Árvores: árvores rubro-negras

Prof. Jefferson T. Oliva

Algoritmos e Estrutura de Dados II (AE23CP)
Engenharia de Computação
Departamento Acadêmico de Informática (Dainf)
Universidade Tecnológica Federal do Paraná (UTFPR)
Campus Pato Branco

DAINF
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
CÂMPUS PATO BRANCO

UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Sumário

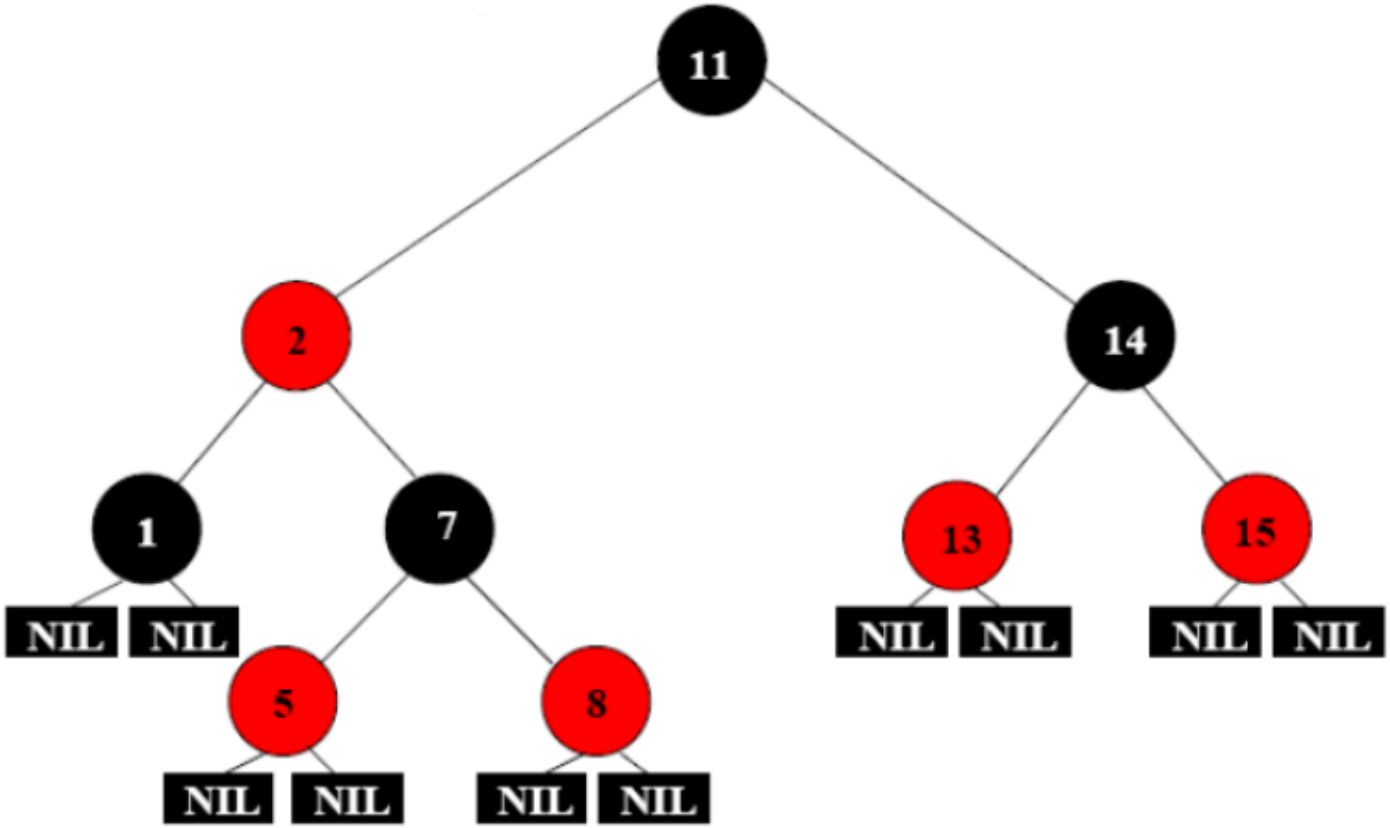
- Propriedades
- Inserção
- Remoção
- Complexidade

Introdução

- As árvores rubro-negras são árvores binárias de busca
- Também conhecidas como vermelha-pretas ou *red-black trees*
- Foram inventadas por Bayer sob o nome "Árvores Binárias Simétricas" em 1972, 10 anos depois das árvores AVL
- As árvores rubro-negras possuem um **bit extra** para armazenar a cor de cada nó, que pode ser VERMELHO ou PRETO
- Exemplos de aplicações de árvores rubro-negras
  - Java
  - Python
  - Kernel Linux
  - Sistema de arquivos

Introdução

- Exemplo de árvore rubro-negra:



Introdução

- Cada nó possui os seguintes campos:
  - Bit referente à cor
  - Valor (informação do nó)
  - Filhos esquerdo e direito
  - Pai
- Se um filho ou o pai de um nó não existir, o ponteiro correspondente é NULL
- Essa árvore é complexa, mas possui bom custo de tempo para a execução de suas operações e é eficiente na prática
  - As operações de busca (pesquisa), inserção e remoção possuem a complexidade de tempo $O(\log n)$

Introdução

- Uma árvore vermelha-preta com n nós tem altura máxima de $2\log(n+1)$

- O balanceamento garante que essas árvores possuam complexidade na ordem de $\log n$ em suas operações

Sumário

Propriedades

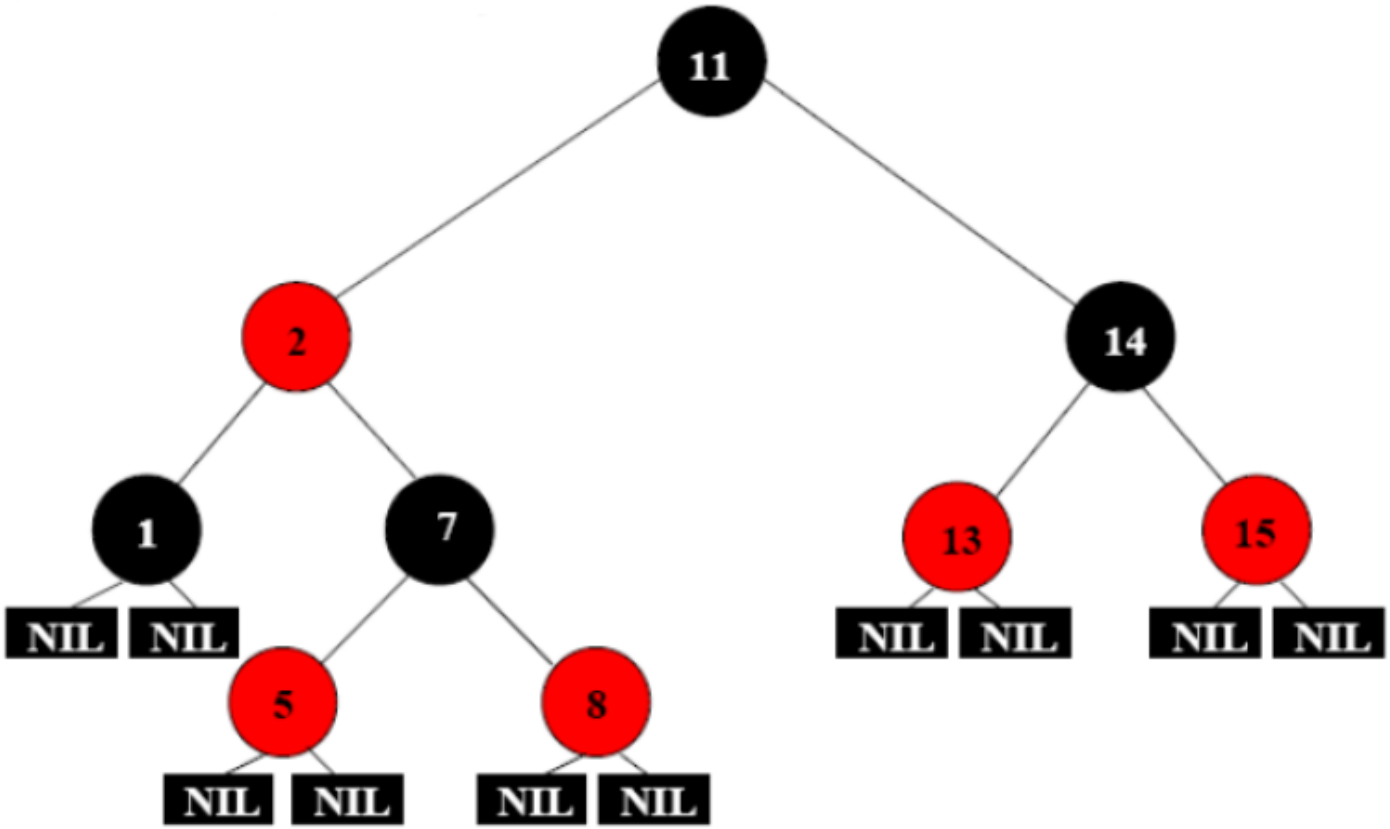
1. Todo nó é vermelho ou preto
2. A raiz é preta
3. Toda folha é preta (nesse tipo de árvore, o ponteiro NULL é considerado folha)
4. Se um nó é vermelho, então os seus filhos são pretos
5. Para cada nó, todos os caminhos simples do nó até folhas descendentes contêm o mesmo número de nós pretos

