INSTITUTO FEDERAL DO MARANHÃO
SISTEMAS DE INFORMAÇÃO
ESTRUTURA DE DADOS II

Autores: José Matheus Aranha Maranhão e Gabriel Vinicius Pinto Portela.

ÁRVORE AVL

Atividade avaliativa apresentada à disciplina de Algoritmo e Estrutura de Dados II..
Prof. Helder Pereira Borges.

São Luís

2022

Sumário

1. Objetivo

Este documento tem como objetivo explicar o funcionamento da árvore AVL, para que possa ser possível o entendimento do algoritmo, vamos abordar diversos assuntos, sendo estes: altura de uma árvore binária, balanceamento, fator de balanceamento, rotação de nós simples e duplos e inserção e remoção em uma árvore AVL.

2. Introdução

Árvores binárias de busca foram projetadas para que o acesso à informação seja rápido, mas para que isso aconteça a árvore deve estar balanceada, ou seja as subárvores esquerda e direita precisam ter aproximadamente a mesma altura. Porém não é possível garantir que uma árvore binária esteja balanceada no decorrer do processo de execução do programa.

Tendo em vista esse problema, os soviéticos Adelson, Velsky e Landis desenvolveram o algoritmo da árvore AVL em 1962, trata-se de uma árvore binária de busca balanceada, a proposta desse algoritmo é manter a cada operação de inserção e remoção as propriedades de uma árvore balanceada.

3. Altura de uma árvore binária

A altura de uma árvore binária é obtida pelo caminho partindo da raiz até o nó folha mais distante, ou seja, se tivermos uma árvore com 3 nós, sendo uma raiz, um filho a direita e um filho a esquerda, podemos dizer que essa árvore possui altura 01, pois o custo do deslocamento tanto para esquerda ou direita é de uma unidade, o exemplo pode ser visto na figura 01.

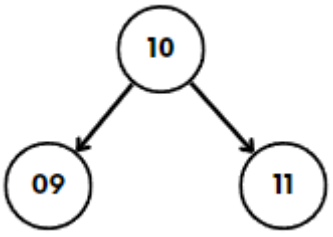

Figura 01. Altura de uma árvore binária.

Código em java para calcular altura:

```
public int calcularAltura(No noAtual) {
        if (noAtual == null) {
            return -1;
        }
        if (noAtual.getEsquerda() == null && noAtual.getDireita() ==
null) {
            return 0;
        } else if (noAtual.getEsquerda() != null &&
noAtual.getDireita() == null) {
            return 1 + calcularAltura(noAtual.getEsquerda());
        } else if (noAtual.getEsquerda() == null &&
noAtual.getDireita() != null) {
            return 1 + calcularAltura(noAtual.getDireita());
        } else {
            return 1 + Math.max(calcularAltura(noAtual.getEsquerda()),
calcularAltura(noAtual.getDireita()));
        }
    }
```

4. Balanceamento

A eficiência de uma árvore depende do seu balanceamento, para que uma árvore seja considerada balanceada, temos como propriedade que as alturas das sub-árvores à esquerda e direita de cada nó precisam diferir no máximo uma unidade, ou seja, a diferença da altura do nó à direita e esquerda precisa ser -1, 0 ou 1.

Podemos afirmar que uma árvore é AVL se e somente se ela respeitar a propriedade de balanceamento. Para manter essa propriedade o algoritmo da AVL propõe a utilização do fator de balanceamento do nó.

Verifique exemplos de árvores balanceadas (a e b) e não balanceadas (c) na figura 02.

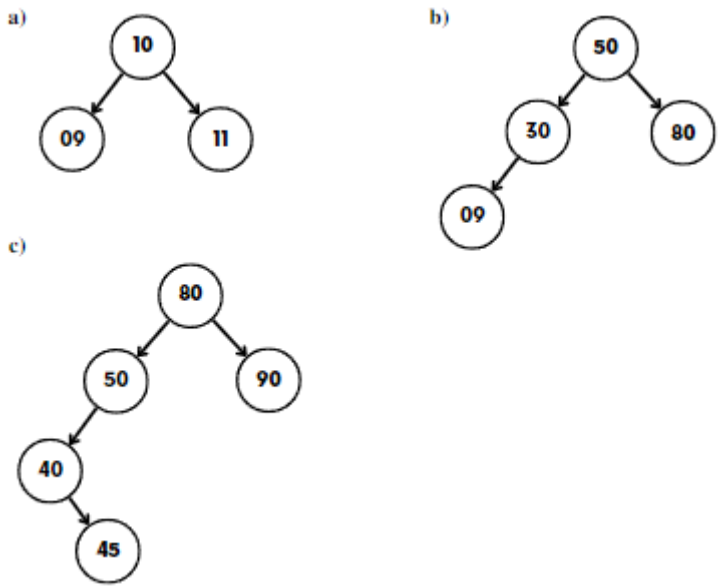

Figura 02. Árvores balanceadas e não balanceadas.

5. Fator de balanceamento

Cada nó da árvore AVL possui uma variável de valor inteiro responsável por guardar o fator de balanceamento desse nó, para definir esse valor é necessário obter a diferença da altura da subárvore à direita e altura da subárvore da esquerda, a cada operação de inserção ou remoção os fatores de balanceamento são recalculados, caso o fator de balanceamento do nó não respeite a propriedade do balanceamento, este nó precisa ser balanceado, para isso utilizamos as operações de rotações. O fator de balanceamento é de suma importância para o funcionamento da árvore AVL, pois é ele que define se a árvore está balanceada ou não, para que uma árvore seja considerada AVL o fator de balanceamento do nó nunca pode ser menor que -1 ou maior que 1.

Vejamos abaixo alguns exemplos do funcionamento do fator de balanceamento:

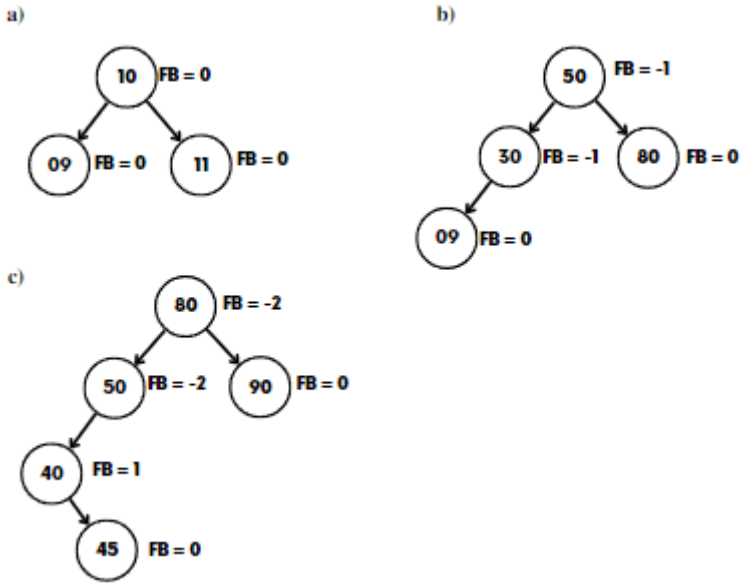

Figura 03. Fator de balanceamento: (a) balanceada; (b) balanceada; (c) desbalanceada

Após analisar as 03 árvores da figura 03, podemos afirmar que apenas as árvores (a) e (b) são AVL, pois o fator de balanceamento de todos os nós da árvore respeitam a regra do balanceamento.

A árvore (c) não é AVL, a mesma não está balanceada pois o fator de balanceamento do nó 80 e 50 não respeitam a regra.

Regra do balanceamento:

```
FB : Fator de Balanceamento
FB >= -1 e FB <=1
```

Fórmula para calcular o fator de balanceamento:

```
FB : Fator de Balanceamento
HD:  Altura a direita do nó atual
HE : Altura a esquerda do nó atual

FB = HD - HE
```

Estrutura do Nó:

```
public class No {
    int elemento;
    No pai;
    No esquerda;
    No direita;
```

```
    int fatorB; // Fator de balanceamento
}
```

Código em java do cálculo do fator de balanceamento:

```
public void calcularFatorBalanceamento(No noAtual) {

        noAtual.setFatorB(calcularAltura(noAtual.getDireita()) -
calcularAltura(noAtual.getEsquerda()));

    }
```

6. Rotações

6.1. Introdução a rotações

O objetivo da rotação é corrigir o fator de balanceamento de cada nó, como visto anteriormente, a cada operação de inserção e remoção na árvore os fatores de balanceamento dos nós são recalculados, quando o fator de balanceamento não respeitar a propriedade do balanceamento da AVL, precisamos executar uma operação que faça com que os nós voltem a ter seus fatores de balanceamentos dentro da regra, para isso utilizamos 04 tipos de métodos, sendo estes:

- Rotação simples à esquerda
- Rotação simples à direita
- Rotação esquerda direita
- Rotação direita esquerda

Cada método de rotação atua de forma diferente na árvore, o fator de balanceamento dos nós informam qual o tipo de rotação deve ser utilizado, para isso vamos verificar cada rotação.

6.2. Rotação simples à esquerda

Uma rotação simples à esquerda ocorre quando o nó raiz e seu filho estão inclinados à direita, ambos possuem o mesmo sentido, formando uma reta. Vejamos esse caso no item (a) da figura 04.

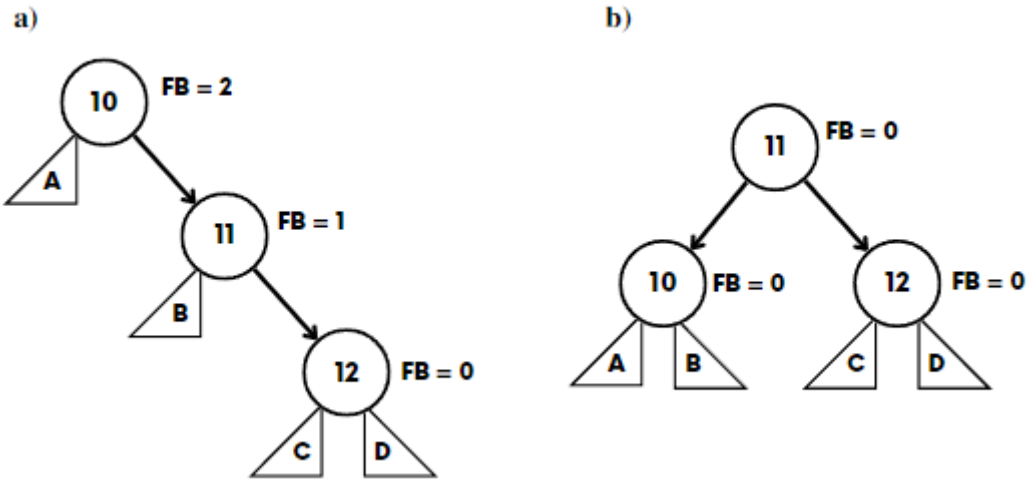

Figura 04. Rotação simples à esquerda: (a) desbalanceado; (b) balanceado.

Conforme o item (a) da figura 04 , o nó 10 possui fator de balanceamento +2 e seu filho possui fator de balanceamento +1, ou seja, possuímos 02 nós a mais à direita.

Para resolvermos isso, temos que rotacionar o nó desbalanceado a esquerda, para isso o filho à direita do nó desbalanceado (11) é escolhido para ser a nova raiz da árvore, a antiga raiz (10) recebe em sua direita os nós que ficavam à esquerda da nova raiz (11) e o (10) é posicionado à esquerda na nova raiz (11), conforme o item (b) da figura 04. Após recalcular os fatores de balanceamento, podemos perceber que o fator de balanceamento de cada nó está dentro da regra.

Código em java da rotação simples à esquerda:

```
public No rotacaoSimplesEsquerda(No noAtual) {

        No novoPai = noAtual.getDireita(); // O novo pai é o no
intermediario

        noAtual.setDireita(novoPai.getEsquerda()); // A direita do
noAtual recebe os valores da esquerda do novoPai
        novoPai.setPai(noAtual.getPai());

        novoPai.setEsquerda(noAtual); // O novoPai agora é a raiz da
subArvore

        if (noAtual.getPai() != null) {
            if (novoPai.getPai().getDireita() == noAtual) {
                novoPai.getPai().setDireita(novoPai);
            } else if (novoPai.getPai().getEsquerda() == noAtual) {
                novoPai.getPai().setEsquerda(novoPai);
            }
```

```
        }

        if (noAtual.getPai() == null) { // Se o no atual foi a raiz
pricipal, devemos informar que o novoPai é a nova
                                        // raiz, caso não seja o novo
pai é filho do pai anterior ao nó atual
            this.raiz = novoPai;
            novoPai.setPai(null);
        }

        noAtual.setPai(novoPai);
        calcularFatorBalanceamento(noAtual);
        calcularFatorBalanceamento(novoPai);

        return novoPai;
    }
```

6.3. Rotação simples à direita

Uma rotação simples à direita ocorre quando o nó raiz e seu filho estão inclinados à esquerda, ambos possuem o mesmo sentido, formando uma reta. Vejamos esse caso no item (a) da figura 05.

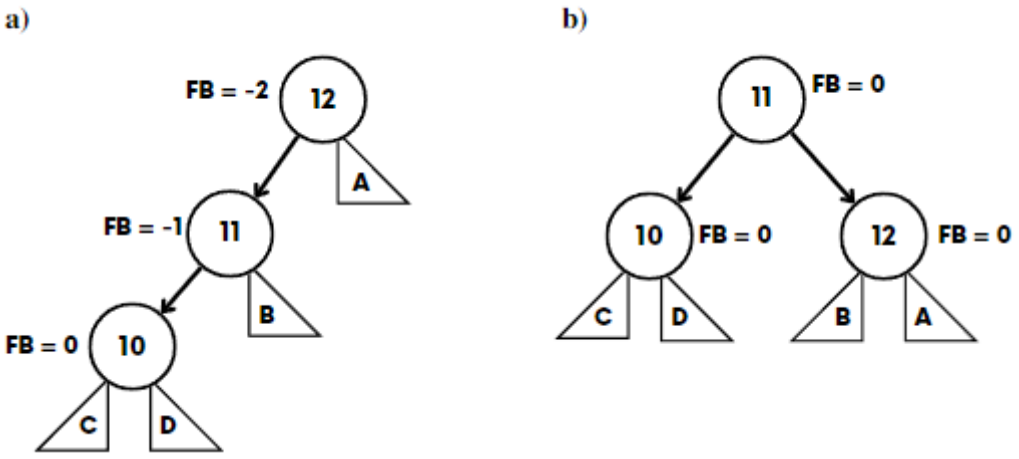

Figura 05. Rotação simples à direita: (a) desbalanceado; (b) balanceado

Conforme o item (a) da figura 05 , o nó 12 possui fator de balanceamento -2 e seu filho possui fator de balanceamento -1, ou seja, possuímos 02 nós a mais na esquerda.

Para resolvermos isso, temos que rotacionar o nó desbalanceado a direita, para isso o filho à esquerda do nó desbalanceado (11) é escolhido para ser a nova raiz da árvore, a antiga raiz (12) recebe a esquerda os nós que ficavam à direita da nova raiz (11) e o (12) é posicionada à direita da nova raiz (11), conforme o item (b) da figura 05. Após recalcular os fatores de balanceamento, podemos perceber que o fator de balanceamento de cada nó está dentro da regra.

Código em java da rotação simples à direita:

```
public No rotacaoSimplesDireita(No noAtual) {

        No novoPai = noAtual.getEsquerda(); // O novo pai é o no
intermediario

        noAtual.setEsquerda(novoPai.getDireita()); // A direita do
noAtual recebe os valores da esquerda do novoPai
        novoPai.setPai(noAtual.getPai());

        novoPai.setDireita(noAtual); // O novoPai agora é a raiz da
subArvore

        if (noAtual.getPai() != null) {
            if (novoPai.getPai().getDireita() == noAtual) {
                novoPai.getPai().setDireita(novoPai);
            } else if (novoPai.getPai().getEsquerda() == noAtual) {
                novoPai.getPai().setEsquerda(novoPai);
            }
        }

        if (noAtual.getPai() == null) { // Se o nó atual foi a raiz
pricipal, devemos informar que o novoPai é a nova
            this.raiz = novoPai;        // raiz, caso não seja o novo
pai é filho do pai anterior ao nó atual
            novoPai.setPai(null);
        }

        noAtual.setPai(novoPai);
        calcularFatorBalanceamento(noAtual);
        calcularFatorBalanceamento(novoPai);

        return novoPai;
    }
```

6.4. Rotação esquerda direita

Uma rotação esquerda direita ocorre quando um nó estiver desbalanceado à esquerda e seu filho à esquerda tem sentido contrário ao pai, formando um joelho.Vejamos esse caso no item (a) da figura 06.

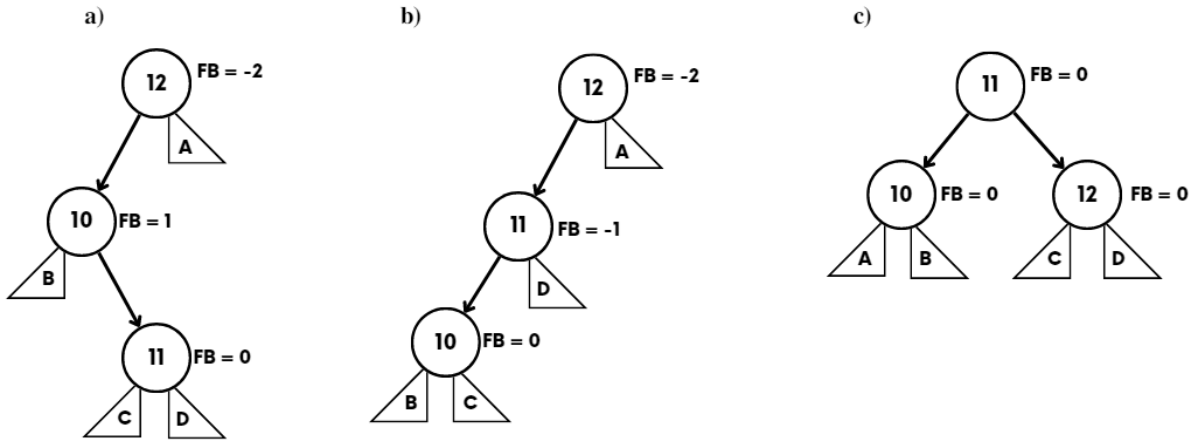

Figura 06. Rotação esquerda direita: (a) desbalanceada; (b) desbalanceada; (c) balanceada.

Conforme o item (a) da figura 06 , o nó 12 possui fator de balanceamento -2 e seu filho a esquerda possui fator de balanceamento 1, perceba que o sentido está invertido, para isso devemos forçar uma rotação a esquerda no filho do nó desbalanceado (10), com isso o sentido do nó (11) é o mesmo de seu pai visto no item (b), logo podemos aplicar o método de rotação a direita, com isso a árvore está balanceada, conforme o item (c).

Código em java da rotação esquerda direita:

```
public void rotacaoEsquerdaDireita(No noAtual) {
        No noFilho = noAtual.getEsquerda();
        rotacaoSimplesEsquerda(noFilho);
        rotacaoSimplesDireita(noAtual);
    }
```

6.5. Rotação direita esquerda

Uma rotação direita esquerda ocorre quando um nó estiver desbalanceado à direita e seu filho à direita tem sentido contrário ao pai, formando um joelho.Vejamos esse caso no item (a) da figura 07.

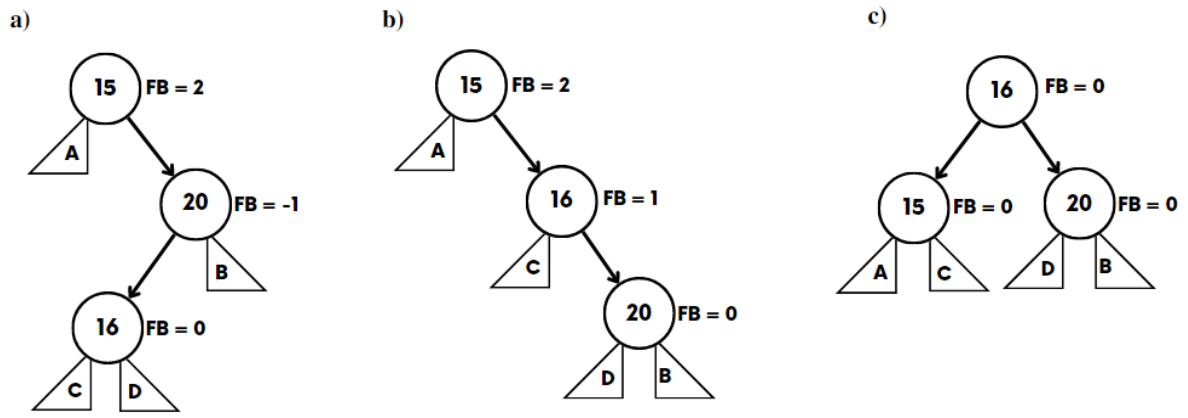

Figura 07. Rotação direita esquerda. (a) desbalanceada; (b) desbalanceada; (c) balanceada.

Conforme o item (a) da figura 07 , o nó 15 possui fator de balanceamento 2 e seu filho a direita possui fator de balanceamento -1, perceba que o sentido está invertido, para isso devemos forçar uma rotação à direita do filho do nó desbalanceado (20), com isso o sentido do nó (16) é o mesmo de seu pai visto no item (b), logo podemos aplicar o método de rotação a esquerda, com isso a árvore está balanceada, conforme o item (c).

Código em java da rotação direita esquerda:

```
public void rotacaoDireitaEsquerda(No noAtual) {
        No noFilho = noAtual.getDireita();
        rotacaoSimplesDireita(noFilho);
        rotacaoSimplesEsquerda(noAtual);
    }
```

7. Inserção na AVL

A inserção na árvore Avl ocorre de maneira parecida com a inserção da árvore binária, primeiramente olhamos se a árvore já possui algum nó, caso a árvore não possua nenhuma informação quer dizer que a árvore está vazia, sendo assim só inserimos a informação na árvore (Figura 08).

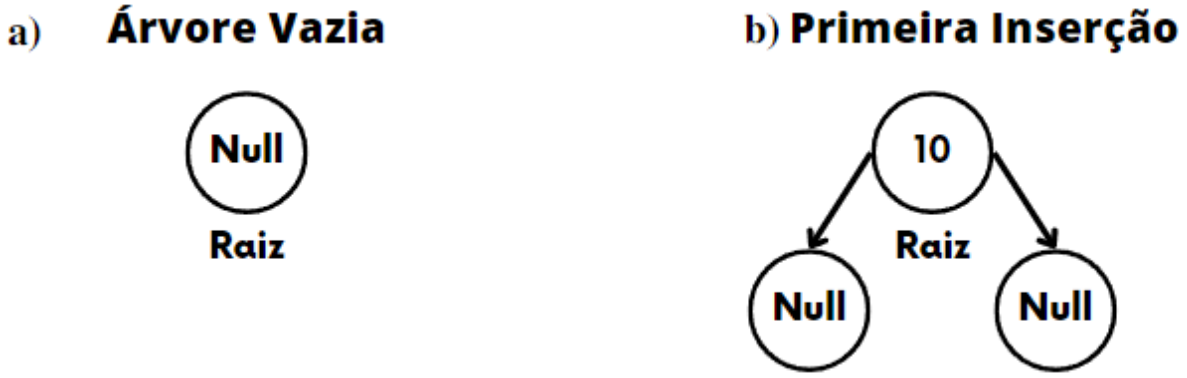

Figura 08. Inserção. (a) Vazia; (b) Primeira Inserção.

Caso a árvore já possua alguma informação, primeiro checamos se a informação que queremos inserir é maior ou menor que a informação que já está na árvore (Figura 09), caso seja menor inserimos para a esquerda (Figura 09), caso seja maior inserimos para a direita (Figura 09).

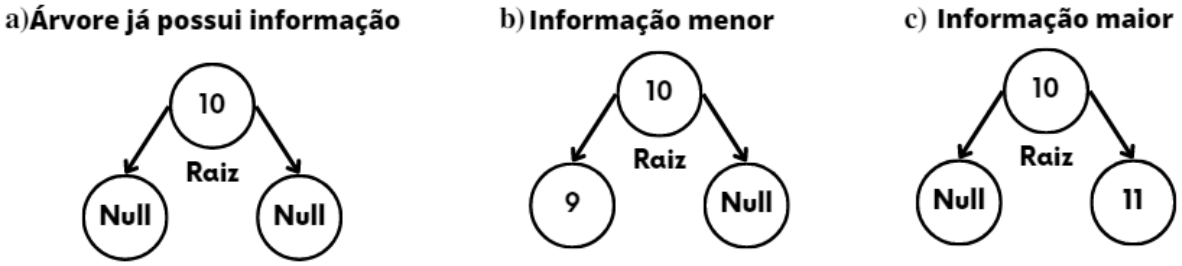

Figura 09. Inserção. (a) Já possui informação; (b) informação menor; (c) Informação maior.

Caso necessário, o algoritmo vai rebalancear a árvore a partir do pai do nó inserido(Figura 10).

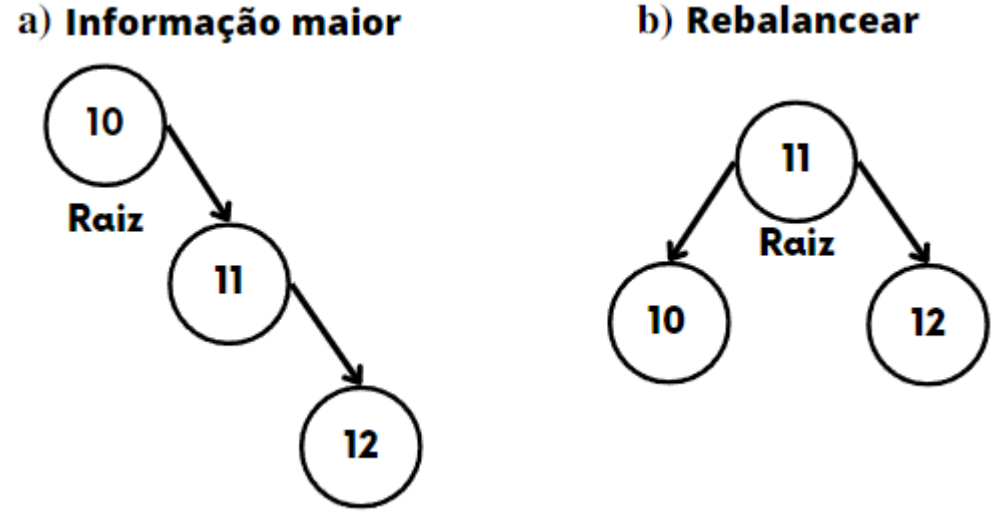

Figura 10. Inserção. (a) Informação maior; (b) Rebalancear.

Código Inserção em java:

```
public void inserirAvl(int elemento, No noAtual) {
        if (arvoreVazia()) {
            No novoNo = new No(elemento);
            this.raiz = novoNo;
            calcularFatorBalanceamento(this.raiz);
        } else {
            if (elemento >= noAtual.getElemento()) { // Inserir
elementos maiores que a raiz
                if (noAtual.getDireita() == null) {
                    No novoNo = new No(elemento);
                    noAtual.setDireita(novoNo);
                    novoNo.setPai(noAtual);
                    balancearArvore(noAtual);
                } else {
                    inserirAvl(elemento, noAtual.getDireita());
                }
            } else if (elemento <= noAtual.getElemento()) { // Inserir
elementos menores que a raiz
                if (noAtual.getEsquerda() == null) {
                    No novoNo = new No(elemento);
                    noAtual.setEsquerda(novoNo);
                    novoNo.setPai(noAtual);
                    balancearArvore(noAtual);
                } else {
                    inserirAvl(elemento, noAtual.getEsquerda());
                }
            }
        }

    }
```

8. Remoção na AVL

A remoção na árvore Avl ocorre da seguinte forma: primeiro procura-se onde a informação se encontra, se ela se encontra na raiz, na subárvore esquerda ou na subárvore direita (Figura 11).

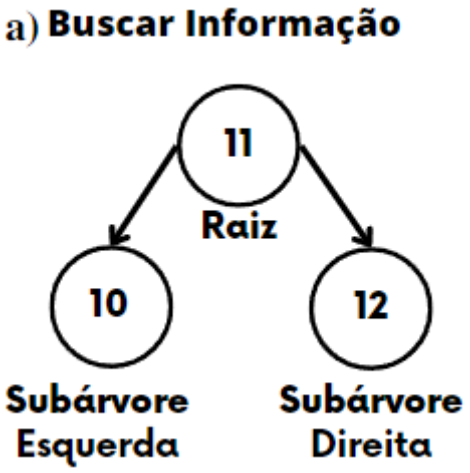

Figura 11. Buscar informação.

Vamos remover a raiz, para isso vamos passar a subárvore direita para a subárvore esquerda (Figura 12). Após isso endereçamos a raiz para a subárvore esquerda e removemos as informações da antiga raiz (Figura 12).

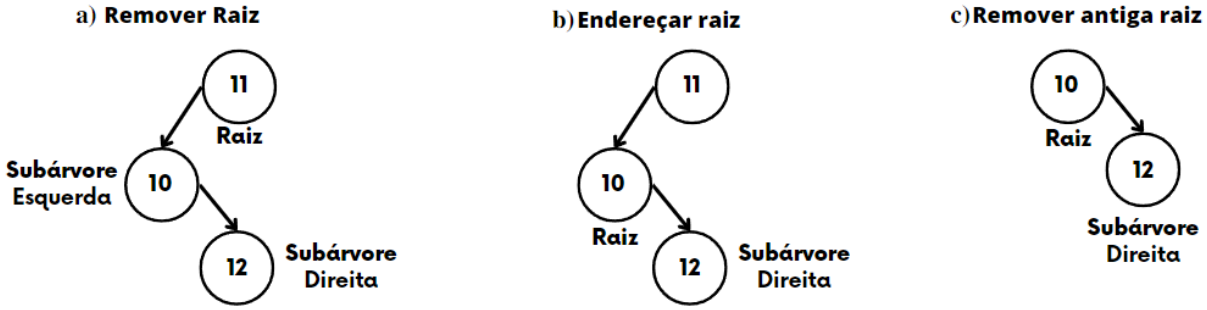

Figura 12. Remover Raiz.

Caso queiramos remover a subárvore direita só a anulamos(Figura 13).

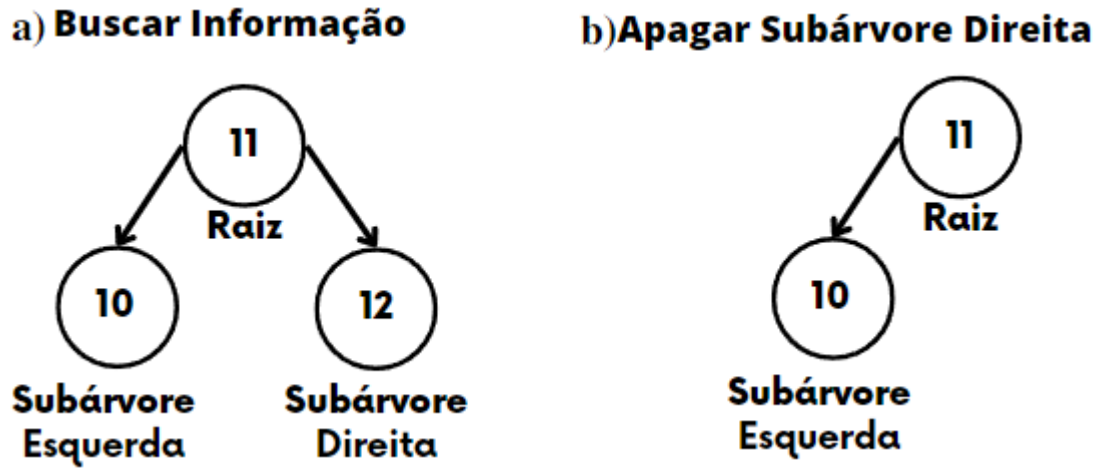

Figura 13. Remover a subárvore direita.

Caso queiramos remover a subárvore esquerda só a anulamos (Figura 14).

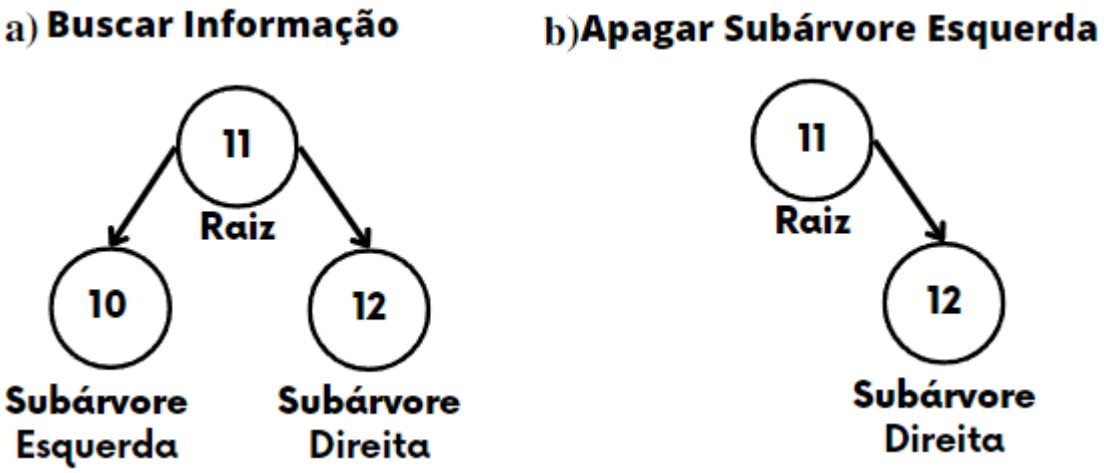

Figura 14. Remover a subárvore esquerda.

Caso necessário, o algoritmo vai rebalancear a árvore usando o pai como referência.

Código de remoção em java:

```
public void removerNo(No noAtual){
    No noAuxiliar;

    if (noAtual.getEsquerda() == null && noAtual.getDireita() == null)
{

        if (noAtual.getPai() == null) {
            this.raiz = null;
            noAtual = null;
            return;
        }
        noAuxiliar = noAtual;

    } else {
        noAuxiliar = procura_sucessor(noAtual);
        noAtual.setElemento(noAuxiliar.getElemento());
    }

    No controle;
    if (noAuxiliar.getEsquerda() != null) {
        controle = noAuxiliar.getEsquerda();
    } else {
        controle = noAuxiliar.getDireita();
    }
```

```
    if (controle != null) {
        controle.setPai(noAuxiliar.getPai());
    }

    if (noAuxiliar.getPai() == null) {
        this.raiz = controle;
    } else {
        if (noAuxiliar == noAuxiliar.getPai().getEsquerda()) {
            noAuxiliar.getPai().setEsquerda(controle);
        } else {
            noAuxiliar.getPai().setDireita(controle);
        }
        calcularFatorBalanceamento(noAuxiliar.getPai());
    }
    noAuxiliar = null;
}
```

Código para buscar um elemento em java:

```
public void buscarAvl(int elemento, No noAtual){
    if (arvoreVazia()) {
        System.out.println("Arvore Vazia!");
    }
    else if (noAtual == null) {
        return;
    }
    else{
        if (elemento > noAtual.getElemento()) {
            buscarAvl(elemento, noAtual.getDireita());
        }
        else if (elemento < noAtual.getElemento()) {
            buscarAvl(elemento, noAtual.getEsquerda());
        }
        else if (elemento == noAtual.getElemento()) {
            removerNo(noAtual);
        }
    }
}
```

Código para encontrar o sucessor para o nó que está sendo removido em java:

```
public No procura_sucessor(No noAtual) {
```

```
    if (noAtual.getEsquerda() != null) {
        No auxiliar = noAtual.getEsquerda();
        while (auxiliar.getDireita() != null) {
            auxiliar = auxiliar.getDireita();
        }
        return auxiliar;
    }
    return noAtual;
}
```

9. Por que utilizar a árvore AVL

Umas das principais funcionalidades da árvore Avl se dá pelo fato da busca ser rápida, em uma árvore binária as inserções e remoções são feitas sem que seja garantido as propriedades de uma árvore balanceada, podendo gerar uma complexidade maior ao buscar um determinado elemento. Com a implementação do algoritmo da árvore AVL essa possibilidade é eliminada pois a árvore é balanceada após cada operação, ou seja, qualquer operação de busca na AVL tende a O(log n).

Por exemplo, se inserirmos as seguintes informações:10,70,20,60,50,40. Essas informações ficam organizadas das seguintes maneiras: na árvore binária e na árvore Avl(Figura 15).

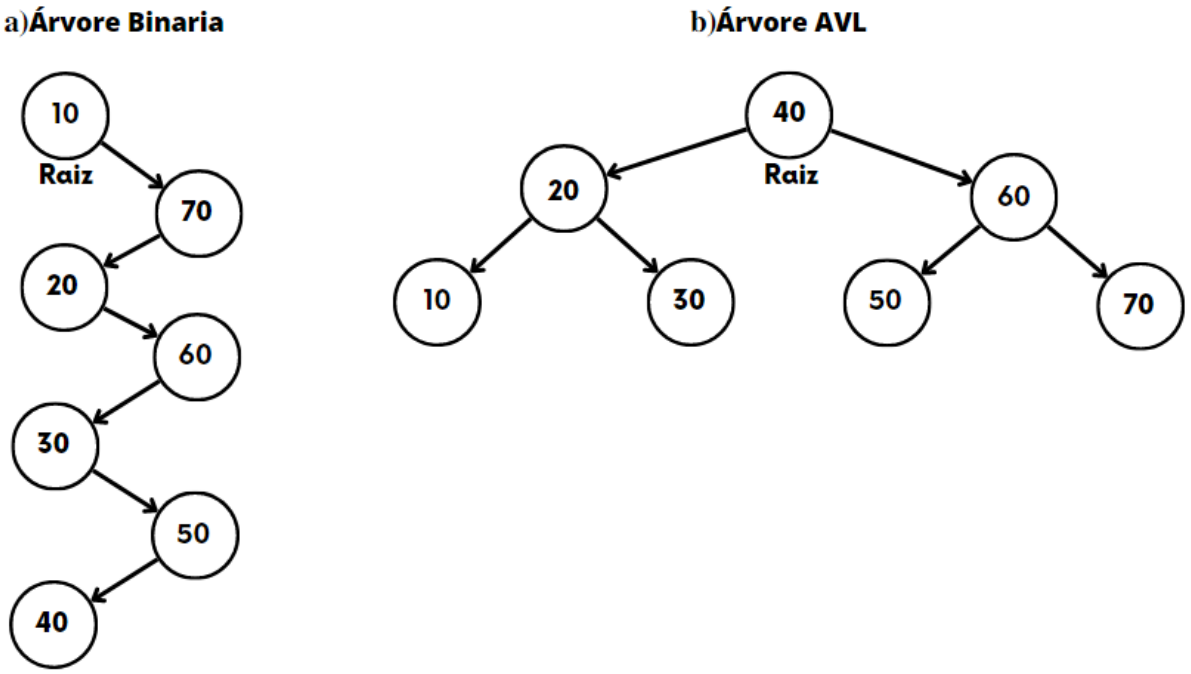

Figura 15. Inserção binária e Avl..

10. Referências

PROFESSOR DOUGLAS MAIOLI. Aula 19 Estrutura de Dados - Árvore AVL (ParteII). YouTube, 01/04/2021. Disponível em:<https://www.youtube.com/watch?v=Huj6CTaaBQI>. Acesso em: 19/11/2022.

GASPAR, Wagner. O que é uma Árvore AVL – Árvore Binária de Busca Balanceada?. Wagner Gaspar, 2021. Disponível em:<https://wagnergaspar.com/o-que-e-uma-arvore-avl-arvore -binaria-de-busca-balanceada/> Acesso em: 19/11/2022.

FEOFILOFF, Paulo. Árvores binárias. IME USP, 2018. Disponível em: <https://www.ime.usp.br/~pf/ algoritmos/aulas/bint.html>. Acesso em: 19/11/2022.

Algoritmos e Estruturas de Dados/Árvores AVL. WIKILIVROS, 2011. Disponível em: <https://pt.wikibooks.org/wiki/Algoritmos_e_Estruturas_de_Dados/%C3%81rvores_AVL>. Acesso em: 19/11/2022.

ÁrvoreAVL. WIKIPEDIA, 2021. Disponível em: <https://pt.wikipedia.org/wiki/%C3%81rvore_AVL>. Acesso em: 20/11/2022.

SOUZA, Jairo. Árvores AVL. UFJF, 2009. Disponível em: <https://www.ufjf.br/jairo_souza/files/2009/12/5-Indexa%C3%A7%C3%A3o-Arvore-AVL.pdf>. Acesso em: 18/11/2022.

Roseli Ap.Francelin Romero. Árvores AVL. Wiki USP. Disponível em:<http://wiki.icmc.usp.br/images/f/f0/AVL.pdf>.Acesso em:16/11/2022

André Backes,Árvore Avl,FACOM.2019.Disponível em:<https://www.facom.ufu.br/~backes/gsi011/Aula11-ArvoreAVL.pdf>.Aceso: 17/11/2022.

PRADO, Simone Domingues. 2007. Apostila 05 - Árvores Balanceadas (AVL). FACOM. Disponívelem:<https://www.facom.ufu.br/~claudio/Cursos/Antigos/EDxxxx/Artigos/bcc-ed05.pdf>. Acesso em: 17/11/2022.