Algoritmos e Estrutura de Dados

Aula 15 – Estrutura de Dados: Árvores Vermelho-Preto

Prof. Tiago A. E. Ferreira

Introdução

- Um árvore vermelho-preto é uma árvore binária onde cada nodo tem um campo extra para a cor do nodo
  - Cor:
    - Vermelho
    - Preto
  - Todo caminho em uma árvore vermelho-preto nunca é maior que o dobro de qualquer outro caminho.
    - Árvore aproximadamente balanceada.

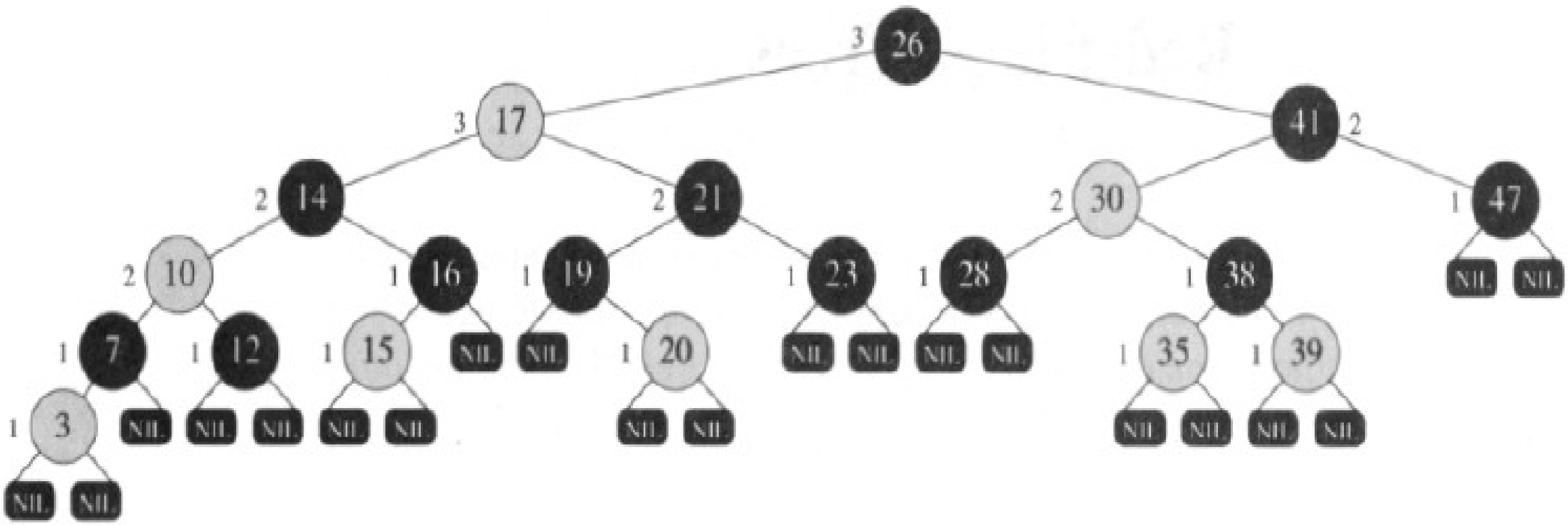
Nodo de Árvores Vermelho-Preto

- De forma geral um nodo de uma árvore vermelho-preta tem os campos:
  - Cor
  - Dados
  - Pai
  - Esquerdo
  - Direito

Condições

- Para uma árvore binária ser vermelho-preta deve:
  - Todo node é vermelho ou preto
  - A raiz é preta
  - Todo filho de uma folha (None) é preto
  - Se um nodo é vermelho, então ambos seus filhos são protos
  - Para cada nodo, todos os caminho de um nodo até as folhas descendentes contêm o mesmo número de nodos pretos.

Árvore Vermelho-Preto



Altura de Preto

- Dada uma árvore vermelho-preto:
  - É possível definir a altura preto de um nodo x, denotada por bh(x), como:
    - Número de nodos pretos em qualquer caminho desde um nodo x, sem incluir esse nodo, até uma folha.

