1.5

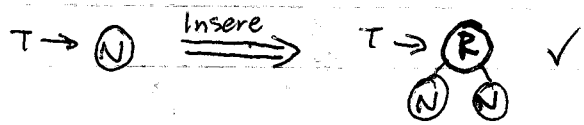
1.0/1.0

(Ex) Demonstre que o algoritmo insira em árvores 2-3-4 funciona, ou seja, que devolve uma árvore 2-3-4. Para isso, apresente as condições corretas (que são a hipótese de indução) e a prova de que tais hipóteses valem desde que a árvore de entrada seja uma árvore 2-3-4 a menos da raiz, que pode ser rubra.

Demonstração: Para a implementação de árvores 2-3-4 vale que:

(a) se um nó T era negro, devolve uma árvore 2-3-4 exceto pela raiz, que pode ser rubra, caso um de seus filhos era rubro antes. ✓

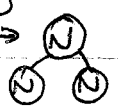
(b) se um nó T era rubro, devolve uma árvore 2-3-4 exceto que a raiz e o nó esquerdo da raiz podem ser rubros. ✓

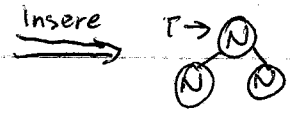
Base: $h(T) = -1$ 

✓

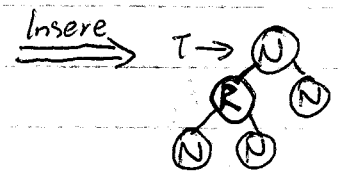
Passo: $h(T) \geq 0$

Caso a: cor(T) = Negro

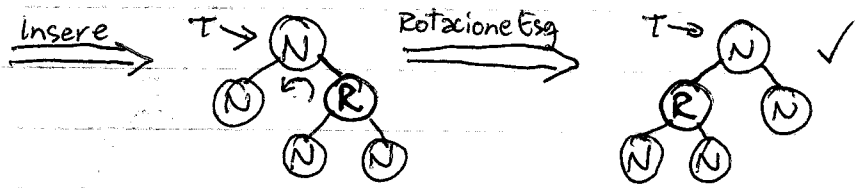
Subcaso 1: 

Subcaso 1.1: 

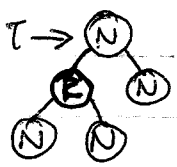
✓

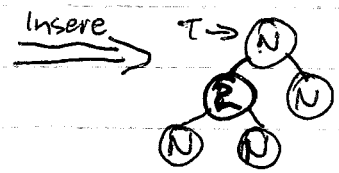
Subcaso 1.2: 

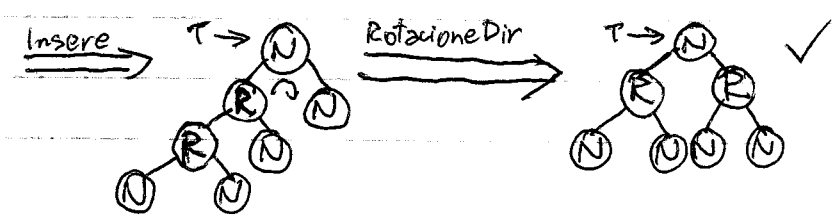
✓

Subcaso 1.3 

✓

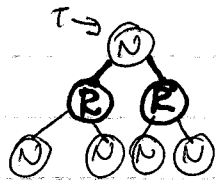
Subcaso 2: 

Subcaso 2.1: 

Subcaso 2.2: 

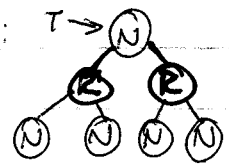
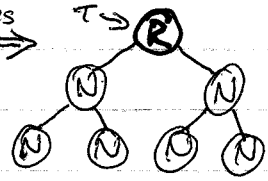
✓

Subcaso 2.3: Insere ⇒

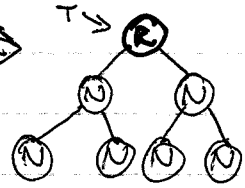


✓

✓

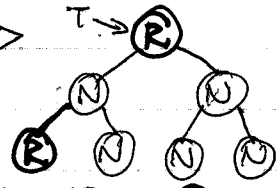
Subcaso 3: T →  TroqueCores ⇒ T → 

Subcaso 3.1: Insere ⇒



✓

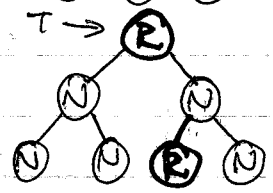
Subcaso 3.2: Insere ⇒



✓

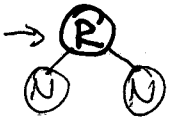
Legal! Separação em casos excelente!

Subcaso 3.3: Insere ⇒

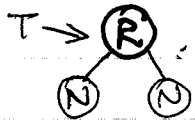


✓

Caso b: cor(T) = **Rubro**

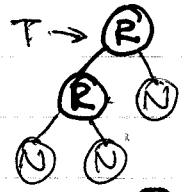
T → 

Subcaso 1: Insere ⇒



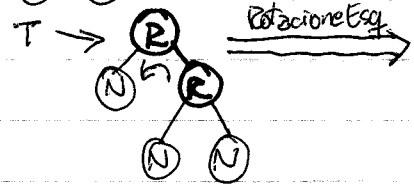
✓

Subcaso 2: Insere ⇒



✓

Subcaso 3: Insere ⇒



✓

Através das condições apresentadas, pode-se provar por indução que o algoritmo insira em árvores 2-3-4 funciona, pois os casos e subcasos apresentados representam todas as possibilidades limitadas pelas condições apresentadas e a árvore resultante de cada caso apresentado é uma árvore 2-3-4 a menos da raiz, que pode ser rubra. ■

Perfeito!

