Relatório Projeto 2 V1.1 AED 2023/2024

Nome: Sérgio Lopes Marques Nº Estudante: 2022222096
PL (inscrição): 1 Email: sergiolmarques2004@gmail.com

IMPORTANTE:

- Os textos das conclusões devem ser manuscritos... o texto deve obedecer a este requisito para não ser penalizado.
- Texto para além das linhas reservadas, ou que não seja legível para um leitor comum, não será tido em conta.
- O relatório deve ser submetido num único PDF que deve incluir os anexos. A não observância deste formato é penalizada.

1. Planeamento

	Semana 1	Semana 2	Semana 3	Semana 4	Semana 5
Árvore Binária	**X**				
Árvore Binária Pesquisa		**X**			
Árvore AVL		**X**	**X**	**X**	**X**
Árvore VP					**X**
Finalização Relatório					**X**

2. Recolha de Resultados *(tabelas)*

Árvore Binária

	Tempo (ms)			
Tamanho Array	Conjunto A	Conjunto B	Conjunto C	Conjunto D
10000	198	175	210	0
15000	503	497	368	5
20000	979	927	1029	14
25000	1658	1107	1831	18
30000	2744	2542	2756	16

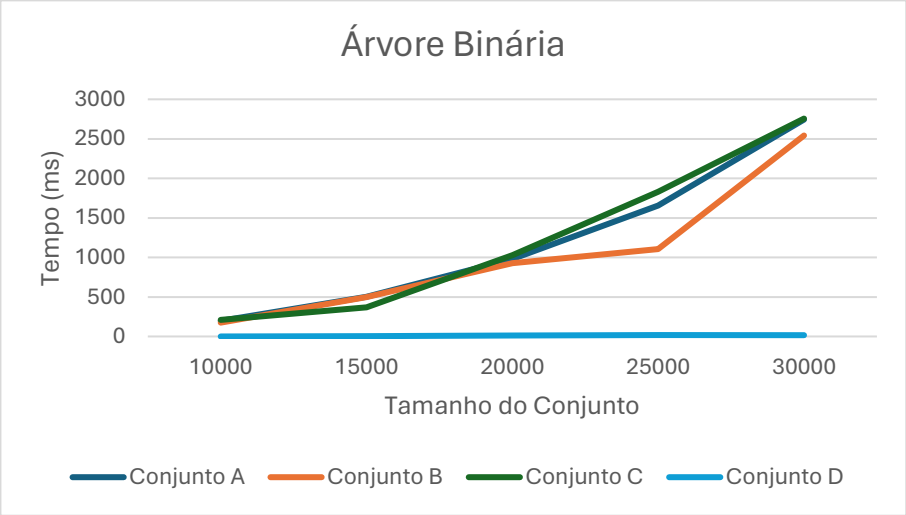
Árvore Binária de Pesquisa

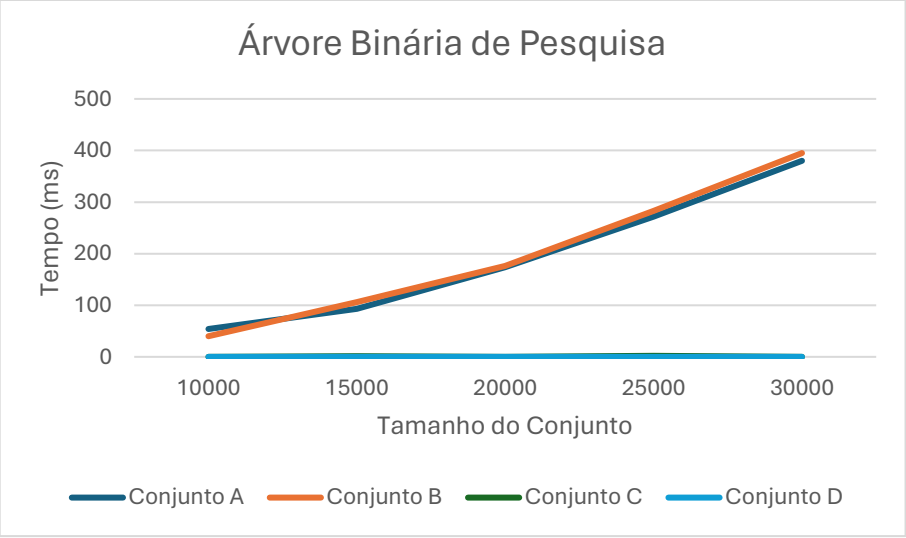
	Tempo (ms)			
Tamanho Array	Conjunto A	Conjunto B	Conjunto C	Conjunto D
10000	54	40	0	0
15000	93	106	1	0
20000	174	176	0	0
25000	272	283	2	0
30000	380	395	0	0

