Relatório Projeto 2 V1.1 AED 2023/2024

Nome: Eduardo Luís Pereira Marques Nº Estudante: 2022231584
PL (inscrição): PL1 Email: eduardo.marques07g@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.

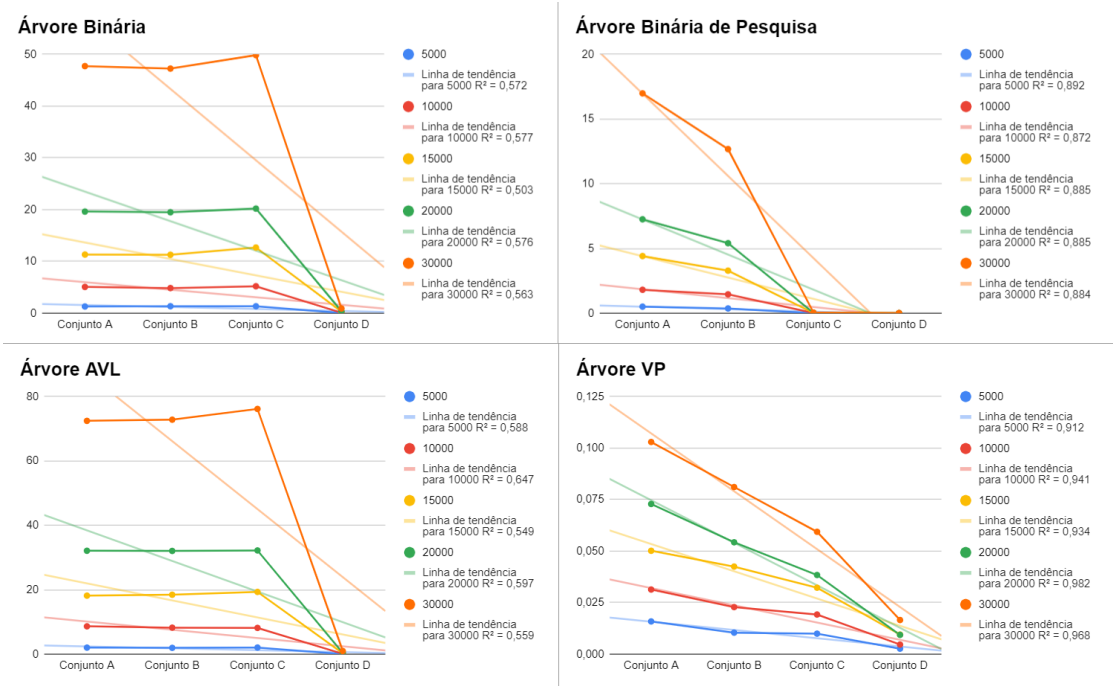
1. Planeamento

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

2. Recolha de Resultados

Árvore Binária	5000	10000	15000	20000	30000
Conjunto A	1,29439	5,076	11,32872	19,62832	47,72033
Conjunto B	1,34793	4,85262	11,2914	19,49653	47,2461
Conjunto C	1,339	5,20634	12,67625	20,20407	49,83783
Conjunto D	0,02905	0,09993	0,28739	0,38498	0,91398
Árvore Binária de Pesquisa	5000	10000	15000	20000	30000
Conjunto A	0,50555	1,81043	4,4151	7,25418	16,98337
Conjunto B	0,36717	1,45627	3,28571	5,40863	12,67751
Conjunto C	0,00602	0,01374	0,01664	0,02459	0,04114
Conjunto D	0,00099	0,00197	0,00495	0,0041	0,00865
Árvore AVL	5000	10000	15000	20000	30000
Conjunto A	2,01428	8,63327	18,15738	32,10406	72,45432
Conjunto B	1,94849	8,22034	18,44368	32,04974	72,816683
Conjunto C	2,0415	8,14383	19,31524	32,20722	76,1539
Conjunto D	0,03047	0,11306	0,23896	0,37183	0,96098
Árvore VP	5000	10000	15000	20000	30000
Conjunto A	0,01586	0,03134	0,05016	0,07287	0,10295
Conjunto B	0,01041	0,02282	0,04248	0,05427	0,08109
Conjunto C	0,00993	0,01922	0,03224	0,0384	0,05937
Conjunto D	0,00265	0,00464	0,00949	0,00934	0,01655

