# UFPI – CCN – DC
# Estrutura de Dados

# Árvores Rubro-Negras

**Prof. Raimundo Moura**
rsm@ufpi.edu.br

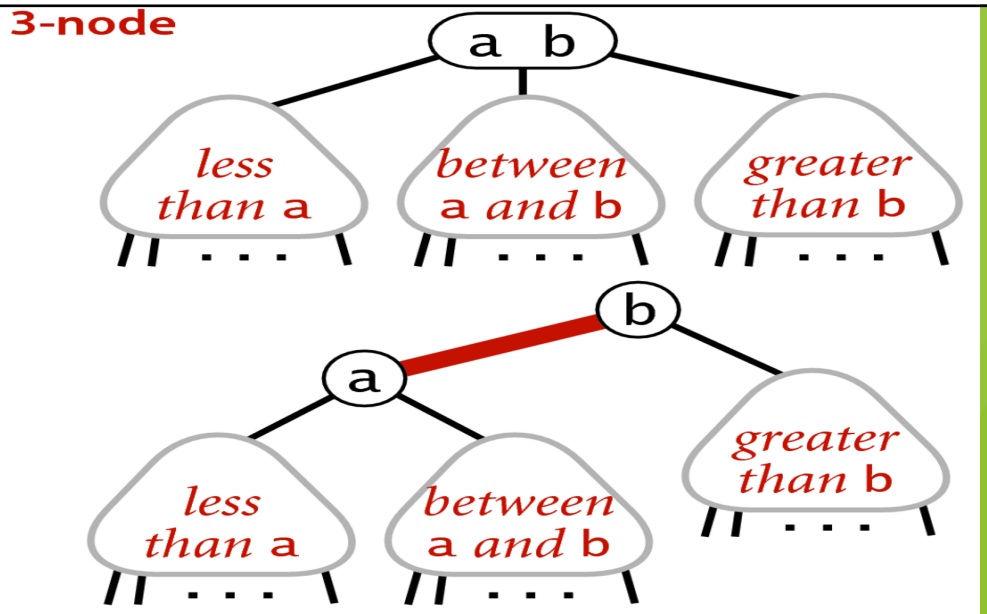
1

## Árvore Rubro-Negra

- Uma BST rubro-negra *(red-black BST),* também conhecida como BST vermelho-preta, é uma BST que *simula* uma árvore 2-3.
- Cada nó duplo da árvore 2-3 é representado por dois nós simples ligados por um link *rubro*.

2

# Árvore Rubro-Negra




3

## Árvore Rubro-Negra: definição

- Uma *BST rubro-negra* é uma BST cujos links são negros e rubros e têm as seguintes popriedades:
  - links rubros se inclinam para a esquerda;
  - nenhum nó incide em dois links rubros;
  - balanceamento negro perfeito: todo caminho da raiz até um link *null* tem o mesmo número de links negros.

4

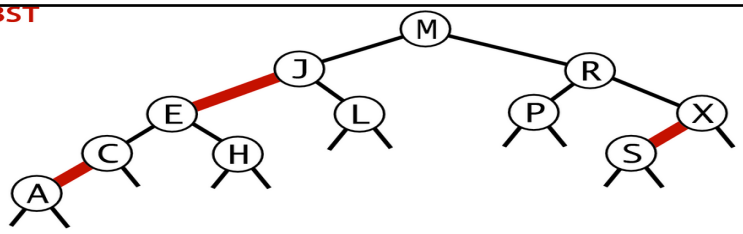
# Árvore Rubro-Negra: curiosidade

- Se os links rubros forem desenhados horizontalmente e depois contraídos, teremos uma árvore 2-3.

