Árvores

Árvores *Red-Black* – Parte I: Definição e Inserção

Prof. Edson Alves - UnB/FGA

2018

Sumário

Definição

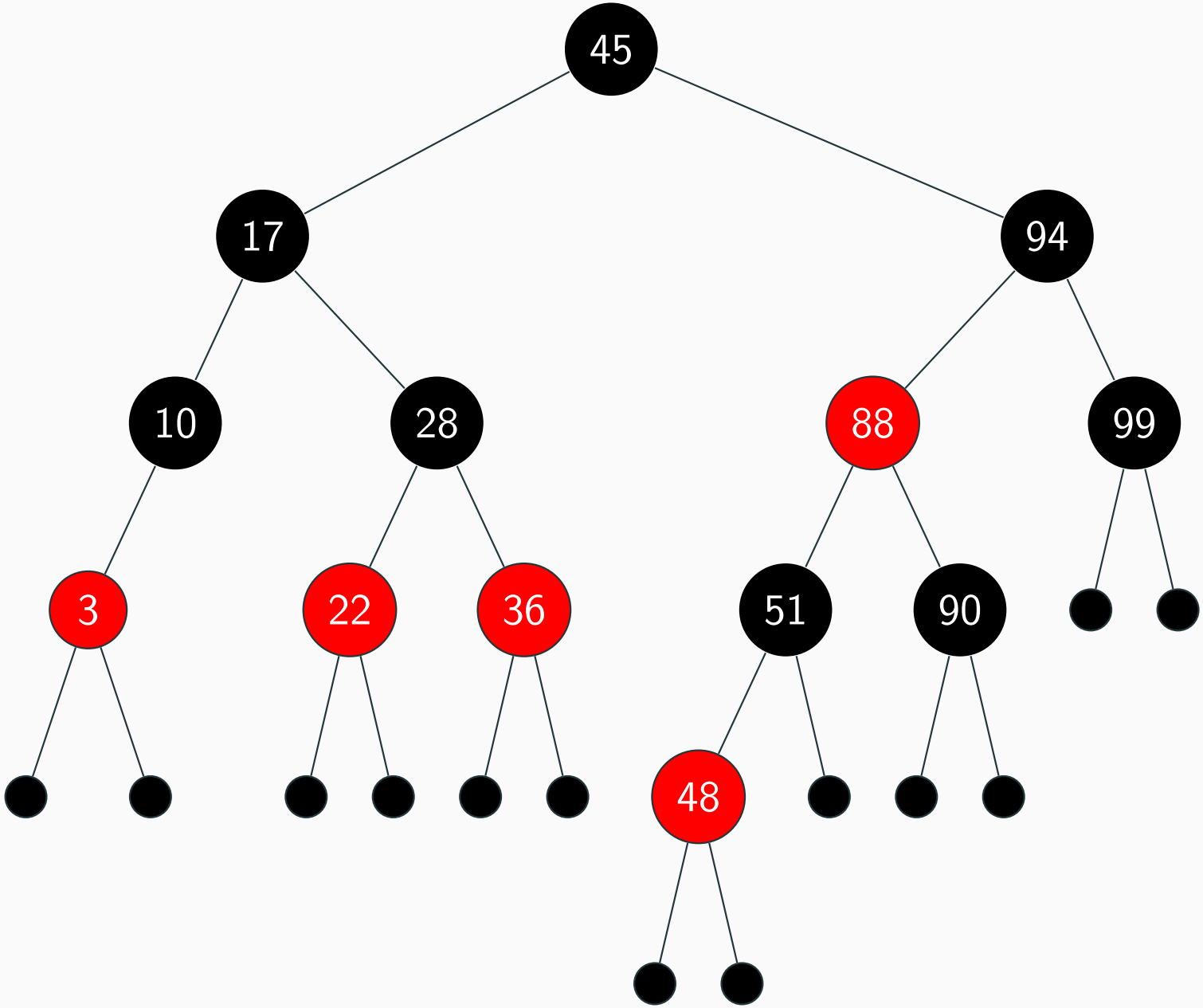
Árvores Red-Black

- Uma árvore *red-black* é uma árvore binária de busca auto-balanceável
- Foi proposta em 1978 por Leonidas J. Guibas e Robert Sedgewick
- A cada um de seus nós é atribuída uma cor: vermelha ou preta
- São estabelecidas 5 propriedades que relacionam os nós e suas cores
- Estas propriedades garantem que a altura h da árvore seja proporcional a $\log N$, onde N é o tamanho da árvore
- As rotinas de inserção e remoção devem preservar as propriedades das árvores *red-black* e, consequentemente, o balanceamento de sua altura

Propriedades de uma árvore red-black

1. Cada nó ou é vermelho ou é preto
2. A raiz é tem a cor preta
3. Todas as folhas são nulas e tem a cor preta
4. Se um nó é vermelho, então todos os seus filhos são pretos
5. Dado um nó n, todos os caminhos de n até um de seus descendentes nulos tem o mesmo número de nós pretos

Exemplo de árvore red-black



Observações sobre árvores red-black

- Cormen et. al propõem e demonstram o seguinte lema: "*A red-black tree with N internal nodes has height at most* $2\log(N+1)$"
- Este lema garante complexidade $O(\log N)$ para as operações de busca, inserção e remoção
- Como uma árvore *red-black* é uma árvore binária de busca, o algoritmo de busca é idêntico ao utilizado em árvores binárias de busca
- As inserções e remoções devem tratar as possíveis violações às propriedades das árvores *red-black*, de modo que as árvores resultantes sejam efetivamente árvores *red-black*
- Para implementar tais operações, é útil manter um ponteiro para o pai de cada nó
- É útil também implementar funções auxiliares que permitam acessar os ponteiros do avó, do tio e do irmão de um dado nó

Definição de uma árvore red-black em C++

```cpp
#include <bits/stdc++.h>

template<typename T>
class RBTree {
private:
    struct Node {