Propriedades

- Um nó que satisfaz as propriedades anteriores é denominado equilibrado, caso contrário é dito desequilibrado
- Em uma árvore vermelha-preta, todos os nós estão equilibrados
- A raiz pode sempre ser alterada de vermelho para preto, mas não o contrário (propriedade 2)
- No caminho da raiz até uma sub-árvore vazia não pode existir dois nós vermelhos consecutivos

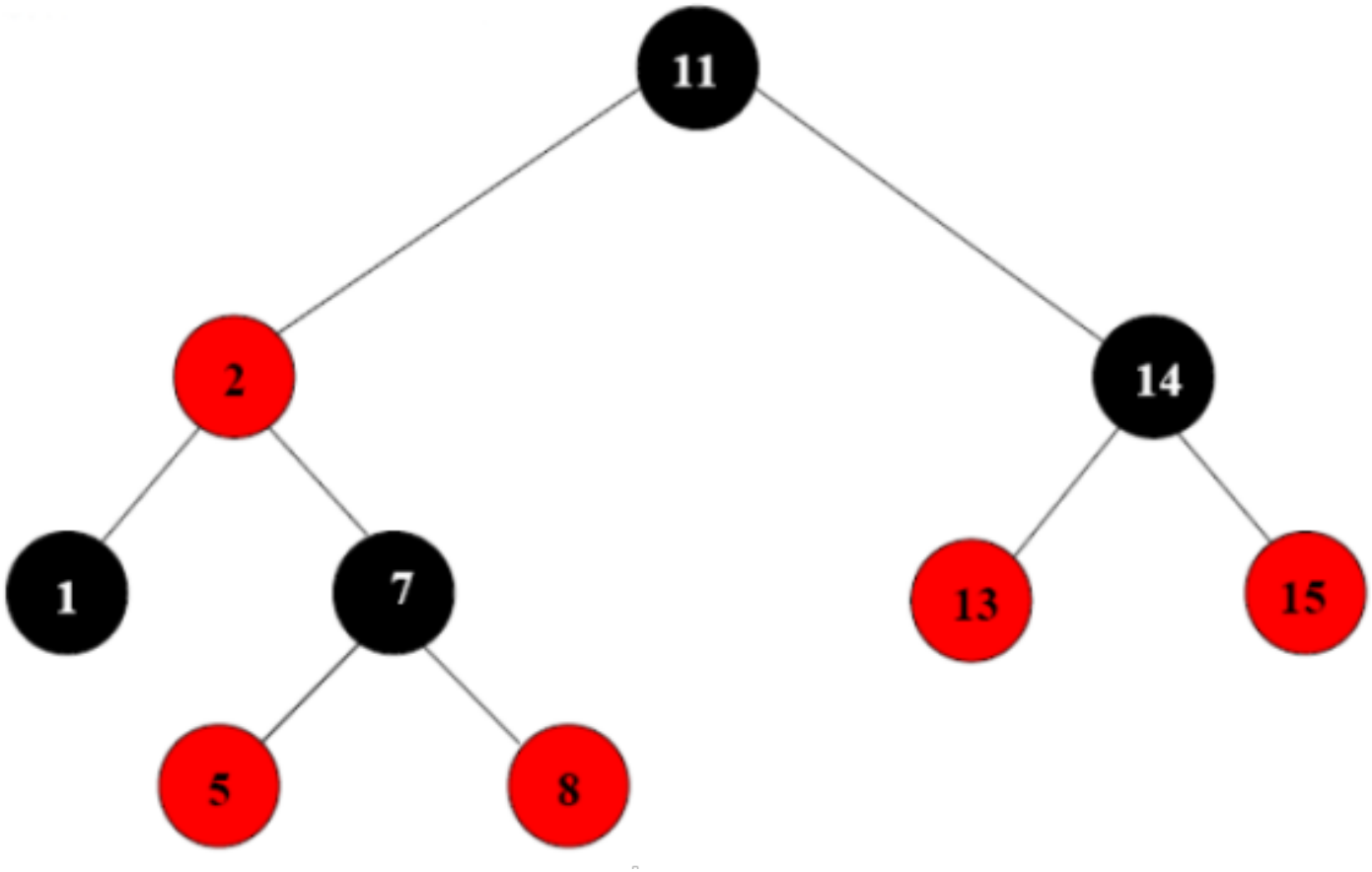
Propriedades

- Formas de representação:



Propriedades

- Formas de representação:



Propriedades

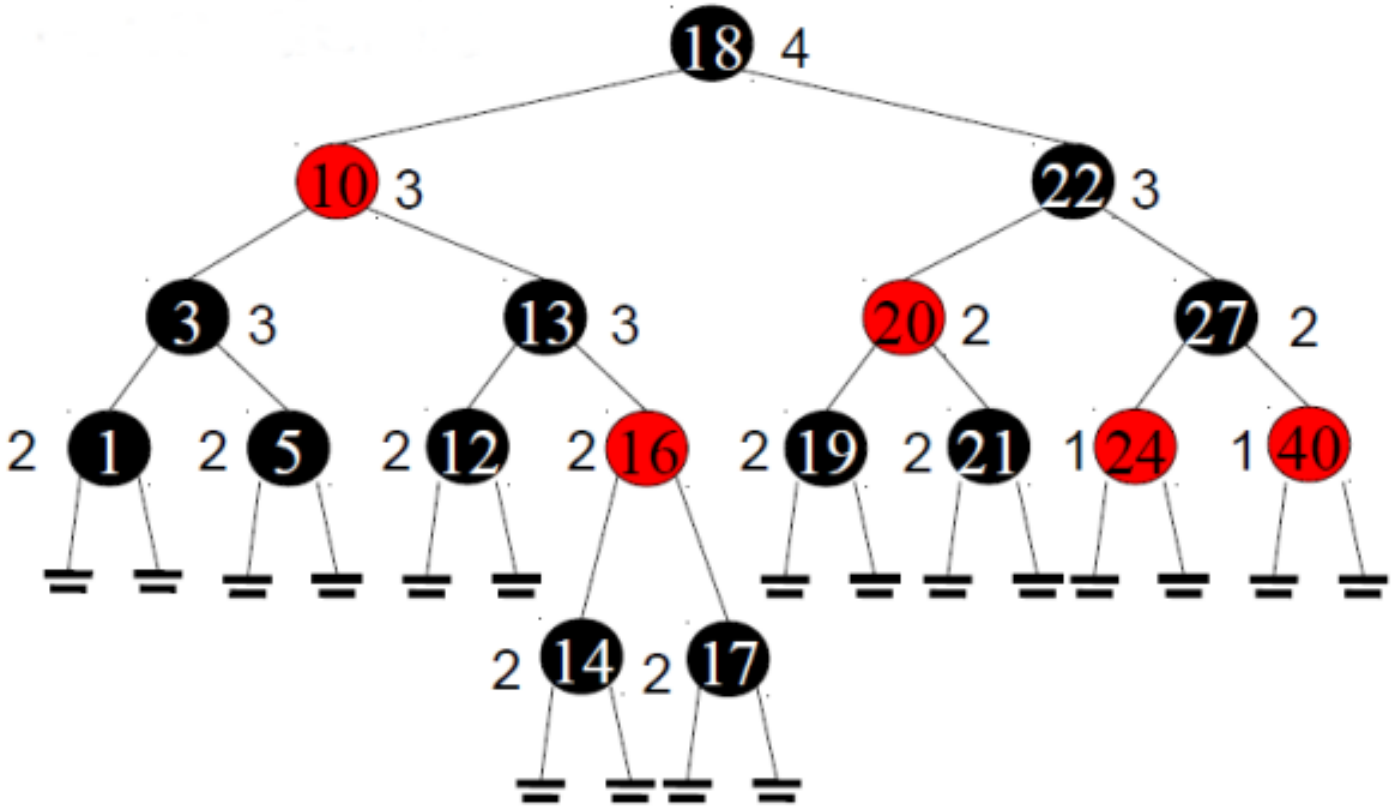
- Cada vez que uma operação de alteração (*e.g.* inserção e remoção) for realizada na árvore, o conjunto de propriedades é testado

- Caso alguma propriedade seja violada, são realizadas rotações e/ou ajustes de cores, de forma que a árvore permaneça "balanceada"

- Restringindo o modo como os nós são coloridos desde a raiz até uma folha, assegura-se que nenhum caminho será maior que duas vezes o comprimento de qualquer outro

Propriedades

- Altura preta: número de nós pretos encontrados até qualquer nó folha



Sumário

Inserção

- As operações inserção e de remoção podem ser implementadas de forma bastante parecida com as respectivas operações nas árvores binárias de busca
- A inserção em uma árvore vermelha-preta começa por uma busca da posição onde o novo nó deve ser inserido
- Essas operações são mais complicadas nas árvores vermelha-pretas em comparação com as árvores binárias de busca
  - Essas operações podem violar alguma propriedade de árvore vermelha-preta

Inserção

- Um nó é inserido sempre na cor vermelha
  - Se o nó fosse inserido na cor preta, invalidaria a propriedade 5 (para cada nó, todos os caminhos do nó para as folhas descendentes contém o mesmo número de nós PRETOS)

- Caso a inserção seja feita em uma árvore vazia, basta alterar a cor do nó para preto

