Resolução de Problemas Estruturados em Computação (Turma B)
Professor: Andrey Cabral Meira
Aluno: Brunno Tatsuo

RA 02:

Árvores binárias e árvores AVL são dois tipos de estruturas de dados na ciência da computação e na teoria de algoritmos. Elas desempenham um papel crucial na organização e busca eficiente de dados em inúmeras aplicações, como bancos de dados, sistemas de arquivos e algoritmos de ordenação.

Uma árvore binária é uma estrutura de dados hierárquica na qual cada nó possui, no máximo, dois filhos: um filho esquerdo e um filho direito. Essa simplicidade permite a criação de estruturas de árvores eficientes para buscas e operações de inserção e remoção de dados. No entanto, árvores binárias podem se tornar desequilibradas e degenerar em listas encadeadas, o que prejudica o desempenho.

Já as árvores AVL são uma forma especial de árvores binárias que mantêm um equilíbrio específico. Elas garantem que a diferença de altura entre as subárvores esquerda e direita de qualquer nó seja no máximo um, o que ajuda a manter o desempenho em operações de busca, inserção e remoção. As árvores AVL são chamadas assim em homenagem a seus inventores, Adelson-Velsky e Landis, e representam uma solução eficaz para evitar o problema de degeneração que pode ocorrer em árvores binárias simples.

Neste relatório se demonstrara a implantação de ambas as Árvores em java, a comparação entre ambas nos métodos de busca e remoção, o desempenho de inserção com 100, 500, 1.000, 10.000, 20.000.