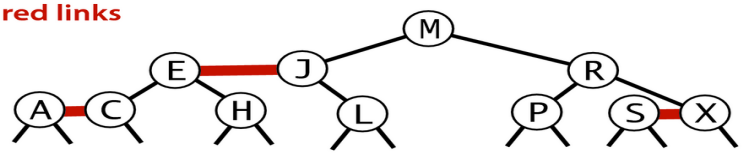
5

# Árvore Rubro-Negra: Anatomia

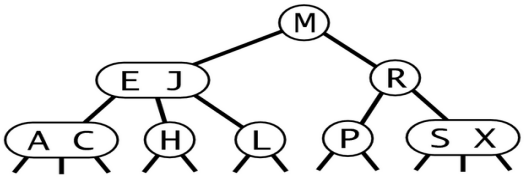
red-black BST




horizontal red links



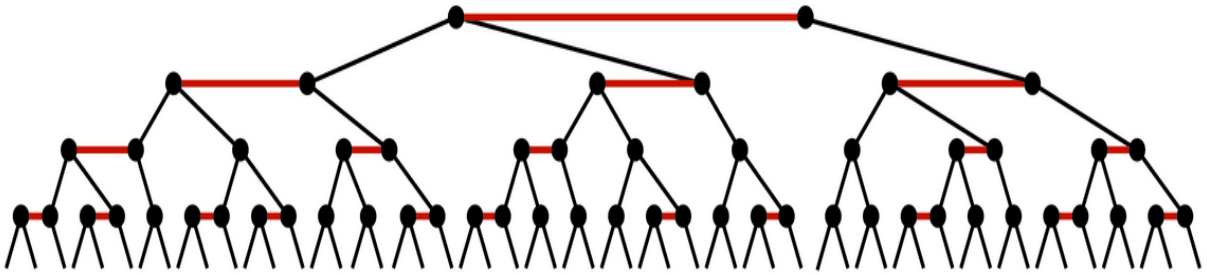

2-3 tree




1-1 correspondence between red-black BSTs and 2-3 trees

6

## TODOS AS LINKS NULL ESTÃO À MESMA PROFUNDIDADE NEGRA



A red-black tree with horizontal red links is a 2-3 tree

7

## Árvore Rubro-Negra: definições

- *Profundidade negra de um nó x:* é o número de links negros no caminho da raiz até x.
- *Altura negra da árvore:* é o máximo da profundidade negra de todos os nós.

8

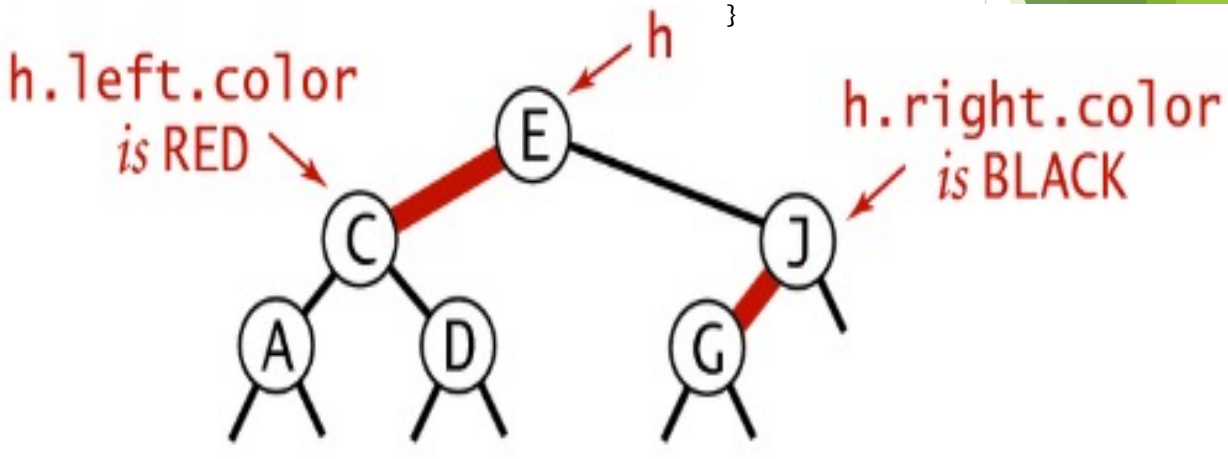
## Árvore Rubro-Negra: considerações

- É inconveniente armazenar a cor de um *link* na estrutura de dados; é mais simples armazenar essa informação nos nós.
- A *cor de um nó* é a cor do único link que entra nele.
- A raiz é considerada negra.

9

## Árvore Rubro-Negra: Representação dos nós

```
class Node {
   Key chave;
   Value valor;
   Node left, right;
   int n;
   boolean cor;
}
```




10

```
private static final boolean RED = true;
private static final boolean BLACK = false;

private class Node {
   Key     key;
   Value   val;
   Node    left, right;
   int     N;              // número de nós nesta subárvore
   boolean color;          // cor do link que aponta para este nó

   Node(Key key, Value val, int N, boolean color) {
      this.key   = key;
      this.val   = val;
      this.N     = N;
      this.color = color;
   }
}

private boolean isRed(Node x) {
   if (x == null) return false;
   return x.color == RED;
}
```

11

## Árvore Rubro-Negra: buscas e inserções

- O código de busca *(get)* para BSTs rubro-negras é exatamente igual ao das BSTs comuns!
- O código de inserção *(put)* é complicado; ele depende de operações de *rotação*.

