

Relatório Projeto 2 V1.1 AED 2023/2024

Nome: João António Faustino Vaz

Nº Estudante: 2022231087

PL (inscrição): 8

Email: joao.vaz1810@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	feito				
Árvore Binária Pesquisa		feito			
Árvore AVL			incompleta	feito	
Árvore VP				feito	
Finalização Relatório					feito

2. Recolha de Resultados (tabelas)

	binaria			
tam	A	B	C	D
200000	334,33	360,9	359,52	19,5
400000	2563,62	2193,7	2819,02	70,09
600000	6848,86	6187,4	6345,3	189,9
800000	9948,23	9872,4	10023,5	350,55
1000000	12242,5	12456,3	14234,2	521,03

	avl			
tam	A	B	C	D
200000	0,034	0,032	0,14	0,07
400000	0,064	0,061	0,27	0,11
600000	0,098	0,11	0,48	0,17
800000	0,11	0,13	0,56	0,2
1000000	0,14	0,14	0,68	0,27

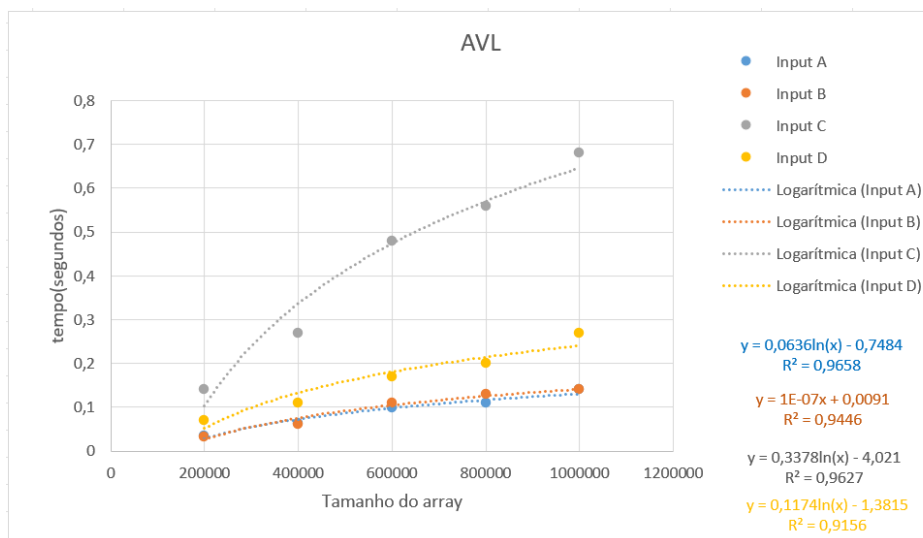
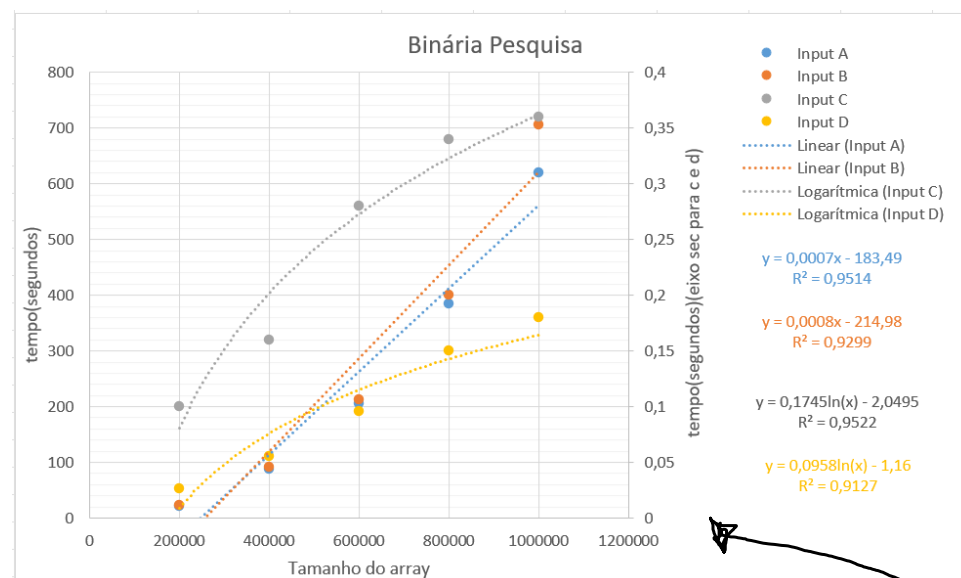
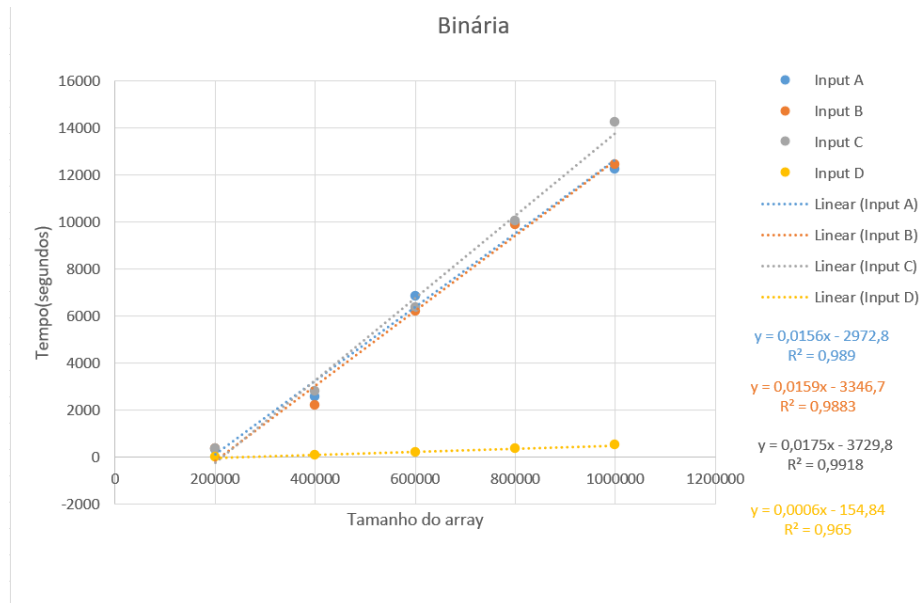
	rotações			
200000	181913	181891	84787	8506
400000	363830	363797	169547	17079
600000	545673	545819	254244	25456
800000	727433	727718	339300	33652
1000000	909592	909405	423149	42340