Inserção

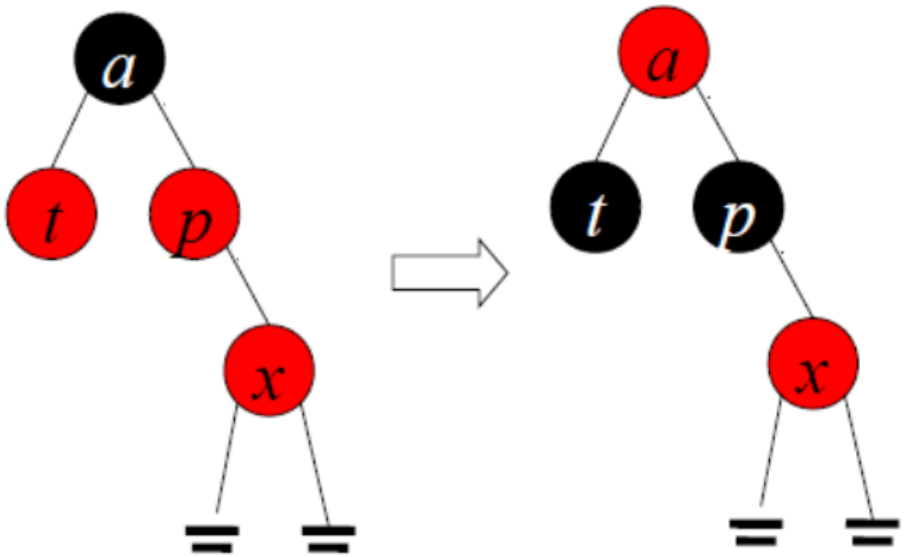
- Após a inserção, um conjunto de propriedades é testado
- Se a árvore não satisfizer essas propriedades, são realizadas rotações e/ou ajustes de cores para o balanceamento da árvore
- Para a correção do desbalanceamento da árvore rubro-negra, são considerados três casos

Inserção

- Caso 1: se esta inserção é feita em uma árvore vazia, basta alterar a cor do nó para preto
  - Satisfaz a propriedade 2 (a raiz é preta)

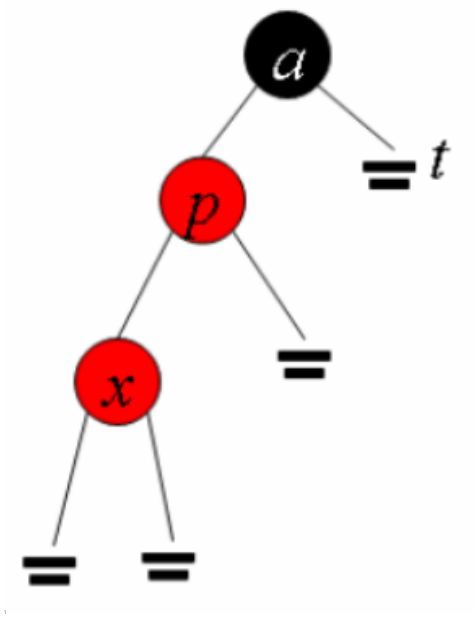
Inserção

- Caso 2: Ao inserir x e respectivo tio é vermelho, é necessário fazer a re-coloração de a (avô), t (tio) e p (pai)
  - Se o pai de a é vermelho, o rebalanceamento deve ser feito novamente
  - Se a é raiz, então ela deve ser mantida preta



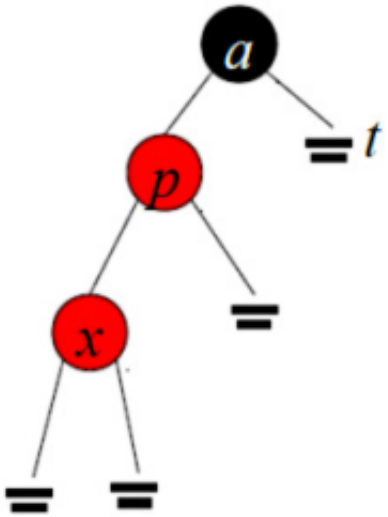
Inserção

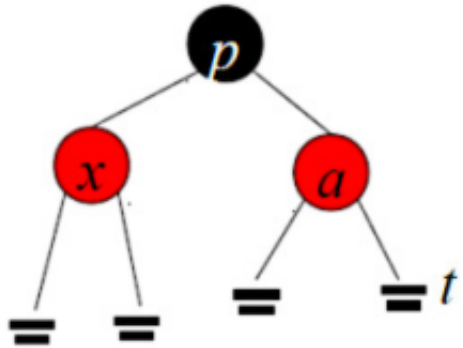
- Caso 3: Suponha que o tio do nó inserido é preto.
  - Para manter a propriedade 4 (se um nó é vermelho, então seus filhos são pretos) é preciso fazer rotações envolvendo a, t, p e x
  - Há 4 sub-casos que correspondem às 4 rotações possíveis



