Relatório Projeto 2

Mestrado em Computação Aplicada – Algoritmos e Programação – Turma 01A

Aluna: Anna Ravaglio // TIA: 72256771

Aluna: Rong Rong Yang // TIA: 72200693

Descrição do projeto

Para este teste, foi utilizado um computador do modelo Dell Gaming 3, com 16GB de memória RAM, processador Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz, a linguagem de programação escolhida foi o Python com a extensão Pandas utilizado no Visual Studio Code e Jupyter Notebook.

Foi escolhida uma base de livros, constituída por detalhamentos dos livros do site de avaliação e biblioteca individual GoodReads(https://www.kaggle.com/datasets/jealousleopard/goodreadsbooks), onde existem os dados:

- ID do Livro no Goodreads
- Título
- Autores
- Média de Avaliação
- ISBN/ISBN13
- Idioma
- Números de páginas
- Contagem de Avaliações
- Contagem de Textos de Avaliações
- Data de Publicação do Livro
- Editora

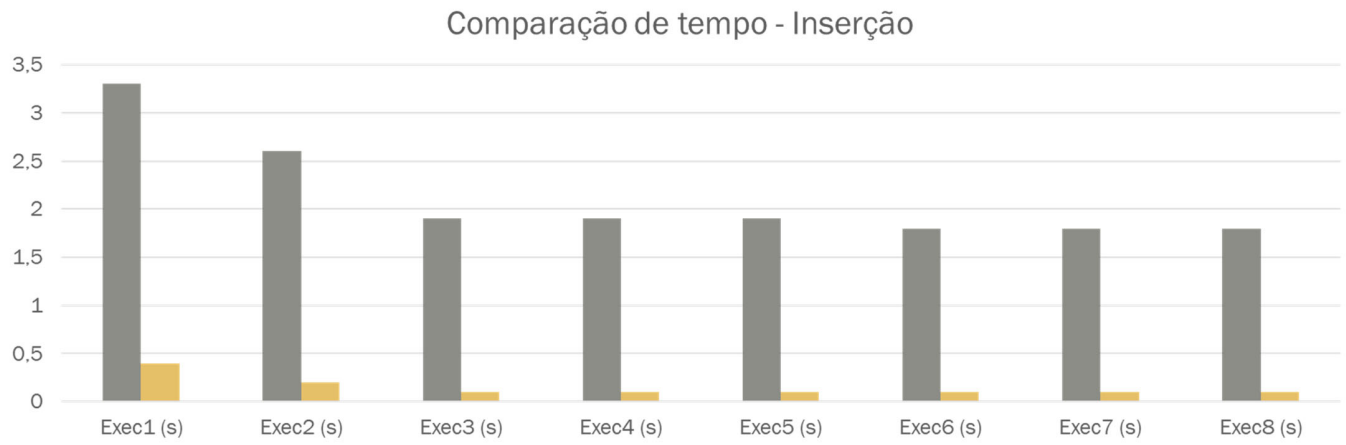
A base foi reduzida de um total de 11000 para 3000 livros para melhor visualização de resultados desta pesquisa por conta da recursão e da capacidade do computador utilizado. Já foi setada a recursão para um valor mais alto pois em testes houve erro de estouro de memória.