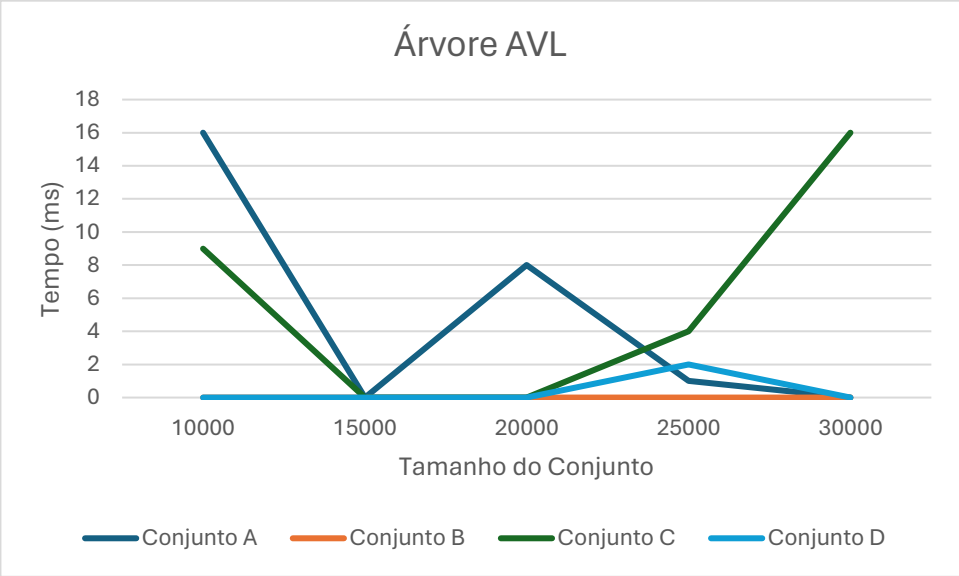
	Tempo (ms)			
Tamanho Array	Conjunto A	Conjunto B	Conjunto C	Conjunto D
10000	16	0	9	0
15000	0	0	0	0
20000	8	0	0	0
25000	1	0	4	2
30000	0	0	16	0

