisar a situação de
 - que é vermelho e se tornou a raiz da subárvore,
 - pois ele pode violar a propriedade 3 com relação a seu pai.
 - Se \oint for a nova raiz da árvore, basta mudar a cor dele para preto.
- Note que, se 🔑 não tiver outro filho, a mesma solução funciona.



Quiz2: Como resolver o caso em que x é inserido como filho direito de y?

Inserção em árvores rubro-negras esquerdistas

Embora nossos tratamentos dos casos anteriores estejam corretos,

- uma vez que eles restauram as propriedades da árvore rubro-negra,
 - o sua implementação é um tanto complicada,
 - por ser necessário manipular antecessores do nó corrente.
- Por isso, vamos estudar a implementação da inserção
 - o de um tipo especial de árvore rubro-negra,
- que é conhecida como árvore rubro-negra esquerdista,
 - o pois os nós vermelhos sempre são filhos esquerdos.

A seguir o código para a estrutura de um nó de árvore rubro-negra:

A seguir apresentamos a implementação recursiva,

• introduzida por Sedgewick, de uma árvore rubro-negra esquerdista.

```
Noh *novoNoh(Chave chave, Item conteudo) {
   Noh *novo;
   novo = (Noh *)malloc(sizeof(Noh));

   novo->vermelho = 1;
   novo->chave = chave;
   novo->conteudo = conteudo;
   novo->esq = NULL;
   novo->dir = NULL;
   return novo;
}
```

```
Arvore inserir(Arvore r) Chave chave, Item conteudo) {
Noh *novo = novoNoh(chave, conteudo);
return insereRN(r) novo);
}

Arvore insereRN(Noh *r, Noh *novo) {

if (r == NULL) { // subárvore era vazia
novo->pai = NULL;
return novo;
}

if (novo->chave (= r->chave) { // desce à esquerda

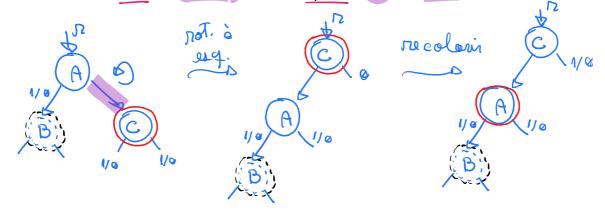
r->esq = insereRN(r->esq, novo);
r->esq->pai = r;
}

else { // desce à direita
r->dir = insereRN(r->dir, novo);

r->dir->pai = r;
}
```

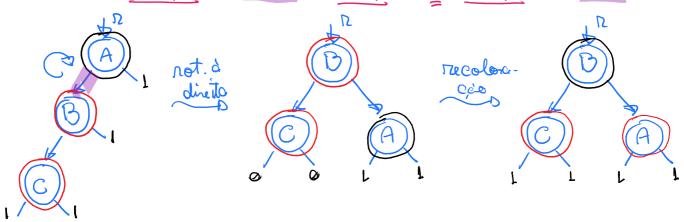
Estamos na volta da recursão (caminho ascendente) e vamos tratar alguns casos:

• Se filho direito é vermelho e filho esquerdo não é vermelho.



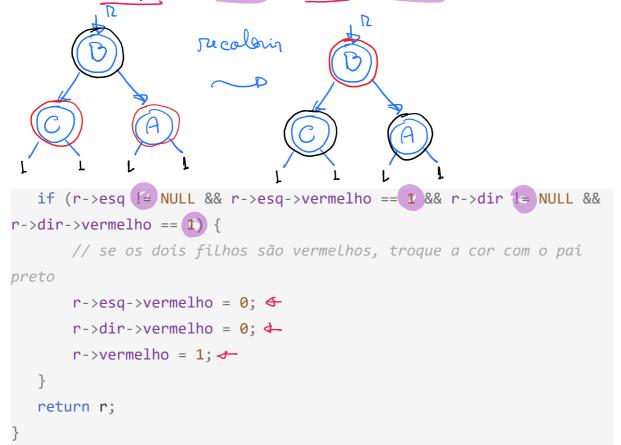
- Note que A pode ser preto ou vermelho,
 - e que sua cor é herdada por C.

• Se filho esquerdo é vermelho e filho esquerdo do filho esquerdo é vermelho.



```
if (r->esq != NULL && r->esq->vermelho == 1 && r->esq->esq !=
NULL && r->esq->esq->vermelho == 1) {
    // se filho e neto esquerdos forem vermelhos faz rotação a
direita e recolore de acordo
    recolore de acordo
    recolore lo;    recolore lo;    recolore lo;    recolore lo;    recolore lo;    recolore lo l
```

• Se filho esquerdo é vermelho e filho direito é vermelho.



É interessante analisar como as operações anteriores

- garantem tanto as propriedades tradicionais das árvores rubro-negras,
 - o quanto a propriedade adicional da árvore rubro-negra esquerdista,
 - i.e., os nós vermelhos sempre são filhos esquerdos.