Disciplina: Algoritmos e Estruturas de Dados II
Professor: Raimundo Osvaldo Vieira

Aula 06

Árvores Red-Black

Referências

CORMEN, Thomas H. **Algoritmos: Teoria e Prática**. Rio de Janeiro: Elsevier, 2012

Resumo Teórico: *O Básico de Árvores*

Uma árvore vermelho-preto é uma árvore de busca binária com um bit extra de armazenamento por nó (sua cor: vermelho ou preto) (Cormen *et al.*, 2012)

A restrição de cores nos nós garante o "balanceamento" da árvore.

Numa árvore vermelho-preto o comprimento de qualquer caminho simples da raiz até uma folha não é maior que duas vezes o de qualquer outro.

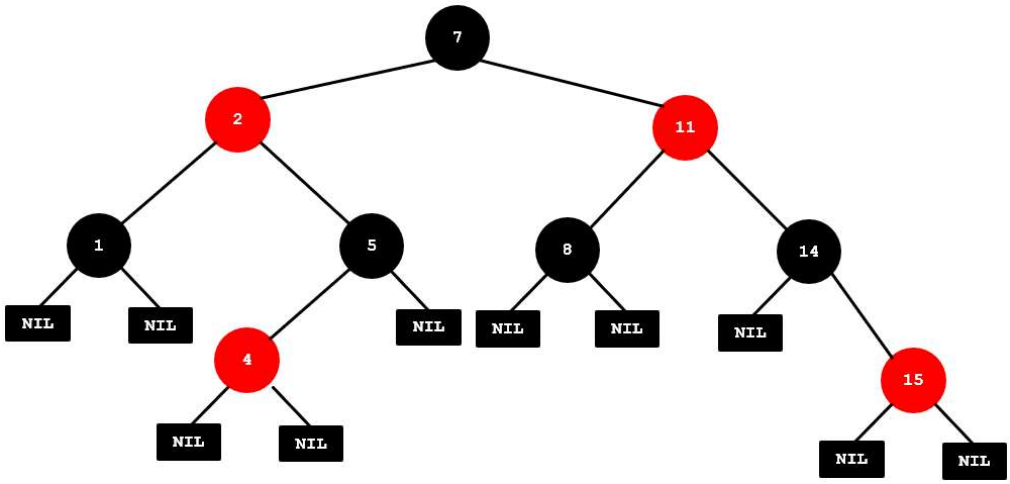

Figura 1 Árvore Vermelho-Preto
Fonte: [Cormen *et al.*, 2012]

Propriedades

1. Todo nó é vermelho ou preto
2. A raiz é preta
3. Toda folha NIL é preta
4. Se um nó é vermelho, ambos os seus filhos são pretos
5. Para cada nó, todos os caminhos, desde um nó até as folhas descendentes, contêm o mesmo número de nós pretos (**altura preta**).

Deve-se notar que a árvore ilustrada na Figura 1 satisfaz todas as propriedades elencadas. De fato, todos os nós são de cor vermelha ou preta, a raiz (7) é preta, todos os nós folha (`null`) são pretos, os nós vermelhos (2, 4, 11 e 15) só possuem filhos pretos e qualquer caminho a partir de qualquer nó até as folhas descendentes possuem o mesmo número de nós pretos.

A quantidade nós pretos contabilizados a partir de um nó x até uma folha descendente (excluindo o nó) é chamada de altura preta desse nó, sendo denotada por $hb(x)$. A altura preta da raiz determina também a altura preta da árvore. Por exemplo, a árvore ilustrada na Figura 1 tem altura preta igual a 2, posto que qualquer caminho a partir da raiz até uma folha tem exatamente 2 nós pretos, excluída a raiz.

Uma vez respeitadas todas as propriedades citadas, garante-se que a altura de uma árvore vermelho-preto não é maior que $2 \cdot \log(n+1)$, em que n é o número de nós da árvore. Esta característica assegura que quaisquer operações básicas são executadas no tempo $T = O(\log n)$.