1 Implementação:

1.1 MENU:

```
import java.util.Random;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Random rd = new Random();
        ArvoreAVL arvoreAVL = new ArvoreAVL();
        ArvoreBinaria arvoreBinaria = new ArvoreBinaria();

        while (true) {
            System.out.println("____MENU____");
            System.out.println("1 - Árvore Binária ");
            System.out.println("2 - Árvore AVL");
            System.out.println("3 - Teste Árvore Binária com X
Valores");
            System.out.println("4 - Teste Árvore AVL com X Valores");
            System.out.println("5 - Teste Árvore AVL  e Binária,
Remover ou Buscar valor dentro da árvore");
            System.out.println("0 - Sair");
            System.out.print("Digite a Opção: ");
            int arvore = sc.nextInt();
```

```
            switch (arvore) {
                case 1:
                    while (true) {
                        System.out.println("_____MENU_____");
                        System.out.println("1 - Inserir ");
                        System.out.println("2 - Deletar");
                        System.out.println("3 - Buscar");
                        System.out.println("0 - Sair");
                        System.out.print("Digite a Opção: ");
                        int bin = sc.nextInt();
                        switch (bin) {
                            case 1:
                                System.out.print("Digite o número para
inserir na Árvore");
                                int valorI = sc.nextInt();
                                arvoreBinaria.inserir(valorI);
                                System.out.println("Pré-ordem da
Árvore Binária");

arvoreBinaria.preOrdem(arvoreBinaria.getRaiz());
                                break;
                            case 2:
                                System.out.print("Digite o número para
deletar da Árvore");
                                int valorD = sc.nextInt();
                                arvoreBinaria.deletar(valorD);
                                System.out.println("Pré-ordem da
Árvore Binária");

arvoreBinaria.preOrdem(arvoreBinaria.getRaiz());
                                break;
                            case 3:
                                System.out.print("Digite o número para
buscar na Árvore");
                                int valorB = sc.nextInt();
                                arvoreBinaria.buscar(valorB);
                                System.out.println("Pré-ordem da
Árvore Binária");

arvoreBinaria.preOrdem(arvoreBinaria.getRaiz());
                                break;
                            case 0:
                                System.exit(0);
                            default:
                                System.out.println("Digite uma opção
válida!");
                        }
                    }
                case 2:
                    while (true) {
                        System.out.println("_____MENU_____");
                        System.out.println("1 - Inserir ");
                        System.out.println("2 - Deletar");
                        System.out.println("3 - Buscar");
                        System.out.println("0 - Sair");
                        System.out.print("Digite a Opção: ");
                        int avl = sc.nextInt();
                        switch (avl) {
                            case 1:
                                System.out.print("Digite o número para
inserir na Árvore");
```

```
                                int valorI = sc.nextInt();
                                arvoreAVL.inserir(valorI);
                                System.out.println("Pré-ordem da
Árvore AVL");

arvoreAVL.preOrdem(arvoreAVL.getRaiz());
                                break;
                            case 2:
                                System.out.print("Digite o número para
deletar da Árvore");
                                int valorD = sc.nextInt();
                                arvoreAVL.deletar(valorD);
                                System.out.println("Pré-ordem da
Árvore AVL");

arvoreAVL.preOrdem(arvoreAVL.getRaiz());
                                break;
                            case 3:
                                System.out.print("Digite o número para
buscar na Árvore");
                                int valorB = sc.nextInt();
                                arvoreAVL.buscar(valorB);
                                System.out.println("Pré-ordem da
Árvore Binária");

arvoreAVL.preOrdem(arvoreAVL.getRaiz());
                                break;
                            case 0:
                                System.exit(0);
                            default:
                                System.out.println("Digite uma opção
válida!");
                        }
                    }
                case 3:
                    System.out.print("Digite o número de inserções:
");
                    int inserB = sc.nextInt();
                    long startB = System.nanoTime();
                    for(int i = 0; i<inserB; i++) {
                        int valorI = rd.nextInt(1,1000);
                        arvoreBinaria.inserir(valorI);
                    }
                    arvoreBinaria.preOrdem(arvoreBinaria.getRaiz());
                    long endB = System.nanoTime();
                    long duracaoB = endB - startB;
                    System.out.println("\nTempo decorrido para criar
Arvore AVL com "+ inserB + " inserções: " + duracaoB + "
nanosegundos");
                    System.exit(0);
                case 4:
                    System.out.print("Digite o número de inserções:
");
                    int inserA = sc.nextInt();
                    long startA = System.nanoTime();
                    for(int i = 0; i<inserA; i++) {
                        int valorI = rd.nextInt(1,1000);
                        arvoreAVL.inserir(valorI);
                    }
                    arvoreAVL.preOrdem(arvoreAVL.getRaiz());
                    long endA = System.nanoTime();
```

```
                    long duracaoA = endA - startA;
                    System.out.println("\nTempo decorrido para criar
Arvore AVL com "+ inserA + " inserções: " + duracaoA + "
nanosegundos");
                    System.exit(0);
                case 5:
                    System.out.print("Digite o número de inserções:
");
                    int inser = sc.nextInt();
                    for(int i = 0; i<inser; i++) {
                        int valor = rd.nextInt(1,1000);
                        arvoreBinaria.inserir(valor);
                        arvoreAVL.inserir(valor);
                    }
                    System.out.println();
                    System.out.println("Pré-ordem da Árvore Binária");
                    arvoreBinaria.preOrdem(arvoreBinaria.getRaiz());
                    System.out.println();
                    System.out.println("Pré-ordem da Árvore AVL");
                    arvoreAVL.preOrdem(arvoreAVL.getRaiz());
                    while (true) {
                        System.out.println();
                        System.out.println("_____MENU_____");
                        System.out.println("1 - Deletar");
                        System.out.println("2 - Buscar");
                        System.out.println("0 - Sair");
                        System.out.print("Digite a Opção: ");
                        int op = sc.nextInt();
                        switch (op) {
                            case 1:
                                System.out.print("Digite o número para
deletar da Árvore: ");
                                int valorD = sc.nextInt();
                                long start1 = System.nanoTime();
                                arvoreBinaria.deletar(valorD);
                                long end1 = System.nanoTime();
                                long start2 = System.nanoTime();
                                arvoreAVL.deletar(valorD);
                                long end2 = System.nanoTime();
                                long duracao1 = end1 - start1;
                                long duracao2 = end2 - start2;
                                System.out.println();
                                System.out.println("Pré-ordem da
Árvore Binária");

arvoreBinaria.preOrdem(arvoreBinaria.getRaiz());
                                System.out.println();
                                System.out.println("Pré-ordem da
Árvore AVL");

arvoreAVL.preOrdem(arvoreAVL.getRaiz());
                                System.out.println();
                                System.out.println("Tempo decorrido
para deletar "+ valorD +" da Arvore Binária: " + duracao1 + "
nanosegundos");
                                System.out.println("Tempo decorrido
para deletar "+ valorD +" da Arvore AVL: " + duracao2 + "
nanosegundos");
                                break;
                            case 2:
                                System.out.print("Digite o número para
```

```
buscar da Árvore: ");
                                int valorB = sc.nextInt();
                                long start3 = System.nanoTime();
                                arvoreBinaria.buscar(valorB);
                                long end3 = System.nanoTime();
                                long start4 = System.nanoTime();
                                arvoreAVL.buscar(valorB);
                                long end4 = System.nanoTime();
                                long duracao3 = end3 - start3;
                                long duracao4 = end4 - start4;
                                System.out.println();
                                System.out.println("Pré-ordem da
Árvore Binária");

arvoreBinaria.preOrdem(arvoreBinaria.getRaiz());
                                System.out.println();
                                System.out.println("Pré-ordem da
Árvore AVL");

arvoreAVL.preOrdem(arvoreAVL.getRaiz());
                                System.out.println();
                                System.out.println("Tempo decorrido
para buscar "+ valorB +" da Arvore Binária: " + duracao3 + "
nanosegundos");
                                System.out.println("Tempo decorrido
para buscar "+ valorB +" da Arvore AVL: " + duracao4 + "
nanosegundos");
                                break;
                            case 0:
                                System.exit(0);
                            default:
                                System.out.println("Digite uma opção
válida!");

                        }
                    }
                case 0:
                    System.exit(0);
                default:
                    System.out.println("Digite uma opção válida!");
            }
        }
    }
}
```