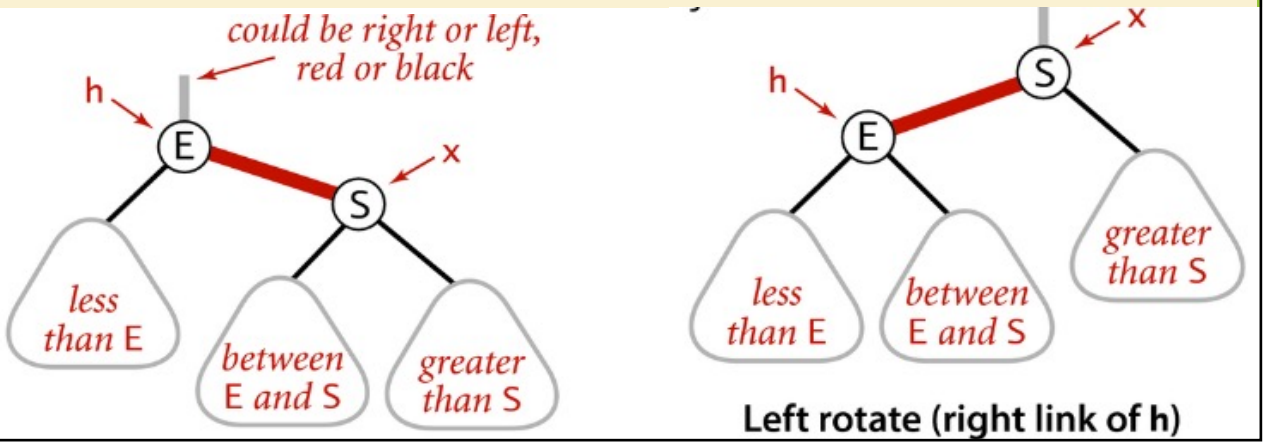
12

# Árvore Rubro-Negra: inserção

- Durante uma operação de inserção, podemos ter, temporariamente, um link rubro inclinado para a direita ou dois links rubros incidindo no mesmo nó. Para corrigir isso, usamos rotações.

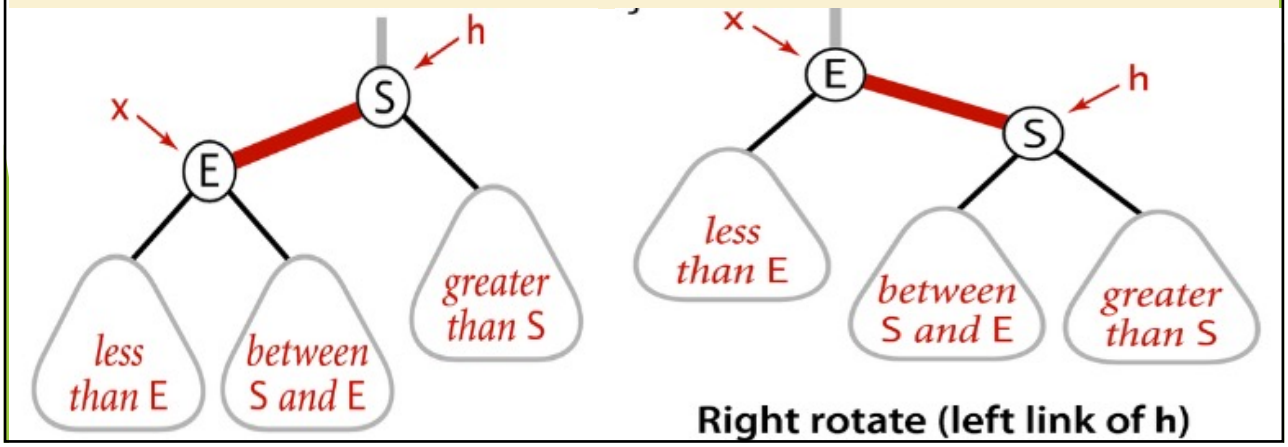
13

- Rotação esquerda (ou anti-horária) em torno de um nó h: o filho direito de h sobe e adota h como seu filho esquerdo. Continuamos tendo uma BST com os mesmos nós, mas raiz diferente:




14

- Rotação direita (ou horária) em torno de um nó h: o filho esquerdo de h sobe e adota h como seu filho direito. Continuamos tendo uma BST com os mesmos nós, mas raiz diferente:




15

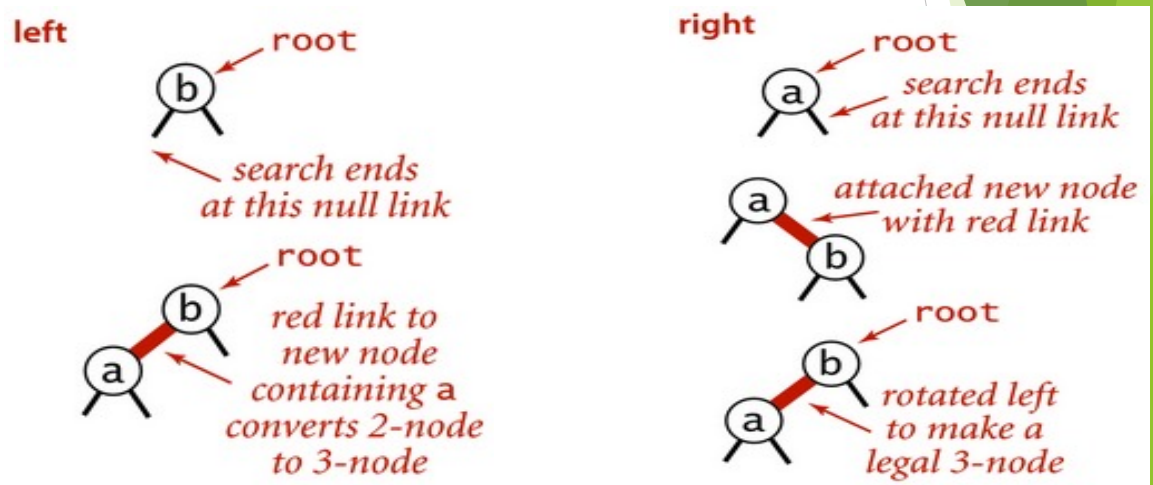
## Árvore Rubro-Negra:

- As operações de rotação são locais (só envolvem um nó e seus vizinhos). Depois de uma rotação, continuamos tendo uma BST com balanceamento negro perfeito. Mas a operação pode ter criado um link rubro inclinado para a lado errado ou dois links rubros seguidos. (Isso será corrigido mais adiante.)

16

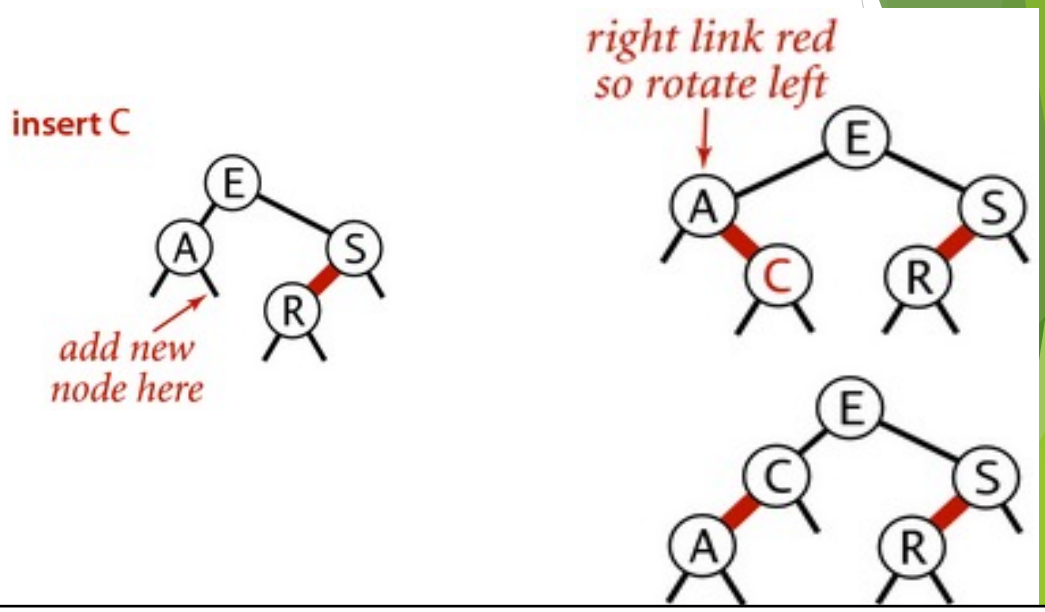
# Árvore Rubro-Negra: inserção

- Um novo nó sempre se liga à árvore por um link *rubro*.