Boas Árvores de Busca

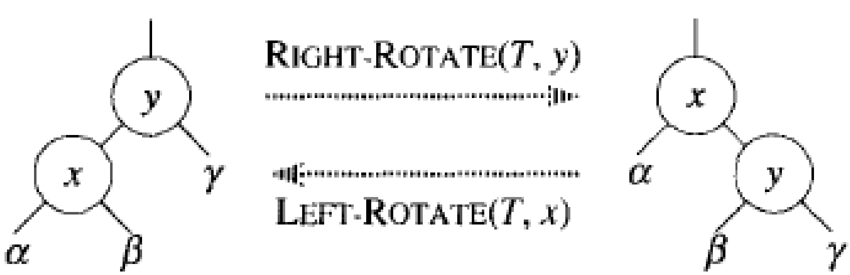
- **Lema 13.1:**
  - Uma árvore vermelho-preta com **n** nodos internos tem altura máxima de **2lg(n+1)**.
- Prova?

Inserção e Exclusão de Elementos

- Dada uma árvore vermelho-preta
  - Ao inserir um novo elemento ou ao deletar um elemento, as propriedades das árvores vermelho-preto podem ser comprometidas
  - Utiliza-se as operações de rotação para corrigir a distribuição de nodos vermelhos e pretos.

Rotação a Esquerda

- Ao realizarmos uma rotação para à esquerda, em torno do nodo x, é suposto que y é não nulo.
- A rotação à esquerda “faz o pivô” na ligação entre x e y.
  - Y vira a nova raiz da árvore
  - X vira o filho esquerdo de Y
  - O filho esquerdo de Y vira filho direito de X



Inserindo Elementos

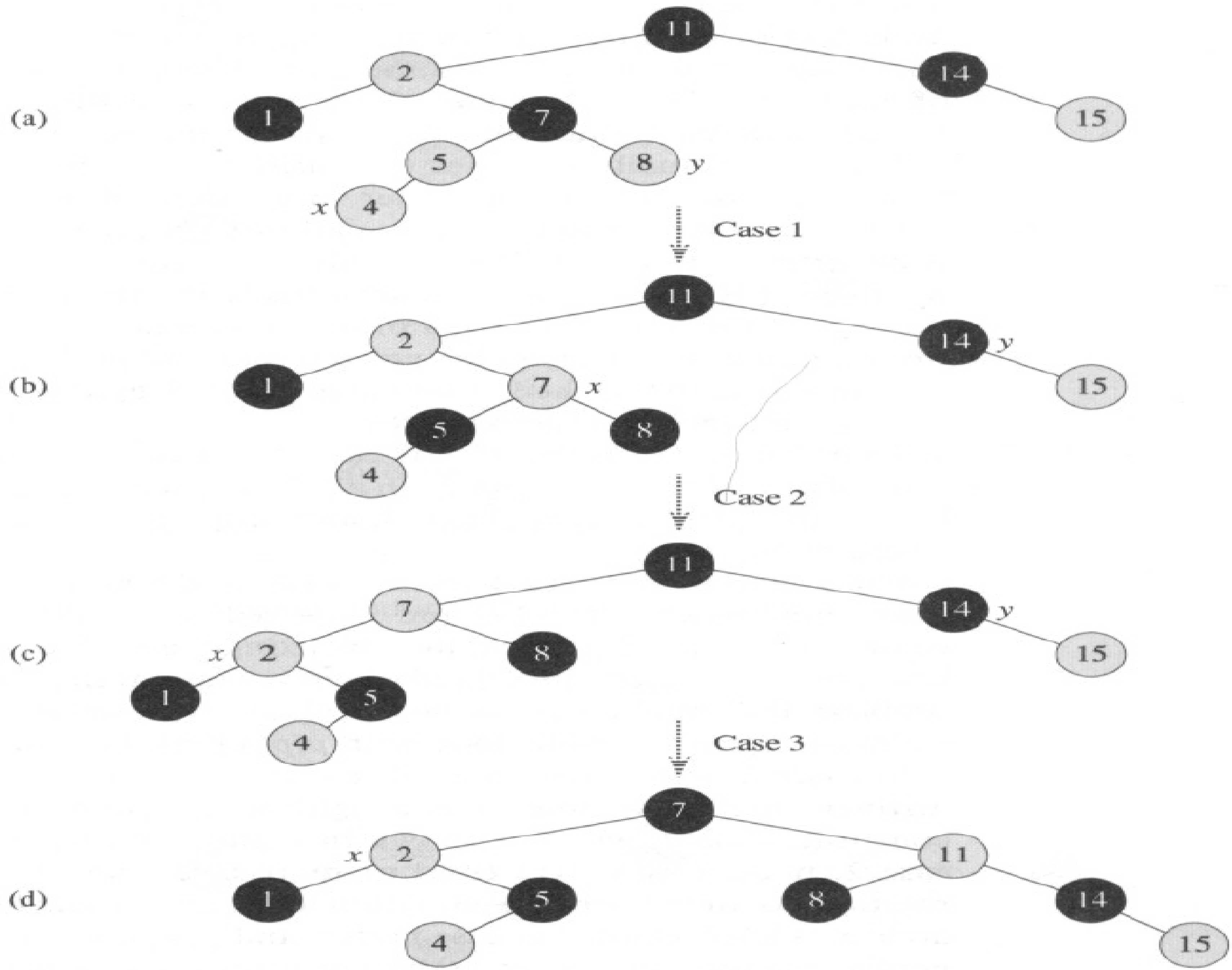
- Para inserir um elemento em uma árvore vermelha-preto:
  - Inicialmente, um nodo Z é inserido como se a árvore fosse uma árvore de pesquisa binária comum.
  - Colori-se Z de vermelho.
  - Invoca-se a função RB-INSERT-FIXUP para recolorir e executar rotações na árvore.