Inserção

- Caso 3a: Rotação à Direita

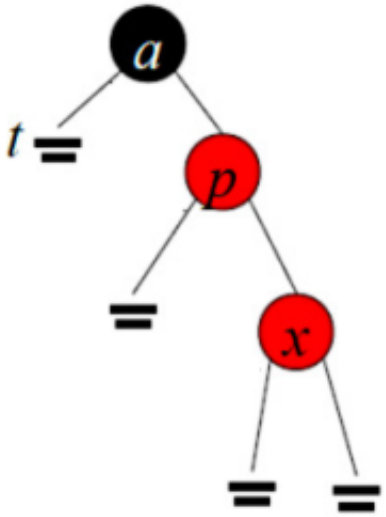


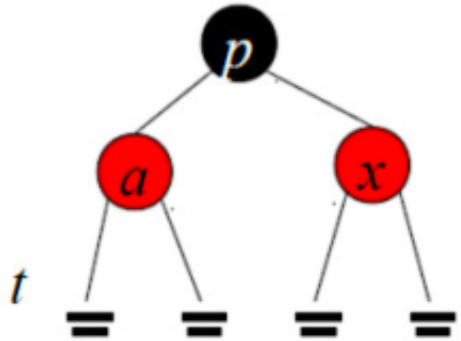


Recoloração de p e a

Inserção

- Caso 3b: Rotação à Esquerda

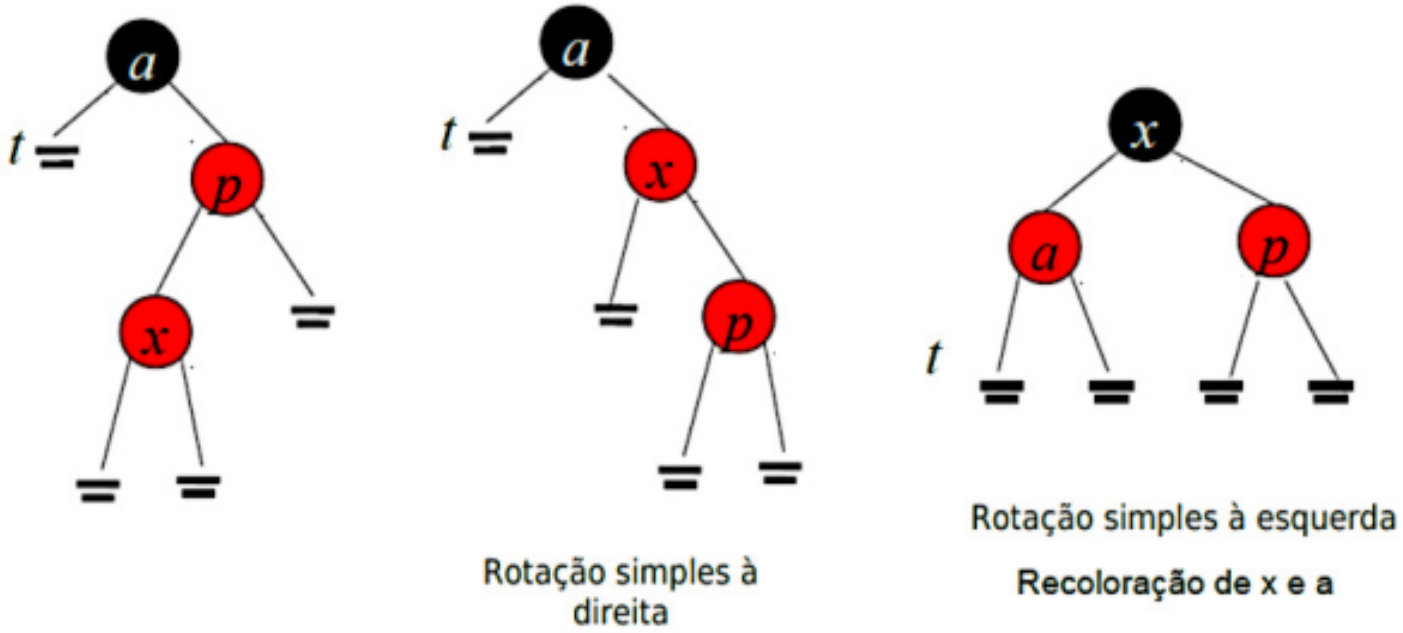




Recoloração de p e a

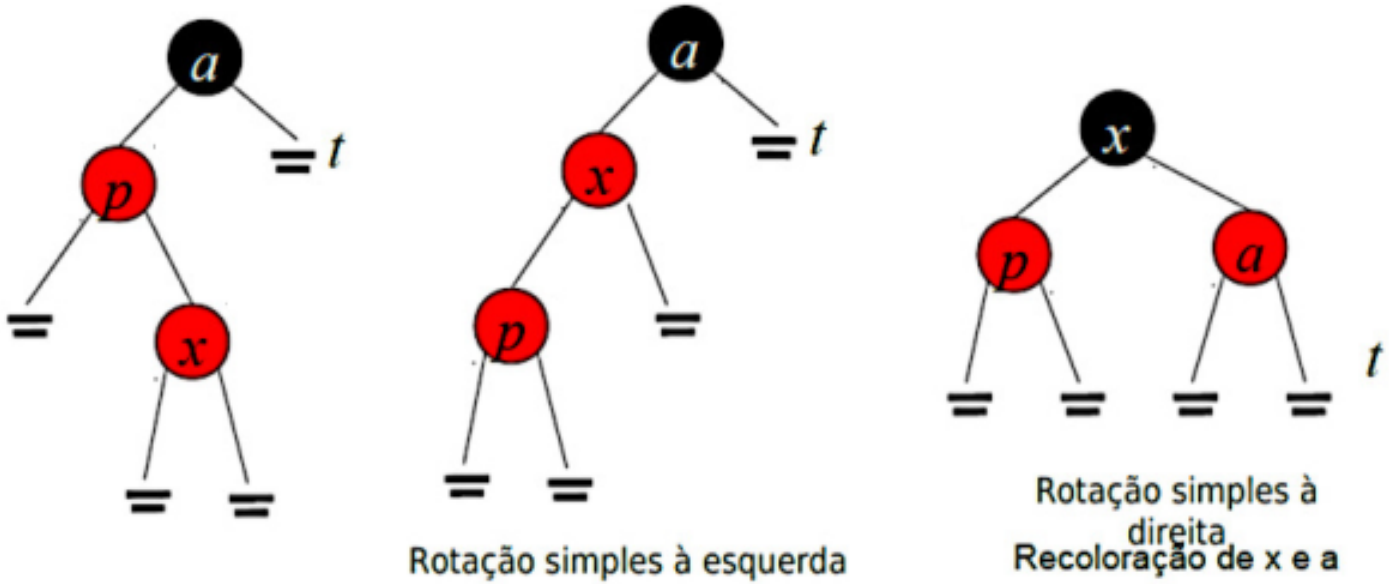
Inserção

- Caso 3c: rotação dupla à esquerda
  - Pode ser visto como um caso 3a seguido do caso 3b



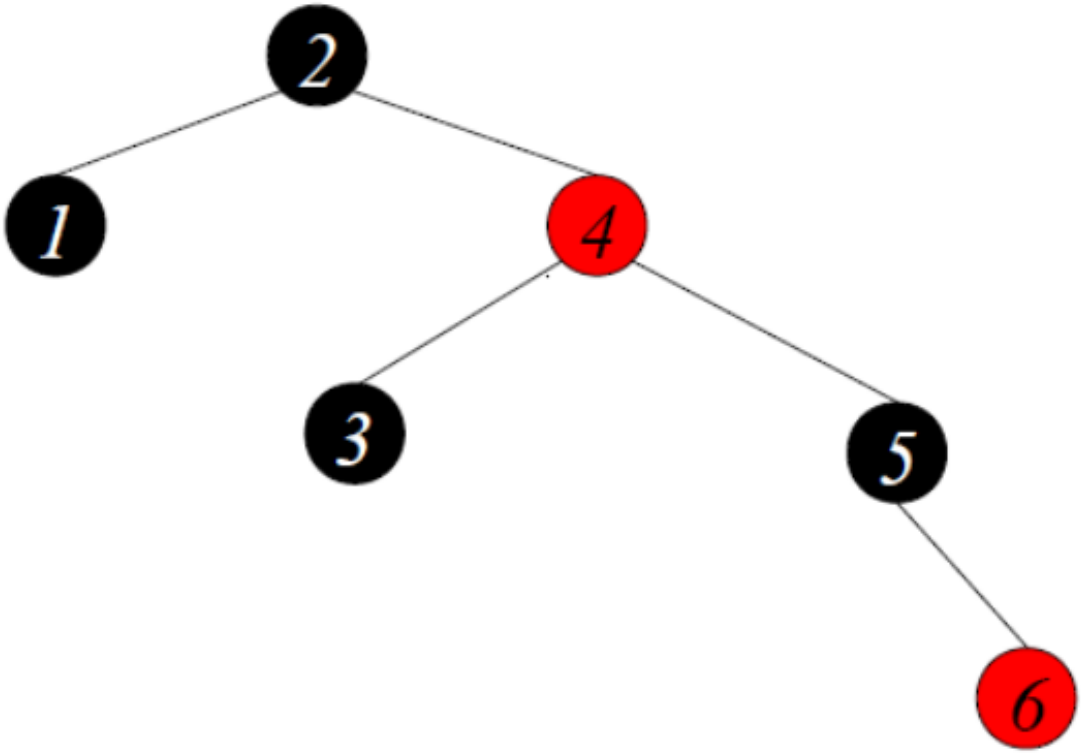
Inserção

- Caso 3d: rotação dupla direita
  - Pode ser visto como um caso 3b seguido do caso 3a



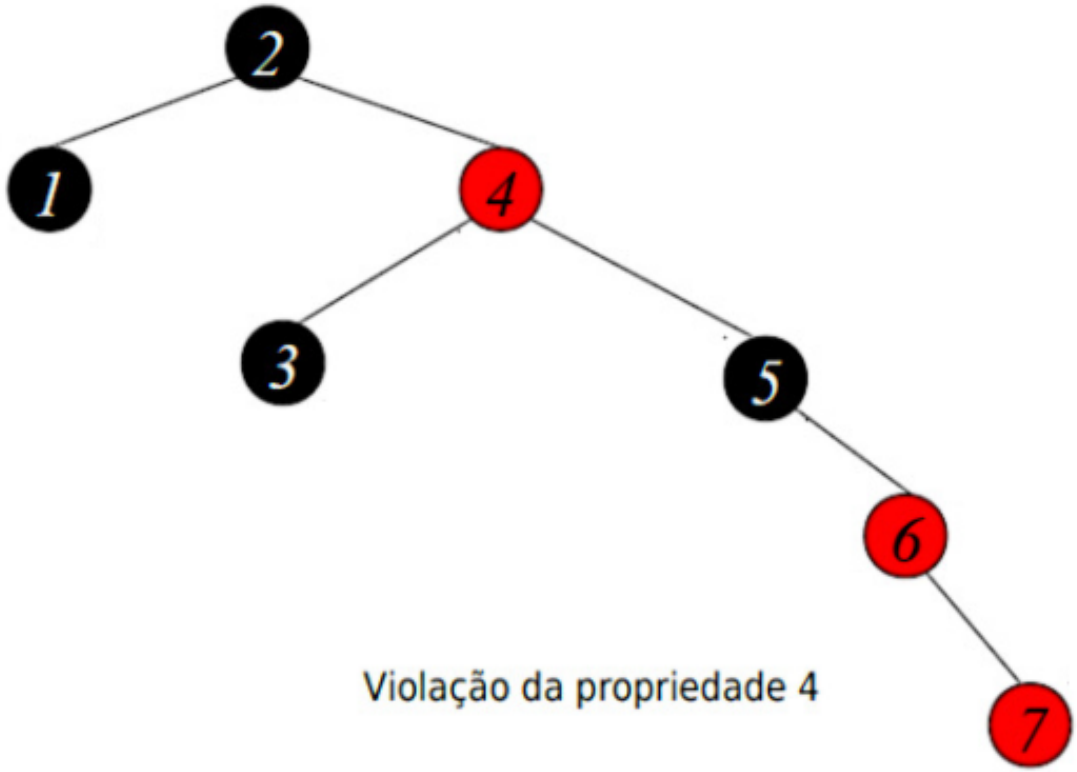
Inserção

- Exemplo
  - Estado inicial da árvore



Inserção

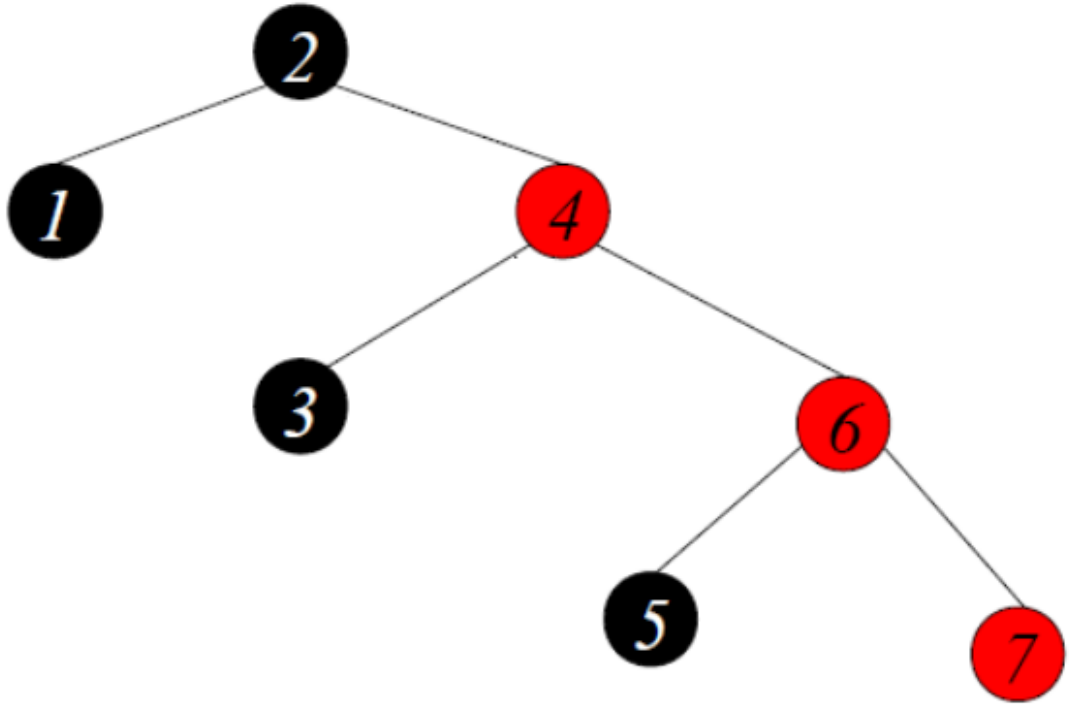
- Exemplo
  - Inserção do nodo 7



(Propriedade 4: se um nó é vermelho, então seus filhos são pretos)