3. Visualização de Resultados



4. Conclusões

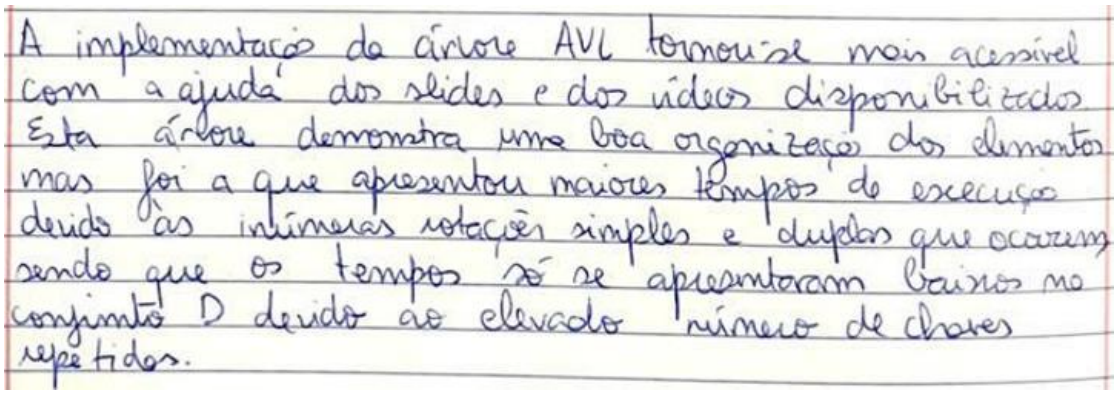
4.1 Tarefa 1

A implementação da árvore binária foi bastante fácil uma vez que os elementos são inseridos aleatoriamente. Os tempos de inserção dos dados revelam um crescimento notório com o aumento do número de chaves, o que seria de esperar. Nesta árvore, os conjuntos não têm grande interferência nos tempos já que os elementos são inseridos aleatoriamente, algo que é provado pelo baixo valor do r^2 - quadrado.

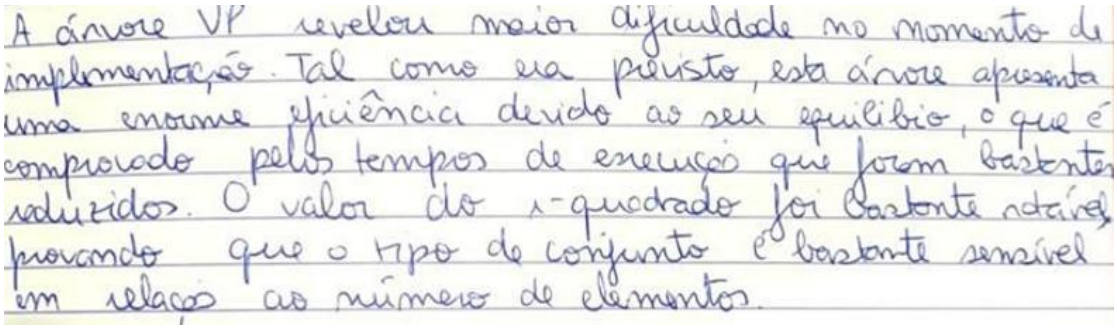
4.2 Tarefa 2

A árvore binária de pesquisa também foi resolvida com bastante facilidade. A particularidade desta é que os elementos maiores são inseridos à direita e os menores à esquerda, o que é garantido com apenas dois ifs. Os tempos de execução também aumentam com o aumento do tamanho da lista e a relação entre esse tamanho e o conjunto em questão é muito grande (elevado valor do r^2 - quadrado)