Pseoudo códogo: RB-Insert(T,z)

```
RB-INSERT(T, z)
y ← nil[T]
x ← raiz[T]
while x ≠ nil[T]
  do y ← x
     if chave[z] < chave[x]
       then x ← esquerda[x]
       else x ← direita[x]
p[z] ← y
if y = nil[T]
  then raiz[T] ← z
  else if chave[z] < chave[y]
         then esquerda[y] ← z
         else direita[y] ← z
esquerda[z] ← nil[T]
direita[z] ← nil[T]
cor[z] ← VERMELHO
RB-INSERT-FIXUP(T, z)
```

Pseudo Código: RB-Insert-Fixup(T,z)

```
RB-INSERT-FIXUP(T, z)
while cor[p[z]] = VERMELHO
    do if p[z] = esquerda[p[p[z]]]
        then y ← direita[p[p[z]]]
            if cor[y] ← VERMELHO
              then cor[p[z]] ← PRETO                ▷ Caso 1
                   cor[y] ← PRETO                   ▷ Caso 1
                   cor[p[p[x]]] ← VERMELHO          ▷ Caso 1
                   z ← p[p[z]]                      ▷ Caso 1
              else if z = direita[p[z]]
                   then z ← p[z]                    ▷ Caso 2
                        LEFT-ROTATE(T, z)           ▷ Caso 2
                   cor[p[z]] ← PRETO                ▷ Caso 3
                   cor[p[p[z]]] ← VERMELHO          ▷ Caso 3
                   RIGHT-ROTATE(T, p[p[z]])         ▷ Caso 3
        else (igual a cláusula then
                   com "direita" e "esquerda" trocadas)
cor[raiz[T]] ← PRETO
```


(a)
11
2
14
1
7
15
5
8
y
x
4
Case 1
(b)
11
2
14
y
1
7
x
15
5
8
4
Case 2
(c)
11
7
14
y
x
2
8
15
1
5
4
Case 3
(d)
7
x
2
11
1
5
8
14
4
15

Entendendo RB-Insert-FixUp

- Passos:
  - Determinar as violações das regras da árvore vermelho-preto com a inserção e coloração do novo nodo z
  - Examinar a meta global do loop while, linhas 1 a 15
  - Examinar os três casos que o loop while se divide
- Quais das propriedades da árvore vermelho-preta podem ser violadas?
  - As propriedade 2 e 4!
    - A raiz deve ser preta.
    - Um nó vermelho não pode ter filho vermelho.