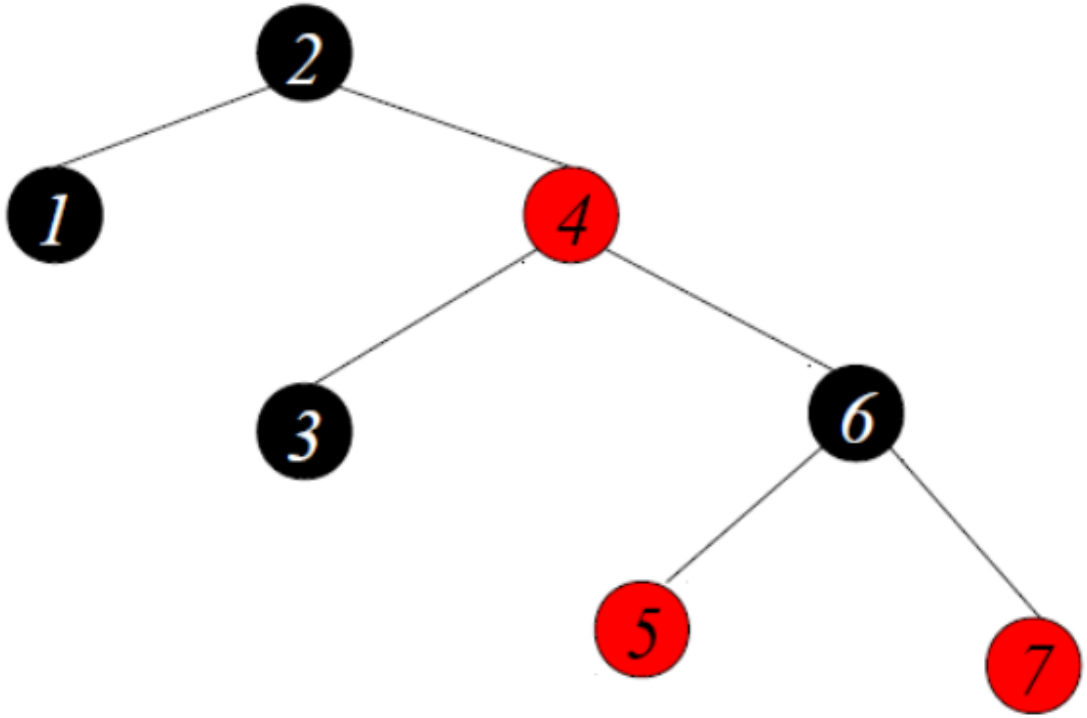
Inserção

- Exemplo
  - Rotação à esquerda dos nodos 5, 6 e 7



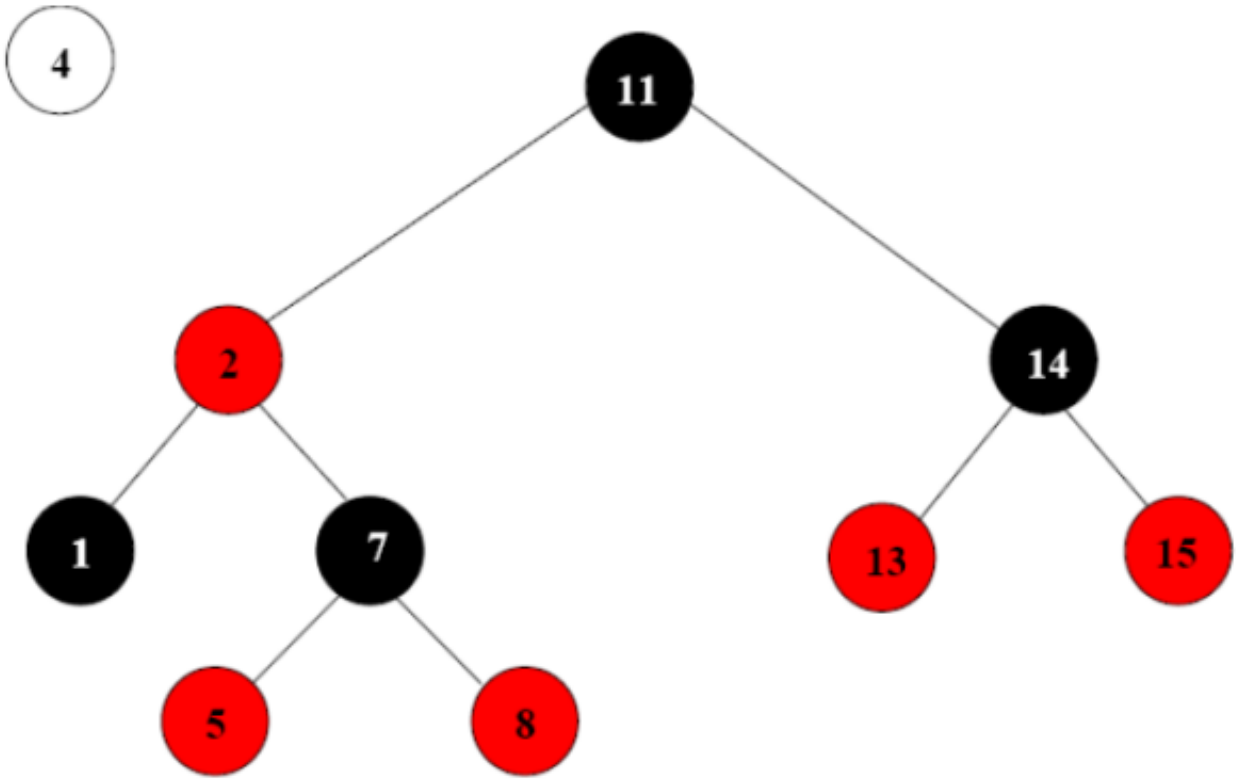
Inserção

- Exemplo
  - Alteração de cor dos nodos 5 e 6



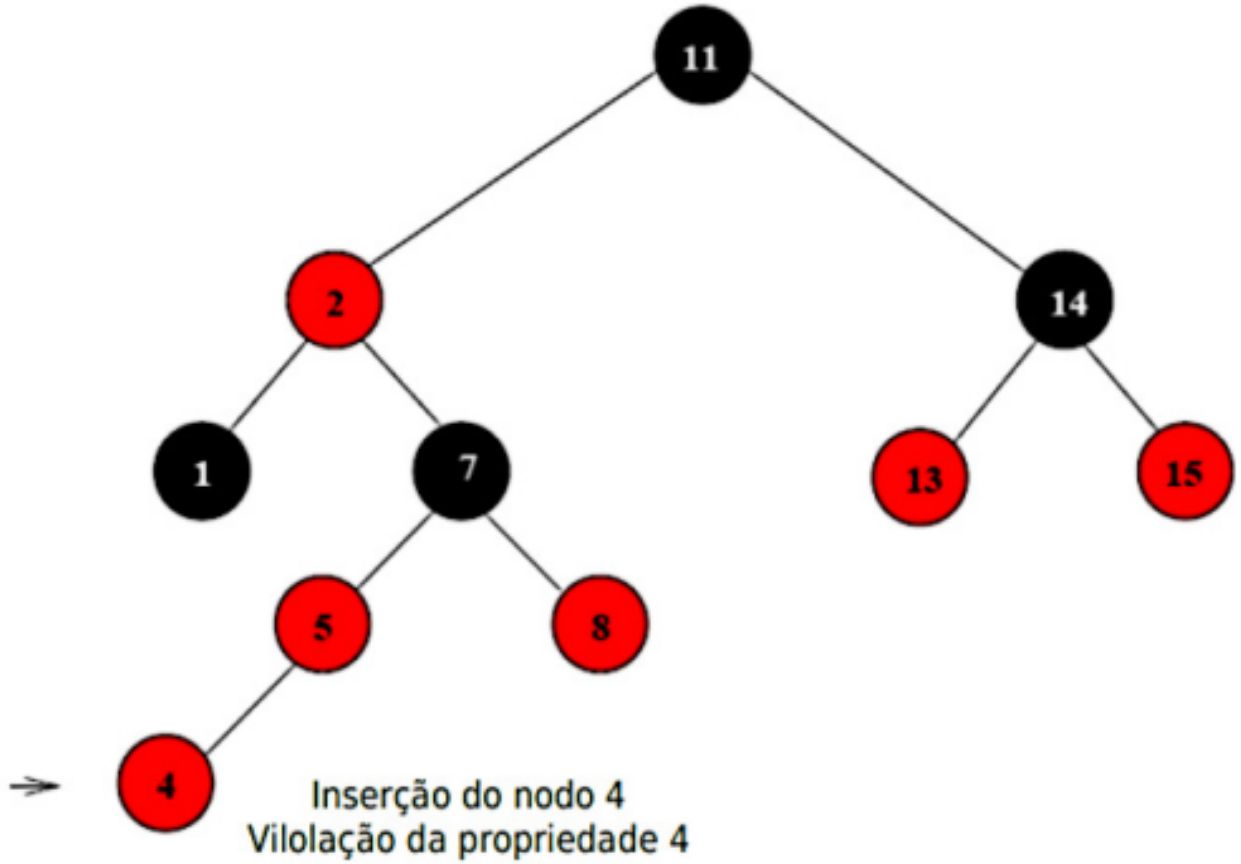
Inserção

- Exemplo
  - Estado inicial da árvore



Inserção

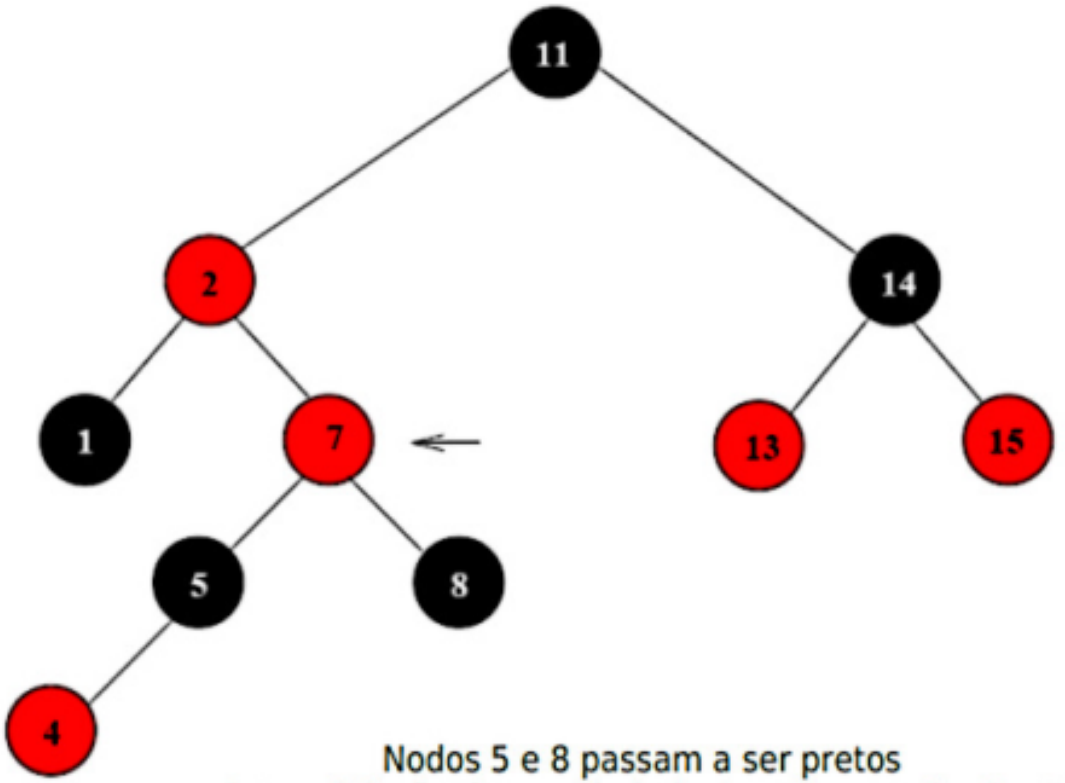
- Exemplo
  - Caso 2: O tio do elemento inserido é vermelho