Uma Implementação

Um nó deve conter os seguintes campos: `cor` (vermelho ou preto), `info` (dados armazenados), `esquerda` (ponteiro para outro nó – filho esquerdo), `direita` (filho direito) e `p` (nó pai). Quando um (ou dois) filho(s) ou o pai de um nó não existir, este será representado por uma referência nula (`null`).

Podemos implementar uma única sentinela para representar o **`null`**(`T.Nil`), que é sempre preta, e cujas alturas pretas são omitidas. O pai da raiz também é uma sentinela (Cormen *et al.*, 2012). Veja a Figura:

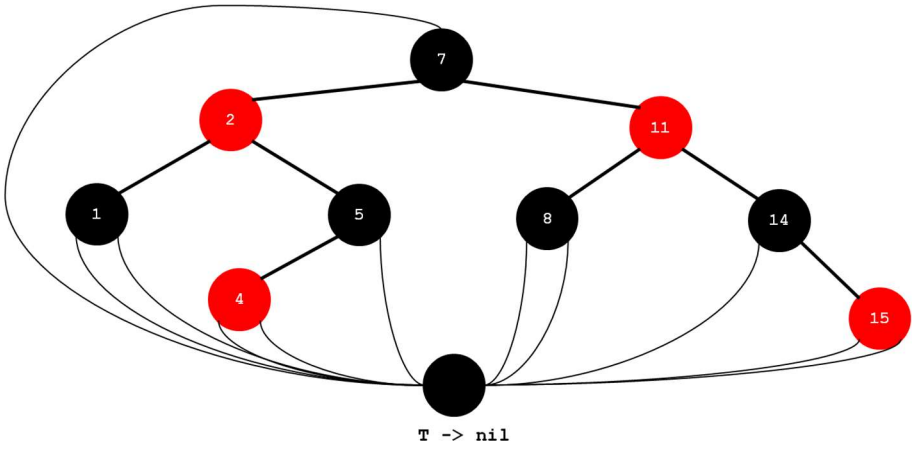

Figura 2 Árvore vermelho-preto com uma única sentinela
Fonte: [CORMEN et. Al, 2012]

Resumo Teórico: *Operações em Árvores Red-Black*

Inserção

A inserção numa árvore vermelho-preto é realizada conforme a inserção em árvores binárias de busca tradicionais. Entretanto, numa árvore vermelho-preto todo novo nó inserido deve ser `VERMELHO`, o que pode violar as propriedades apresentadas anteriormente. É preciso, portanto, garantir que essas propriedades sejam preservadas. Para isso, faz-se necessária a definição de um procedimento que realize rotações e recoloração dos nós de modo a garantir que sejam atendidas todas as propriedades das árvores vermelho-preto.

```
RB-insert(T, z){
   y = T.Nil;
   x = T.raiz;
   while (x != T.Nil){
      y = x;
      if(z.chave < x.chave)
         x = x.esquerda;
      else
         x = x.direita;
   }
   z.p = y;
   if(y == T.Nil)
      T.raiz = z;
   else{
```

```
      if(z.chave < y.chave)
         y.esquerda = z;
      else
         y.direita = z;
   }
   z.esquerda = T.Nil;
   z.direita = T.Nil;
   z.cor = VERMELHO;
   RB-insert-fixup(T, z);
}
```

Após a inserção de um novo nó n numa árvore vermelho-preto, as propriedades 1, 3 e 5 continuam intactas enquanto as propriedades 2 e 4 podem ser violadas. Para melhor entendimento, é possível descrever as violações dividindo-as em 3 casos, conforme ilustrado na Figura 3.