O Loop While

- No início, o *loop while* tem três invariantes:
  - O nodo z é vermelho
  - Se P[z] é a raiz, então P[z] é preto.
  - Se existir algum violação das propriedades vermelho-preto, existirá apenas uma única violação:
    - Se z é a raiz e é vermelho (propriedade 2)
    - Se o pai de z é vermelho e z é vermelho (propriedade 4)

Loop Invariante

- Inicialização:
  - Começa-se com uma árvore vermelho-preto sem violações e começamos com a inserção de z.
    - Quando RB-Insert-FixUp é chamado z é o nodo vermelho que foi adcionado
    - Se P[z] é a raiz, ete começou preto e não é modificado
    - Se houver violações, são da propriedade 2 ou da propriedade 4 (exclusivamente).

Loop Invariante

- Término:
  - Quando o loop termina, P[z] é preto (propriedade 4)
  - A propriedade 2 é garantida na linha 16.
- Manutenção:
  - Só entra-se no loop se P[z] for vermelho, caso contrário, P[z] é preto.

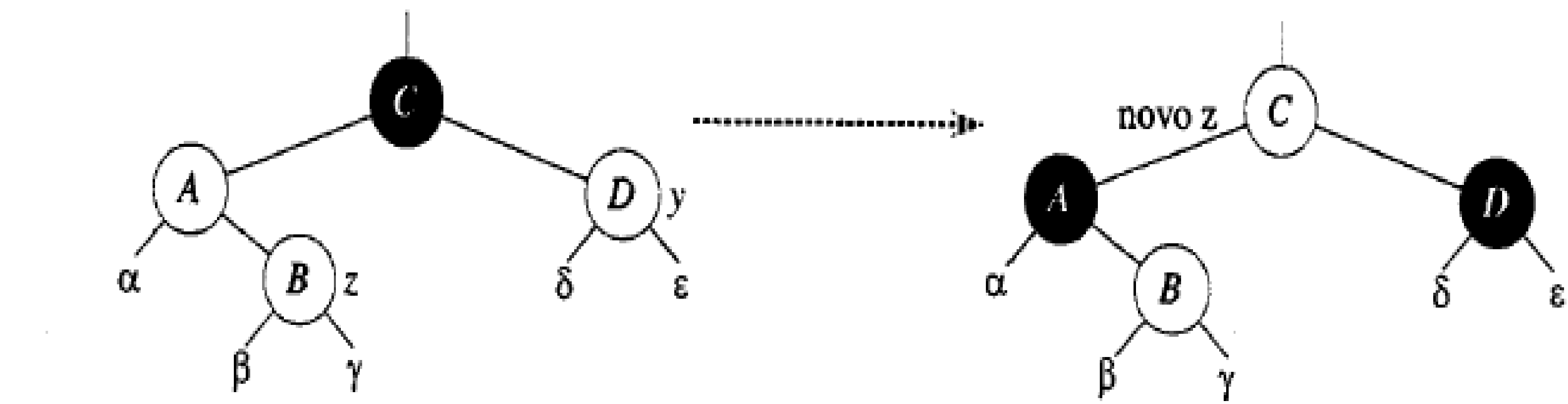
Inserir e Deletar

- Dada uma árvore binária, é possível inserir e/ou deletar nodos da árvore.
  - Porém, as propriedades da árvore devem ser mantidas!