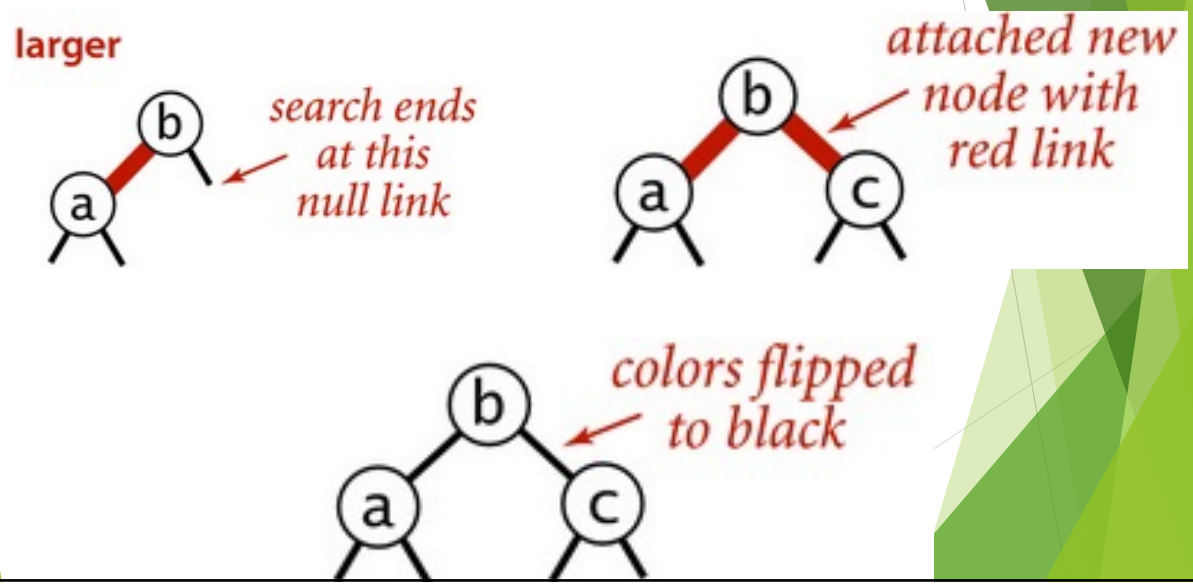

17

# Árvore Rubro-Negra: inserção



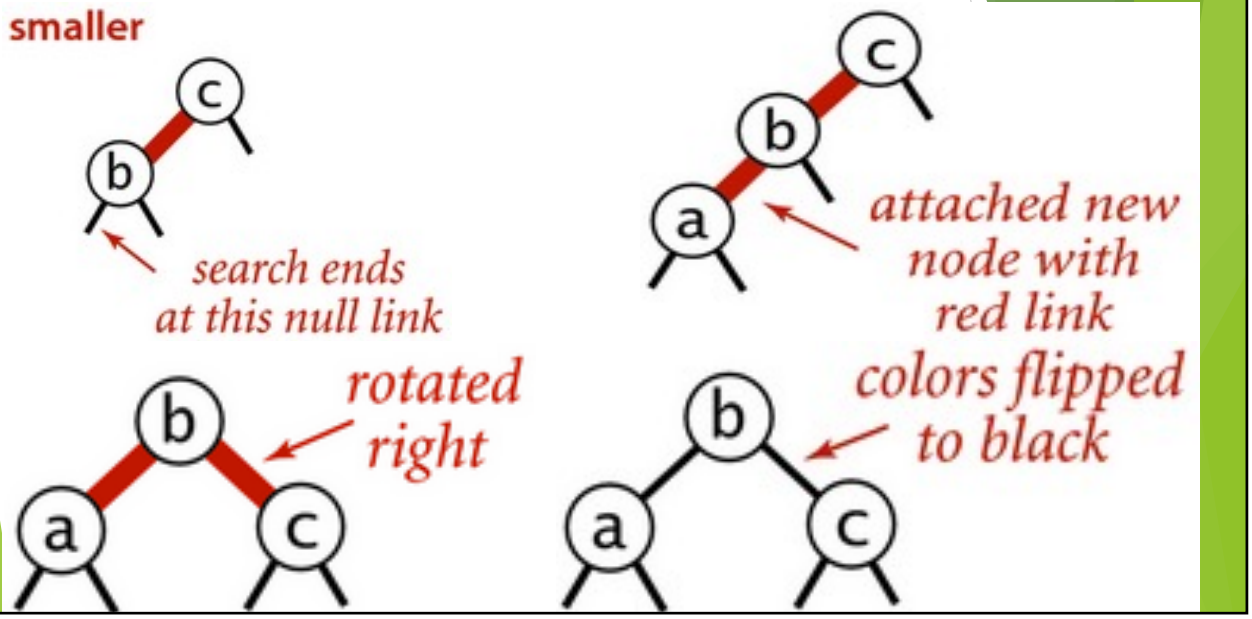

18

# Árvore Rubro-Negra: inserção 1



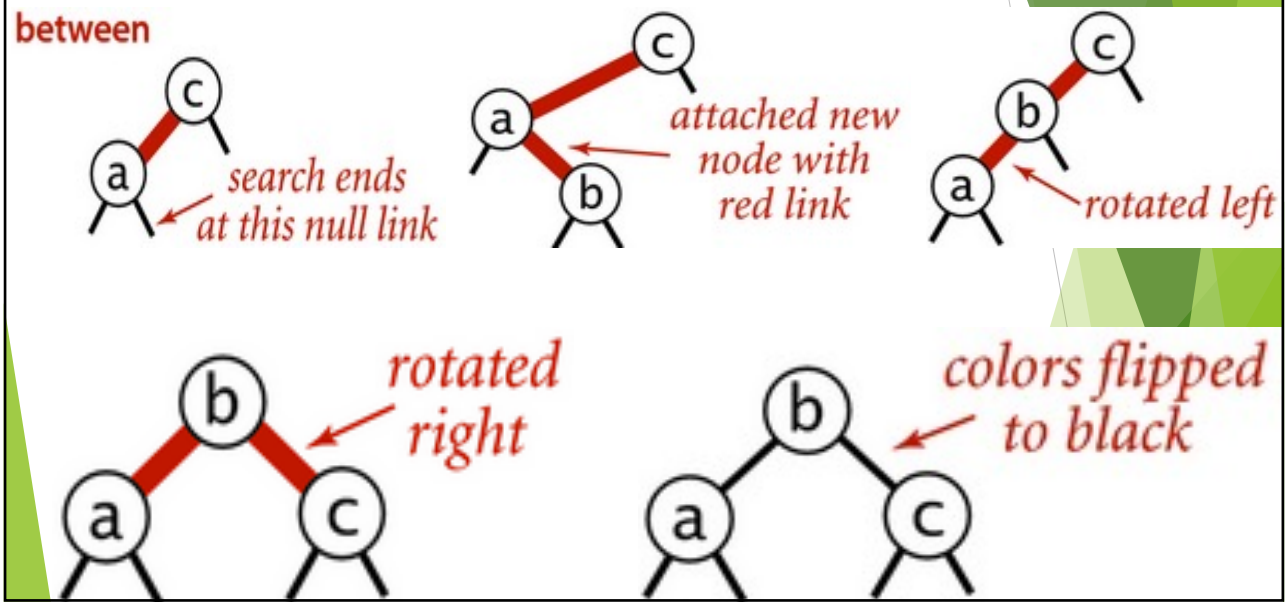

19

# Árvore Rubro-Negra: inserção 2



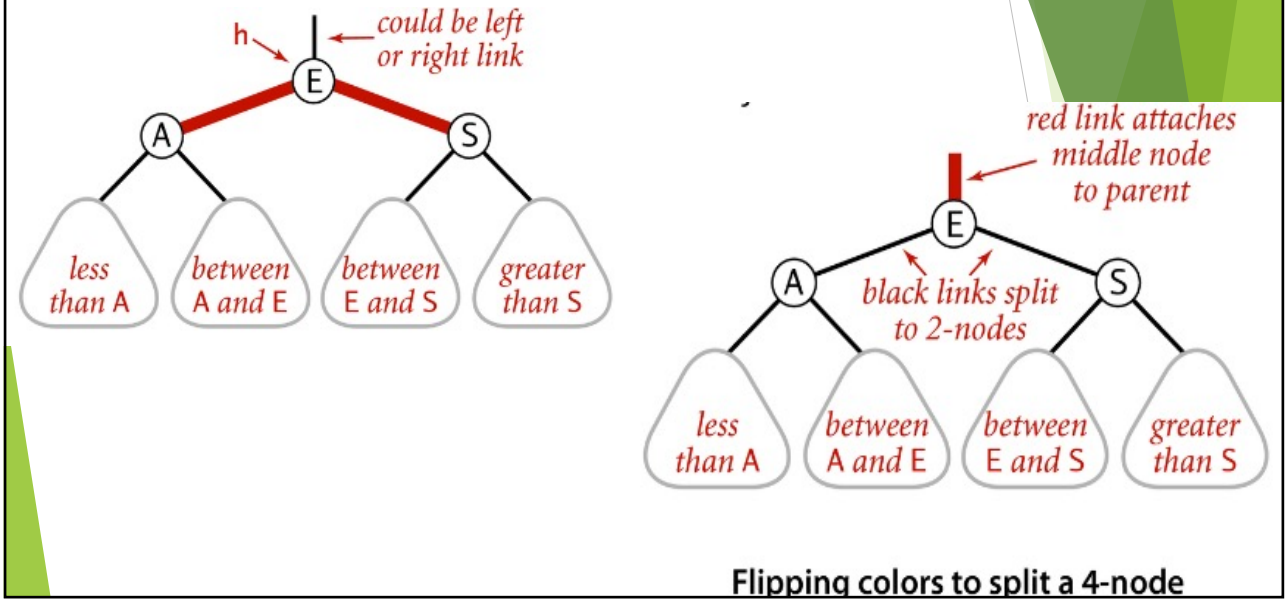

20

# Árvore Rubro-Negra: inserção 3




21