// usar o espaço que considerar necessário //

3. Visualização de Resultados *(gráficos)*



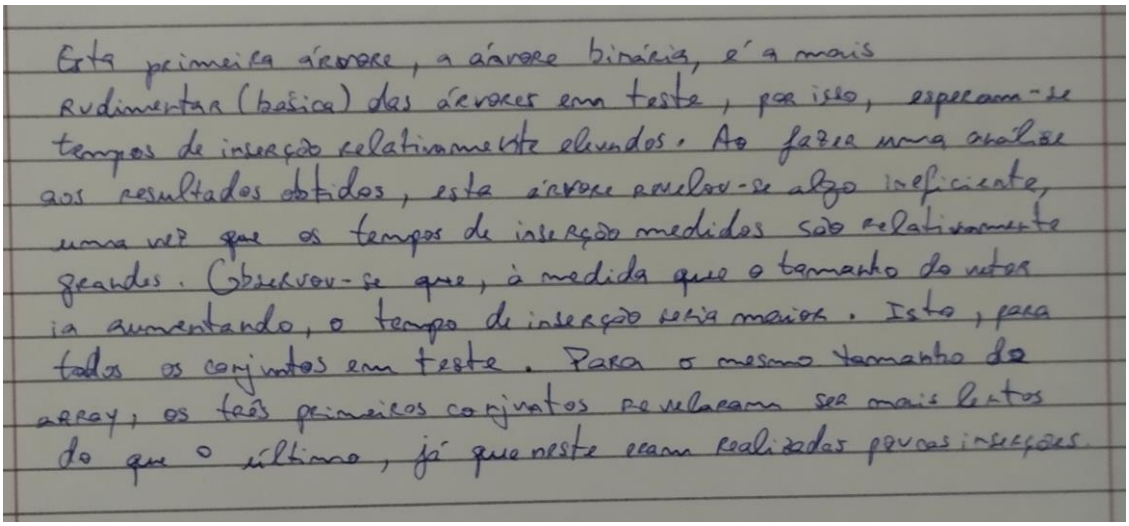




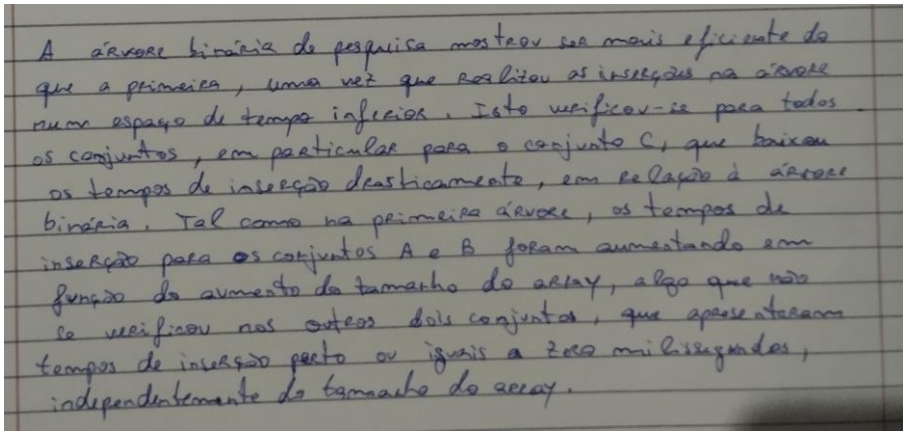
// usar o espaço que considerar necessário //

4. Conclusões *(as linhas desenhadas representam a extensão <u>máxima</u> de texto manuscrito)*

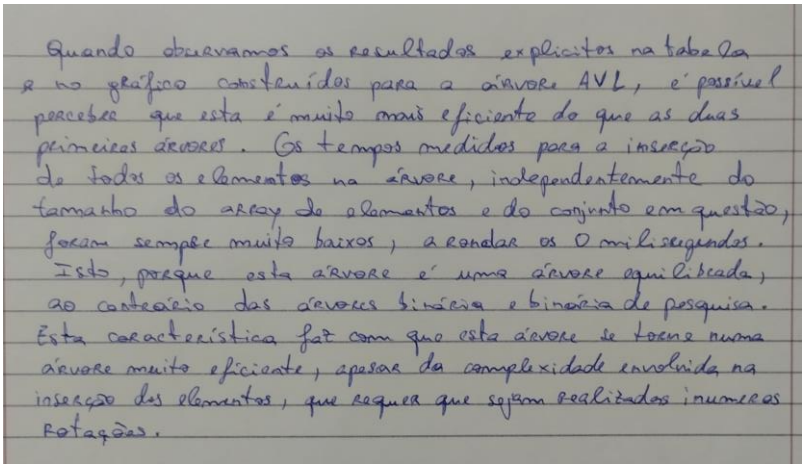
4.1 Tarefa 1


Esta primeira árvore, a árvore binária, é a mais rudimentar (básica) das árvores em teste, por isso, esperam-se tempos de inserção relativamente elevados. Ao fazer uma análise aos resultados obtidos, esta árvore revelou-se algo ineficiente, uma vez que os tempos de inserção medidos são relativamente grandes. Observou-se que, à medida que o tamanho do vetor ia aumentando, o tempo de inserção seria maior. Isto, para todos os conjuntos em teste. Para o mesmo tamanho do array, os três primeiros conjuntos revelaram ser mais lentos do que o último, já que neste foram realizadas poucas inserções.

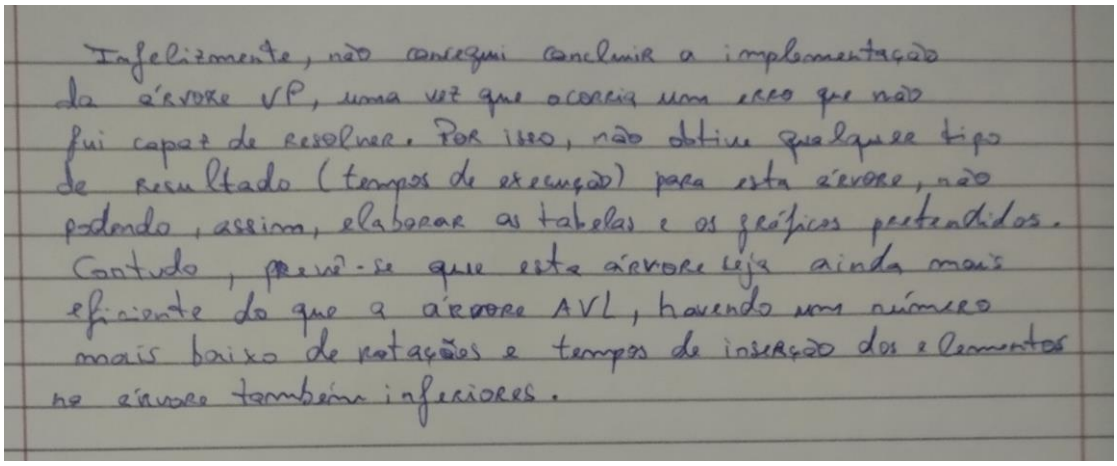
4.2 Tarefa 2


A árvore binária de pesquisa mostrou ser mais eficiente do que a primeira, uma vez que realizou as inserções na árvore num espaço de tempo inferior. Isto verificou-se para todos os conjuntos, em particular para o conjunto C, que baixou os tempos de inserção drasticamente, em relação à árvore binária. Tal como na primeira árvore, os tempos de inserção para os conjuntos A e B foram aumentando em função do aumento do tamanho do array, algo que não se verificou nos outros dois conjuntos, que apresentaram tempos de inserção perto ou iguais a zero milissegundos, independentemente do tamanho do array.

4.3 Tarefa 3

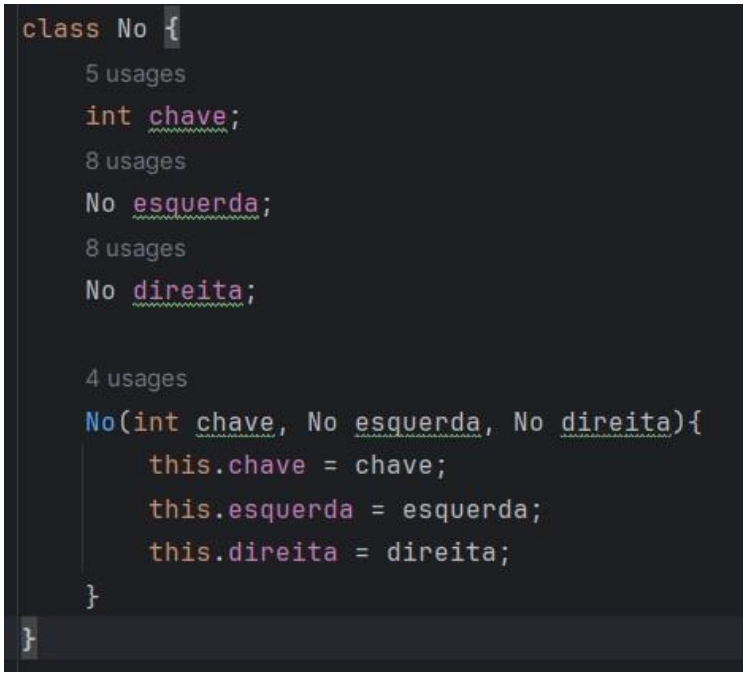

Quando observamos os resultados explícitos na tabela e no gráfico construídos para a árvore AVL, é possível perceber que esta é muito mais eficiente do que as duas primeiras árvores. Os tempos medidos para a inserção de todos os elementos na árvore, independentemente do tamanho do array de elementos e do conjunto em questão, foram sempre muito baixos, a rondar os 0 milissegundos. Isto, porque esta árvore é uma árvore equilibrada, ao contrário das árvores binária e binária de pesquisa. Esta característica fez com que esta árvore se torne numa árvore muito eficiente, apesar da complexidade envolvida na inserção dos elementos, que requer que sejam realizadas inúmeras rotações.

4.4 Tarefa 4


Infelizmente, não consegui concluir a implementação da árvore VP, uma vez que ocorria um erro que não fui capaz de resolver. Por isso, não obtive qualquer tipo de resultado (tempos de execução) para esta árvore, não podendo, assim, elaborar as tabelas e os gráficos pretendidos. Contudo, prevê-se que esta árvore seja ainda mais eficiente do que a árvore AVL, havendo um número mais baixo de rotações e tempos de inserção dos elementos na árvore também inferiores.

Anexo A – Código

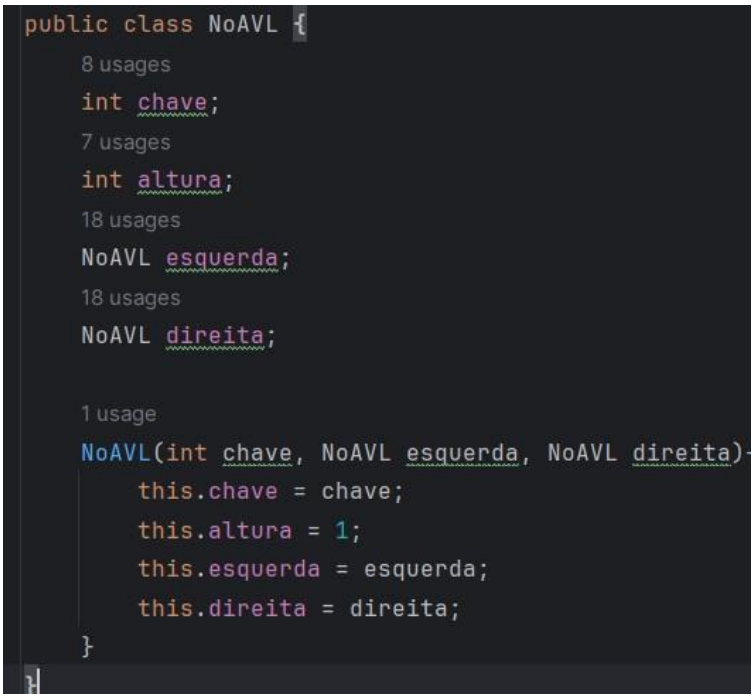
Classe Nó



```
class No {
    5 usages
    int chave;
    8 usages
    No esquerda;
    8 usages
    No direita;

    4 usages
    No(int chave, No esquerda, No direita){
        this.chave = chave;
        this.esquerda = esquerda;
        this.direita = direita;
    }
}
```