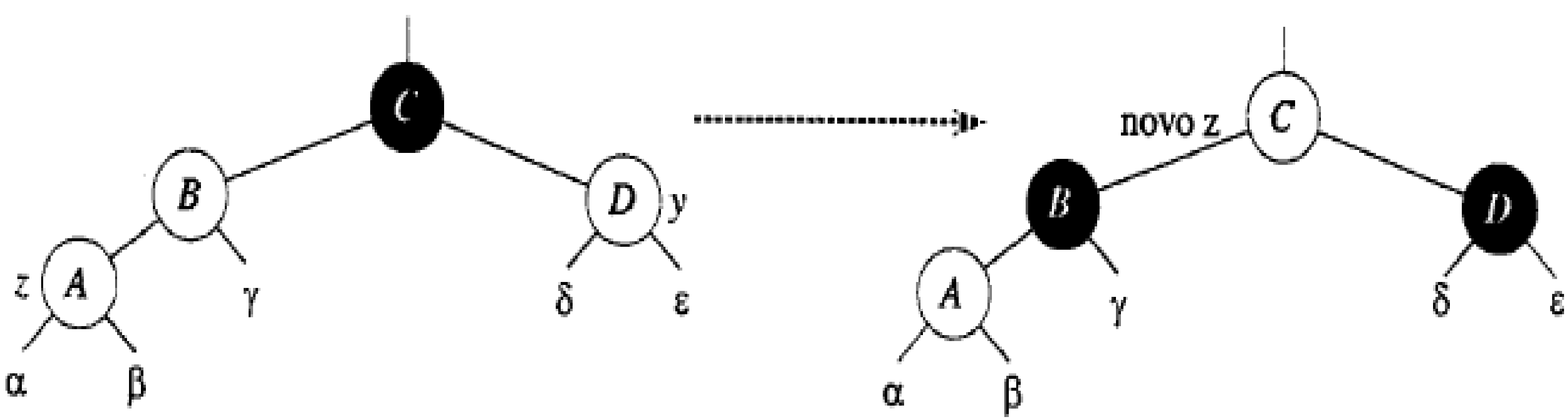
Analisando: Caso 1

Z e P[z] são vermelhos:

(a)

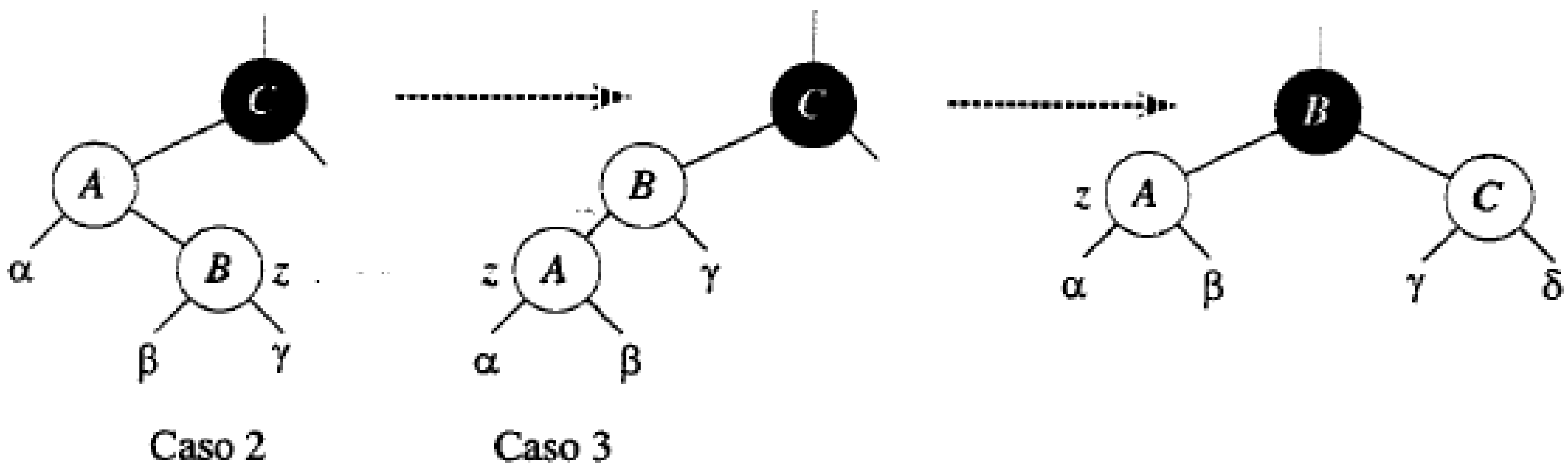


(b)



Análise: Casos 2 e 3

- Caso 2: o tio y de z é preto e z é filho da direita
- Caso 3: o tio y de z é preto e z é filho da esquerda



Exercícios Práticos

- Exercício : Implementar uma classe do nodo vermelho-preto. Implementar também uma classe de uma árvore vermelho-preta com a função de inserção de um elemento.