1. **Caso 01** – **o tio do novo nó n é `VERMELHO`** (Figura 3a): neste caso, a propriedade 4 é violado, visto que n e seu pai são vermelhos. Como o tio de n também é vermelho a restauração das propriedades se dá pela recoloração de n, de seu pai e de seu tio. Após isso, o ponteiro n é movimentado para cima na árvore. O resultado é mostrado na Figura 3b.
2. **Caso 02** – **o tio de n é `PRETO` e n é um filho direito** (Figura 3b): neste caso, a propriedade 4 permanece violada, porém o tio de n é preto. Além disso, n é um filho à direita e a violação da propriedade ocorreu à esquerda da raiz. A solução para este caso consiste na execução de uma rotação à esquerda em $n.p$, transformando o arranjo para o caso 3. O resultado é mostrado na Figura 3c.

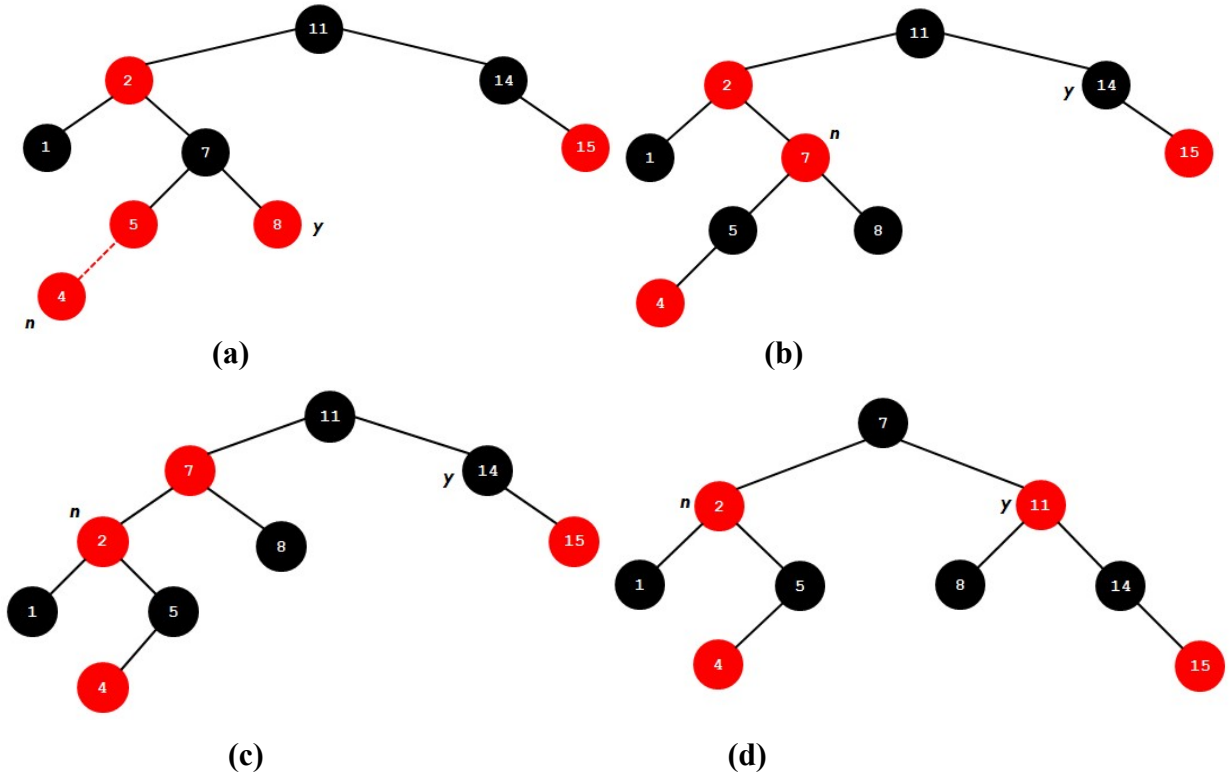

Figura 3 Operações de correção após inserção de um novo nó numa árvore vermelho-preto