        T info;
        enum { RED, BLACK } color;
        Node *left, *right, *parent;
    };

    Node *root;

public:
    RBTree() : root(nullptr) {}

```

Funções auxiliares

```
17 private:
18     Node * parent(Node *node)
19     {
20         return node ? node->parent : nullptr;
21     }
22
23     Node * grandparent(Node* node)
24     {
25         return parent(parent(node));
26     }
27
28     Node* sibling(Node* node)
29     {
30         auto p = parent(node);
31         return p ? (node == p->left ? p->right : p->left) : nullptr;
32     }
33
34     Node * uncle(Node* node)
35     {
36         return sibling(parent(node));
37     }
```

Funções auxiliares

```
38
39      void rotate_left(Node *G, Node *P, Node *C)
40      {
41          if (G != nullptr)
42              G->left == P ? G->left = C : G->right = C;
43
44          P->right = C->left;
45          C->left = P;
46      }
47
48      void rotate_right(Node *G, Node *P, Node *C)
49      {
50          if (G != nullptr)
51              G->left == P ? G->left = C : G->right = C;
52
53          P->left = C->right;
54          C->right = P;
55      }
56
```

Inserção

Inserção em árvores red-black

- A inserção em uma árvore *red-black* consiste em duas etapas
- A primeira é a inserção de um nó vermelho, nos mesmos moldes da inserção em uma árvore binária de busca
- A segunda etapa consiste em corrigir possíveis violações às propriedades de uma árvore *red-black*
- O algoritmo $O(N \log N)$ que resulta em uma árvore balanceada é
  1. armazene os N elementos a serem inseridos em um vetor v
  2. ordene v em ordem crescente
  3. insira o elemento central x do vetor na árvore
  4. continue o processo no subvetor à esquerda de x
  5. finalize o processo no subvetor à direita de x

Referências

1. Red-Black Trees, acesso em 27/03/2019.
2. 8.2 Red-Black Trees, acesso em 27/03/2019.
3. Wikipédia. *Red-Black Tree*, acesso em 27/03/2019.[1]

[1] https://en.wikipedia.org/wiki/Red-black_tree