(Propriedade 4: se um nó é vermelho, então seus filhos são pretos)

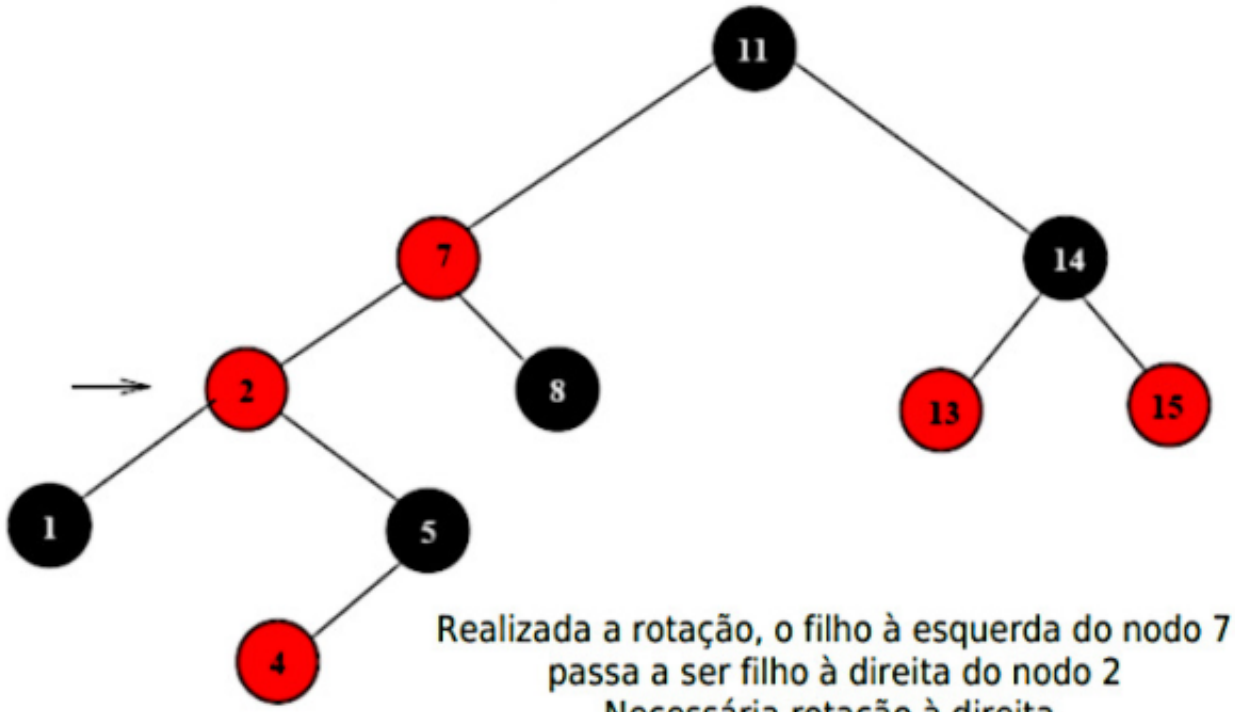
Inserção

- Exemplo
  - Caso 3d: resolver o caso 3b, onde o tio do elemento é preto e o elemento é filho à direita



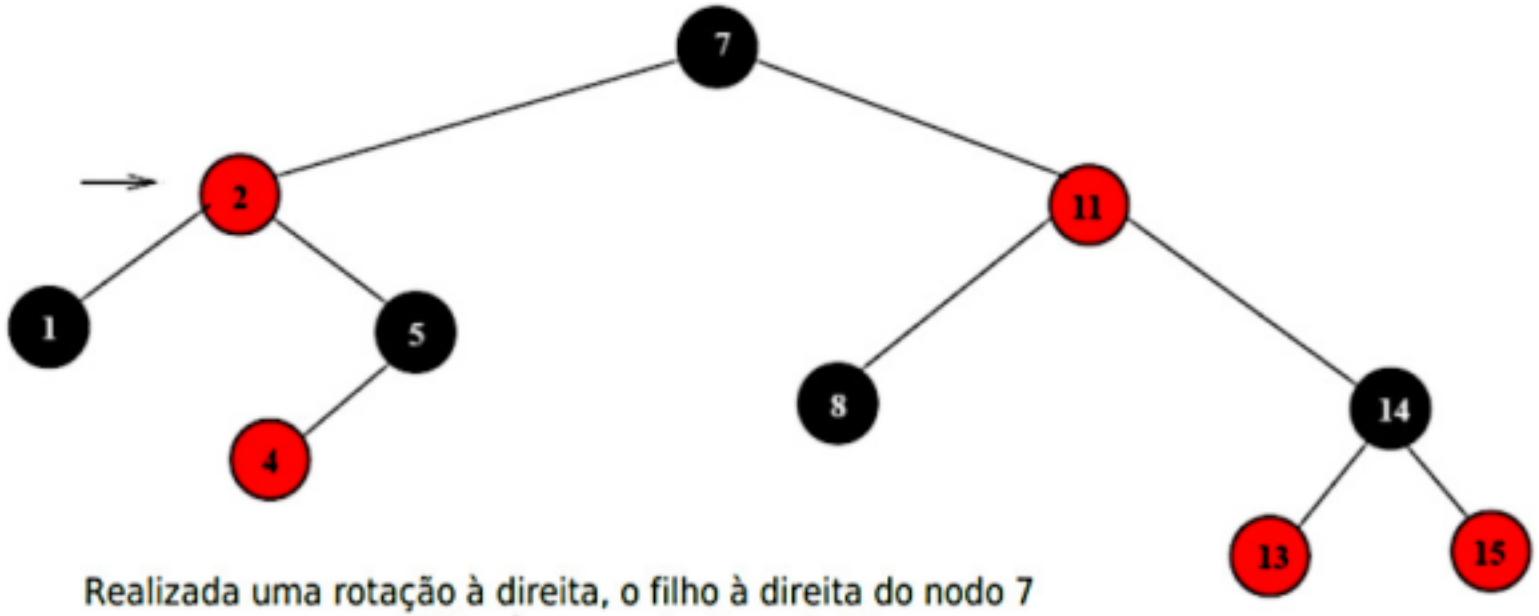
Inserção

- Exemplo
  - Caso 3d: resolver o caso 3a, onde o tio do elemento é preto e o elemento é filho à esquerda