Metodologia de Pesquisa

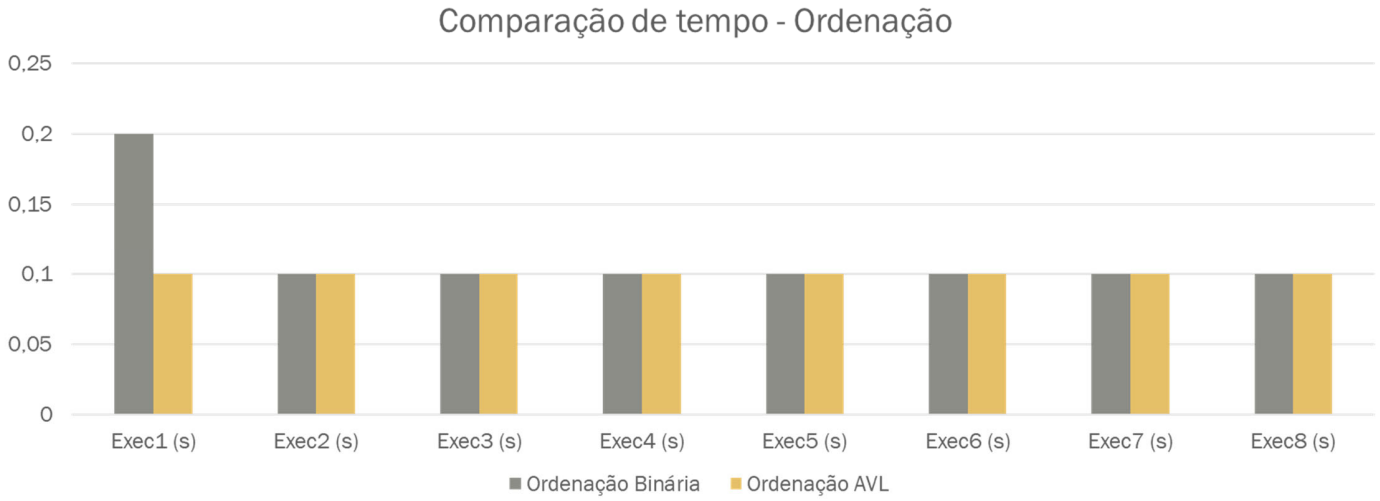
- Foi escolhido para fazer a árvore o dado de índice, pois a base já veio com índice dos livros e com pulos entre os índices.
- Como no caso dos livros o índice deles começam em 1 e terminam em 11037 para os 3000, foi utilizado este range para a definição dos índices de busca.
- Para a pesquisa, foram escolhidas 3 amostras fixas (Início, meio e final), 4 aleatórias geradas apenas uma vez, dois à direita e dois à esquerda e 1 amostra não existente.
- Os índices de busca selecionados foram: Fixos(1, 5518, 11037), Aleatórios(2802, 3939, 8249, 10964), Inexistente(5000).
- Foi criada uma estrutuda para a contagem do tempo padrão para todas as pesquisas, uma para cada índice selecionado, descrita abaixo:

Resultados

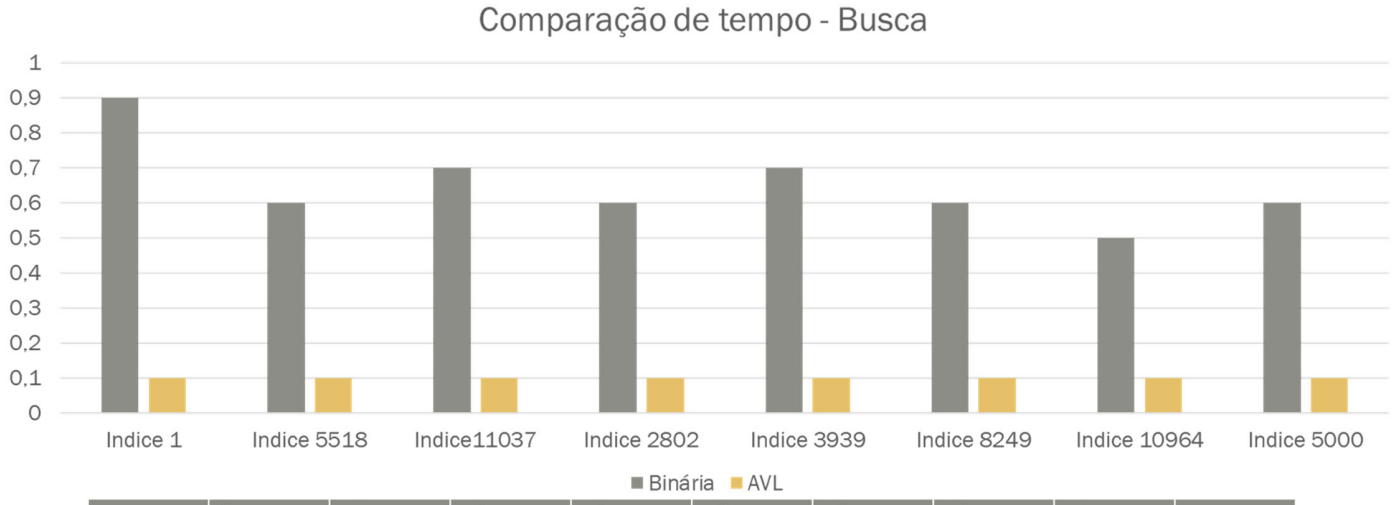
- Os resultados foram melhores que o esperado, obtivemos uma diferença grande entre a busca de ávore AVL e de árvore Binária, sendo a AVL a melhor opção pois executou praticamente em tempo de execução.
- Para outras buscas sem serem com o índice já utilizado, no caso da nossa base de dados, para um atributo não chave seria a busca por Autor + Título que retornaria um livro específico, pois eles não estão duplicados na base, ou por outra chave, como o ISBN, pois o restante das informações são muito genéricas para serem utilizadas.
- A altura da árvore AVL ficou em 12, a da binária não fizemos a avaliação.



Função	Exec 1	Exec 2	Exec 3	Exec 4	Exec 5	Exec 6	Exec 7	Exec 8	Média
Inserção Binária	3.3	2.6	1.9	1.9	1.9	1.8	1.8	1.8	2.125
Inserção AVL	0.4	0.2	0.1	0.1	0.1	0.1	0.1	0.1	0.15



Função	Exec 1	Exec 2	Exec 3	Exec 4	Exec 5	Exec 6	Exec 7	Exec 8	Média
Ordenação Binária	0.2	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1125
Ordenação AVL	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1



Busca	Índice 1	Índice 5518	Índice 11037	Índice 2802	Índice 3939	Índice 8249	Índice 10964	Índice 5000	Média
Binária	0.9	0.6	0.7	0.6	0.7	0.6	0.5	0.6	0.65
AVL	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1

```
import pandas as pd
import numpy as np import sys
sys.setrecursionlimit(5000000)
arquivo = pd.read_csv("books.csv", sep=";") display(arquivo
```

	bookID	title	authors	average_rating	isbn	isbn13	language_code	num_pages	ratings_count	text_reviews_count	publication_date	publisher
0	1	Harry Potter and the Half-Blood Prince (Harry ...	J.K. Rowling/Mary GrandPré	4.57	439785960	9,78044E+12	eng	652	2095690	27591	9/16/2006	Scholastic Inc.
1	2	Harry Potter and the Order of the Phoenix (Har...	J.K. Rowling/Mary GrandPré	4.49	439358078	9,78044E+12	eng	870	2153167	29221	09/01/2004	Scholastic Inc.
2	4	Harry Potter and the Chamber of Secrets (Harry...	J.K. Rowling	4.42	439554896	9,78044E+12	eng	352	6333	244	11/01/2003	Scholastic
3	5	Harry Potter and the Prisoner of Azkaban (Harr...	J.K. Rowling/Mary GrandPré	4.56	043965548X	9,78044E+12	eng	435	2339585	36325	05/01/2004	Scholastic Inc.
4	8	Harry Potter Boxed Set Books 1-5 (Harry Potte...	J.K. Rowling/Mary GrandPré	4.78	439682584	9,78044E+12	eng	2690	41428	164	9/13/2004	Scholastic
...	...	...	...	...	...	...	...	...	...	...	...	...
2995	11027	No me cogeréis vivo: artículos 2001-2005	Arturo Pérez-Reverte	4.16	8420469432	9,78842E+12	spa	537	199	5	11/10/2005	Alfaguara
2996	11031	The Flanders Panel	Arturo Pérez-Reverte/Margaret Jull Costa	3.79	156029588	9,78016E+12	eng	295	12998	589	06/07/2004	Mariner Books
2997	11033	The Seville Communion	Arturo Pérez-Reverte/Sonia Soto	3.68	156029812	9,78016E+12	eng	400	3912	209	06/07/2004	Mariner Books
2998	11036	Little Birds	Anaïs Nin	3.72	156029049	9,78016E+12	eng	148	7888	366	02/02/2004	Mariner Books
2999	11037	The Diary of Anaïs Nin Vol. 2: 1934-1939	Anaïs Nin/Gunther Stuhlmann	4.18	156260263	9,78016E+12	eng	372	1775	47	3/25/1970	Mariner Books

3000 rows × 12 columns

Código Fonte Utilizado:

```
import time
#Funções para formatar o tempo
def formatTime(diff):     r =
diff % 3600
    minutes, seconds = divmod(r, 60)
    print("{minutes:0>2}:{seconds:05.3f}".format(minutes=minutes,
seconds=seconds))

#Cálculo do tempo #Marca
tempo inicial initIndice
= time.time() #Marca
tempo final fimIndice =
time.time()
#Subtrai inicial do final para o total de tempo utilizado na busca
totalIndice = fimIndice - initIndice #Formata o tempo total de
execução da busca formatTime(totalIndice)
```

Criando estrutura de Árvore Binária de Busca

https://algoritmosempython.com.br/cursos/algoritmos-python/estruturas-dados/ arvores/

```
class NodoArvore:     def __init__(self, chave=None,
esquerda=None, direita=None):         self.chave = chave
self.esquerda = esquerda         self.direita = direita

    def __repr__(self):         return '%s <- %s ->
%s' % (self.esquerda and self.esquerda.chave,
                                    self.chave,
self.direita and self.direita.chave)

def em_ordem_binaria(raiz):     if
not raiz:         return

    # Visita filho da esquerda.
em_ordem_binaria(raiz.esquerda)

    # Visita nodo corrente.
    print(raiz.chave),

    # Visita filho da direita.
em_ordem_binaria(raiz.direita)

def insere_binaria(raiz, nodo):
    """Insere um nodo em uma árvore binária de pesquisa."""
```

```
    # Nodo deve ser inserido na raiz.
if raiz is None:          raiz = nodo

    # Nodo deve ser inserido na subárvore direita.
elif raiz.chave < nodo.chave:          if
raiz.direita is None:              raiz.direita =
nodo          else:
insere_binaria(raiz.direita, nodo)

    # Nodo deve ser inserido na subárvore esquerda.
else:          if raiz.esquerda is None:
raiz.esquerda = nodo          else:
insere_binaria(raiz.esquerda, nodo)
def busca_binaria(raiz, chave):
    """Procura por uma chave em uma árvore binária de pesquisa."""
    # Trata o caso em que a chave procurada não está presente.
if raiz is None:          return None

    # A chave procurada está na raiz da árvore.
if raiz.chave == chave:          return raiz

    # A chave procurada é maior que a da raiz.
if raiz.chave < chave:          return
busca_binaria(raiz.direita, chave)

    # A chave procurada é menor que a da raiz.
return busca_binaria(raiz.esquerda, chave)
```

Inserindo índices na árvore e fazendo a Pesquisa na Busca Binária

```
arvorebinaria = NodoArvore(0)
initInsercaoBin = time.time() for i in
range(0,2999,+1):      add =
arquivo.iloc[i,0]      node =
NodoArvore(add)
insere_binaria(arvorebinaria, node)
fimInsercaoBin = time.time()
insercaoBin = fimInsercaoBin - initInsercaoBin
formatTime(insercaoBin) print(insercaoBin)

initOrdena = time.time()
em_ordem_binaria(arvorebinaria)
fimOrdena = time.time()
ordenaBin = fimInsercaoBin - initInsercaoBin

initIndice1 = time.time()
busca_binaria(arvorebinaria,1)
fimIndice1 = time.time()
```

```
totalIndice1 = fimIndice1 - initIndice1
formatTime(totalIndice1)
print(totalIndice1)

initIndice2 = time.time()
busca_binaria(arvorebinaria,5518)
fimIndice2 = time.time()
totalIndice2 = fimIndice2 - initIndice2
formatTime(totalIndice2)
print(totalIndice2)
initIndice3 = time.time()
busca_binaria(arvorebinaria,11037)
fimIndice3 = time.time()
totalIndice3 = fimIndice3 - initIndice3
formatTime(totalIndice3)
print(totalIndice3)

initIndice4 = time.time()
busca_binaria(arvorebinaria,2802)
fimIndice4 = time.time()
totalIndice4 = fimIndice4 - initIndice4
formatTime(totalIndice4)
print(totalIndice4)

initIndice5 = time.time()
busca_binaria(arvorebinaria,8249)
fimIndice5 = time.time()
totalIndice5 = fimIndice5 - initIndice5
formatTime(totalIndice5)
print(totalIndice5)

initIndice6 = time.time()
busca_binaria(arvorebinaria,10964)
fimIndice6 = time.time()
totalIndice6 = fimIndice6 - initIndice6
formatTime(totalIndice6)
print(totalIndice6)

initIndice7 = time.time()
busca_binaria(arvorebinaria,5000)
fimIndice7 = time.time()
totalIndice7 = fimIndice7 - initIndice7
formatTime(totalIndice7)
print(totalIndice7)

initIndice8 = time.time()
busca_binaria(arvorebinaria,3939)
fimIndice8 = time.time()
```

```
totalIndice8 = fimIndice8 - initIndice8
formatTime(totalIndice8)
print(totalIndice8)
```

Criando estrutura de Árvore AVL

https://www.programiz.com/dsa/avl-tree - estrutura

https://cppsecrets.com/users/29798104117109105107974610997107119971109750565757641031099710510846991111 09/Python-program-to-find-an-element-into-AVLtree.php - Busca

```
# AVL tree implementation in Python

import sys
from tkinter.tix import Tree
from setuptools import find_packages

# Create a tree node class
TreeNode(object):     def
__init__(self, key):
self.key = key
self.left = None
self.right = None
self.height = 1 class
AVLTree(object):

    # Function to insert a node
def insert_node(self, root, key):

        # Find the correct location and insert the node
if not root:              return TreeNode(key)          elif
key < root.key:              root.left =
self.insert_node(root.left, key)          else:
root.right = self.insert_node(root.right, key)

        root.height = 1 +
max(self.getHeight(root.left),self.getHeight(root.right))

        # Update the balance factor and balance the tree
balanceFactor = self.getBalance(root)          if
balanceFactor > 1:              if key < root.left.key:
return self.rightRotate(root)              else:
                root.left = self.leftRotate(root.left)
return self.rightRotate(root)

        if balanceFactor < -1:
if key > root.right.key:
                return self.leftRotate(root)
else:                   root.right =
self.rightRotate(root.right)                  return
self.leftRotate(root)          return root
```

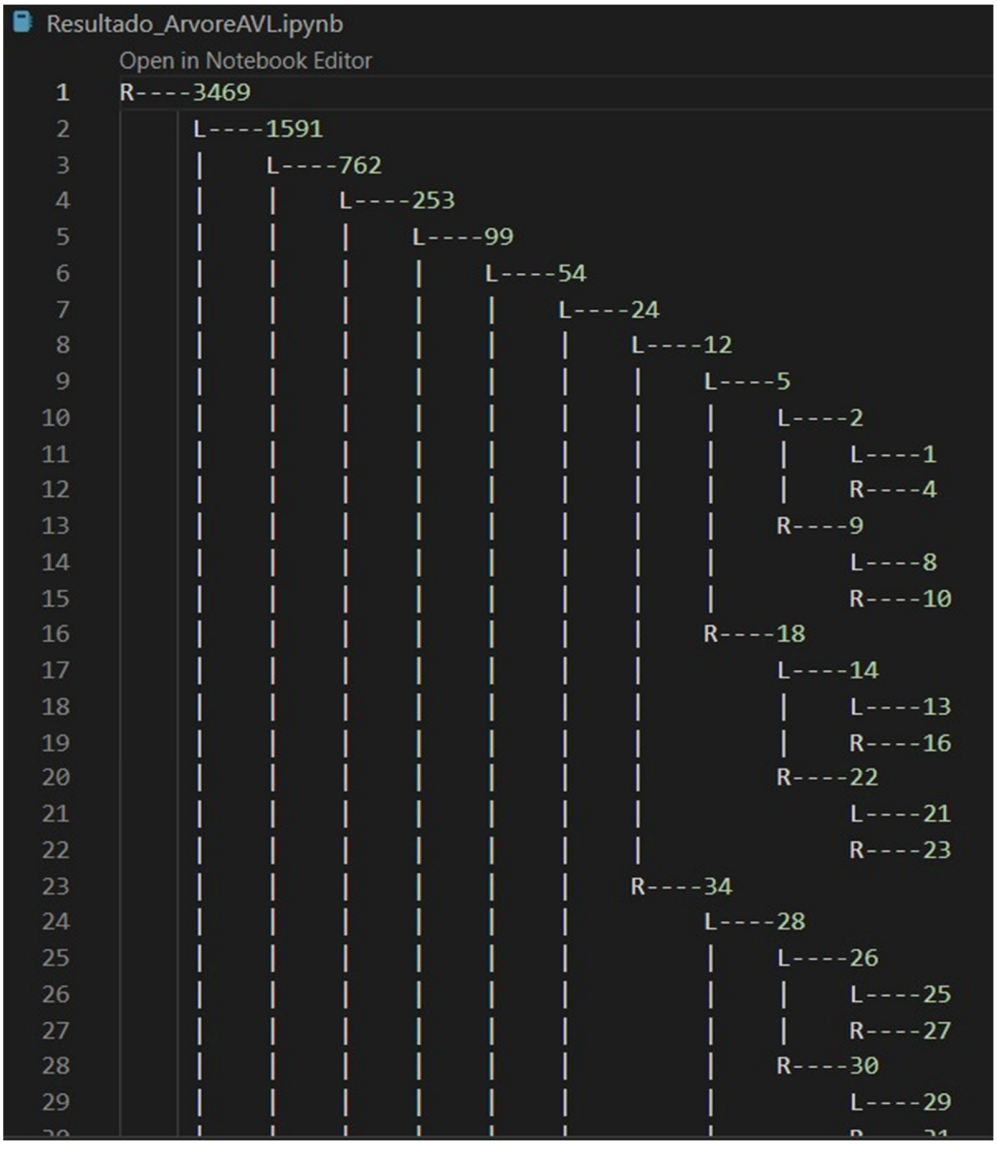
```
    def find_node(self,root,val):
if (root is None):
return False          elif
(root.key == val):
return True          elif(root.key
< val):              return
self.find_node(root.right,val)
return
self.find_node(root.left,val)


    # Function to delete a node
def delete_node(self, root, key):

        # Find the node to be deleted and remove it
if not root:              return root          elif
key < root.key:
            root.left = self.delete_node(root.left, key)
elif key > root.key:              root.right =
self.delete_node(root.right, key)          else:
if root.left is None:                  temp = root.right
root = None                  return temp              elif
root.right is None:                   temp = root.left
root = None                  return temp
            temp = self.getMinValueNode(root.right)
            root.key = temp.key
            root.right = self.delete_node(root.right, temp.key)
if root is None:              return root

        # Update the balance factor of nodes
root.height = 1 + max(self.getHeight(root.left),
self.getHeight(root.right))          balanceFactor =
self.getBalance(root)

        # Balance the tree          if balanceFactor > 1:
if self.getBalance(root.left) >= 0:
return self.rightRotate(root)              else:
root.left = self.leftRotate(root.left)
return self.rightRotate(root)          if balanceFactor <
-1:             if self.getBalance(root.right) <= 0:
return self.leftRotate(root)              else:
root.right = self.rightRotate(root.right)
return self.leftRotate(root)
        return root

    # Function to perform left rotation
def leftRotate(self, z):          y =
```

```
z.right          T2 = y.left
y.left = z
        z.right = T2
        z.height = 1 +
max(self.getHeight(z.left),self.getHeight(z.right))
y.height = 1 +
max(self.getHeight(y.left),self.getHeight(y.right))
return y

    # Function to perform right rotation
def rightRotate(self, z):          y =
z.left          T3 = y.right
y.right = z
        z.left = T3
        z.height = 1 +
max(self.getHeight(z.left),self.getHeight(z.right))
y.height = 1 +
max(self.getHeight(y.left),self.getHeight(y.right))
return y

    # Get the height of the node
def getHeight(self, root):
if not root:              return
0         return root.height

    # Get balance factore of the node     def getBalance(self, root):
if not root:              return 0          return
self.getHeight(root.left) - self.getHeight(root.right)

    def getMinValueNode(self, root):
if root is None or root.left is None:
return root          return
self.getMinValueNode(root.left)

    def preOrder(self, root):
if not root:
return
        print("{0} ".format(root.key), end="")
self.preOrder(root.left)
self.preOrder(root.right)
    # Print the tree     def printHelper(self,
currPtr, indent, last):          if currPtr !=
None:              sys.stdout.write(indent)
if last:                  sys.stdout.write("R----
")                 indent += "     "
else:                   sys.stdout.write("L----")
indent += "|    "
print(currPtr.key)
```

```
            self.printHelper(currPtr.left, indent, False)
self.printHelper(currPtr.right,  indent,  True)
```

Inserindo índices na árvore e fazendo a Pesquisa na Busca Binária

```
arvoreAVL = AVLTree()
root = None
initInsercaoAVL = time.time() for i in
range(0,2999,+1):      add =
arquivo.iloc[i,0]      root =
arvoreAVL.insert_node(root,add)
fimInsercaoAVL = time.time()
insercaoAVL = fimInsercaoAVL - initInsercaoAVL
formatTime(insercaoAVL) print(insercaoAVL)

arvoreAVL.printHelper(root, "", True)
```



```
initIndice1 = time.time()
print(arvoreAVL.find_node(root,1))
fimIndice1 = time.time() totalIndice1 =
fimIndice1 - initIndice1
formatTime(totalIndice1)
print(totalIndice1)

initIndice2 = time.time()
print(arvoreAVL.find_node(root,5518))
fimIndice2 = time.time() totalIndice2 =
fimIndice2 - initIndice2
formatTime(totalIndice2)
print(totalIndice2)

initIndice3 = time.time()
print(arvoreAVL.find_node(root,11037))
fimIndice3 = time.time() totalIndice3 =
fimIndice3 - initIndice3
formatTime(totalIndice3)
print(totalIndice3)
initIndice4 = time.time()
print(arvoreAVL.find_node(root,2802))
fimIndice4 = time.time() totalIndice4 =
fimIndice4 - initIndice4
formatTime(totalIndice4)
print(totalIndice4)

initIndice5 = time.time()
print(arvoreAVL.find_node(root,8249))
fimIndice5 = time.time() totalIndice5 =
fimIndice5 - initIndice5
formatTime(totalIndice5)
print(totalIndice5)

initIndice6 = time.time()
print(arvoreAVL.find_node(root,10964))
fimIndice6 = time.time() totalIndice6 =
fimIndice6 - initIndice6
formatTime(totalIndice6)
print(totalIndice6)

initIndice7 = time.time()
print(arvoreAVL.find_node(root,5000))
fimIndice7 = time.time() totalIndice7 =
fimIndice7 - initIndice7
formatTime(totalIndice7)
print(totalIndice7)

initIndice8 = time.time()
print(arvoreAVL.find_node(root,3939))
```

```
fimIndice8 = time.time() totalIndice8 =
fimIndice8 - initIndice8
formatTime(totalIndice8)
print(totalIndice8)
```