3. **Caso 3 – o tio de n é `PRETO` e n é um filho à esquerda** (Figura 3c): neste caso, temos uma situação semelhando ao caso 2, com a diferença de que n, estando à esquerda da raiz, é filho esquerdo de $n.p$. A solução agora consiste em recolorir o pai e o avô de n e realizar uma rotação à direita em $n.p$. O resultado é a árvore vermelho-preta válida exibida na Figura 3d.

```
RB-insert-fixup(T, z){
   while(z.p.cor == VERMELHO){
      if(z.p == z.p.p.esquerda){
         y = z.p.p.direita;
         if(y.cor == VERMELHO){
            z.p.cor = PRETO;
            y.cor = PRETO;
            z.p.p.cor = VERMELHO;
            z = z.p.p;
         }
         else{
            if(z = z.p.direita){
               z = z.p;
               Rotação-esquerda(z);
            }
            z.p.cor = PRETO;
            z.p.p.cor = VERMELHO;
            Rotação-direita(z.p.p);
         }
      }
      else{
         //A solução para o lado direito é simétrica
      }
   }
   T.raiz.cor = PRETO;
}
```

Remoção

O procedimento de remoção de um nó em uma árvore vermelho-preto é composto de uma etapa de remoção em árvore binária de busca seguido de uma etapa de balanceamento, caso alguma das propriedades apresentadas na seção anterior tenha sido violada durante a operação. O processo de exclusão considera três situações:

1 Quando o nó z a ser excluído não possui filho esquerdo, neste caso basta realizar uma chamada ao procedimento `RB-transplant` para que $z.direita$ assuma o lugar de z na árvore.

```
RB-transplant(T, u, v){
   if(u.p == T.Nil)
      T.raiz = v;
   else{
      if(u == u.p.esquerda)
         u.p.esquerda = v;
      else
         u.p.direita = v;
   }
   v.p = u.p;
}
```

2 De modo semelhante, se z não possui filho direito, $z.esquerda$ é transplantado para o lugar de z na árvore.

3 Quando z possui ambos os filhos, é preciso encontrar um nó que possa substituí-lo, que é o menor elemento de sua subárvore direita. Isto é feito através de uma chamada à função `tree-minimum`. A posição original do nó y é ocupada por outro nó (x).

```
RB-remove(T, z){
   y = z;
   y.corOriginal = y.cor;
   if(z.esquerda == T.Nil){
      x = z.direita;
      RB-transplant(T, z, z.direita);
   }
   else{
      if(z.direita == T.Nil){
         x = z.esquerda;
         RB-transplant(T, z, z.esquerda);
      }
      else{
         y = tree-minimum(z.direita);
         y.corOriginal = y.cor;
         x = y.direita;
         if(y.p == z){
            x.p = y;
         }
         else{
            RB-transplant(T, y, y.direita);
            y.direita = z.esquerda;
            y.esquerda.p = y;
            y.cor = z.cor;
         }
      }
   }
   if(y.corOriginal == PRETO)
      RB-remove-fixup(T, x);
}
```

Como é possível excluir qualquer nó de uma árvore vermelho-preto é necessário prever duas situações: 1) o nó excluído é VERMELHO; 2) o nó excluído é PRETO. A situação 1 não gera preocupações, visto que a eliminação de um nó VERMELHO não viola nenhuma das propriedades. Entretanto, se o nó excluído for PRETO podem ser violadas as propriedades 2, 4 e 5. A violação da propriedade 2 ocorre quando a raiz é excluída e um filho VERMELHO passa a ser a nova raiz. A propriedade 4 é desrespeitada quando $\boldsymbol{x}$ (que assume o lugar de $\boldsymbol{y}$) e $\boldsymbol{y.pai}$ é VERMELHO. Por fim, a propriedade 5 é rompida quando a eliminação do nó $\boldsymbol{y}$ provoca um desnível na altura preta de um ou mais caminhos, o que é corrigido quando se considera que o nó $\boldsymbol{x}$, que assume o lugar de $\boldsymbol{y}$, carrega consigo um PRETO "extra". Para melhor entendimento, é possível descrever tais violações em quatro casos, conforme ilustrado na Figura 4.

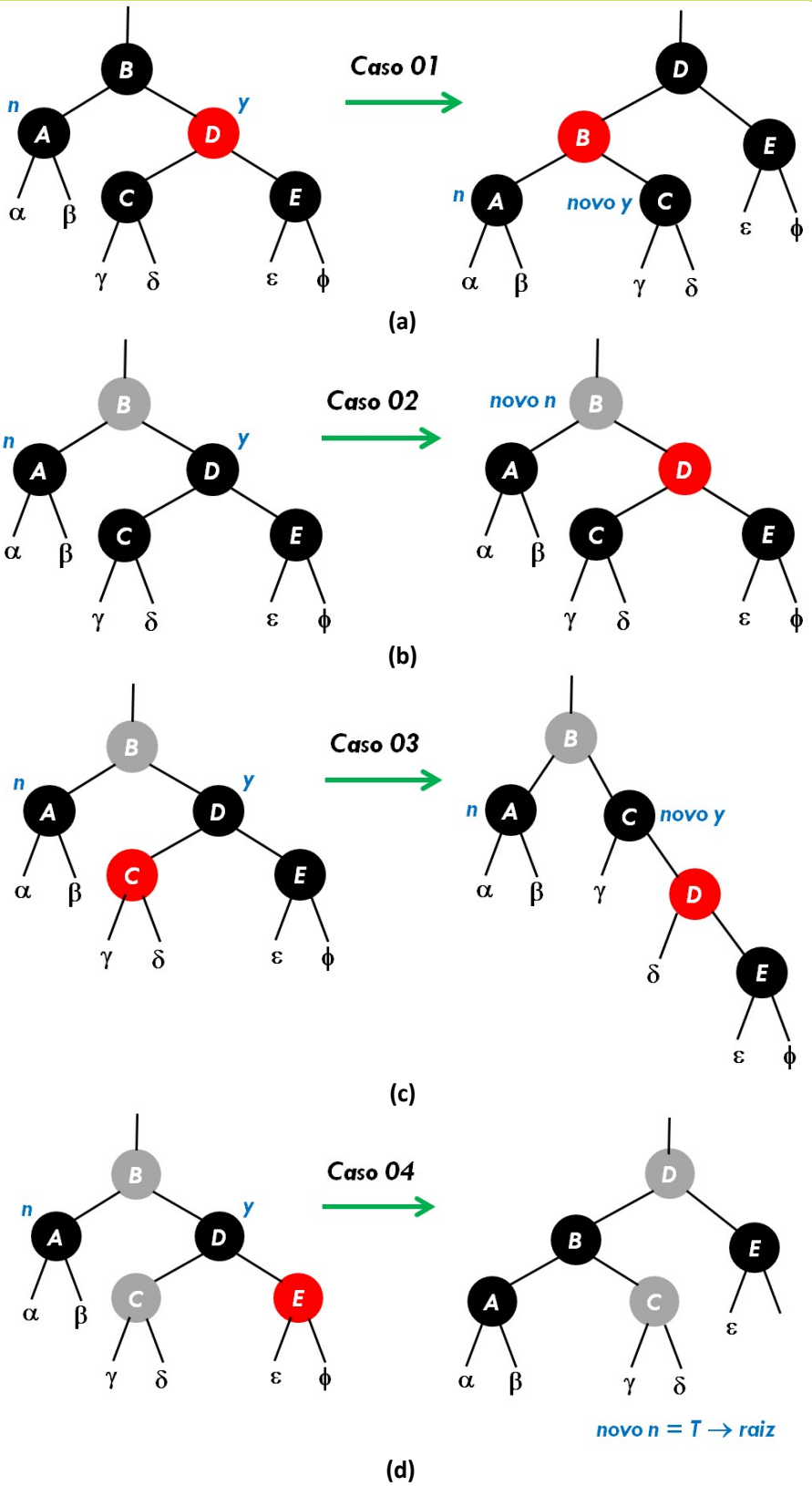

Figura 4 Remoção de um nó com dois filhos
Fonte: [Cormen *et al.*, 2012]

1. **Caso 01 – o irmão de n (y) é VERMELHO** (Figura 4a): neste caso a solução é transformar este caso em qualquer um dos demais. Para isso, devem ser trocadas as cores do nó pai e do nó y e, em seguida, realizar uma rotação à esquerda em $n.p$
2. **Caso 02 – o irmão de n (y) é PRETO e ambos os filhos de y são PRETO** (Figura 4b): neste caso, o nó n carrega o PRETO "extra", que deve ser eliminado, jogando-o para cima até que seja encontrado um nó vermelho ou até que se atinja a raiz

3. **Caso 03 – o irmão de n (y) é PRETO, o filho esquerdo de y é VERMELHO e o filho direito é PRETO** (Figura 4c): neste caso, é necessário fazer uma conversão para o caso 4. Para isso, são trocadas as cores de y e de seu filho esquerdo. Em seguida, executa-se uma rotação à direita em y.
4. **Caso 04 – o irmão de n (y) é PRETO e o filho direito de y é VERMELHO** (Figura 4d): neste caso, o PRETO "extra" é eliminado permutando-se as cores dos nós y e $y.direita$ e realizando-se uma rotação à esquerda em $n.p$. Por fim, n é definido como raiz para que seja encerrado o laço de repetição.

```
RB-remove-fixup(T, x){
   while(x != T.raiz && x.cor == PRETO){
      if(x == x.p.esquerda){
         w = x.p.direita;
         if(w.cor == VERMELHO){
            w.cor = PRETO;
            x.p.cor = VERMELHO;
            Rotação-esquerda(T, x.p);
            w = x.p.direita;
         }
         if(w.esquerda.cor == PRETO && w.direita.cor == PRETO){
            w.cor = VERMELHO;
            x = x.p;
         }
         else{
            if(w.direita.cor == PRETO){
               w.esquerda.cor = PRETO;
               w.cor = VERMELHO;
               Rotação-direita(T, w)
               w = x.p.direita;
            }
            w.cor = x.p.cor;
            x.p.cor = PRETO;
            w.direita.cor = PRETO;
            Rotação-esquerda(T, x.p);
            x = T.raiz;
         }
      }
      else{
         //A solução para o lado direito é simétrica
      }
   }
   x.cor = PRETO;
}
```

Deve-se destacar que a estrutura `while` é repetida até que o `PRETO` "extra" seja eliminado, o que ocorrer pela execução dos casos 2, 3 e/ou 4. A eliminação do `PRETO` "extra" pelo caso 2 merece atenção. Essa eliminação só se dá no caso 2 quando este é executado em sequência ao caso 1, visto que o nó B (Figura 4b) é `VERMELHO` e, com isso, torna possível a tal eliminação. Quando o caso 2 é executado sem que o caso 1 o preceda, o nó B é preto e, deste modo, ao receber x, passa a ser o `PRETO` "extra", provocando uma repetição do corpo do `while`.

Busca

A operação de busca é a mais simples das operações básicas em árvores vermelho-preto e consiste em percorrer a árvore até que o dado procurado seja encontrado ou que se chegue a um nó folha, o que representa que o dado não foi encontrado. Esta função inicia a busca pela raiz da árvore, desloca-se para direita ou para a esquerda de acordo com o resultado de sua comparação com a chave armazenada em cada nó. A implementação é conforme apresentado para as árvores binárias de busca (Aula 04).

Esquema de Estudo

Parte I: Fundamentos e Propriedades

1. Definição e Objetivo Explique o que diferencia uma Árvore Vermelho-Preto de uma árvore binária de busca comum. Qual é o mecanismo principal utilizado para garantir o balanceamento da estrutura? *Dica: Verifique a definição sobre o bit extra de armazenamento e a restrição de caminhos no Resumo Teórico.*

2. As 5 Propriedades Canônicas Para que uma árvore seja considerada Vermelho-Preto, ela deve satisfazer 5 propriedades rigorosas. Liste essas propriedades relacionando-as aos nós, à raiz, às folhas e aos caminhos. *Dica: As propriedades estão listadas na tabela da Figura 1.*

3. Altura Preta (Black Height) Defina o conceito de "altura preta". Como ela é calculada e qual a relação da altura preta da raiz com a altura da árvore? *Dica: Consulte a explicação sobre a contagem de nós pretos até as folhas descendentes.*

Parte II: Estrutura e Implementação

4. Estrutura do Nó Quais são os campos necessários para implementar um nó de uma Árvore Vermelho-Preto? Como são representados os filhos ou pais inexistentes? *Dica: Analise a seção "Uma Implementação" para identificar os ponteiros e o campo de cor.*

5. O Papel da Sentinela (`T.Nil`) Na implementação sugerida, como é tratado o valor `null`? Qual é a cor da sentinela e por que ela é utilizada? *Dica: Veja a descrição da sentinela única e a Figura 2*

Parte III: Operações - Inserção

6. Algoritmo de Inserção e Cor Inicial Ao inserir um novo nó na árvore, qual cor ele recebe inicialmente? Essa escolha preserva quais propriedades imediatamente, mas arrisca violar quais outras? *Dica: O texto menciona que todo novo nó é VERMELHO e discute a violação das propriedades 2 e 4.*

7. Casos de Correção na Inserção (Fixup) A correção após a inserção depende fundamentalmente da cor do "tio" do novo nó. Descreva a diferença na solução para:

- Caso 1: Quando o tio do novo nó é VERMELHO.
- Caso 2 e 3: Quando o tio do novo nó é PRETO (diferenciando se o nó é filho à direita ou à esquerda). *Dica: Utilize as descrições dos Casos 01, 02 e 03 na seção de Inserção.*

Parte IV: Operações - Remoção

8. Remoção e Impacto nas Cores A remoção de um nó VERMELHO causa problemas estruturais na árvore? E a remoção de um nó PRETO? Explique quais propriedades podem ser violadas na remoção de um nó preto. *Dica: Consulte o parágrafo que descreve as duas situações de exclusão e as violações das propriedades 2, 4 e 5.*

9. O Conceito de "Preto Extra" Durante a correção da remoção (fixup), o texto menciona que um nó pode carregar um "PRETO extra". Qual é o objetivo desse conceito e como ele ajuda a restaurar a propriedade da altura preta? *Dica: Verifique a explicação sobre a violação da propriedade 5 e os casos de ajuste na Figura 4.*

10. Transplant Qual é a função do procedimento RB-transplant durante a remoção de um nó? *Dica: Analise o código e a explicação sobre substituir um nó por seu filho ou sucessor.*

Parte V: Análise de Desempenho

11. Complexidade de Tempo Qual é a garantia de altura máxima de uma Árvore Vermelho-Preto com nós? Qual é a complexidade de tempo (Big-O) para as operações básicas (inserção, remoção, busca) garantida por essa estrutura? *Dica: O texto afirma que a altura não é maior que.*

Tarefas Propostas

1. Implemente o TAD Árvore Red-Black.
2. Mostre as árvores Red-Black que resultam após a inserção sucessiva das chaves 41, 38, 31, 12, 19, 8 em uma árvore inicialmente vazia.
3. Considerando aárvore final da questão anterior, mostre as árvores Red-Black resultantes da eliminação sucessiva das chaves na seguinte ordem 8, 12, 19, 31, 38, 41.
4. Suponha que um nó x seja inserido numa árvore Red-Black com RB-insert e então imediatamente eliminado com RB-remove. A árvore Red-Black resultante é igual à árvore Red-Black Inicial? Justifique sua resposta.