# Árvore Rubro-Negra: inversão de cores




Flipping colors to split a 4-node

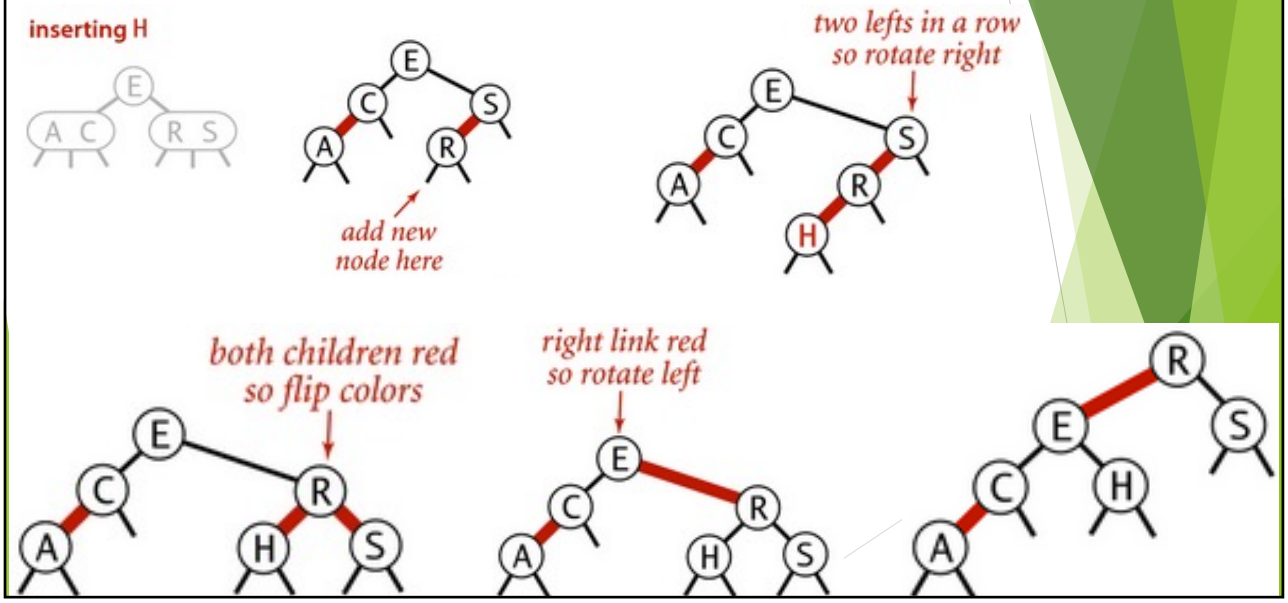
22

## Árvore Rubro-Negra: inversão de cores

- A operação é local (só envolve um nó e seus vizinhos) e preserva o balanceamento negro perfeito (embora possa ter criado dois links rubros incidentes no mesmo nó). A