Inserção

- Exemplo
  - Árvore vermelha-preta resultante



Realizada uma rotação à direita, o filho à direita do nodo 7
passa a ser filho à esquerda do nodo 11
O nodo 7 é colorido de preto, é restaurada a propriedade 4
e nenhuma outra é violada

Sumário

Remoção

- A remoção nas árvores vermelho-pretas se inicia com uma etapa de busca e remoção como nas árvores binárias de busca convencionais

- Caso o nó removido seja vermelho, as propriedades da árvore rubro-negra são mantidas
  - Se os descendentes forem nulos, basta liberar o espaço do nó
  - Se o nó substituto for vermelho, a troca é feita sem a necessidade de mudanças adicionais
  - Se o nó substituto for preto, dois nós descendentes deverão ter a cor mudada de preta para vermelha

Remoção

- Na remoção, se um nó retirado for preto, a propriedade 5 (slide 6) é violada!
  - Em outras palavras, a árvore deve ser rebalanceada
- Na inserção, as operações de rotação são baseadas na coloração do nó tio
- Na remoção, essas operações são baseadas na coloração do nó irmão

Remoção

- Na inserção, o desbalanceamento ocorre quando um determinado nó e um de seus filhos são vermelhos

- Na remoção, o desbalanceamento ocorre quando o nó retirado for preto
  - Nesse cenário, há quatro casos a serem tratados (estão no apêndice deste material)

Complexidade

Complexidade

- Complexidade das principais operações em árvores vermelha-pretas
  - Espaço: $O(n)$
  - Busca: $O(1)$ (melhor caso) e $O(\log n)$ (médio e pior caso)
  - inserção: $O(1)$ (melhor caso) e $O(\log n)$ (médio e pior caso)
  - remoção: $O(1)$ (melhor caso) e $O(\log n)$ (médio e pior caso)

Exercício

- Criar uma árvore vermelha-preta inserindo as seguintes chaves, na ordem em que são apresentadas:
  - 41 - 38 - 31 - 12 - 19 - 8 - 25 - 54 - 65

Referências I


Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C.
Introduction to Algorithms.
Third edition, The MIT Press, 2009.

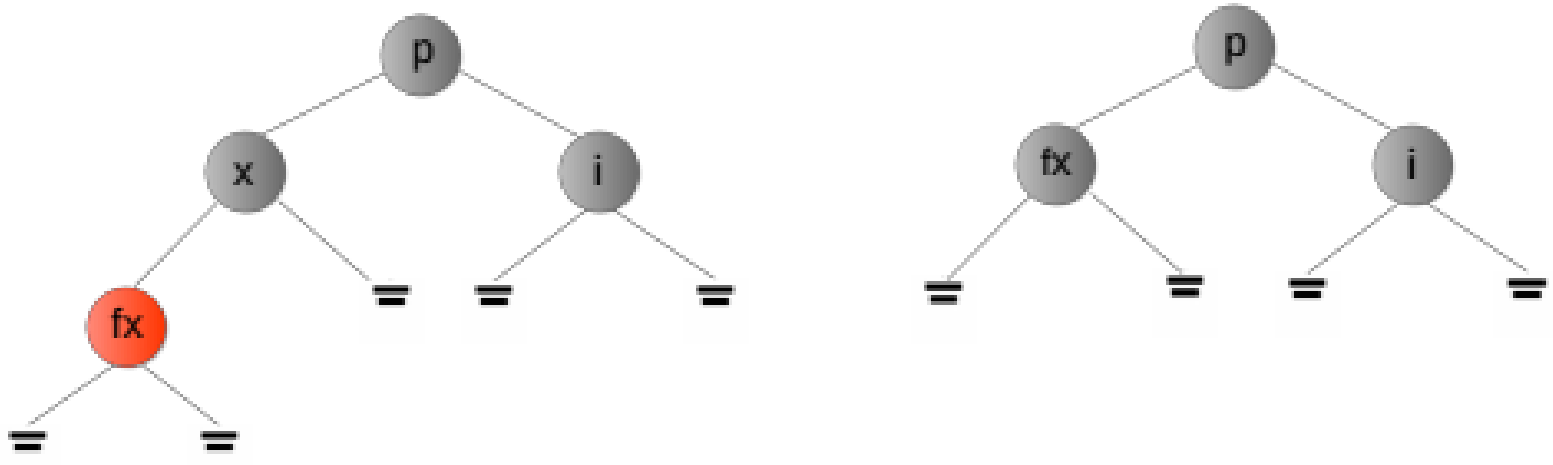

Marin, L. O.
Árvores Vermelho-Preta (Rubro-Negra) (RB-Tree). AE23CP – Algoritmos e Estrutura de Dados II.
Slides. Engenharia de Computação. Dainf/UTFPR/Pato Branco, 2017.


Song, S. W.
Árvore Rubro-Negra. MAC 508 – Estrutura de Dados.
Slides. Ciência da Computação. IME/USP/São Paulo, 2008.

Apêndice: remoção em árvores rubro-negras

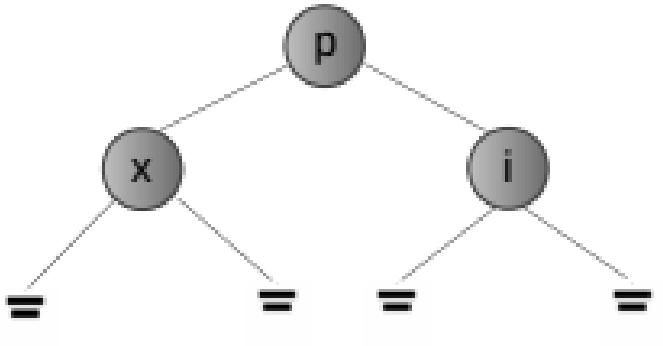
Remoção

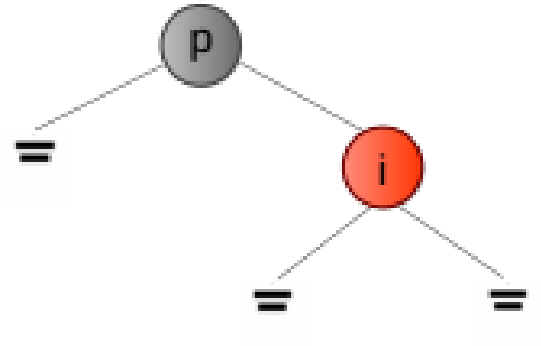
- Caso 1: se nó possui um filho vermelho (fx), o mesmo é recolorido para preto



Remoção

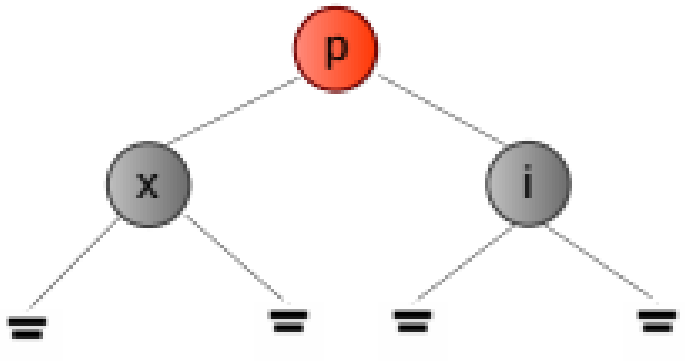
- Caso 2: se o irmão e os sobrinhos forem pretos, o nó irmão é recolorido para vermelho
  - Se o nó pai for vermelho, o mesmo deve ter a cor mudada para preta