0.5
0.5

(Ex) A implementação do Sedgewick de árvores rubro-negras, acessível em <<URL>>, é um pouco diferente da vista em aula. Simule a inserção das chaves 40, 20, 50, 10, 30, 25, 35, nesta ordem com essa implementação. Como é a árvore rubro-negra que você obtem? Você vê algum problema?

Resolução: Início: root → null

Processo de inserção do 40: Insert ⇒ root → 40 ⇒ root → 40 ✓

Processo de inserção do 20: Insert ⇒ root → 40, 20

Processo de inserção do 50: Insert ⇒ root → 40, 20, 50 ⇒ rotateLeft ⇒ root → 50, 40, 20 ⇒ rotateRight ⇒ root → 40, 20, 50 ✓

Processo de inserção do 10: colorFlip ⇒ root → 40, 20, 50 ⇒ Insert ⇒ 40, 20, 50, 10 ⇒ root → 40, 20, 50, 10 ✓

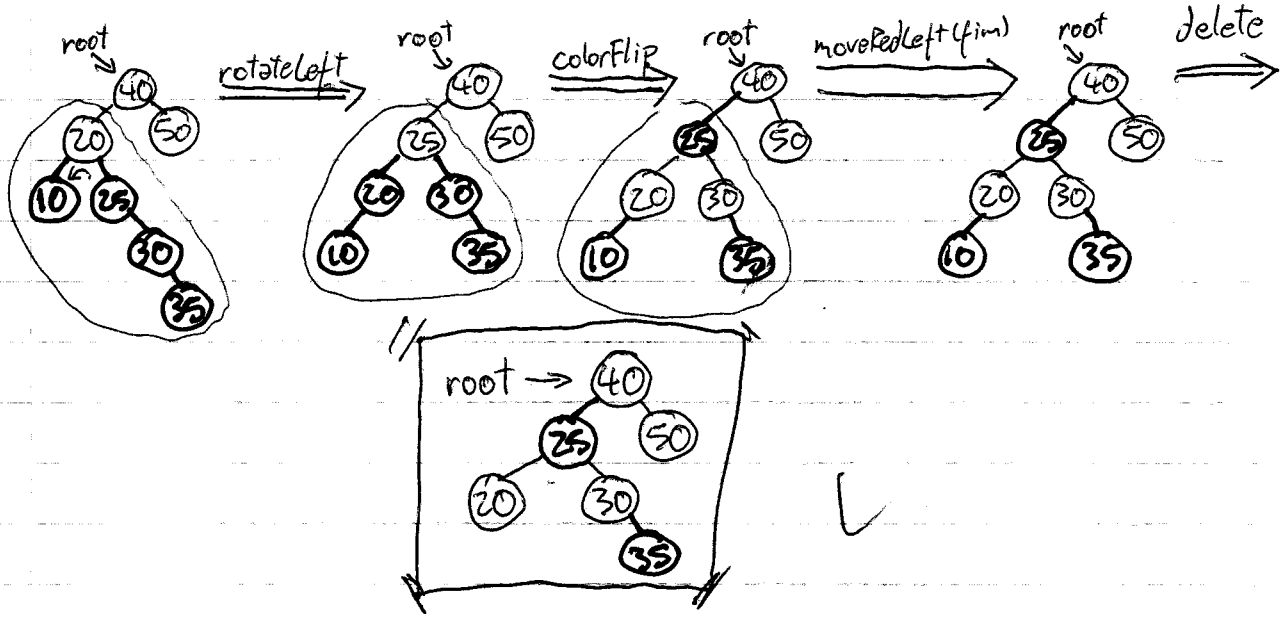
Processo de inserção do 30: Insert ⇒ root → 40, 20, 50, 10, 30 ⇒ rotateLeft ⇒ root → 40, 30, 50, 20, 10 ⇒ rotateRight ⇒ root → 40, 20, 50, 10, 30 ✓

Processo de inserção do 25: colorFlip ⇒ root → 40, 20, 50, 10, 30 ⇒ Insert ⇒ root → 40, 20, 50, 10, 30, 25

Processo de inserção do 35: insert ⇒ root → 40, 20, 50, 10, 30, 25, 35 ✓

Processo de remoção do 10: moveRedLeft(início) ⇒ root → 40, 20, 50, 10, 30, 25, 35 ⇒ colorFlip ⇒ root → 40, 20, 50, 10, 30, 25, 35 ⇒ rotateRight

④



O nó de valor 30 possui um nó folha à esquerda e um nó rubro à direita.
Tal situação não se adequa à característica de ser uma árvore rubro-negra left-leaning 2-3-4, proposta pelo próprio Sedgewick.
Comparando o código do Sedgewick aos slides de aula, verificamos o código a seguir, onde o trecho em **vermelho** é o pedaço de código **ausente** no código do Sedgewick, que garante a propriedade de left-leaning para 3-nós, para a remoção proposta no enunciado:

```
private Node moveRedLeft(Node h)
{
    colorFlip(h);
    if (isRed(h.right.left))
    {
        h.right = rotateRight(h.right);
        h = rotateLeft(h);
        colorFlip(h);
        if (isRed(h.right.right))
            h.right = rotateLeft(h.right);
    }
    return h;
}
```

Legal!