ESTRUTURA DE DADOS

Rafael Albuquerque

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS

Balanceamento de árvore

Objetivos de aprendizagem

Ao final deste texto, você deve apresentar os seguintes aprendizados:

- Identificar as definições de árvores AVL.
- Reconhecer os conceitos de árvores AVL — rotação direita.
- Definir os conceitos de árvores AVL — rotação esquerda.

Introdução

Uma das principais vantagens das árvores binárias em relação a outras estruturas é a eficiência na operação de busca. Uma árvore binária que minimiza o número esperado de comparações para um certo conjunto de valores e probabilidades é denominada como ótima. No entanto, não existe um algoritmo eficiente para a construção de uma árvore binária dessa magnificência para o caso geral. O que existe são métodos que aproximam essa eficiência, o que permite construir árvores quase perfeitas em tempo O(n) — linear. Para isso, um dos métodos explorados, conhecido como método de balanceamento, destina-se a encontrar um valor de simetria (–1, 0 ,1) e estabelecer um elemento *k* como raiz da árvore binária, com *k(l)* até *k(i – 1)* em sua subárvore esquerda, e *k(i + 1)* até *k(n)* em sua subárvore direita.

Neste capítulo, você estudará sobre as definições de árvores AVL e os conceitos de árvores AVL com rotação para a direita e para a esquerda. Além disso, você verá como implementar os métodos de rotação na linguagem de programação em Python 3.X pela ferramenta *on-line* Repl.

1 O que são árvores AVL?

De acordo com Piva Junior *et al.* (2014), as árvores AVL têm como principal característica a distribuição homogênea, a qual conhecemos como balanceamento em uma estrutura de dados de árvore, podendo esta ser do tipo binária ou N-ária. No que tange ao balanceamento, a ideia central é que, para cada

novo elemento adicionado ou removido da árvore, seja realizada uma reorganização para que a distribuição dos elementos, conforme a sua subárvore, continue homogênea.

Logo, mesmo se tratando da nomenclatura árvores AVL, estas contêm as mesmas propriedades das árvores binárias, com métodos de inserção, remoção e busca. Conforme Piva Junior *et al.* (2014), as árvores AVL correspondem à família das árvores binárias, na qual a distribuição dos elementos é feita de acordo com determinadas condições, que são necessárias para garantir o balanceamento da árvore. Segundo Tenenbaum, Langsam e Augenstein (1995), uma árvore balanceada é assim chamada por alguns critérios:

- Altura: a altura de uma árvore é o nível máximo de suas folhas. Em algumas literaturas, é determinado como profundidade de uma árvore. A altura de uma árvore nula, por exemplo, é definida como –1, e uma árvore binária balanceada é uma árvore na qual as alturas de suas subárvores (esquerda e direita) de todo nó nunca diferem em mais de 1.
- Balanceamento: o balanceamento de uma árvore é definido como a diferença entre a altura de sua subárvore esquerda em relação à altura de sua subárvore direita. Para melhor compreensão, cada nó da árvore carrega consigo o valor de seu balanceamento, podendo ser de: 1, –1 ou 0, o que depende da altura de sua subárvore esquerda ser maior, menor ou igual à altura de sua subárvore direita, respectivamente. A Figura 1 ilustra bem os valores mencionados para o balanceamento armazenado em cada nó.

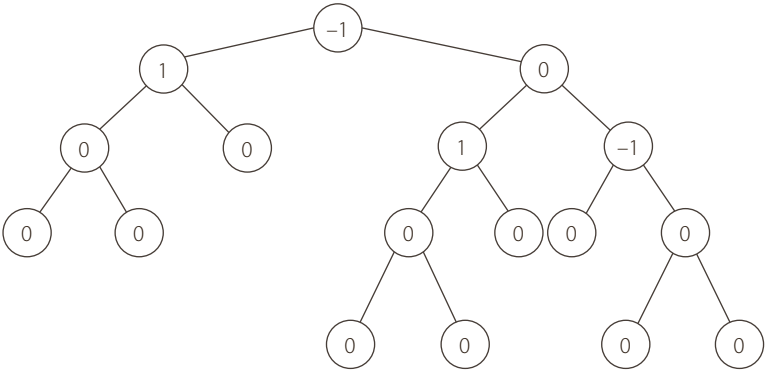

Figura 1. Árvore binária balanceada com seus valores de balanceamento.
Fonte: Adaptada de Tenenbaum, Langsam e Augenstein (1995).