Atividade Prática em Equipe

Realizar essa atividade nas mesmas equipes dos trabalhos anteriores.
Data para Entrega: 30/ 12/ 2025
Apresentações por equipe: 13 e 15/ 01/ 2026 (agendar horário com o professor)

Parte 1: Auditoria e Garantia de Propriedades (Corretude)

Com base na API *Red-Black* implementada, você deve desenvolver um módulo de testes unitários ou uma função de validação chamada `verificarIntegridade(T)` que percorra a árvore e assegure que nenhuma das propriedades fundamentais foi violada após operações de inserção e remoção.

Seu verificador deve garantir os seguintes pontos descritos:

1. Raiz e Folhas: A raiz deve ser PRETA e todas as folhas (T.Nil) devem ser PRETAS.

2. Propriedade Vermelha: Se um nó é VERMELHO, ambos os filhos devem ser PRETOS (não podem haver dois vermelhos consecutivos).

3. Altura Preta (Black-Height - hb(x)): Para cada nó, todos os caminhos até as folhas descendentes devem conter exatamente o mesmo número de nós pretos.

Cenários de Teste Obrigatórios: Você deve criar um script que force a execução dos algoritmos de correção (fixup) descritos no material:

- **Teste de Inserção (Casos 1, 2 e 3):** Insira chaves que forcem o "Caso 1" (Tio Vermelho -> apenas recoloração) e os "Casos 2 e 3" (Tio Preto -> exigem rotação).
- **Teste de Remoção de Nó Preto:** Remova nós pretos para forçar o tratamento do "Preto Extra" através dos 4 casos de remoção descritos na Figura 4 do material, validando se a altura preta permanece constante após a operação.

Parte 2: Estudo de Caso – "Logs em Tempo Real vs. Arquivo Morto"

Nesta etapa, você realizará um estudo comparativo entre sua implementação Red-Black e uma implementação de árvore AVL (implementação já realizada). O objetivo é analisar o *trade-off* entre a rigidez do balanceamento da AVL e a flexibilidade da Red-Black.

O Cenário

Simule um sistema de processamento de dados com três fases distintas. Para cada fase, meça o Tempo de CPU e o Número de Rotações realizadas.

Fase A: Ingestão (Escrita Intensiva)

• **Ação:** Inserir de inteiros aleatórios.

• **Análise Teórica:** Na Red-Black, todo novo nó entra como VERMELHO. Muitas correções (Caso 1) são resolvidas apenas recolorindo nós, sem rotações físicas.

• **Expectativa:** Compare a velocidade de inserção e a quantidade total de rotações da Red-Black versus a AVL.

Fase B: Consulta (Leitura Intensiva)

• **Ação:** Realizar buscas na estrutura preenchida. Metade das chaves devem existir, metade não.

• **Análise Teórica:** A altura da Red-Black é garantida como sendo no máximo. A AVL é mais estritamente balanceada.

• **Expectativa:** Verifique se a busca na AVL é perceptivelmente mais rápida e meça a altura máxima final de ambas as árvores.

Fase C: Limpeza (Remoção e Rebalanceamento)

• **Ação:** Remover 10% dos nós inseridos (ex: os primeiros 100.000).

• **Análise Teórica:** A remoção de um nó PRETO na Red-Black exige o tratamento complexo do "Preto Extra", podendo propagar o rebalanceamento até a raiz (loops no RB-remove-fixup).

• **Expectativa:** Analise o impacto no desempenho quando muitas remoções consecutivas exigem rebalanceamento.

Entregáveis

1. Código Fonte: A classe ArvoreRB contendo os métodos RB-insert, RB-remove e seus respectivos fixup, instrumentados com contadores para rotações.

2. Relatório Técnico:

◦ **Tabela comparativa contendo:** Altura Final, Total de Rotações (Inserção), Tempo de Inserção, Tempo de Busca e Tempo de Remoção.

◦ **Discussão**: Os resultados confirmam a teoria de que a Red-Black executa operações básicas em tempo com menos *overhead* de rotação que a AVL?