Criado para realizar a interação do programa com o usuário.

1.2 NO:

```
public class Node {
    private Integer info;
    private Node filhoEsq;
    private Node filhoDir;
    private int altura = 1;

    public Node(Integer info){
        this.info = info;
    }

    public Integer getInfo() {
        return info;
```

```
    }

    public void setInfo(Integer info) {
        this.info = info;
    }

    public Node getFilhoEsq() {
        return filhoEsq;
    }

    public void setFilhoEsq(Node filhoEsq) {
        this.filhoEsq = filhoEsq;
    }

    public Node getFilhoDir() {
        return filhoDir;
    }

    public void setFilhoDir(Node filhoDir) {
        this.filhoDir = filhoDir;
    }

    public int getAltura() {
        if (this == null){
            return 0;
        }
        return altura;
    }

    public void setAltura(int altura) {
        this.altura = altura;
    }
}
```

Classe Node, representa a classe de um nó da árvore, tendo como atributo um Intenger nomeado “info”, onde se encontra as informações do nó. Dois Node nomeados “filhoEsq” e “filhoDir”, que tem como função apontar para os próximos nó a partir do nó atual. Um int nomeado “altura” para identificar a altura do nó na arvore em que ele esta inserido. Os métodos get e set são criados de forma padrão, contudo no método “getAltura”, há um adendo de que se o nó que ele estiver lendo for nulo ele retornara 0.

1.3 BINÁRIA:

```
public class ArvoreBinaria {
    private Node raiz;

    public ArvoreBinaria() {
        this.raiz=null;
    }

    public Node getRaiz() {
        return raiz;
    }

    private boolean vazia() {
        return raiz == null; // se a raiz estiver vazia retorna
```

```
verdadeiro.
    }
```

Classe ArvoreBinaria, representa a classe de uma árvore binária. Tendo como atributo um Node nomeado raiz. Tem um construtor que inicia arvore com uma raiz nula. Tem um getRaiz como método de obter a raiz e o método vazia para verificar se a árvore está vazia.

```
public void inserir(Integer info) {
    if (vazia()) {
        raiz = new Node(info);
    } else {
        inserir(info, raiz);
    }
}
private void inserir(Integer info,Node noAtual) {
    if (info < noAtual.getInfo()) {
        if (noAtual.getFilhoEsq() == null) { // se o no atual não
possuir um filho a esquerda, um novo no sera adicionado
            noAtual.setFilhoEsq(new Node(info));
        } else {
            inserir(info, noAtual.getFilhoEsq()); // chamada recursiva
para percorrer o caminho até encontrar a posição correta
        }
    }else if (info >= noAtual.getInfo()){
        if (noAtual.getFilhoDir() == null) { // se o no atual não
possuir um filho a direira, um novo no sera adicionado
            noAtual.setFilhoDir(new Node(info));
        } else {
            inserir(info,noAtual.getFilhoDir());
        }
    }
}
```

O método público "inserir(Integer info)" é responsável por adicionar um valor à árvore binária. Ele começa verificando se a árvore está vazia, criando um novo nó como raiz se for o caso. Se a árvore já possui uma raiz, a inserção é realizada de forma recursiva pelo método privado "inserir(Integer info, Node noAtual)". Esse método privado segue o princípio de uma árvore binária, direcionando a inserção para a esquerda se o valor a ser inserido for menor ou igual ao valor do nó atual, e para a direita caso contrário. Essa abordagem garante que os valores sejam colocados em posições apropriadas na árvore, mantendo a estrutura binária.

```
public void deletar(Integer info) {
    raiz = deletar(info, raiz);
}
private Node deletar(Integer info, Node noAtual) {
    if (noAtual == null) {
        System.out.println("Valor não encontrado");
        return  null;
    }
    if (info < noAtual.getInfo()) {
        noAtual.setFilhoEsq(deletar(info,noAtual.getFilhoEsq()));
    } else if (info > noAtual.getInfo()) {
        noAtual.setFilhoDir(deletar(info,noAtual.getFilhoDir()));
    } else {
        if (noAtual.getFilhoEsq() == null) {
            return noAtual.getFilhoDir();
        } else if (noAtual.getFilhoDir() == null) {
```

```
            return noAtual.getFilhoEsq();
        }
        noAtual.setInfo(maximo(noAtual.getFilhoEsq())); // Quando um
nó tem dois filhos, precisamos escolher o predescessor ou o sucessor,
neste caso estamos escolhendo o predescessor.

noAtual.setFilhoEsq(deletar(noAtual.getInfo(),noAtual.getFilhoEsq()));
    }
    return noAtual;
}
```