De modo mais prático, a Figura 2 ilustra uma representatividade de árvores binárias AVL e árvores binárias sem balanceamento, com chaves do tipo inteiro adicionado a elas. Perceba que, na Figura 2a, a distância entre as subárvores esquerda e direita não difere de 1 entre elas. Todavia, na Figura 2b, é fácil notar que a subárvore direita é menor que a subárvore esquerda, diferindo em 2, conforme apresentado ao lado de cada nó.

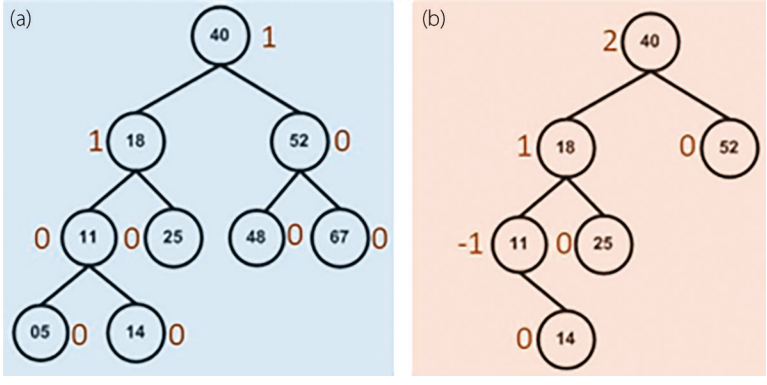

Figura 2. Árvores (a) balanceada e (b) não balanceada.
Fonte: Adaptada de Piva Junior *et al.* (2014).

Mesmo árvores AVL contendo as mesmas prioridades de uma árvore binária, como busca, inserção e remoção, a garantia do balanceamento é feita por meio da verificação da condicional de balanceamento, aqui tratada de acordo com a altura, de modo a não afetar o ordenamento das chaves. Em se tratando de árvores AVL, identificar a altura de cada nó é primordial e, para encontrá-la, será apresentada, a seguir, a implementação dessas funções, alinhadas à criação de uma rede binária simples.

Implementação das funções bases

Como supracitado, a Figura 3 apresenta uma árvore binária com seus valores bem distribuídos, conforme as regras dessa árvore. Podemos observar que aqui já temos que tratar a sua altura, pois a função de altura será fundamental para construirmos uma árvore binária com balanceamento.

```
def altura(self, no : No) -> int :
  altura_dir, altura_esq = 0,0
  if no == None:
    return -1
  else:
    altura_dir = self.altura(no.dir)
    altura_esq = self.altura(no.esq)

    # retorna a altura do lado esquerdo ou direito
    # uso do operador ternário do Python
    return (altura_dir + 1) if (altura_dir > altura_esq) else altura_esq + 1
```

Teste:

```
if __name__ == "__main__":
  # inicia a árvore
  arvore = Arvore()
  lista = [85, 75, 30, 50, 91,20, 98,95, 53]

  for chave in lista:
    arvore.inserir(chave)
  print("Altura da árvore: {}\n".format(arvore.medir_altura(arvore.raiz)))
```

Resultado:

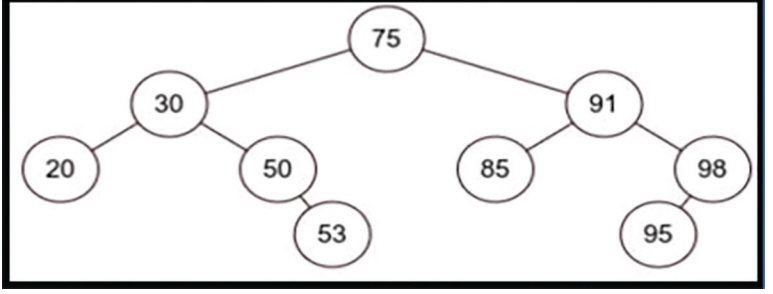

Figura 3. Apresentação de uma rede binária com testes.