altura negra da árvore aumenta em 1 se e somente se h é a raiz da árvore.

23

## Árvore Rubro-Negra: inversão de cores




24

# Árvore Rubro-Negra

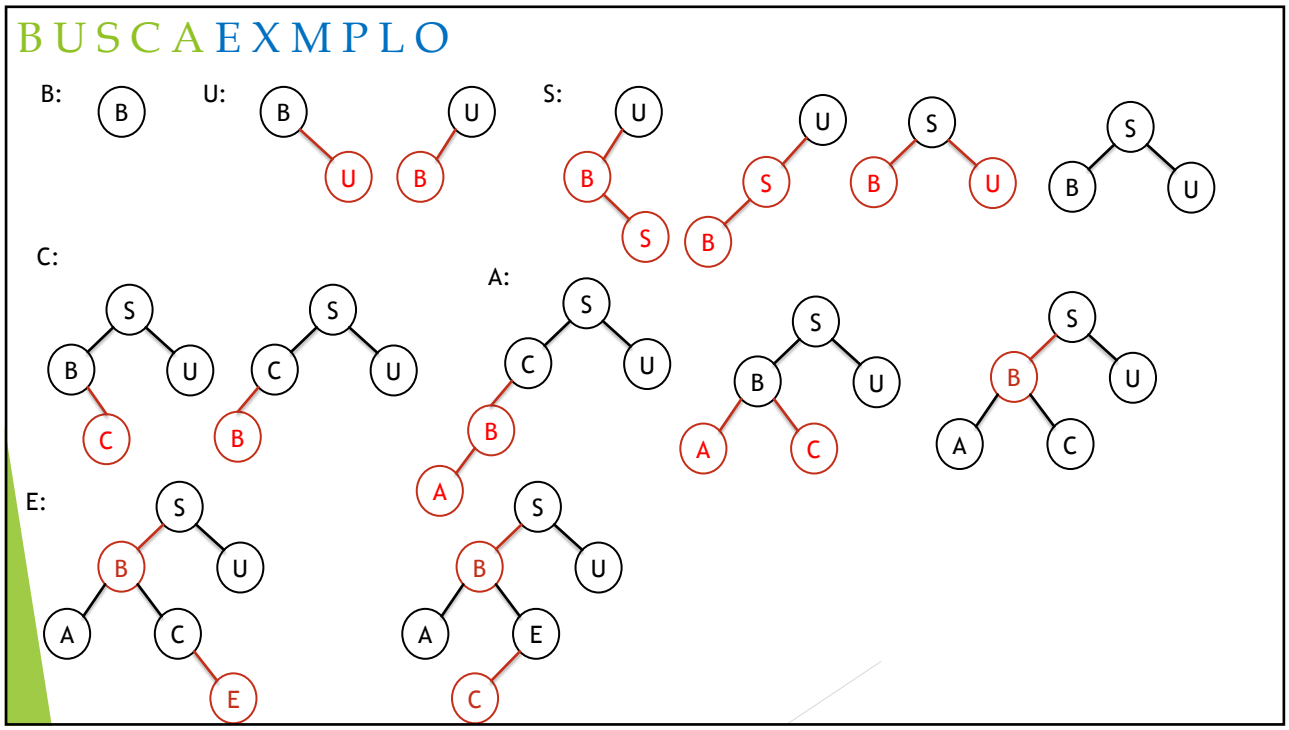
- Ver o vídeo que ilustra a inserção da sequência de chaves S E A R C H E X A M P L E
  - algs4.cs.princeton.edu/lectures/33DemoRedBlackBST.mov