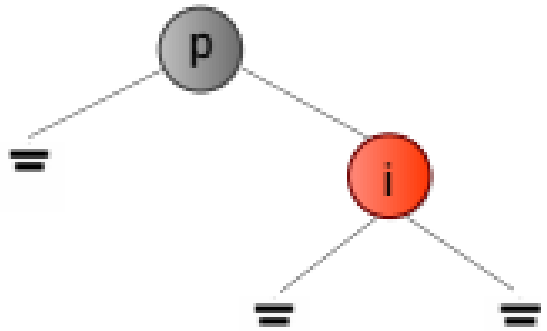




Remoção

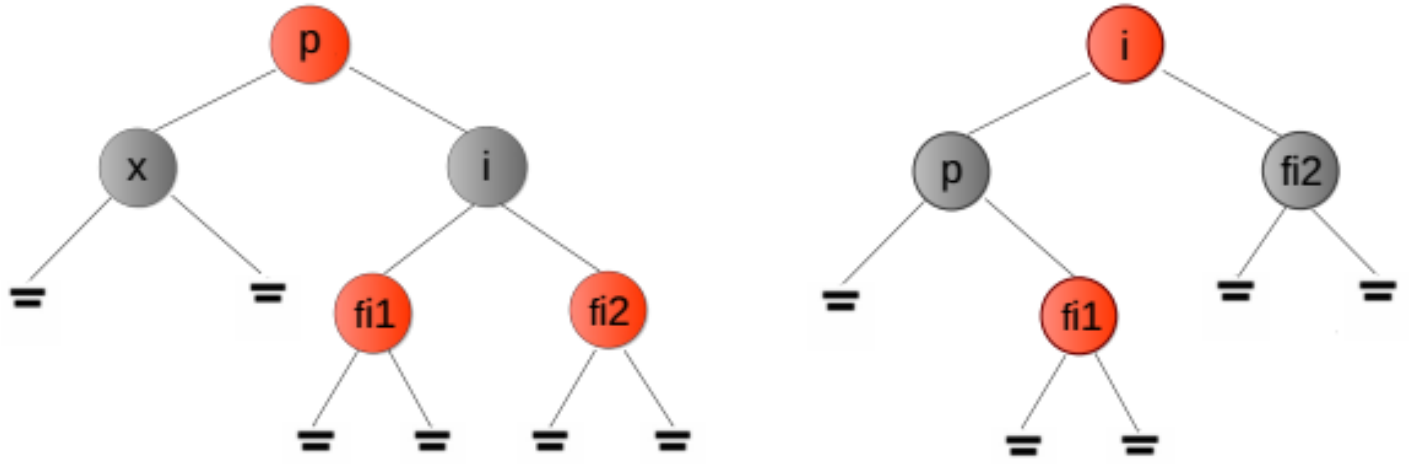
- Caso 2: se o irmão e o sobrinho forem pretos, o nó irmão é recolorido para vermelho
  - Se o nó pai for vermelho, o mesmo também deve ter a cor mudada para preta (assim, não haveria dois nós vermelhos consecutivos)





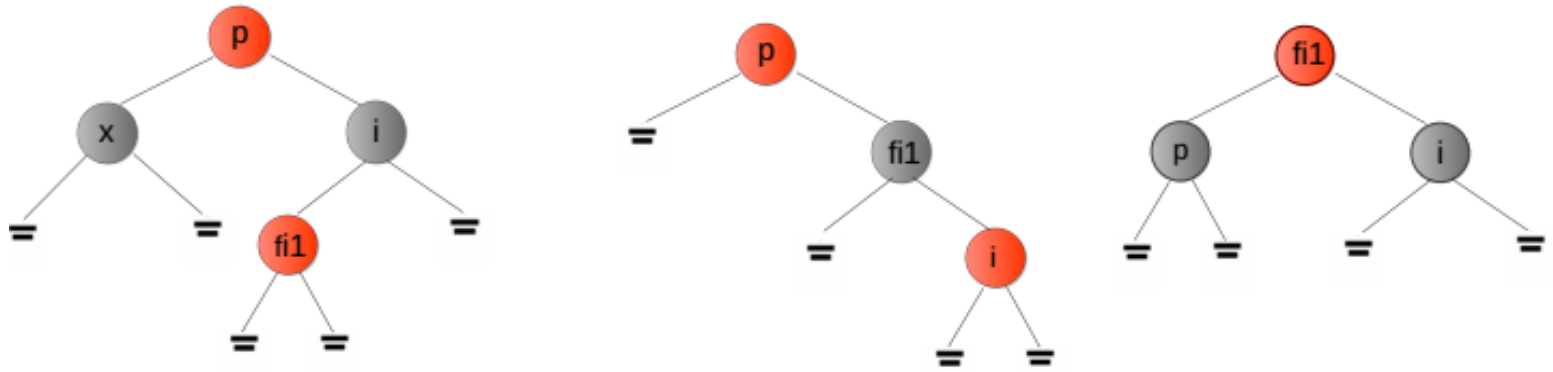
Remoção

- Caso 3: se o irmão for preto e pelo menos um dos sobrinhos (*fi*) vermelhos
  - É necessário fazer algum tipo de rotação
    - Depende de qual *fi* é vermelho (por exemplo se ambos forem vermelhos, pode ser aplicado algum tipo de rotação simples)
  - O nó que irá ocupar o lugar de *p* terá a mesma cor que o respectivo nó tinha



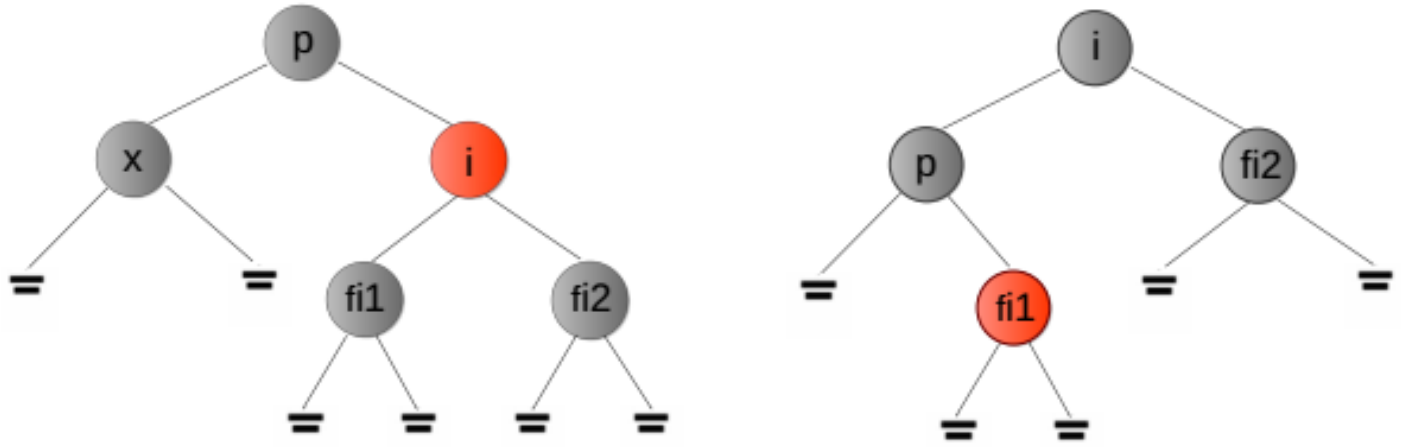
Remoção

- Caso 3: se o irmão for preto e pelo menos um dos sobrinhos (*fi*) vermelhos
  - É necessário fazer algum tipo de rotação
    - Depende de qual *fi* é vermelho (por exemplo se ambos forem vermelhos, pode ser aplicado algum tipo de rotação simples)
  - O nó que irá ocupar o lugar de *p* terá a mesma cor que o respectivo nó tinha



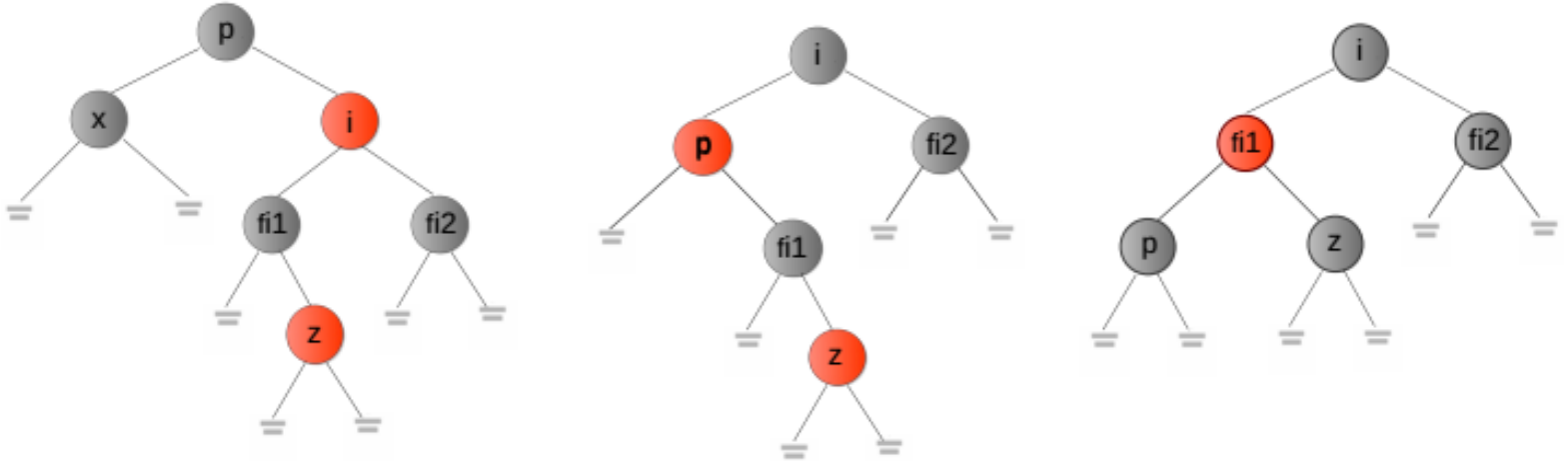
Remoção

- Caso 4: o irmão do nó removido é vermelho



Remoção

- Caso 4: o irmão do nó removido é vermelho



Remoção

- Exercício: implemente um algoritmo para a remoção de um nó em uma árvore rubro-negra de forma que a mantenha balanceada.