	Binária Pesquisa			
tam	A	B	C	D
200000	21,9	23,01	0,1	0,026
400000	87,83	92,17	0,16	0,055
600000	206,1	212,72	0,28	0,096
800000	383,88	400,7	0,34	0,15
1000000	619,39	704,86	0,36	0,18

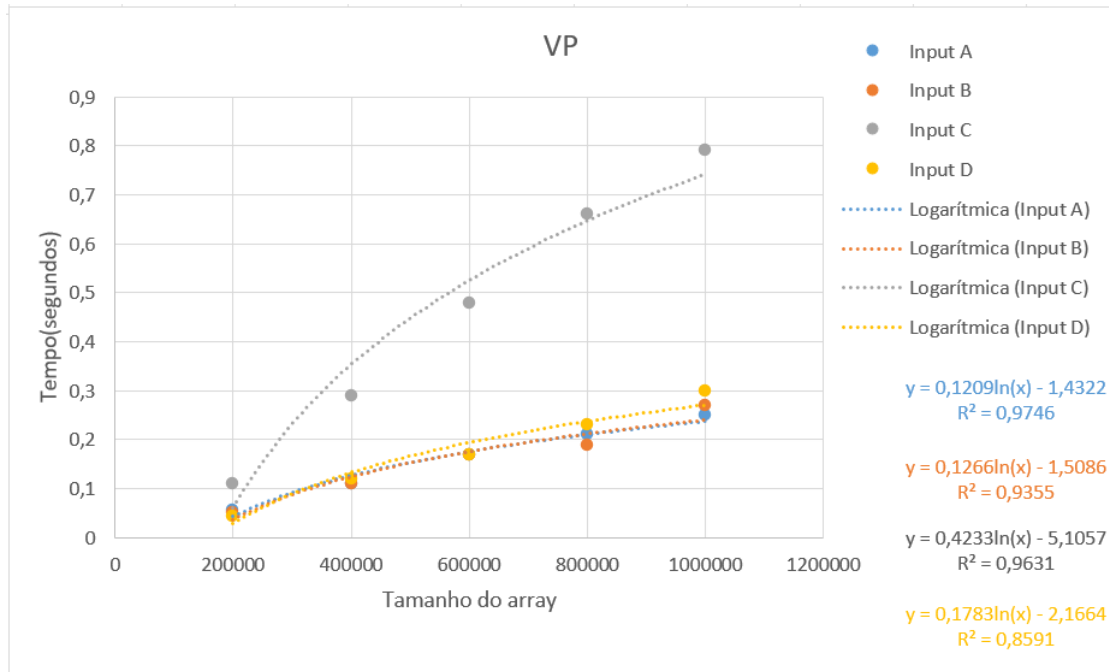
	vp			
tam	A	B	C	D
200000	0,055	0,05	0,11	0,045
400000	0,11	0,11	0,29	0,12
600000	0,17	0,17	0,48	0,17
800000	0,21	0,19	0,66	0,23
1000000	0,25	0,27	0,79	0,3

	rotações			
200000	181867	181836	70585	7106
400000	363782	363789	141262	14062
600000	545633	545678	212292	21073
800000	727548	727575	282395	28020
1000000	909321	909542	353327	35238

3. Visualização de Resultados (gráficos)



lixo
secundário
para o C
e D



4. Conclusões (as linhas desenhadas representam a extensão máxima de texto manuscrito)

4.1 Tarefa 1

Relativamente à árvore binária simples podemos ver através do gráfico que os tempos de execução para os três primeiros inputs (A, B e C) são muito parecidos. No input D já temos uma diminuição dos tempos de execução muito significativa devido ao facto de o input ter 90% dos números repetidos. Em relação à complexidade é $O(n)$, ou seja, percorremos os nós caso a árvore toda e isso observa-se no gráfico.

4.2 Tarefa 2

Relativamente à árvore binária de pesquisa os tempos são muito inferiores. Optei por usar a regressão linear nos inputs A e B visto que a árvore será toda percorrida nesses inputs e tenderá ao para um lado como se fosse uma lista. Nos inputs C e D isso não acontece

e termos uma árvore mais equilibrada sendo que a complexidade é $O(\log n)$ e portanto utilizei regressões logarítmicas na sua representação.

4.3 Tarefa 3

Na AVL, vemos claramente uma redução do tempo. A complexidade é $O(\log n)$ e conseguimos tirar várias conclusões: o tempo de execução para o input A e B é muito similar, no C o tempo aumenta o que nos indica que as Árvores AVL não se comportam de forma muito rápida quando os inputs são estes ordenados. Para além disso, podemos ver que o número de rotações é significativamente maior no conjunto D, o que faz sentido devido aos tais 90% repetidos.

4.4 Tarefa 4

Em relação às VP, a complexidade é igualmente $O(\log n)$ mas podemos ver uma melhoria de desempenho principalmente em relação aos conjuntos C e D. É interessante ver que em termos de tempo as AVL e VP são muito similares, no entanto as rotações nas VP são maiores o que demonstra uma vantagem sobre as AVL.

Anexo B - Código de Autor

Todo o código feito na árvore binária simples, exceto a função de dar print à árvore e do funcionamento da stack em java.

Todo o código feito Binária de pesquisa à exceção da função de dar print à árvore.

Na AVL, as funções insert, balancefactor, height, updateheight e rebalance.

Na VP, a função getcolor.

Anexo C - Referências

<https://www.geeksforgeeks.org/stack-class-in-java/>

Função de dar print à árvore – chatgpt

Função rotateleft e rotateright na AVL – chatgpt

Função insert, rotateleft e rotateright na VP - chatgpt

Código Binária

```
1 import java.util.*;
2
3 public class Binaria {
4     public class Node {
5         int data;
6         Node left;
7         Node right;
8         public Node(int data) {
9             this.data = data;
10            this.left = null;
11            this.right = null;
12        }
13    }
14    Node root=null;
15    public void insert(int data) {
16        if (root == null) {
17            root=new Node(data);
18            return;
19        }
20        Random random = new Random();
21        Node atual = root;
22        while (true) {
23            if (random.nextBoolean()) {
24                if (atual.left == null) {
25                    atual.left = new Node(data);
26                    return;
27                } else {
28                    atual = atual.left;
29                }
30            } else {
31                if (atual.right == null) {
32                    atual.right = new Node(data);
33                    return;
34                } else {
35                    atual = atual.right;
36                }
37            }
38        }
39    }
40
41    public boolean find(int data) {
42        if (root == null) {
43            return false;
44        }
45
46        Stack<Node> pilha = new Stack<>();
47        pilha.push(root);
48        while (!pilha.isEmpty()) {
49            Node atual = pilha.pop();
50            if (atual.data == data) {
51                return true;
52            }
53            if (atual.right != null) {
54                pilha.push(atual.right);
55            }
56            if (atual.left != null) {
57                pilha.push(atual.left);
58            }
59        }
60        return false;
61    }
62
63    public void printTree(Node node) {
64        printTreeHelper(node, 0);
65    }
66
67    private void printTreeHelper(Node node, int level) {
68        if (node != null) {
69            printTreeHelper(node.right, level + 1);
70            for (int i = 0; i < level; i++) {
71                System.out.print(" ");
72            }
73            System.out.println(node.data);
74            printTreeHelper(node.left, level + 1);
75        }
76    }
77
78    public static void main(String[] args) {
79        int size = 1000000;
80        ArrayList<Integer> teste=new ArrayList<>();
81        for(int i=0;i<20;i++){
82            teste.set(i,i);
83        }
84        ArrayList<Integer> input = new ArrayList<>();
85        Random random = new Random();
86        for (int i = 0; i < size; i++) {
87            input.add(i);
88        }
89
90        for (int j = (int)(0.9*size); j < size; j++) {
91            int num = random.nextInt(size);
92            input.set(j, num);
93        }
94        Collections.sort(input);
95        ArrayList<Integer> inputb = new ArrayList<>(input);
96        inputb.sort(Collections.reverseOrder());
97        ArrayList<Integer> inputc = new ArrayList<>(inputb);
98        Collections.shuffle(inputc);
99        ArrayList<Integer> inputd = new ArrayList<>(inputc);
100
101        int numDistinct = size / 10;
102        for (int k = numDistinct; k < size; k++) {
103            inputd.set(k, inputd.get(k % numDistinct));
104        }
105        Collections.shuffle(inputd);
106        Binaria arvoreBinaria=new Binaria();
107        long ti=System.nanoTime();
108        for(int i=0;i<size;i++){
109            if(!arvoreBinaria.find(inputd.get(i))){
110                arvoreBinaria.insert(inputd.get(i));
111            }
112        }
113        long tf=System.nanoTime();
114        System.out.println(tf-ti);
115    }
116 }
```

Binária Pesquisa

```

1 import java.util.*;
2
3 public class BinariaPesquisa {
4     public class Node {
5         int data;
6         Node left;
7         Node right;
8
9         public Node(int data) {
10             this.data = data;
11             this.left = null;
12             this.right = null;
13         }
14     }
15     Node root = null;
16     public boolean insert(int data) {
17         if (root == null) {
18             root = new Node(data);
19             return false;
20         }
21         Node atual = root;
22         while (true) {
23             if (data < atual.data) {
24                 if (atual.left == null) {
25                     atual.left = new Node(data);
26                     return false;
27                 }
28                 atual = atual.left;
29             } else if (data > atual.data) {
30                 if (atual.right == null) {
31                     atual.right = new Node(data);
32                     return false;
33                 }
34                 atual = atual.right;
35             } else {
36                 return true;
37             }
38         }
39     }
40     public void printTree(Node node) {
41         printTreeHelper(node, 0);
42     }
43
44     private void printTreeHelper(Node node, int level) {
45         if (node != null) {
46             printTreeHelper(node.right, level + 1);
47             for (int i = 0; i < level; i++) {
48                 System.out.print("  ");
49             }
50             System.out.println(node.data);
51             printTreeHelper(node.left, level + 1);
52         }
53     }
54
55     public static void main(String[] args) {
56         int size = 1000000;
57         ArrayList<Integer> teste = new ArrayList<>();
58         for (int i = 0; i < 20; i++) {
59             teste.set(i, i);
60         }
61         ArrayList<Integer> input = new ArrayList<>();
62         Random random = new Random();
63         for (int i = 0; i < size; i++) {
64             input.add(i);
65         }
66         for (int j = (int)(0.9*size); j < size; j++) {
67             int num = random.nextInt(size);
68             input.set(j, num);
69         }
70         Collections.sort(input);
71         ArrayList<Integer> inputb = new ArrayList<>(input);
72         inputb.sort(Collections.reverseOrder());
73         ArrayList<Integer> inputc = new ArrayList<>(inputb);
74         Collections.shuffle(inputc);
75         ArrayList<Integer> inputd = new ArrayList<>(input);
76         int numDistinct = size / 10;
77         for (int k = numDistinct; k < size; k++) {
78             inputd.set(k, inputd.get(k % numDistinct));
79         }
80         Collections.shuffle(inputd);
81         BinariaPesquisa arvoreBinariaPesquisa = new BinariaPesquisa();
82         long ti = System.nanoTime();
83         for (int i = 0; i < size; i++) {
84             arvoreBinariaPesquisa.insert(inputc.get(i));
85         }
86         long tf = System.nanoTime();
87         System.out.println((tf-ti));
88     }
89 }

```

AVL

```

1 import java.util.*;
2
3 public class AVL {
4     public class Node {
5         int data;
6         Node left;
7         Node right;
8         int height;
9
10        public Node(int data) {
11            this.data = data;
12            this.left = null;
13            this.right = null;
14            this.height = 1;
15        }
16    }
17    Node root = null;
18    int rotationCount = 0;
19    public int height(Node node) {
20        if (node != null) {
21            return node.height;
22        } else {
23            return 0;
24        }
25    }
26    public void updateHeight(Node node) {
27        if (node != null) {
28            int leftChildHeight = height(node.left);
29            int rightChildHeight = height(node.right);
30            node.height = Math.max(leftChildHeight, rightChildHeight) + 1;
31        }
32    }
33    public int balanceFactor(Node node) {
34        if (node != null) {
35            return height(node.left) - height(node.right);
36        } else {
37            return 0;
38        }
39    }
40    public Node insert(Node node, int data) {
41        if (node == null) {
42            return new Node(data);
43        }
44        if (data < node.data) {
45            node.left = insert(node.left, data);
46        } else if (data > node.data) {
47            node.right = insert(node.right, data);
48        } else {
49            return node;
50        }
51        updateHeight(node);
52        return rebalance(node);
53    }
54    private Node rotateRight(Node node) {
55        Node leftChild = node.left;
56        node.left = leftChild.right;
57        leftChild.right = node;
58        updateHeight(node);
59        updateHeight(leftChild);
60        return leftChild;
61    }
62    private Node rotateLeft(Node node) {
63        Node rightChild = node.right;
64        node.right = rightChild.left;
65        rightChild.left = node;
66        updateHeight(node);
67        updateHeight(rightChild);
68        return rightChild;
69    }
70    public Node rebalance(Node node) {
71        updateHeight(node);
72        int balanceFactor = balanceFactor(node);
73
74        if (balanceFactor > 1) {
75            if (height(node.left.left) >= height(node.left.right)) {
76                node = rotateRight(node);
77            } else {
78                node.left = rotateLeft(node.left);
79                node = rotateRight(node);
80            }
81            rotationCount++;
82        }
83        else if (balanceFactor < -1) {
84            if (height(node.right.right) >= height(node.right.left)) {
85                node = rotateLeft(node);
86            } else {
87                node.right = rotateRight(node.right);
88                node = rotateLeft(node);
89            }
90            rotationCount++;
91        }
92        return node;
93    }
94
95    public void printTree(Node node) {
96        printTreeHelper(node, 0);
97    }
98    private void printTreeHelper(Node node, int level) {
99        if (node != null) {
100            printTreeHelper(node.right, level + 1);
101            for (int i = 0; i < level; i++) {
102                System.out.print("  ");
103            }
104            System.out.println(node.data);
105            printTreeHelper(node.left, level + 1);
106        }
107    }
108    public static void main(String[] args) {
109        int size = 1000000;
110        ArrayList<Integer> teste = new ArrayList<>();
111        for (int i = 0; i < 20; i++) {
112            teste.set(i, i);
113        }
114        ArrayList<Integer> input = new ArrayList<>();
115        Random random = new Random();
116        for (int i = 0; i < size; i++) {
117            input.add(i);
118        }
119        for (int j = (int)(0.9*size); j < size; j++) {
120            int num = random.nextInt(size);
121            input.set(j, num);
122        }
123        Collections.sort(input);
124        ArrayList<Integer> inputb = new ArrayList<>(input);
125        inputb.sort(Collections.reverseOrder());
126        ArrayList<Integer> inputc = new ArrayList<>(inputb);
127        Collections.shuffle(inputc);
128        ArrayList<Integer> inputd = new ArrayList<>(inputc);
129        int numDistinct = size / 10;
130        for (int k = numDistinct; k < size; k++) {
131            inputd.set(k, inputd.get(k % numDistinct));
132        }
133        Collections.shuffle(inputd);
134        AVL avl = new AVL();
135        long ti = System.nanoTime();
136        for (int i = 0; i < size; i++) {
137            avl.root = avl.insert(avl.root, inputc.get(i));
138        }
139        long tf = System.nanoTime();
140        System.out.println("Tempo de inserção: " + (tf - ti) + " nanossegundos");
141        System.out.println("Número total de rotações durante a inserção: " + avl.rotationCount);
142    }
143 }
144

```

VP

```

1 import java.util.ArrayList;
2 import java.util.Collections;
3 import java.util.Random;
4
5 public class VP {
6     public class Node {
7         int data;
8         Node left;
9         Node right;
10        char color;
11
12        public Node(int data) {
13            this.data = data;
14            this.left = null;
15            this.right = null;
16            this.color = 'r';
17        }
18    }
19
20    Node root = null;
21    int rotationCount = 0;
22
23    char getColor(Node node) {
24        if (node != null) {
25            return node.color;
26        } else {
27            return 'b';
28        }
29    }
30
31    public Node insert(Node node, int data) {
32        if (node == null) {
33            Node newNode = new Node(data);
34            if (root == null) {
35                newNode.color = 'b';
36            }
37            return newNode;
38        }
39
40        if (data < node.data) {
41            node.left = insert(node.left, data);
42        } else if (data > node.data) {
43            node.right = insert(node.right, data);
44        } else {

```

```

81        } else {
82            node.color = 'r';
83            node.left.color = 'b';
84            node.right.color = 'b';
85        }
86    }
87
88    if (root == node) {
89        node.color = 'b';
90    }
91
92    return node;
93 }
94
95
96 private Node rotateRight(Node node) {
97     Node leftChild = node.left;
98     node.left = leftChild.right;
99     leftChild.right = node;
100    leftChild.color = 'b';
101    node.color = 'r';
102    return leftChild;
103 }
104
105 private Node rotateLeft(Node node) {
106     Node rightChild = node.right;
107     node.right = rightChild.left;
108     rightChild.left = node;
109     rightChild.color = 'b';
110     node.color = 'r';
111     return rightChild;
112 }
113
114 public void printTree(Node node) {
115     printTreeHelper(node, 0);
116 }
117
118 private void printTreeHelper(Node node, int level) {
119     if (node != null) {
120         printTreeHelper(node.right, level + 1);
121         for (int i = 0; i < level; i++) {
122             System.out.print(" ");
123         }
124         System.out.println(node.data + "(" + node.

```

```

45         return node;
46     }
47
48     if (getColor(node.left) == 'r' && getColor(node.
49         left.left) == 'r') {
50         if (getColor(node.right) == 'b' || getColor(
51             node.right) == 'r') {
52             node = rotateRight(node);
53             rotationCount++;
54         } else {
55             node.color = 'r';
56             node.left.color = 'b';
57             node.right.color = 'b';
58         }
59     } else if (getColor(node.left) == 'r' && getColor(
60         node.left.right) == 'r') {
61         if (getColor(node.right) == 'b' || getColor(
62             node.right) == 'r') {
63             node.left = rotateLeft(node.left);
64             node = rotateRight(node);
65             rotationCount++;
66         } else {
67             node.color = 'r';
68             node.left.color = 'b';
69             node.right.color = 'b';
70         }
71     } else if (getColor(node.right) == 'r' && getColor
72         (node.right.left) == 'r') {
73         if (getColor(node.left) == 'b' || getColor(
74             node.left) == 'r') {
75             node.right = rotateRight(node.right);
76             node = rotateLeft(node);
77             rotationCount++;
78         } else {
79             node.color = 'r';
80             node.left.color = 'b';
81             node.right.color = 'b';
82         }
83     } else if (getColor(node.right) == 'r' && getColor
84         (node.right.right) == 'r') {
85         if (getColor(node.left) == 'b' || getColor(
86             node.left) == 'r') {
87             node = rotateLeft(node);
88             rotationCount++;
89         }
90     }
91 }

```

```

124 color + ")");
125     printTreeHelper(node.left, level + 1);
126 }
127 }
128
129 public static void main(String[] args) {
130     int size = 1000000;
131     ArrayList<Integer> teste = new ArrayList<>();
132     for (int i = 0; i < 20; i++) {
133         teste.set(i, i);
134     }
135     ArrayList<Integer> input = new ArrayList<>();
136     Random random = new Random();
137     for (int i = 0; i < size; i++) {
138         input.add(i);
139     }
140     for (int j = (int)(0.9*size); j < size; j++) {
141         int num = random.nextInt(size);
142         input.set(j, num);
143     }
144     Collections.sort(input);
145     ArrayList<Integer> inputb = new ArrayList<>(input
146 );
147     inputb.sort(Collections.reverseOrder());
148     ArrayList<Integer> inputc = new ArrayList<>(
149         inputb);
150     Collections.shuffle(inputc);
151     ArrayList<Integer> inputd = new ArrayList<>(input
152 );
153     int numDistinct = size / 10;
154     for (int k = numDistinct; k < size; k++) {
155         inputd.set(k, inputd.get(k % numDistinct));
156     }
157     Collections.shuffle(inputd);
158     VP arvoreVP = new VP();
159     long ti = System.nanoTime();
160     for (int i = 0; i < size; i++) {
161         arvoreVP.root = arvoreVP.insert(arvoreVP.root
162             , inputc.get(i));
163     }
164     long tf = System.nanoTime();
165     System.out.println("Tempo de inserção: " + (tf -
166         ti) + " nanossegundos");
167     System.out.println("Número total de rotações

```

```

162 durante a inserção: " + arvoreVP.rotationCount);
163 }
164 }
165

```