Perceba que, em resultados, não há nenhum sinal de desbalanceamento. Caso seja adicionado a essa mesma árvore o elemento 52, o que provavelmente aconteceria? A resposta é nada, e o resultado seria conforme apresentado na Figura 4. Veja que existe um desbalanceamento no nível 2, com o nó de valor 50, no qual existem dois níveis a mais em sua ramificação direita, enquanto não existe nenhuma à sua esquerda. Esse problema, que afeta diretamente a ordenação dos elementos da árvore, é contornado com as rotações tanto para a esquerda como para a direita, simples ou duplo, que veremos em seguida.

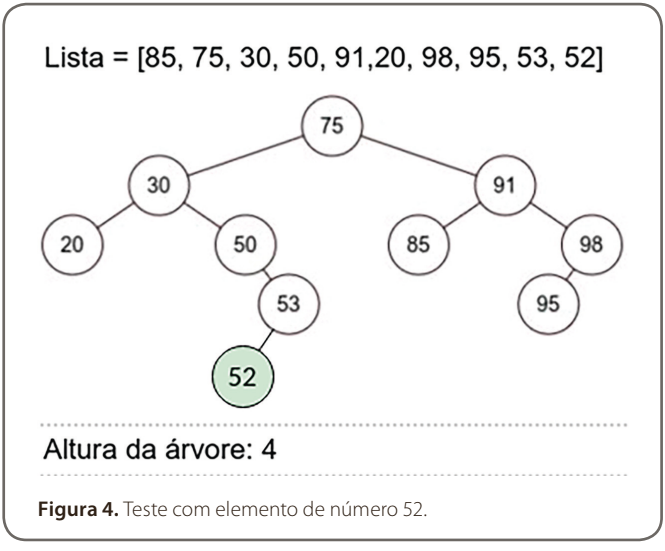

Figura 4. Teste com elemento de número 52.

Fique atento

Até aqui abordamos apenas o método de balanceamento por altura, mas não se deixe enganar, pois existem outras formas de se balancear uma árvore, como o balanceamento por peso — no qual são observados o número de nós nulos de cada subárvore — e o balanceamento de Tarjan, no qual, para todo nó nd, o comprimento do maior segmento a partir de nd até um nó externo (nó nulo) é, no máximo, o dobro do comprimento do menor segmento a partir de nd até um nó externo.

2 Rotação direita em árvores AVL

Em uma árvore que exige balanceamento, como mencionado anteriormente, cada nó necessita da informação de altura, portanto, é necessário adicionarmos esse atributo na classe, conforme ilustrado na Figura 5.

```
class No (object):
  def __ini__(self, chave):
    self.chave = chave
    self.esq = None
    self.dir = None
    self.altura = 1
```

Figura 5. Modificação da classe Nó para adição do balanceamento.

As rotações para a direita podem ser do tipo simples, quando é necessário apenas uma rotação para o lado em desequilíbrio. Vamos acompanhar os tipos de movimentos para a esquerda, conforme a Figura 6.

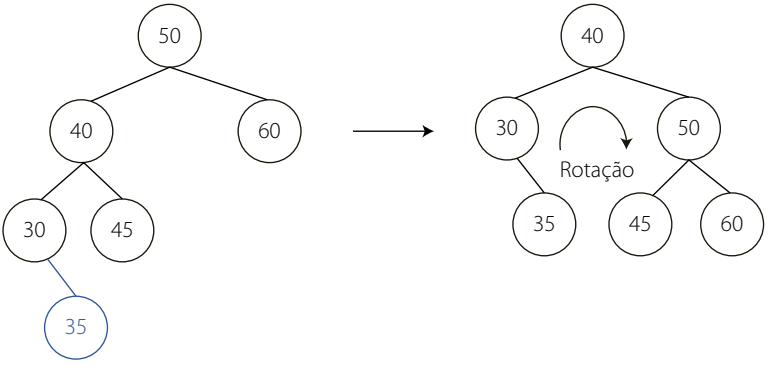

Figura 6. Rotação para a direita.

Fonte: Adaptada de Piva Junior *et al.* (2014).