Classe Nó AVL



```
public class NoAVL {
    8 usages
    int chave;
    7 usages
    int altura;
    18 usages
    NoAVL esquerda;
    18 usages
    NoAVL direita;

    1 usage
    NoAVL(int chave, NoAVL esquerda, NoAVL direita)
        this.chave = chave;
        this.altura = 1;
        this.esquerda = esquerda;
        this.direita = direita;
    }
}
```

Classe Nó VP

```java
public class NoVP {

    5 usages
    int chave;
    22 usages
    NoVP pai;
    18 usages
    NoVP esquerda;
    15 usages
    NoVP direita;
    19 usages
    int cor;

    1 usage
    NoVP(int chave, NoVP pai, NoVP esquerda, NoVP direita){
        this.chave = chave;
        this.pai = pai;
        this.esquerda = esquerda;
        this.direita = direita;
        this.cor = 1;
    }
}
```

Classe Árvore Binária

```java
import java.util.Random;

4 usages
class Binaria {

    8 usages
    No raiz;

    2 usages
    Binaria(No raiz){
        this.raiz = raiz;
    }

    3 usages
    boolean verificaChaves(No no, int chave){

        if(no == null){
            return false;
        }
        if(no.chave == chave){
            return true;
        }
        else{
            return(verificaChaves(no.esquerda, chave) || verificaChaves(no.direita, chave));
        }
    }
     No inserir(No no, int chave){

         if(verificaChaves(no, chave) == false){

             if(no == null){
                 no = new No(chave,  esquerda: null,  direita: null);
                 return no;
             }

             Random rand = new Random();
             int num = rand.nextInt( origin: 0, bound: 2);

             if(num == 0){
                 no.esquerda = inserir(no.esquerda, chave);
             }
             else if(num == 1){
                 no.direita = inserir(no.direita, chave);
             }
         }
         else{
             return no;
         }
         return no;
     }
}
```

Classe Árvore Binária Pesquisa

```java
class BinariaPesquisa {

    8 usages
    No raiz;

    2 usages
    BinariaPesquisa(No raiz){
        this.raiz = raiz;
    }

    4 usages
    No inserir(No no, int chave){

        if(no == null){
            no = new No(chave, esquerda: null, direita: null);
            return no;
        }

        while(no != null){
            if(chave < no.chave){
                if(no.esquerda == null){
                    no.esquerda = new No(chave, esquerda: null, direita: null);
                    break;
                }
                else{
                    no = no.esquerda;
                }
            }
            else if(chave > no.chave){
                if(no.direita == null){
                    no.direita = new No(chave, esquerda: null, direita: null);
                    break;
                }
                else{
                    no = no.direita;
                }
            }
            else{
                break;
            }

        }

        return no;
    }
}
```

Classe Árvore AVL

```
class AVL {
    10 usages
    NoAVL raiz;
    7 usages
    int contaRotacoes;

    2 usages
    AVL(NoAVL raiz){
        this.raiz = raiz;
        this.contaRotacoes = 0;
    }

    4 usages
    int altura(NoAVL no){
        if(no == null){
            return 0;
        }
        return no.altura;
    }
    5 usages
    int alturaMaxima(NoAVL no1, NoAVL no2){
        return Math.max(altura(no1), altura(no2));
    }
int equilibrio(NoAVL no){
    if(no == null){
        return 0;
    }
    return (altura(no.esquerda)) - (altura(no.direita));
}

3 usages
NoAVL rotacaoDireita(NoAVL no){
    NoAVL esquerda = no.esquerda;
    NoAVL esqDireita = esquerda.direita;

    esquerda.direita = no;
    no.esquerda = esqDireita;

    no.altura = alturaMaxima(no.esquerda, no.direita) + 1;
    esquerda.altura = alturaMaxima(esquerda.esquerda, esquerda.direita) + 1;

    this.contaRotacoes = this.contaRotacoes + 1;

    return esquerda;
}
NoAVL rotacaoEsquerda(NoAVL no){
    NoAVL direita = no.direita;
    NoAVL dirEsquerda = direita.esquerda;

    direita.esquerda = no;
    no.direita = dirEsquerda;

    no.altura = alturaMaxima(no.esquerda, no.direita) + 1;
    direita.altura = alturaMaxima(direita.esquerda, direita.direita) + 1;

    this.contaRotacoes = this.contaRotacoes + 1;

    return direita;
}
```

```
NoAVL inserir(NoAVL no, int chave){

    if(no == null){
        no = new NoAVL(chave, esquerda: null, direita: null);
        return no;
    }

    if(chave < no.chave){
        no.esquerda = inserir(no.esquerda, chave);
    }
    else if(chave > no.chave){
        no.direita = inserir(no.direita, chave);
    }
    else{
        return no;
    }

    no.altura = alturaMaxima(no.esquerda, no.direita) + 1;

        if(equilibrio(no) > 1){
            if(chave < (no.esquerda).chave){
                return rotacaoDireita(no);
            }
            else if(chave > (no.esquerda).chave){
                no.esquerda = rotacaoEsquerda(no.esquerda);
                return rotacaoDireita(no);
            }
        }
        else if(equilibrio(no) < -1){
            if(chave > (no.direita).chave){
                return rotacaoEsquerda(no);
            }
            else if(chave < (no.direita).chave){
                no.direita = rotacaoDireita(no.direita);
                return rotacaoEsquerda(no);
            }
        }
        return no;
    }
}
```

Classe Árvore VP

```
class VP {
    12 usages
    NoVP raiz;
    7 usages
    int contaRotacoes;

    2 usages
    VP(NoVP raiz){
        this.raiz = raiz;
        this.contaRotacoes = 0;
    }
```

```
void rotacaoDireita(NoVP no){
    NoVP esquerda = no.esquerda;
    no.esquerda = esquerda.direita;
    if(no.esquerda != null){
        no.esquerda.pai = no;
    }
    esquerda.pai = no.pai;
    if(no.pai == null){
        this.raiz = esquerda;
    }
    else if(no == (no.pai).esquerda){
        (no.pai).esquerda = esquerda;
    }
    else{
        (no.pai).direita = esquerda;
    }
    esquerda.direita = no;
    no.pai = esquerda;

    this.contaRotacoes = this.contaRotacoes + 1;
}
```

```
void rotacaoEsquerda(NoVP no){
    NoVP direita = no.direita;
    no.direita = direita.esquerda;
    if(no.direita != null){
        (no.direita).pai = no;
    }
    direita.pai = no.pai;
    if(no.pai == null){
        this.raiz = direita;
    }
    else if(no == (no.pai).esquerda){
        (no.pai).esquerda = direita;
    }
    else{
        (no.pai).direita = direita;
    }
    direita.esquerda = no;
    no.pai = direita;

    this.contaRotacoes = this.contaRotacoes + 1;
}
```

```
NoVP inserir(NoVP no, int chave){

    if(no == null){
        no = new NoVP(chave, pai: null, esquerda: null, direita: null);
        return no;
    }
    if(chave < no.chave){
        no.esquerda = inserir(no.esquerda, chave);
        (no.esquerda).pai = no;
    }
    else if(chave > no.chave){
        no.direita = inserir(no.direita, chave);
        (no.direita).pai = no;
    }
    else{
        return no;
    }

    while((no.pai).cor == 1){

        NoVP pai = no.pai;
        NoVP avo = pai.pai;
```

```
if (pai == avo.esquerda) {
    NoVP tio = avo.direita;
    if (tio != null && tio.cor == 1) {
        avo.cor = 1;
        pai.cor = 0;
        tio.cor = 0;
        no = avo;
    } else {
        if (no == pai.direita) {
            no = pai;
            rotacaoEsquerda(no);
        }
        rotacaoDireita(avo);
        int temp = pai.cor;
        pai.cor = avo.cor;
        avo.cor = temp;
    }
```

```
            } else {
                NoVP tio = avo.esquerda;
                if (tio != null && tio.cor == 1) {
                    avo.cor = 1;
                    pai.cor = 0;
                    tio.cor = 0;
                    no = avo;
                } else {
                    if (no == pai.esquerda) {
                        no = pai;
                        rotacaoDireita(no);
                    }
                    rotacaoEsquerda(avo);
                    int temp = pai.cor;
                    pai.cor = avo.cor;
                    avo.cor = temp;
                }

            }
        }
        return no;
    }
}
```

Main

```java
import java.util.Arrays;
import java.util.Collections;
import java.util.Random;

public class Main {

    public static void main(String[] args) {

        int tamanho = 10000;

        for(int z = 0; z < 5; z++) {

            Random rand = new Random();
            String[] letras = {"A", "B", "C", "D"};

            Integer[] conjuntoA = new Integer[tamanho];
            for(int x = 0; x < tamanho; x++){
                conjuntoA[x] = rand.nextInt( origin: 1, bound: tamanho + 1);
            }
            Arrays.sort(conjuntoA);

            Integer[] conjuntoB = new Integer[tamanho];
            for(int x = 0; x < tamanho; x++){
                conjuntoB[x] = rand.nextInt( origin: 1, bound: tamanho + 1);
            }
            Arrays.sort(conjuntoB, Collections.reverseOrder());

            Integer[] conjuntoC = new Integer[tamanho];
            for(int x = 0; x < tamanho; x++){
                conjuntoC[x] = rand.nextInt( origin: 1, bound: tamanho + 1);
            }
```

```java
            Integer[] conjuntoD = new Integer[tamanho];
            int repetidos = (int) (tamanho * 0.9);
            int num = rand.nextInt( origin: 1, bound: 101);
            for(int x = 0; x < repetidos; x++) {
                conjuntoD[x] = num;
            }
            for(int x = repetidos; x < tamanho; x++){
                conjuntoD[x] = rand.nextInt( origin: 1,  bound: tamanho + 1);
            }

            Integer[][] conjuntos = new Integer[4][tamanho];
            System.arraycopy(conjuntoA,  srcPos: 0, conjuntos[0],  destPos: 0, conjuntoA.length);
            System.arraycopy(conjuntoB,  srcPos: 0, conjuntos[1],  destPos: 0, conjuntoB.length);
            System.arraycopy(conjuntoC,  srcPos: 0, conjuntos[2],  destPos: 0, conjuntoA.length);
            System.arraycopy(conjuntoD,  srcPos: 0, conjuntos[3],  destPos: 0, conjuntoA.length);

            long inicio, fim, duracao;
            System.out.println("TAMANHO DO ARRAY: "+tamanho+"\n");
            System.out.println("==ÁRVORE BINÁRIA==");
            for (int i = 0; i < conjuntos.length; i++) {
                Binaria arvoreBinaria = new Binaria( raiz: null);
                System.out.println("--Conjunto " + letras[i] + "--\n");
                inicio = System.currentTimeMillis();
                arvoreBinaria.raiz = arvoreBinaria.inserir(arvoreBinaria.raiz, conjuntos[i][0]);
                for (int j = 1; j < conjuntos[i].length; j++) {
                    arvoreBinaria.inserir(arvoreBinaria.raiz, conjuntos[i][j]);
                }
                fim = System.currentTimeMillis();
                duracao = fim - inicio;
                System.out.println("Tempo de inserção: " + duracao + " milisegundos\n");
```

```java
            System.out.println("==ÁRVORE BINÁRIA PESQUISA==");
            for (int i = 0; i < conjuntos.length; i++) {
                BinariaPesquisa arvorePesquisa = new BinariaPesquisa( raiz: null);
                System.out.println("--Conjunto " + letras[i] + "--\n");
                inicio = System.currentTimeMillis();
                arvorePesquisa.raiz = arvorePesquisa.inserir(arvorePesquisa.raiz, conjuntos[i][0]);
                for (int j = 1; j < conjuntos[i].length; j++) {
                    arvorePesquisa.inserir(arvorePesquisa.raiz, conjuntos[i][j]);
                }
                fim = System.currentTimeMillis();
                duracao = (fim - inicio);
                System.out.println("Tempo de inserção: " + duracao + " milisegundos\n");
            }
            System.out.println("==ÁRVORE AVL==");
            for (int i = 0; i < conjuntos.length; i++) {
                AVL arvoreAVL = new AVL( raiz: null);
                System.out.println("--Conjunto " + letras[i] + "--\n");
                inicio = System.currentTimeMillis();
                arvoreAVL.raiz = arvoreAVL.inserir(arvoreAVL.raiz, conjuntos[i][0]);
                for (int j = 1; j < conjuntos[i].length; j++) {
                    arvoreAVL.raiz = arvoreAVL.inserir(arvoreAVL.raiz, conjuntos[i][j]);
                }
                fim = System.currentTimeMillis();
                duracao = (fim - inicio);
                System.out.println("Tempo de inserção: " + duracao + " milisegundos");
                System.out.println("Número de rotações: " + arvoreAVL.contaRotacoes);
                System.out.println("\n");
            }
        }
    }
}
```

```
        System.out.println("==ÁRVORE VP==");
        for (int i = 0; i < conjuntos.length; i++) {
            VP arvoreVP = new VP( raiz: null);
            System.out.println("--Conjunto " + letras[i] + "--\n");
            inicio = System.currentTimeMillis();
            arvoreVP.raiz = arvoreVP.inserir(arvoreVP.raiz, conjuntos[i][0]);
            for (int j = 1; j < conjuntos[i].length; j++) {
                arvoreVP.raiz = arvoreVP.inserir(arvoreVP.raiz, conjuntos[i][j]);
            }
            fim = System.currentTimeMillis();
            duracao = (fim - inicio);
            System.out.println("Tempo de inserção: " + duracao + " milisegundos");
            System.out.println("Número de rotações: " + arvoreVP.contaRotacoes);
            System.out.println("\n");
        }

        tamanho += 5000;
    }

int[] conjuntoTeste = {10, 1, 1, 3, 4, 20, 42, 42, 37, 28, 30, 2, 45, 26, 26, 13, 5, 8, 41, 12};
Binaria arvoreBinaria = new Binaria( raiz: null);
BinariaPesquisa arvorePesquisa = new BinariaPesquisa( raiz: null);
AVL arvoreAVL = new AVL( raiz: null);
VP arvoreVP = new VP( raiz: null);

System.out.println("== Visualização das Árvores (Teste) ==\n");

System.out.println("--Árvore Binária--\n");
arvoreBinaria.raiz = arvoreBinaria.inserir(arvoreBinaria.raiz, conjuntoTeste[0]);
for (int j = 1; j < conjuntoTeste.length; j++) {
    arvoreBinaria.inserir(arvoreBinaria.raiz, conjuntoTeste[j]);
}
imprimirBinaria(arvoreBinaria.raiz,  espaco: 0);
System.out.println("\n");

System.out.println("--Árvore Binária de Pesquisa--\n");
arvorePesquisa.raiz = arvorePesquisa.inserir(arvorePesquisa.raiz, conjuntoTeste[0]);
for (int j = 1; j < conjuntoTeste.length; j++) {
    arvorePesquisa.inserir(arvorePesquisa.raiz, conjuntoTeste[j]);
}
imprimirBinaria(arvorePesquisa.raiz,  espaco: 0);
System.out.println("\n");

    System.out.println("--Árvore AVL--\n");
    arvoreAVL.raiz = arvoreAVL.inserir(arvoreAVL.raiz, conjuntoTeste[0]);
    for (int j = 1; j < conjuntoTeste.length; j++) {
        arvoreAVL.raiz = arvoreAVL.inserir(arvoreAVL.raiz, conjuntoTeste[j]);
    }
    imprimirAVL(arvoreAVL.raiz,  espaco: 0);
    System.out.println("\n");
    System.out.println("Número de rotações: " + arvoreAVL.contaRotacoes);
    System.out.println("\n");
    System.out.println("--Árvore VP--\n");
    arvoreVP.raiz = arvoreVP.inserir(arvoreVP.raiz, conjuntoTeste[0]);
    for (int j = 1; j < conjuntoTeste.length; j++) {
        arvoreVP.raiz = arvoreVP.inserir(arvoreVP.raiz, conjuntoTeste[j]);
    }
    imprimirVP(arvoreVP.raiz,  espaco: 0);
    System.out.println("\n");
    System.out.println("Número de rotações: " + arvoreVP.contaRotacoes);
    System.out.println("\n");
}
```

```java
public static void imprimirBinaria(No raiz, int espaco) {

    if (raiz == null) {
        return;
    }
    espaco += 10;

    imprimirBinaria(raiz.direita, espaco);

    System.out.print("\n");
    for (int i = 10; i < espaco; i++) {
        System.out.print(" ");
    }
    System.out.print(raiz.chave + "\n");

    imprimirBinaria(raiz.esquerda, espaco);
}
```

```java
public static void imprimirAVL(NoAVL raiz, int espaco) {

    if (raiz == null) {
        return;
    }
    espaco += 10;

    imprimirAVL(raiz.direita, espaco);

    System.out.print("\n");
    for (int i = 10; i < espaco; i++) {
        System.out.print(" ");
    }
    System.out.print(raiz.chave + "\n");

    imprimirAVL(raiz.esquerda, espaco);
}
```

```java
    public static void imprimirVP(NoVP raiz, int espaco) {

        if (raiz == null) {
            return;
        }
        espaco += 10;

        imprimirVP(raiz.direita, espaco);

        System.out.print("\n");
        for (int i = 10; i < espaco; i++) {
            System.out.print(" ");
        }
        if(raiz.cor == 1){
            System.out.print(raiz.chave + "| V \n");
        }
        else{
            System.out.print(raiz.chave + "| P \n");
        }

        imprimirVP(raiz.esquerda, espaco);
    }
}
```

Anexo B - Código de Autor

//indicar as linhas de código de que é autor//

A implementação das árvores binária e binária de pesquisa são da minha autoria.

A implementação da árvore AVL e da árvore VP foram, também, da minha autoria, apesar de retirar alguma informação dos slides e dos vídeos disponibilizados pelo professor.

Anexo C - Referências

//indicar as linhas de código de que não é autor, e qual a fonte//

As únicas funções das quais não sou autor são as funções que permitem imprimir as árvores.

O modelo dessas funções foi retirado do seguinte link:

https://www.geeksforgeeks.org/print-binary-tree-2-dimensions/