25

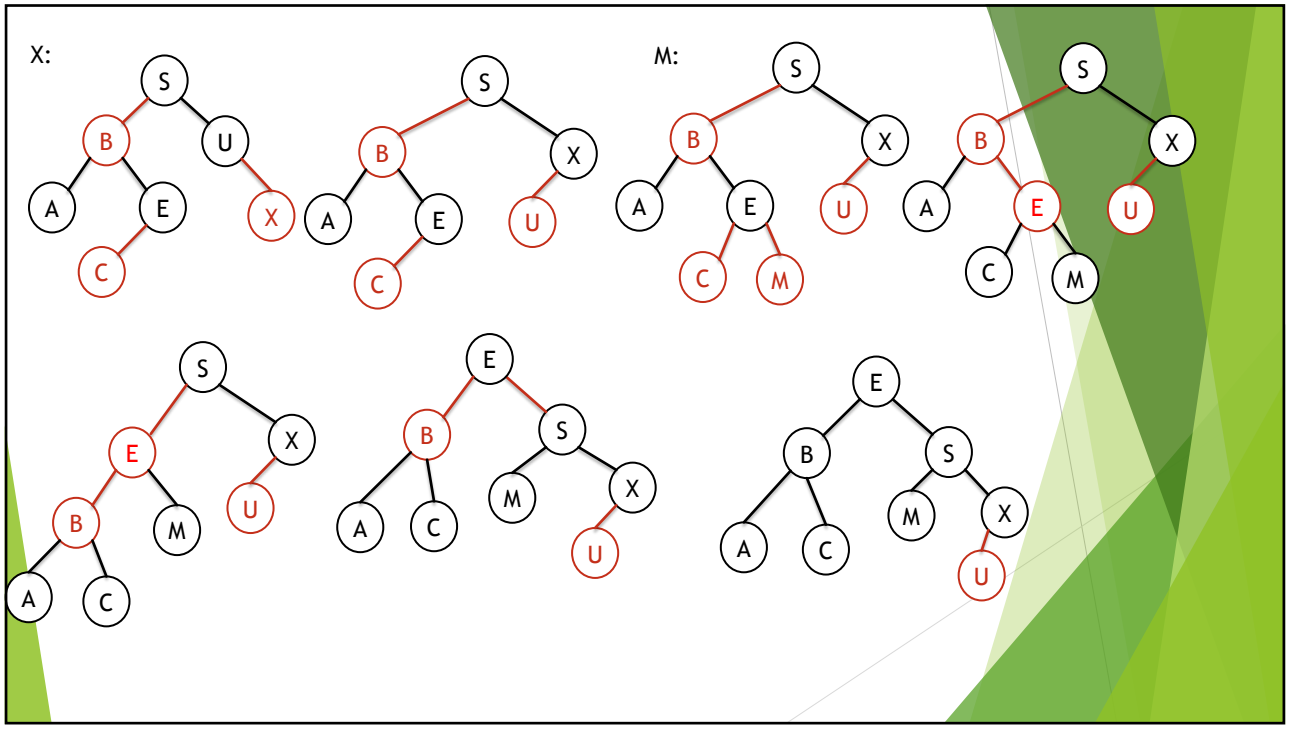
# Atividade de participação

1) Inserir as seguintes chaves em uma Árvore de Busca Rubro-Negra:
   a) B U S C A E X M P L O (completar a solução apresentada em sala de aula)
   b) U F I E X A M P L S R C H (Precisa fazer)

26




27




28

## Em uma árvore Red-Black, inserir as seguintes chaves: A B R C I D E F

29