A rotação à direita vem munida de algumas sequências que podem facilitar em seu entendimento:

1. o nó raiz da subárvore é deslocado para a posição de seu nó filho à direita;
2. o nó filho à direita, por sua vez, continua a ser o nó filho à direita do nó deslocado — (60) ainda é filho de (50);
3. o nó filho à direita do nó filho à esquerda da raiz — (45) é deslocado para ser o nó filho à esquerda do nó raiz deslocado — (45) passa a ser o filho à esquerda de (50);
4. por fim, o nó filho à esquerda do nó raiz ocupa seu antigo lugar, sendo a nova raiz.

A implementação dessa rotação segue a seguinte lógica (Figura 7).

```
def rotaciona_direita(self, no_z):
    #   T1, T2, T3 e T4 são subárvores.
    #         z                                       y
    #        / \                                    /   \
    #       y   T4      Rot. direita (z)          x      z
    #      / \          - - - - - - - - ->       /  \    /  \
    #     x   T3                                T1  T2  T3  T4
    #    / \
    # T1   T2
    no_y = no_z.esq
    T3 = no_y.dir

    # rotaciona
    no_y.dir = no_z
    no_z.esq = T3

    # atualiza as alturas
    no_z.altura = 1 + max(self.medir_altura(no_z.esq),
                          self.medir_altura(no_z.dir))

    no_y.altura = 1 + max(self.medir_altura(no_y.esq),
                          self.medir_altura(no_y.dir))
    return no_y
```

Figura 7. Implementação do método de rotação à direita.

Vejamos um exemplo de implementação da seguinte sequência com desequilíbrio para a esquerda: [14, 20, 9, 12, 5, 3]. Assim, ao aplicar a função de rotação para a direita, teremos o seguinte resultado, quando comparado ao resultado que seria obtido por meio de uma árvore binária sem balanceamento (Figura 8).

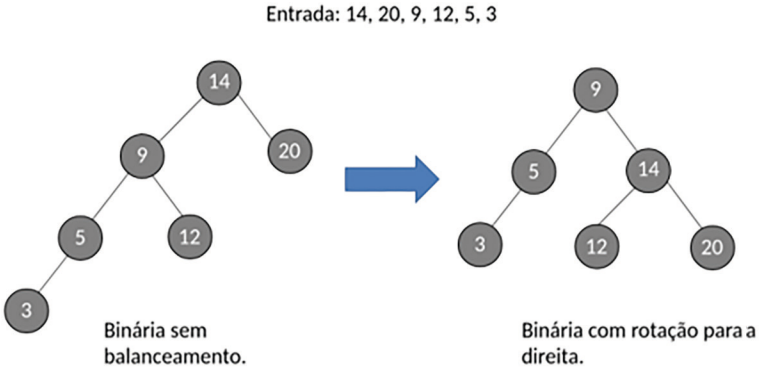

Figura 8. Comparação de uma árvore binária para uma rotação à direita.

A função de inserção já contempla o balanceamento, portanto, ao invés de primeiro criar a árvore para em seguida balanceá-la, é mais fácil procurar métodos que deixem a árvore balanceada, mesmo que não esteja perfeitamente balanceada, mantendo o objetivo em obter bons tempos de pesquisa próximos do tempo ótimo da árvore completamente balanceada, mas sem pagar muito para inserir ou retirar da árvore (ZIVIANI, c2011).

Saiba mais

Quando se menciona rotacionamento duplo, leva-se em consideração as seguintes possibilidades: uma rotação para a direita e outra para a esquerda; ou uma rotação para a direita e outra para a esquerda. Tais rotacionamentos são realizados sequencialmente.

No *link* a seguir, você pode acompanhar o processo de balanceamento da árvore de forma dinâmica.

https://qrgo.page.link/FDq2P

3 Rotação esquerda em árvores AVL

A rotação esquerda é realizada por meio dos seguintes passos, tendo como orientação a Figura 9:

1. o nó raiz (50) da subárvore é deslocado para a posição de seu nó filho à esquerda;
2. o nó filho à esquerda continua a ser o nó filho à esquerda do nó deslocado — (40) ainda é filho de (50);
3. o nó filho à esquerda do nó filho à direita da raiz é deslocado para ser o nó filho à direita do nó raiz deslocado — (55) passa a ser filho à direita do nó (50);
4. o nó filho à direita do nó raiz ocupa o seu antigo lugar, passando a ser a nova raiz — (60) passa a ser a nova raiz da árvore.