4.3 Tarefa 3



4.4 Tarefa 4



Anexo A - Código

```python
import random
import time

class arvoreBinaria:
    def __init__(self,value):
        self.value = value
        self.left = None
        self.right = None

    def duplicado(self, value):
        if self.value == value:
            return True
        if self.left is not None and self.left.duplicado(value):
            return True
        if self.right is not None and self.right.duplicado(value):
            return True
        return None

    def inserir(self,value):
        if self.value is None:
            self.value = value
        elif self.duplicado(value) is None:
            if self.left is None:
                self.left = arvoreBinaria(value)
            elif self.right is None:
                self.right = arvoreBinaria(value)
            else:
                boolean = random.choice([True, False])
                if boolean:
                    self.left.inserir(value)
                else:
                    self.right.inserir(value)

    def imprimir(self, espaco):
        if self is None:
            return
        espaco += 10
        if self.right:
            self.right.imprimir(espaco)
        print("\n" + " " * espaco + str(self.value))
        if self.left:
            self.left.imprimir(espaco)
```

```python
class arvorePesquisa:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None

    def inserir(self, value):
        if self.value is None:
            self.value = value
            return
        current = self
        while current:
            if value < current.value:
                if current.left is None:
                    current.left = arvorePesquisa(value)
                    break
                else:
                    current = current.left
            elif value > current.value:
                if current.right is None:
                    current.right = arvorePesquisa(value)
                    break
                else:
                    current = current.right
            else:
                break

    def imprimir(self, espaco):
        if self is None:
            return
        espaco += 10
        if self.right:
            self.right.imprimir(espaco)
        print("\n" + " " * espaco + str(self.value))
        if self.left:
            self.left.imprimir(espaco)
```

```python
class NoAVL:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None

class arvoreAVL:
    def __init__(self):
        self.no = None
        self.rotates = 0

    def inserir(self, valor):
        self.no = self.inserirRec(self.no, valor)

    def inserirRec(self,new,value):
        if new is None:
            new = NoAVL(value)
        elif value < new.value:
            if new.left is None:
                new.left = NoAVL(value)
            else:
                new.left = self.inserirRec(new.left, value)
        elif value > new.value:
            if new.right is None:
                new.right = NoAVL(value)
            else:
                new.right = self.inserirRec(new.right, value)
        return self.corrigeEquilibrio(new)

    def corrigeEquilibrio(self, node):
        altura = self.equilibrio(node)
        if altura > 1:
            if self.equilibrio(node.left) > 0:
                node = self.rotacaoDireita(node)
            else:
                node = self.rotacaoEsquerdaDireita(node)
        elif altura < -1:
            if self.equilibrio(node.right) < 0:
                node = self.rotacaoEsquerda(node)
            else:
                node = self.rotacaoDireitaEsquerda(node)
        return node
```

```
def equilibrio(self, node):
    alturaEsquerda = 0
    alturaDireita = 0
    if node.left:
        alturaEsquerda = self.altura(node.left)
    if node.right:
        alturaDireita = self.altura(node.right)
    difAltura = alturaEsquerda - alturaDireita
    return difAltura

def altura(self, node):
    alturaEsquerda = 0
    alturaDireita = 0
    if node.left:
        alturaEsquerda = self.altura(node.left)
    if node.right:
        alturaDireita = self.altura(node.right)
    if alturaDireita >= alturaEsquerda:
        return alturaDireita + 1
    else:
        return alturaEsquerda + 1

def rotacaoEsquerda(self, node):
    tmp = node.right
    node.right = tmp.left
    tmp.left = node
    self.rotates += 1
    return tmp

def rotacaoDireita(self, node):
    tmp = node.left
    node.left = tmp.right
    tmp.right = node
    self.rotates += 1
    return tmp

def rotacaoEsquerdaDireita(self, node):
    node.left = self.rotacaoEsquerda(node.left)
    self.rotates += 2
    return self.rotacaoDireita(node)
```

```
def rotacaoDireitaEsquerda(self, node):
    node.direita = self.rotacaoDireita(node.right)
    self.rotates += 2
    return self.rotacaoEsquerda(node)

def imprimir(self):
    if self.no:
        self.imprimirRec(self.no, 0)

def imprimirRec(self, no, espaco):
    if no is None:
        return
    espaco += 10
    if no.right:
        self.imprimirRec(no.right, espaco)
    print("\n" + " " * espaco + str(no.value))
    if no.left:
        self.imprimirRec(no.left, espaco)
```

```
class NoVP():
    def __init__(self,value):
        self.value = value
        self.parent = None
        self.left = None
        self.right = None
        self.color = 1

class arvoreVP():
    def __init__(self):
        self.NULL = NoVP(0)
        self.NULL.color = 0
        self.root = self.NULL
        self.rotates = 0

    def inserir(self,value):
        new = NoVP(value)
        new.parent = None
        new.value = value
        new.left = self.NULL
        new.right = self.NULL
        new.color = 'red'
        parent = None
        atual = self.root
        while atual != self.NULL:
            parent = atual
            if new.value < atual.value:
                atual = atual.left
            elif new.value > atual.value:
                atual = atual.right
            else:
                return
        new.parent = parent
        if parent == None:
            self.root = new
        elif new.value < parent.value:
            parent.left = new
        else:
            parent.right = new
        if new.parent == None:
            new.color = 'black'
            return
        self.corrige(new)
```

```
def rotacaoEsquerda(self,node):
    right_child = node.right
    node.right = right_child.left
    if right_child.left != self.NULL :
        right_child.left.parent = node
    right_child.parent = node.parent
    if node.parent == None :
        self.root = right_child
    elif node == node.parent.left :
        node.parent.left = right_child
    else :
        node.parent.right = right_child
    right_child.left = node
    node.parent = right_child
    self.rotates += 1

def rotacaoDireita(self,node):
    left_child = node.left
    node.left = left_child.right
    if left_child.right != self.NULL :
        left_child.right.parent = node

    left_child.parent = node.parent
    if node.parent == None :
        self.root = left_child
    elif node == node.parent.right :
        node.parent.right = left_child
    else :
        node.parent.left = left_child
    left_child.right = node
    node.parent = left_child
    self.rotates += 1
```

```
def corrige(self, new):
    while new.parent.color == 'red':
        if new.parent == new.parent.parent.right:
            aux = new.parent.parent.left
            if aux.color == 'red':
                aux.color = 'black'
                new.parent.color = 'black'
                new.parent.parent.color = 'red'
                new = new.parent.parent
            else:
                if new == new.parent.left:
                    new = new.parent
                    self.rotacaoDireita(new)
                new.parent.color = 'black'
                new.parent.parent.color = 'red'
                self.rotacaoEsquerda(new.parent.parent)
        else:
            aux = new.parent.parent.right
            if aux.color == 'red':
                aux.color = 'black'
                new.parent.color = 'black'
                new.parent.parent.color = 'red'
                new = new.parent.parent
            else:
                if new == new.parent.right:
                    new = new.parent
                    self.rotacaoEsquerda(new)
                new.parent.color = 'black'
                new.parent.parent.color = 'red'
                self.rotacaoDireita(new.parent.parent)
        if new == self.root:
            break
    self.root.color = 'black'

def imprimir(self):
    if self.root:
        self.imprimirRec(self.root, 0)

def imprimirRec(self, no, espaco):
    if no is None or no == self.NULL:
        return
    espaco += 10
    if no.right:
        self.imprimirRec(no.right, espaco)
    print("\n" + " " * espaco + str(no.value))
    if no.left:
        self.imprimirRec(no.left, espaco)
```

```python
def calcularA(arvore,tamanho):
    conjuntoA = [random.randint(0, tamanho + 1) for i in range(tamanho)]
    conjuntoA.sort()
    inicio = time.time()
    for valor in conjuntoA:
        arvore.inserir(valor)
    fim = time.time()
    return format(fim - inicio, ".5f")

def calcularB(arvore,tamanho):
    conjuntoB = [random.randint(0, tamanho + 1) for i in range(tamanho)]
    conjuntoB.sort(reverse = True)
    inicio = time.time()
    for valor in conjuntoB:
        arvore.inserir(valor)
    fim = time.time()
    return format(fim - inicio, ".5f")

def calcularC(arvore,tamanho):
    conjuntoC = [random.randint(0, tamanho + 1) for i in range(tamanho)]
    inicio = time.time()
    for valor in conjuntoC:
        arvore.inserir(valor)
    fim = time.time()
    return format(fim - inicio, ".5f")

def calcularD(arvore,tamanho):
    conjuntoD = []
    repetidos = int(0.9 * tamanho)
    resto = tamanho - repetidos
    num = random.randint(1,tamanho + 1)
    for i in range (repetidos):
        conjuntoD.append(num)
    for i in range(resto):
        conjuntoD.append(random.randint(1,tamanho + 1))
    inicio = time.time()
    for valor in conjuntoD:
        arvore.inserir(valor)
    fim = time.time()
    return format(fim - inicio, ".5f")
```

```python
class main():
    tamanho = 1000
    print("\nNÚMERO DE CHAVES:",tamanho,"\n")
    print("- ARVORE BINÁRIA -")
    arvoreBinariaA = arvoreBinaria(None)
    tempo = calcularA(arvoreBinariaA,tamanho)
    print("Conjunto A - Tempo decorrido:",tempo)
    arvoreBinariaB = arvoreBinaria(None)
    tempo = calcularB(arvoreBinariaB,tamanho)
    print("Conjunto B - Tempo decorrido:",tempo)
    arvoreBinariaC = arvoreBinaria(None)
    tempo = calcularC(arvoreBinariaC,tamanho)
    print("Conjunto C - Tempo decorrido:",tempo)
    arvoreBinariaD = arvoreBinaria(None)
    calcularD(arvoreBinariaD,tamanho)
    print("Conjunto D - Tempo decorrido:",tempo)
    print("\n-------------------------------------\n")
    print("- ARVORE PESQUISA -")
    arvorePesquisaA = arvorePesquisa(None)
    tempo = calcularA(arvorePesquisaA,tamanho)
    print("Conjunto A - Tempo decorrido:",tempo)
    arvorePesquisaB = arvorePesquisa(None)
    tempo = calcularB(arvorePesquisaB,tamanho)
    print("Conjunto B - Tempo decorrido:",tempo)
    arvorePesquisaC = arvorePesquisa(None)
    tempo = calcularC(arvorePesquisaC,tamanho)
    print("Conjunto C - Tempo decorrido:",tempo)
    arvorePesquisaD = arvorePesquisa(None)
    tempo = calcularD(arvorePesquisaD,tamanho)
    print("Conjunto D - Tempo decorrido:",tempo)
    print("\n-------------------------------------\n")
    print("- ARVORE AVL -")
    arvoreAVLA = arvoreAVL()
    tempo = calcularA(arvoreAVLA,tamanho)
    print("Conjunto A - Tempo decorrido:",tempo)
    print("  - Rotações simples:",arvoreAVLA.rotates)
    arvoreAVLB = arvoreAVL()
    tempo = calcularB(arvoreAVLB,tamanho)
    print("Conjunto B - Tempo decorrido:",tempo)
    print("  - Rotações simples:",arvoreAVLB.rotates)
    arvoreAVLC = arvoreAVL()
    tempo = calcularC(arvoreAVLC,tamanho)
```

```
print("Conjunto C - Tempo decorrido:",tempo)
print("  - Rotações simples:",arvoreAVLC.rotates)
arvoreAVLD = arvoreAVL()
tempo = calcularD(arvoreAVLD,tamanho)
print("Conjunto D - Tempo decorrido:",tempo)
print("  - Rotações simples:",arvoreAVLD.rotates)
print("\n-------------------------------------\n")
print("- ARVORE VP -")
arvoreVPA = arvoreVP()
tempo = calcularA(arvoreVPA,tamanho)
print("Conjunto A - Tempo decorrido:",tempo)
print("  - Rotações simples:",arvoreVPA.rotates)
arvoreVPB = arvoreVP()
tempo = calcularB(arvoreVPB,tamanho)
print("Conjunto B - Tempo decorrido:",tempo)
print("  - Rotações simples:",arvoreVPB.rotates)
arvoreVPC = arvoreVP()
tempo = calcularC(arvoreVPC,tamanho)
print("Conjunto C - Tempo decorrido:",tempo)
print("  - Rotações simples:",arvoreVPC.rotates)
arvoreVPD = arvoreVP()
tempo = calcularD(arvoreVPD,tamanho)
print("Conjunto D - Tempo decorrido:",tempo)
print("  - Rotações simples:",arvoreVPD.rotates)
print('\n')

exemploBin = arvoreBinaria(None)
exemploPesq = arvorePesquisa(None)
exemploAVL = arvoreAVL()
exemploVP = arvoreVP()
listaTeste = [5,8,23,56,12,82,5,2,6,2,5,16,23,12,34,12,53]
for chave in listaTeste:
    exemploBin.inserir(chave)
    exemploPesq.inserir(chave)
    exemploAVL.inserir(chave)
    exemploVP.inserir(chave)

print('\n- Exemplos - \n')
exemploBin.imprimir(0)
print('\n-------------------------------------------------------------------------------\n')
exemploPesq.imprimir(0)
print('\n-------------------------------------------------------------------------------\n')
exemploAVL.imprimir()
print('\n-------------------------------------------------------------------------------\n')
exemploVP.imprimir()
```

Anexo B - Código de Autor

Desenvolvimento dos códigos da árvore binária e da árvore binária de pesquisa. Para além disso, a árvore AVL com ajuda dos slides e dos vídeos disponibilizados e um pouco da árvore VP. A implementação das funções para criação dos conjuntos e para o cálculo dos tempos de execução e desenvolvimento do *main*.

Anexo C - Referências

Grande parte da implementação da árvore VP com ajuda encontrada no seguinde website: https://favtutor.com/blogs/red-black-tree-python

Para além disso, as funções de imprimir as árvores. Website: https://www.geeksforgeeks.org/print-binary-tree-2-dimensions/