O método público "deletar(Integer info)" permite a remoção de um valor da árvore binária, e ele invoca o método privado "deletar(info, raiz)" para realizar a operação. O método privado "deletar(Integer info, Node noAtual)" é responsável por gerenciar a exclusão do valor especificado a partir do nó atual da árvore. Ele lida de forma eficiente com diversas situações, incluindo casos em que o nó a ser excluído não possui filhos (zero filhos), tem um único filho (um filho), ou possui dois filhos. Isso garante a manutenção da estrutura da árvore binária após a remoção do valor desejado.

```
public Integer maximo(Node noAtual) {
    if(noAtual.getFilhoDir() != null){
        return maximo(noAtual.getFilhoDir());
    }
    return noAtual.getInfo();
}
```

O método público “máximo” permite encontrar o maior valor da subárvore a partir do nó “noAtual”, servindo de auxílio para a remoção de um nó com dois filhos.

```
public void preOrdem(Node noAtual) {
    if(noAtual != null) {
        System.out.print(noAtual.getInfo() + " ");
        preOrdem(noAtual.getFilhoEsq());
        preOrdem(noAtual.getFilhoDir());
    }
}
```

O método público “preOrdem” realiza o print da árvore no console usando pré-ordem como referência. Começando na raiz da árvore e, em seguida, visita todos os nós à esquerda antes de seguir para os nós à direita.

```
public void buscar(Integer info) {
    if (vazia()) {
        System.out.println("Árvore Vazia");
    } else {
        buscar(info, raiz);
    }
}
private void buscar(Integer info, Node noAtual) {
    if( noAtual == null) {
        System.out.println("Valor não encontrado na Árvore");
        return;
    }
    if (info != noAtual.getInfo()) {
```

```
        if ( info < noAtual.getInfo()) {
            buscar(info,noAtual.getFilhoEsq());
        } else {
            buscar(info,noAtual.getFilhoDir());
        }
    }
    if(info == noAtual.getInfo()){
        System.out.println("Valor  encontrado na Árvore");
    }
}
```

O método público "buscar(Integer info)" permite a busca de um valor específico na árvore binária, e ele invoca o método privado "buscar(info, raiz)" para realizar a operação. O método privado "buscar(Integer info, Node noAtual)" é encarregado de executar a busca a partir do nó atual da árvore. Ele avalia se o valor de interesse é igual, menor ou maior em relação ao valor no nó atual, direcionando a busca para a esquerda ou direita da árvore, dependendo da relação encontrada. Dessa forma, a busca é conduzida de forma eficaz na estrutura da árvore, determinando se o valor procurado está presente ou ausente na árvore binária.

1.4 AVL:

A implementação da Árvore AVL é uma extensão da árvore binária que visa manter o equilíbrio da árvore, otimizando as operações de busca, inserção e exclusão.

```
public class ArvoreAVL {
    private Node raiz;

    public ArvoreAVL(){
        this.raiz = null;
    }

    public Node getRaiz() {
        return raiz;
    }

    private boolean vazia() {
        return raiz == null;
    }
```

Classe ArvoreAVL, representa a classe de uma árvore AVL. A mesma estrutura da Árvore Binária.

```
public void inserir(Integer info) {
        raiz = inserir(info, raiz);
}

private Node inserir(Integer info, Node noAtual) {
    if (noAtual == null) {
        return new Node(info);
    }
    if (info < noAtual.getInfo()) {
        noAtual.setFilhoEsq(inserir(info, noAtual.getFilhoEsq()));
    } else if (info >= noAtual.getInfo()) {
        noAtual.setFilhoDir(inserir(info, noAtual.getFilhoDir()));
```

```
    } else {
        return noAtual;
    }
    ajustarAltura(noAtual);
    return rotacao(noAtual);
}
```

Tem a mesma implementação da Árvore Binária, com um adendo, ao final de cada inserção de nó, chama-se o método “ajustarAltura” e “rotação” para manter o equilíbrio da árvore.

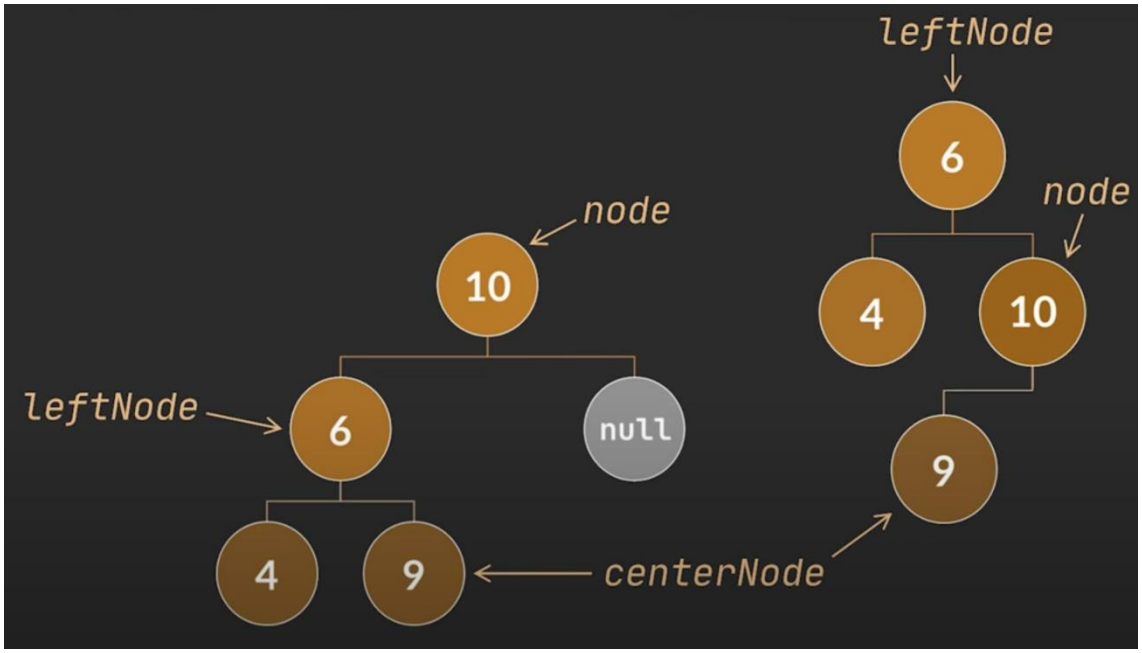
```
public void deletar(Integer info) {
    raiz = deletar(info, raiz);
}
private Node deletar(Integer info, Node noAtual) {
    if (noAtual == null) {
        System.out.println("Valor não encontrado");
        return  null;
    }
    if (info < noAtual.getInfo()) {
        noAtual.setFilhoEsq(deletar(info,noAtual.getFilhoEsq()));
    } else if (info > noAtual.getInfo()) {
        noAtual.setFilhoDir(deletar(info,noAtual.getFilhoDir()));
    } else {
        if (noAtual.getFilhoEsq() == null) {
            return noAtual.getFilhoDir();
        } else if (noAtual.getFilhoDir() == null) {
            return noAtual.getFilhoEsq();
        }
        noAtual.setInfo(maximo(noAtual.getFilhoEsq())); // Quando um
nó tem dois filhos, precisamos escolher o predescessor ou o sucessor,
neste caso estamos escolhendo o predescessor.

noAtual.setFilhoEsq(deletar(noAtual.getInfo(),noAtual.getFilhoEsq()));
    }
    ajustarAltura(noAtual);
    return rotacao(noAtual);
}
```

Assim como inserir, tem a mesma implementação da Árvore Binária, com um adendo, ao final de cada remoção de nó, chama-se o método “ajustarAltura” e “rotação” para manter o equilíbrio da árvore.

```
private void ajustarAltura(Node noAtual) {
    int altura;
    if ( altura(noAtual.getFilhoEsq()) >
altura(noAtual.getFilhoDir()) ) {
        altura = altura(noAtual.getFilhoEsq());
    } else {
        altura = altura(noAtual.getFilhoDir());
    }
    noAtual.setAltura(altura + 1);
}
private int altura(Node noAtual) {
    return noAtual != null ? noAtual.getAltura() : 0;
}
```

O método "ajustarAltura" é responsável por manter a altura dos nós da árvore AVL atualizada, garantindo que o cálculo do fator de equilíbrio seja preciso. Ele determina a altura de um nó considerando a altura das subárvores esquerda e direita, selecionando a maior delas e acrescentando um nível. O método "altura" calcula a altura de um nó individual, levando em conta os nós nulos, que têm altura igual a zero.

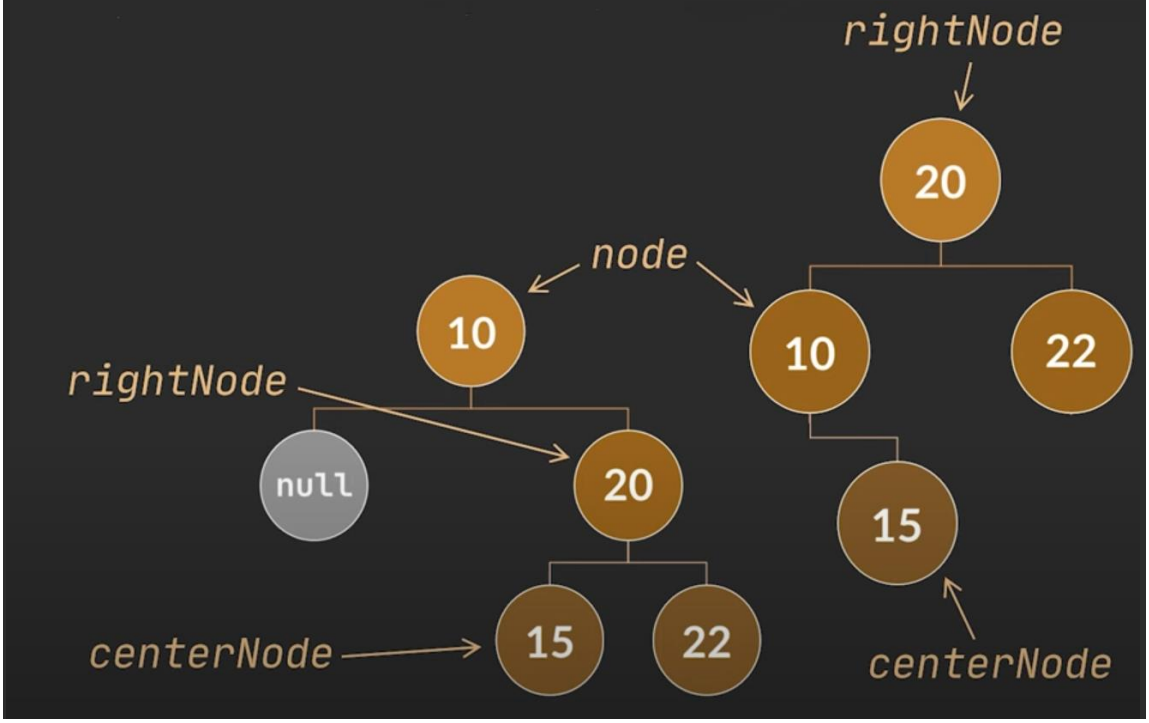
```
private Node rotacaoDir(Node noAtual) {
    Node noEsq = noAtual.getFilhoEsq();
    Node noCentral = noEsq.getFilhoDir();
    noEsq.setFilhoDir(noAtual);
    noAtual.setFilhoEsq(noCentral);
    ajustarAltura(noAtual);
    ajustarAltura(noEsq);
    return noEsq;
}
```



Primeiro, cria-se uma referência (noEsq) para o nó à esquerda do nó atual, que é onde ocorreu o desequilíbrio. Em seguida, obtém-se o nó à direita da subárvore esquerda (noCentral), que será usado para reconfigurar as conexões. A rotação envolve fazer com que o nó à esquerda (noEsq) se torne o pai do nó atual, substituindo o que era o filho direito do nó à esquerda. Simultaneamente, o nó atual passa a ter como filho esquerdo o nó à direita da subárvore esquerda (noCentral). Após a rotação, a altura dos nós afetados é ajustada para refletir as mudanças na estrutura da árvore. Finalmente, o método retorna o nó à esquerda (noEsq), que agora se tornou o novo nó raiz da subárvore, restabelecendo o equilíbrio na direção direita.

```
private Node rotacaoEsq(Node noAtual) {
    Node noDir = noAtual.getFilhoDir();
    Node noCentral = noDir.getFilhoEsq();
    noDir.setFilhoEsq(noAtual);
    noAtual.setFilhoDir(noCentral);
```

```
    ajustarAltura(noAtual);
    ajustarAltura(noDir);
    return noDir;
}
```



Assim como a rotação direita, faz se os mesmos passos, sendo que eles são simétricos, trocando a referência da esquerda para a direita.

```
private Node rotacao(Node noAtual) {
    int balanceamento = balanceamento(noAtual);
    if (balanceamento > 1) { // valor 2
        if (balanceamento(noAtual.getFilhoEsq()) < 0) {
            noAtual.setFilhoEsq(rotacaoEsq(noAtual.getFilhoEsq()));
        }
        return rotacaoDir(noAtual);
    }
    if (balanceamento < -1) { // valor -2
        if (balanceamento(noAtual.getFilhoDir()) > 0) {
            noAtual.setFilhoDir(rotacaoDir(noAtual.getFilhoDir()));
        }
        return rotacaoEsq(noAtual);
    }
    return noAtual;
}
```

O método “rotação” é responsável por manter o equilíbrio de uma Árvore AVL. Primeiro, ele calcula o fator de equilíbrio do nó atual, que é a diferença entre a altura da subárvore esquerda e da subárvore direita. Se o fator de equilíbrio for maior que 1, indica um desequilíbrio na direção esquerda. Dentro desse bloco, verifica se a subárvore esquerda do nó atual também está desequilibrada para a direita. Se sim, aplica uma rotação à esquerda nessa subárvore para corrigir o desequilíbrio. Em seguida, aplica uma rotação à direita no nó atual para ajustar o equilíbrio. Se o fator de equilíbrio for menor que -1, indica um desequilíbrio na direção direita. Dentro desse bloco, verifica se a subárvore direita do nó atual também está desequilibrada para a esquerda. Se sim, aplica uma rotação à direita nessa subárvore para corrigir o desequilíbrio.

Em seguida, aplica uma rotação à esquerda no nó atual para ajustar o equilíbrio. Se o fator de equilíbrio estiver dentro dos limites aceitáveis (-1, 0, 1), o nó atual não requer rotações e é retornado inalterado.

```
private int balanceamento(Node noAtual) {
    return  noAtual != null ? (altura(noAtual.getFilhoEsq()) -
altura(noAtual.getFilhoDir())) : 0;
}
```

O método privado “balanceadmento” calcula o fator de equilíbrio de um nó, que é a diferença entre a altura da subárvore esquerda e da subárvore direita.

2 Resultados:

2.1 Inserção Árvore Binária:

Inserções	Tentativa 1	Tentativa 2	Tentativa 3	Média
100	2551400	2482200	2462900	2498833
500	5358600	5108400	5470900	5312633
1.000	7505800	8426400	7720800	7884333
10.000	31922500	30613800	28919000	30485100
20.000	50641100	51368400	51200700	51070067

2.2 Inserção Árvore AVL:

Inserções	Tentativa 1	Tentativa 2	Tentativa 3	Média
100	2810700	3447300	3825500	3361167
500	6163000	6923800	7442200	6843000
1.000	7548200	8253900	8683500	8161867
10.000	35737200	32495400	32564600	33599067
20.000	55765800	58499600	54795000	56353467

Com inserções de pouco valores 100 e 500, temos uma diferença de 30% na performance, tendo o um desempenho parecido com 1.000 sendo que a partir de 10.000 valores a diferença fica em 10%

2.3 Buscar Valor na Árvore Binária e AVL:

2.3.1 100 valores:

```
Digite o número para buscar da Árvore: 518

Pré-ordem da Árvore Binária
714 660 489 219 190 88 76 72 161 159 155 116 158 162 214 191 423 230
Pré-ordem da Árvore AVL
653 316 190 88 76 72 159 155 116 158 161 162 234 219 214 191 222 221
Tempo decorrido para buscar 518 da Arvore Binária: 9800 nanosegundos
Tempo decorrido para buscar 518 da Arvore AVL: 9500 nanosegundos
```

2.3.2 500 valores:

```
Digite o número para buscar da Árvore: 517

Pré-ordem da Árvore Binária
404 313 80 6 5 1 46 36 33 11 9 6 9 10 23 20 13 18 25 24 24 29 27 30 30
Pré-ordem da Árvore AVL
404 221 80 46 33 11 6 5 1 9 6 9 10 25 23 18 13 20 24 24 29 27 30 30 36
Tempo decorrido para buscar 517 da Arvore Binária: 29600 nanosegundos
Tempo decorrido para buscar 517 da Arvore AVL: 14500 nanosegundos
```

2.3.3 1.000 valores:

```
Digite o número para buscar da Árvore: 519

Pré-ordem da Árvore Binária
659 222 66 62 46 13 2 3 8 6 3 8 10 8 8 9 14 16 14 33 26 25 16 32 28 27
Pré-ordem da Árvore AVL
509 222 107 66 46 14 8 3 2 6 3 10 8 8 8 9 13 14 33 28 25 16 16 26 26 27
Tempo decorrido para buscar 519 da Arvore Binária: 12500 nanosegundos
Tempo decorrido para buscar 519 da Arvore AVL: 4000 nanosegundos
```

2.3.4 10.000 valores:

```
Digite o número para buscar da Árvore: 520

Pré-ordem da Árvore Binária
434 242 227 192 102 7 4 3 1 2 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 3 3 3
Pré-ordem da Árvore AVL
407 192 112 55 32 16 9 5 3 2 1 1 1 1 1 1 1 1 1 2 1 2 2 2 2 2 2 4 3 3
Tempo decorrido para buscar 520 da Arvore Binária: 13900 nanosegundos
Tempo decorrido para buscar 520 da Arvore AVL: 4900 nanosegundos
```

2.3.5 20.000 valores:

```
Digite o número para buscar da Árvore: 519

Pré-ordem da Árvore Binária
186 59 13 12 11 7 6 4 3 2 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
Pré-ordem da Árvore AVL
583 322 186 110 59 37 17 9 6 4 3 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2
Tempo decorrido para buscar 519 da Arvore Binária: 14200 nanosegundos
Tempo decorrido para buscar 519 da Arvore AVL: 5300 nanosegundos
```

2.4 Deletar Valor na Árvore Binária e AVL:

2.4.1 100 valores:

```
Digite o número para deletar da Árvore: 518

Pré-ordem da Árvore Binária
714 660 489 219 190 88 76 72 161 159 155 116 158 162 214 191 423 230 22
Pré-ordem da Árvore AVL
653 316 190 88 76 72 159 155 116 158 161 162 234 219 214 191 222 221 23
Tempo decorrido para deletar 518 da Arvore Binária: 12700 nanosegundos
Tempo decorrido para deletar 518 da Arvore AVL: 7200 nanosegundos
```

2.4.2 500 valores:

```
Digite o número para deletar da Árvore: 517

Pré-ordem da Árvore Binária
404 313 80 6 5 1 46 36 33 11 9 6 9 10 23 20 13 18 25 24 24 29 27 30 30 3
Pré-ordem da Árvore AVL
404 221 80 46 33 11 6 5 1 9 6 9 10 25 23 18 13 20 24 24 29 27 30 30 36 3
Tempo decorrido para deletar 517 da Arvore Binária: 14300 nanosegundos
Tempo decorrido para deletar 517 da Arvore AVL: 6700 nanosegundos
```

2.4.3 1.000 valores:

```
Digite o número para deletar da Árvore: 519

Pré-ordem da Árvore Binária
659 222 66 62 46 13 2 3 8 6 3 8 10 8 8 9 14 16 14 33 26 25 16 32 28 27
Pré-ordem da Árvore AVL
509 222 107 66 46 14 8 3 2 6 3 10 8 8 8 9 13 14 33 28 25 16 16 26 26 27
Tempo decorrido para deletar 519 da Arvore Binária: 13700 nanosegundos
Tempo decorrido para deletar 519 da Arvore AVL: 5200 nanosegundos
```

2.4.4 10.000 valores:

```
Digite o número para deletar da Árvore: 520

Pré-ordem da Árvore Binária
434 242 227 192 102 7 4 3 1 2 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 3 3 3 3
Pré-ordem da Árvore AVL
407 192 112 55 32 16 9 5 3 2 1 1 1 1 1 1 1 1 1 2 1 2 2 2 2 2 2 4 3 3
Tempo decorrido para deletar 520 da Arvore Binária: 16200 nanosegundos
Tempo decorrido para deletar 520 da Arvore AVL: 8200 nanosegundos
```

2.4.5 20.000 valores:

```
Digite o número para deletar da Árvore: 519

Pré-ordem da Árvore Binária
186 59 13 12 11 7 6 4 3 2 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
Pré-ordem da Árvore AVL
583 322 186 110 59 37 17 9 6 4 3 2 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2
Tempo decorrido para deletar 519 da Arvore Binária: 17900 nanosegundos
Tempo decorrido para deletar 519 da Arvore AVL: 8100 nanosegundos
```

Como já visto na teoria o tempo de resposta da Arvore AVL menor do que da Árvore binaria, neste estudo a performance de buscar e deletar valores na Árvore AVL foi 2 vezes mais eficaz que a performance de buscar e deletar na Árvore Binária.