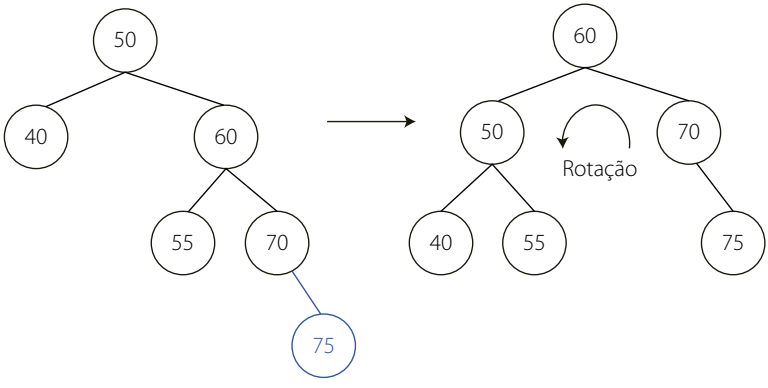

Figura 9. Rotação para a esquerda.

Fonte: Adaptada de Piva Junior *et al.* (2014).

Sua implementação é realizada da seguinte forma (Figura 10).

```
def rotaciona_direita(self, no_z):
    #   T1, T2, T3 e T4 são subárvores.
    #         z                                       y
    #        / \                                    /   \
    #       y   T4      Rot. direita (z)          x      z
    #      / \          - - - - - - - - ->       /  \    /  \
    #     x   T3                                T1  T2  T3  T4
    #    / \
    # T1   T2
    no_y = no_z.esq
    T3 = no_y.dir

    # rotaciona
    no_y.dir = no_z
    no_z.esq = T3

    # atualiza as alturas
    no_z.altura = 1 + max(self.medir_altura(no_z.esq),
                          self.medir_altura(no_z.dir))

    no_y.altura = 1 + max(self.medir_altura(no_y.esq),
                          self.medir_altura(no_y.dir))
    return no_y
```

Figura 10. Função de rotação para a esquerda.

Os códigos seguidos de testes podem ser acompanhados na Figura 11, na qual são apresentadas as funções inserir e remover, além das funções auxiliares de medida de balanço das subárvores, medição de altura e de valor mínimo, utilizadas na função de remover, que buscarão o menor valor pela ramificação à esquerda, partindo de uma ramificação à direita. Não foram apresentadas as funções de rotação na Figura 11, por elas já terem sido apresentadas nas Figuras 7 e 10.

Figura 11. Função de rotação para a esquerda.

As operações de rotação em árvores AVL são responsáveis por manter o balanceamento na árvore, no entanto, em aplicações que necessitam realizar frequentemente operações de inserção ou remoção, não se indica utilizar árvores AVL, pois o processo será custoso devido às diversas operações de rotação, que podem ser necessárias, sendo que as árvores AVL são indicadas apenas para aplicações em que a operação de busca seja mais frequente.

Referências

PIVA JUNIOR, D. *et al. Estrutura de dados e técnicas de programação*. Rio de Janeiro: Elsevier, 2014.

TENENBAUM, A. M.; LANGSAM, Y.; AUGENSTEIN, M. J. *Estruturas de dados em C*. São Paulo: Makron Books, 1995.

ZIVIANI, N. *Projeto de algoritmos*: com implementações em PASCAL e C. 3. ed. São Paulo: Cengage Learning, c2011.

Fique atento

Os *links* para *sites* da *web* fornecidos neste capítulo foram todos testados, e seu funcionamento foi comprovado no momento da publicação do material. No entanto, a rede é extremamente dinâmica; suas páginas estão constantemente mudando de local e conteúdo. Assim, os editores declaram não ter qualquer responsabilidade sobre qualidade, precisão ou integralidade das informações referidas em tais *links*.

Encerra aqui o trecho do livro disponibilizado para esta Unidade de Aprendizagem. Na Biblioteca Virtual da Instituição, você encontra a obra na íntegra.

Conteúdo:

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS