| | |
|---|---|
| 1 2 9 0 FACULDADE DE CIÊNCIAS E TECNOLOGIA UNIVERSIDADE D COIMBRA DEPARTAMENTO DE ENGENHARIA INFORMÁTICA | **Trabalho no 3 v3.1 – Lista de Referências**<br>**Algoritmos e Estruturas de Dados**<br>**2019/2020 – 2º Semestre**<br><br>Upload: (link a disponibilizar no infoestudante)<br>Data Limite: 17/Abril/2020, 18h00<br>Data Limite (PL1 e PL7): 24/Abril/2020, 18h00 |

**O RELATÓRIO E LISTAGEM DO CÓDIGO DESENVOLVIDO DEVEM SER SUBMETIDOS NUM ÚNICO DOCUMENTO PDF**

Nome: _Dário Filipe Torres Félix_ nº _2018275530_ PL: _PL2_

Nº de horas de trabalho: Aulas Práticas de Laboratório: _10_ H Fora de Sala de Aula: _12_ H

| CLASSIFICAÇÃO:<br>(A Preencher pelo Docente) |
|---|

**Análise Empírica de Complexidade**

Tarefa preparatória para o desenvolvimento desta ficha:

- Fazer o download dos 4 textos disponibilizados.
- Caracterizar cada texto em termos de número de palavras distintas, evidência de alguma ordem pré-estabelecida para as palavras, extensão do texto. Considere essa caracterização quando relevante na análise qualitativa que lhe é pedida mais adiante.
- Calcular na tabela abaixo os tempos[1] para as três/quatro versões do trabalho relativos às operações indicadas (a tarefa B é opcional).
- Analisar o número de rotações que vão ocorrer no carregamento do texto A (segunda tabela).

[1] **Usar o tempo médio de 20 execuções do respetivo comando**

**TEXTO A**

Núm. palavras distintas: _2 920_.

Algum ordenamento? Caracterize? _Numerosas palavras distintas distribuídas aleatoriamente pelo texto_.

Núm. total de palavras: _10 000_.

**TEXTO B**

Núm. palavras distintas: _2 922_.

Algum ordenamento? Caracterize? _Numerosas palavras distintas em ordem alfabética no texto_.

Núm. total de palavras: _10 000_.

**TEXTO C**

Núm. palavras distintas: _2 922_.

Algum ordenamento? Caracterize? _Numerosas palavras distintas em ordem alfabética-inversa no texto_.

Núm. total de palavras: _10 038_.

**TEXTO D**

Núm. palavras distintas: _84_.

Algum ordenamento? Caracterize? _Poucas palavras distintas distribuídas aleatoriamente pelo texto_.

Núm. total de palavras: _10 000_.

| # | Tarefa<br>Tempos em [ Milissegundo - ms ]<br>Operação | A0 | A1 | A2 | B |
|---|---|---|---|---|---|
| 1 | Carregamento **(Texto A)** | 436.167 | 234.865 | 176.083 | 349.833 |
| 2 | Carregamento **(Texto B)** | 647.908 | 207.742 | 209.454 | 105.174 |
| 3 | Carregamento **(Texto C)** | 165.922 | 243.894 | 242.207 | 88.298 |
| 4 | 50 chamadas do comando "LINHAS" com diferentes palavras do texto (escolha aleatória das palavras) **(Texto A)** | 0.771 | 0.778 | 0.802 | 1.622 |
| 5 | 50 chamadas do comando "ASSOC" com diferentes palavras do texto (escolha aleatória das palavras) e na linha definida aleatoriamente, dentro dos limites do texto **(Texto A)** | 0.653 | 1.089 | 0.943 | 1.647 |
| 6 | 50 chamadas do comando "LINHAS" usando somente 10 palavras (T**exto D)** (escolha aleatória das palavras) | 11.168 | 10.672 | 10.388 | 11.226 |
| 7<br>Estrutura de dados auxiliar usada em cada uma das abordagens? | | Possuem em todas a Ordenação por Inserção e Pesquisa Binária para a lista com os números das linhas em cada palavra. Na A0, também é aplicado na estrutura principal (lista/array das palavras). | | | |

| # | Tarefa<br>Número Total de Rotações Simples<br>Operação | A1 | A2 | B |
|---|---|---|---|---|
| 8 | Carregamento do texto D | 56 | 44 | 96 774 |

**Reflexão sucinta sobre os resultados obtidos**

*(Formato de referência: Helvetica 10pt; texto para além do número de linhas não é considerado e desvaloriza o relatório)*

1. Comente os resultados obtidos na tarefa A1 para os textos A, B e C.

Apesar de os valores de carregamento serem muito próximos, para os Textos B e C era esperado valores altos devido a sobrecarga permanente num dos lados (mais rotações por inserção). Assim, prova a alta instabilidade (equilíbrio) mesmo para casos do tipo Texto A.

2. Comente os resultados obtidos nas tarefas A0 a A2 e a B (opcional) para o texto A

Apesar de todas as tarefas terem complexidade O(log2(N)), no carregamento, A2 tem o melhor valor, enquanto A0 e B (devido ao splaying) têm o pior desempenho, A2 supera A1, pois precisa menos rotações para estar equilibrada. Nos comandos, Arvore B é a pior devido ao splaying num contexto de palavras aleatórias, as restantes apresentam valores aproximados - [arvore] pesquisa binária.

3. Compare os resultados obtidos nas operações 5 e 6 e relacione com as opções tomadas em termos de estrutura auxiliar de dados. Se achar que não há relação justifique. Comente os resultados obtidos para a tarefa B com estas duas operações.

Esperava-se que as operações de comando “ASSOC” (envolve pesquisar num array) fossem mais demoradas que o comando “LINHAS”, mas verifica-se o contrário num fator 10x. De facto, a Ordenação na Inserção e a Pesquisa Binária do “array de ocorrências” tem complexidade O(log2(N)) tornando o comando “ASSOC” num simples check rápido. O comando “LINHAS” precisa de converter cada elemento do array numa única string e imprimi-lo, ou seja, 2 x O(N). Outro fator é que a comparação envolve o comando “ASSOC” no contexto do Texto A (as palavras ocorrem em menos linhas) e que o comando “LINHAS” está no contexto do Texto D (as palavras repetem-se e ocorrem significativamente em mais linhas). Esperava-se que o comando “LINHAS” usando somente 10 palavras tivesse um valor baixo em B, mas, além das razões anteriores, também poderá ter sido camuflado pela sobrecarga das rotações do splaying.

4. Analise e comente os resultados da operação 8.

A2 precisa de menos rotações para ser considerada equilibrada do que a A1. No B e devido ao splaying sempre que se insere um novo elemento na arvore ou atualize-o há rotações que aumentam em número á medida que a arvore cresça (pela inserção) para que um nó nas folhas chegue á raiz.

Bom trabalho, os Docentes da Disciplina,

Carlos L Bento e Catarina Silva

```python
# v2 - Melhoramentos: Retirei "in" em "x in array"; implementei pesquisa binaria e ordenaçao por inserçao; print_array; etc.
# 2 ARRAYS: 1 para palavras, 1 para guardar o numero em que ocorre
# *** VERSAO RELATORIO ***


# #### BIBLIOTECAS ####
import sys
import time
import msvcrt
from io import StringIO


# #### CONSTANTES ####
CMD_IN_LINHAS = "LINHAS"
CMD_OUT_NULO = "-1"
CMD_IN_ASSOC = "ASSOC"
CMD_OUT_NAOENCONTRADA = "NAO ENCONTRADA."
CMD_OUT_ENCONTRADA = "ENCONTRADA."
CMD_IN_TERMINADO = "TCHAU\n"
CMD_IN_TERMINADO2 = "TCHAU"
CMD_IN_TEXTO = "TEXTO\n"
CMD_IN_FIM = "FIM.\n"
CMD_OUT_GUARDADO = "GUARDADO."


# #### VARS GLOBAIS ####
acumula_linhas = 0
acumula_assoc = 0


# #### FUNCOES ####
def main():
    # ### FUNCAO ### Funcao Principal
    textos_relatorio = ["A", "B", "C", "D"]
    global acumula_linhas
    global acumula_assoc
    for n_texto in textos_relatorio:
        print("# # # # # # TEXTO " + n_texto + " # # # # # #")
        acumula_texto = 0
        acumula_linhas = 0
        acumula_assoc = 0
        for i in range(20):
            array_palavras = []  # Varias Palavras
            array_ocorrencias = [[]]
            while msvcrt.kbhit():   # Clean stdin Windows
                msvcrt.getch()
            if n_texto == "A" or n_texto == "D":
                nome_fich = "./StdinsCalculaTempos/StdinTexto" + n_texto + "RelatorioN" + str(i) + ".txt"
            else:
                nome_fich = "./StdinsCalculaTempos/StdinTexto" + n_texto + "RelatorioN" + str(0) + ".txt"
            print("########" + nome_fich + "########")
            my_file = open(nome_fich, "r")
            my_stdin = my_file.read()
            my_file.close()
            sys.stdin = StringIO(my_stdin)
            if sys.stdin.readline() == CMD_IN_TEXTO:
                start_texto = time.time()
                array_palavras, array_ocorrencias = input_texto(array_palavras, array_ocorrencias)
                end_texto = time.time()
                tempo_texto = end_texto - start_texto
            else:
                sys.exit("Erro - Sem Comando Incial: " + CMD_IN_TEXTO)
            input_cmd(array_palavras, array_ocorrencias)
            acumula_texto = acumula_texto + tempo_texto
            print("########################################\n")
        acumula_texto = acumula_texto / 20.0
        acumula_linhas = acumula_linhas / 20.0
        acumula_assoc = acumula_assoc / 20.0
        print("* * * * * Tempo Medio em Segundos Input TEXTO = " + str(acumula_texto) + " * * * * *")
        print("* * * * * Tempo Medio em Segundos Input CMDs - LINHAS = " + str(acumula_linhas) + " * * * * *")
        print("* * * * * Tempo Medio em Segundos Input CMDs - ASSOC = " + str(acumula_assoc) + " * * * * *")
        print("# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #\n\n")
    return 0
```

```python
def input_texto(array_palavras, array_ocorrencias):
    # ### FUNCAO ### Le e manipula o texto do stdin ate CMD_IN_FIM
    count = 0
    conta_palavras_diferente = 0
    conta_palavras = 0
    for linha in sys.stdin:
        if count == 0 and linha == "":
            sys.exit("Erro - Sem Texto para input")

        if linha == CMD_IN_FIM:
            break
        palavra = ""
        for ch in linha:
            if ch == '\n':
                if len(palavra) > 0:
                    conta_palavras = conta_palavras + 1
                    palavra = palavra.lower()
                    indice_palavra = pesquisa_binaria(array_palavras, palavra)
                    if not (indice_palavra == -1):
                        if not (count == array_ocorrencias[indice_palavra][-1]):
                            array_ocorrencias[indice_palavra].append(count)
                    else:
                        array_palavras, array_ocorrencias = inserir_array(array_palavras, palavra, array_ocorrencias, count)
                        conta_palavras_diferente = conta_palavras_diferente + 1
                palavra = ""
            elif ch == ' ' or ch == '.' or ch == ',' or ch == ';' or ch == '(' or ch == ')':
                if len(palavra) > 0:
                    conta_palavras = conta_palavras + 1
                    palavra = palavra.lower()
                    indice_palavra = pesquisa_binaria(array_palavras, palavra)
                    if not (indice_palavra == -1):
                        if not (count == array_ocorrencias[indice_palavra][-1]):
                            array_ocorrencias[indice_palavra].append(count)
                    else:
                        array_palavras, array_ocorrencias = inserir_array(array_palavras, palavra, array_ocorrencias, count)
                        conta_palavras_diferente = conta_palavras_diferente + 1
                indice_palavra = pesquisa_binaria(array_palavras, ch)
                if not (indice_palavra == -1):
                    if not (count == array_ocorrencias[indice_palavra][-1]):
                        array_ocorrencias[indice_palavra].append(count)
                else:
                    array_palavras, array_ocorrencias = inserir_array(array_palavras, ch, array_ocorrencias, count)
                palavra = ""
            else:
                palavra = palavra + ch
        count += 1
    print(CMD_OUT_GUARDADO)
    print("* * * * * Palavras Total no Texto = " + str(conta_palavras) + " * * * * *")
    print("* * * * * Palavras Diferentes no Texto = " + str(conta_palavras_diferente) + " * * * * *")
    return array_palavras, array_ocorrencias
```

```python
def input_cmd(array_palavras, array_ocorrencias):
    # ### FUNCAO ### Le, executa e escreve no stdout os comandos no stdin, ate CMD_IN_TERMINADO
    fator_assoc = 0
    fator_linhas = 0
    inicio_assoc = 0
    inicio_linhas = 0
    global acumula_assoc
    global acumula_linhas
    for linha in sys.stdin:
        if linha == CMD_IN_TERMINADO2:
            break
        elif linha == CMD_IN_TERMINADO:
            break
        elif linha == "":
            break
        elif (CMD_IN_LINHAS in linha) and (linha.index(CMD_IN_LINHAS) == 0):
            if fator_assoc == 1:
                acumula_assoc = acumula_assoc + (time.time() - inicio_assoc)
                fator_assoc = -1
            if fator_linhas == 0:
                inicio_linhas = time.time()
                fator_linhas = 1
            palavra = linha[len(CMD_IN_LINHAS)+1:len(linha)-1]
            palavra = palavra.lower()

            indice_palavra = pesquisa_binaria(array_palavras, palavra)
            if not (indice_palavra == -1):
                print(print_array(array_ocorrencias[indice_palavra]))
            else:
                print(CMD_OUT_NULO)
        elif (CMD_IN_ASSOC in linha) and (linha.index(CMD_IN_ASSOC) == 0):
            if fator_linhas == 1:
                acumula_linhas = acumula_linhas + (time.time() - inicio_linhas)
                fator_linhas = -1
            if fator_assoc == 0:
                inicio_assoc = time.time()
                fator_assoc = 1
            palavras = linha.split(' ')
            palavras[2] = (palavras[2])[:len(palavras[2])-1]
            palavras[1] = palavras[1].lower()

            indice_palavra = pesquisa_binaria(array_palavras, palavras[1])
            if not (indice_palavra == -1):
                if not (pesquisa_binaria(array_ocorrencias[indice_palavra], int(palavras[2])) == -1):   # Confirmar Alteracao
                    print(CMD_OUT_ENCONTRADA)
                else:
                    print(CMD_OUT_NAOENCONTRADA)
            else:
                print(CMD_OUT_NAOENCONTRADA)
        else:
            sys.exit("Erro - Interpretacao dos comandos pos-texto")
    if fator_linhas == 1:
        acumula_linhas = acumula_linhas + (time.time() - inicio_linhas)
    if fator_assoc == 1:
        acumula_assoc = acumula_assoc + (time.time() - inicio_assoc)
    return 0
```

```python
def pesquisa_binaria(array, valor):
    # ### FUNCAO ### Pesquisa Binaria Classica num Array/Lista, input array e valor, return indice ou -1 se nao existir
    inicio = 0
    fim = len(array)-1
    if fim == -1:
        return -1
    while inicio <= fim:
        meio = inicio + (fim - inicio) // 2    # Divisao Real, Arredonda para baixo
        if array[meio] == valor:       # Valor esta no meio?
            return meio
        elif array[meio] < valor:        # Se valor e maior que o meio, ignora metade inferior
            inicio = meio + 1
        else:                          # Se for menor que o meio, ignora metade superior
            fim = meio - 1
    return -1                            # Nao existe


def print_array(array):
    # ### FUNCAO ### Transforma os dados num array numa string com espacos
    string = ""
    for num in array:
        string = string + " " + str(num)
    return string[1:]


def inserir_array(array_palavras, palavra, array_ocorrencias, count):
    # ### FUNCAO ### Inserir palavra e n-linha pela primeira vez nos arrays, ordenacao por insercao
    index = len(array_palavras)
    if index == 0:   # Se primeira palavra no array
        array_palavras.append(palavra)
        array_ocorrencias[0].append(count)
        return array_palavras, array_ocorrencias
    for i in range(len(array_palavras)):    # Procura pela posicao
        if i == index:
            break
        if array_palavras[i] > palavra:   # Ordenacao por insercao
            index = i
            break
    array_palavras = array_palavras[:index] + [palavra] + array_palavras[index:]    # Inserir
    array_ocorrencias = array_ocorrencias[:index] + [[count]] + array_ocorrencias[index:]
    return array_palavras, array_ocorrencias


if __name__ == '__main__':
    # ### START ###
    main()
```

```python
# v3 - Melhoramentos: Retirei "in" em "x in array"; implementei pesquisa binaria; print_array; etc.
# v3 Abordagem Ate as folhas, depois de Baixo-para-Cima, Recursiva
# pai.direcao = return no filho da recursividade
# *** VERSAO RELATORIO ***


# #### BIBLIOTECAS ####
import sys
import time
import msvcrt
from io import StringIO


# #### CONSTANTES ####
CMD_IN_LINHAS = "LINHAS"
CMD_OUT_NULO = "-1"
CMD_IN_ASSOC = "ASSOC"
CMD_OUT_NAOENCONTRADA = "NAO ENCONTRADA."
CMD_OUT_ENCONTRADA = "ENCONTRADA."
CMD_IN_TERMINADO = "TCHAU\n"
CMD_IN_TERMINADO2 = "TCHAU"
CMD_IN_TEXTO = "TEXTO\n"
CMD_IN_FIM = "FIM.\n"
CMD_OUT_GUARDADO = "GUARDADO."


# #### VARS GLOBAIS ####
acumula_linhas = 0
acumula_assoc = 0
acumula_rotacoes_simples = 0


# #### FUNCOES ####
class Elemento:
    def __init__(self, input_palavra, input_ocorrencias):
        self.palavra = input_palavra
        self.ocorrencias = []
        self.ocorrencias.append(input_ocorrencias)

    def add_ocorrencia(self, count):
        if not count == self.ocorrencias[-1]:
            self.ocorrencias.append(count)


class No:
    def __init__(self, input_elemento=None, input_esquerda=None, input_direita=None):
        self.elemento = input_elemento
        self.esquerda = input_esquerda
        self.direita = input_direita
        self.altura = 1


class ArvoreAVL:
    def __init__(self, input_raiz=None):
        self.raiz = input_raiz
```

```python
    def rotacao_esq(self, input_no_k1):   # Faz rotacao simples com filho k2 a direita, E <- D
        # ### FUNCAO ### Rotacao Simples Esquerda (Direcao <-)
        global acumula_rotacoes_simples
        acumula_rotacoes_simples = acumula_rotacoes_simples + 1
        no_k2 = input_no_k1.direita
        no_k3 = no_k2.esquerda
        no_k2.esquerda = input_no_k1
        input_no_k1.direita = no_k3
        input_no_k1.altura = 1 + max(self.get_altura(input_no_k1.esquerda),
                                     self.get_altura(input_no_k1.direita))  # Cumprir ordem para obter altura coerente
        no_k2.altura = 1 + max(self.get_altura(no_k2.esquerda),
                               self.get_altura(no_k2.direita))   # Altura anterior + 1 (para incluir o no atual)
        return no_k2  # Nova raiz da sub-arvore

    def rotacao_dir(self, input_no_k1):   # Faz rotacao simples com filho k2 a esquerda, E -> D
        # ### FUNCAO ### Rotacao Simples Direita ( Direcao ->)
        global acumula_rotacoes_simples
        acumula_rotacoes_simples = acumula_rotacoes_simples + 1
        no_k2 = input_no_k1.esquerda
        no_k3 = no_k2.direita
        no_k2.direita = input_no_k1
        input_no_k1.esquerda = no_k3
        input_no_k1.altura = 1 + max(self.get_altura(input_no_k1.esquerda), self.get_altura(input_no_k1.direita))
        no_k2.altura = 1 + max(self.get_altura(no_k2.esquerda), self.get_altura(no_k2.direita))
        return no_k2

    def rotacao_esq_dir(self, input_no_k1):   # Faz rotacao com filho k2 a direita | Faz rotacao com filho k2 a esquerda
        # ### FUNCAO ### Rotacao Dupla Esquerda-Direita ( Direcao <- e ->)
        input_no_k1.esquerda = self.rotacao_esq(input_no_k1.esquerda)
        return self.rotacao_dir(input_no_k1)

    def rotacao_dir_esq(self, input_no_k1):   # Faz rotacao com filho k2 a esquerda | Faz rotacao com filho k2 a direita
        # ### FUNCAO ### Rotacao Dupla Direita-Esquerda ( Direcao -> e <-)
        input_no_k1.direita = self.rotacao_dir(input_no_k1.direita)
        return self.rotacao_esq(input_no_k1)

    def procura_palavra(self, input_palavra):
        # ### FUNCAO ### Procura Palavra na Arvore e return esse elemento, se nao existe retorna: None
        no = self.raiz
        while no is not None:
            if compara_str(input_palavra, no.elemento.palavra) == 0:
                return no.elemento
            elif compara_str(input_palavra, no.elemento.palavra) == 1:
                no = no.direita
            else:
                no = no.esquerda
        return None
```

```python
    def inserir_elemento(self, input_raiz, input_elemento):     # input_raiz -> raiz ou no da sub-arvore
        # ### FUNCAO ### Inserir Elementos na Arvore AVP, recursivamente, ate chegar as folhas nulas, inserindo-o
        if input_raiz is None:                          # Insere o elemento
            novo_no = No(input_elemento)
            return novo_no
        elif compara_str(input_raiz.elemento.palavra, input_elemento.palavra) == 1:    # Se a str 1 (no da arvore) e maior
            input_raiz.esquerda = self.inserir_elemento(input_raiz.esquerda, input_elemento)
        else:                                                               # Se a str 2 (novo no) e maior
            input_raiz.direita = self.inserir_elemento(input_raiz.direita, input_elemento)

        input_raiz.altura = 1 + max(self.get_altura(input_raiz.esquerda),
                                    self.get_altura(input_raiz.direita))   # Altura anterior + 1 (para incluir o no atual)

        # ----------------------- Verificar Equilibrio, fazer rotacoes para corrigir ----------------------
        equilibrio = self.get_equilibrio(input_raiz)

        if equilibrio > 1:       # Lado Esquerdo MAIOR que o Direito (na sub-arvore do no atual: input_raiz)
            if compara_str(input_raiz.esquerda.elemento.palavra,
                           input_elemento.palavra) == 1:  # str 1 (Palavra no->esquerdo) MAIOR que str 2 (Palavra nova inserida)
                # Se Caminho entre Avo-Pai-Filho -> Esq-Esq
                return self.rotacao_dir(input_raiz)
            else:        # str 2 (Palavra nova inserida) MAIOR que str 1 (Palavra no->esquerdo)
                # Se Caminho entre Avo-Pai-Filho -> Esq-Dir
                return self.rotacao_esq_dir(input_raiz)
        if equilibrio < -1:                # Lado Direito MAIOR que o Esquerdo (na sub-arvore do no atual: input_raiz)
            if compara_str(input_raiz.direita.elemento.palavra,
                           input_elemento.palavra) == 2:  # str 1 (Palavra no->esquerdo) MAIOR que str 2 (Palavra nova inserida)
                # Se Caminho entre Avo-Pai-Filho -> Dir-Dir
                return self.rotacao_esq(input_raiz)
            else:      # str 2 (Palavra nova inserida) MAIOR que str 1 (Palavra no->esquerdo)
                # Se Caminho entre Avo-Pai-Filho -> Dir-Esq
                return self.rotacao_dir_esq(input_raiz)

        return input_raiz      # Sem rotacoes

    def get_altura(self, input_no):
        # ### FUNCAO ### Get Altura guardado no atributo do no, ou 0 se o no e nulo
        if input_no is None:
            return 0
        return input_no.altura

    def get_equilibrio(self, input_no):
        # ### FUNCAO ### Get Equilibrio atraves da altura guardado no atributo do no, ou 0 se o no e nulo
        if input_no is None:
            return 0
        return self.get_altura(input_no.esquerda) - self.get_altura(input_no.direita)  # Equilibrio da sub-arvore


def compara_str(str1, str2):
    # ### FUNCAO ### str1 maior: return 1, str2 maior: return 2, iguais: return 0
    if str1 > str2:          # Str1 Maior
        return 1
    elif str1 < str2:         # Str2 Maior
        return 2
    else:                    # Iguais
        return 0
```

```python
def input_texto(arvore_avl):
    # ### FUNCAO ### Le e manipula o texto do stdin ate CMD_IN_FIM
    count = 0
    for linha in sys.stdin:
        if count == 0 and linha == "":
            sys.exit("Erro - Sem Texto para input")
        if linha == CMD_IN_FIM:
            break
        palavra = ""
        for ch in linha:
            if ch == '\n':
                if len(palavra) > 0:
                    palavra = palavra.lower()
                    elemento = arvore_avl.procura_palavra(palavra)
                    if elemento is not None:
                        elemento.add_ocorrencia(count)
                    else:
                        elemento = Elemento(palavra, count)
                        arvore_avl.raiz = arvore_avl.inserir_elemento(arvore_avl.raiz, elemento)
                palavra = ""
            elif ch == ' ' or ch == '.' or ch == ',' or ch == ';' or ch == '(' or ch == ')':
                if len(palavra) > 0:
                    palavra = palavra.lower()
                    elemento = arvore_avl.procura_palavra(palavra)
                    if elemento is not None:
                        elemento.add_ocorrencia(count)
                    else:
                        elemento = Elemento(palavra, count)
                        arvore_avl.raiz = arvore_avl.inserir_elemento(arvore_avl.raiz, elemento)
                elemento = arvore_avl.procura_palavra(ch)
                if elemento is not None:
                    elemento.add_ocorrencia(count)
                else:
                    elemento = Elemento(ch, count)
                    arvore_avl.raiz = arvore_avl.inserir_elemento(arvore_avl.raiz, elemento)
                palavra = ""
            else:
                palavra = palavra + ch
        count += 1
    print(CMD_OUT_GUARDADO)
    return 0
```

```
206 def input_cmd(arvore_avl):
207     # ### FUNCAO ### Le, executa e escreve no stdout os comandos no stdin, ate CMD_IN_TERMINADO
208     fator_assoc = 0
209     fator_linhas = 0
210     inicio_assoc = 0
211     inicio_linhas = 0
212     global acumula_assoc
213     global acumula_linhas
214     for linha in sys.stdin:
215         if linha == CMD_IN_TERMINADO2:
216             break
217         elif linha == CMD_IN_TERMINADO:
218             break
219         elif linha == "":
220             break
221         elif (CMD_IN_LINHAS in linha) and (linha.index(CMD_IN_LINHAS) == 0):
222             if fator_assoc == 1:
223                 acumula_assoc = acumula_assoc + (time.time() - inicio_assoc)
224                 fator_assoc = -1
225             if fator_linhas == 0:
226                 inicio_linhas = time.time()
227                 fator_linhas = 1
228             palavra = linha[len(CMD_IN_LINHAS)+1:len(linha)-1]
229             palavra = palavra.lower()
230             elemento = arvore_avl.procura_palavra(palavra)
231             if elemento is not None:
232                 print(print_array(elemento.ocorrencias))
233             else:
234                 print(CMD_OUT_NULO)
235         elif (CMD_IN_ASSOC in linha) and (linha.index(CMD_IN_ASSOC) == 0):
236             if fator_linhas == 1:
237                 acumula_linhas = acumula_linhas + (time.time() - inicio_linhas)
238                 fator_linhas = -1
239             if fator_assoc == 0:
240                 inicio_assoc = time.time()
241                 fator_assoc = 1
242             palavras = linha.split(' ')
243             palavras[2] = (palavras[2])[:len(palavras[2])-1]
244             palavras[1] = palavras[1].lower()
245 
246             elemento = arvore_avl.procura_palavra(palavras[1])
247             if elemento is not None:
248                 if not (pesquisa_binaria(elemento.ocorrencias, int(palavras[2])) == -1):
249                     print(CMD_OUT_ENCONTRADA)
250                 else:
251                     print(CMD_OUT_NAOENCONTRADA)
252             else:
253                 print(CMD_OUT_NAOENCONTRADA)
254         else:
255             sys.exit("Erro - Interpretacao dos comandos pos-texto")
256     if fator_linhas == 1:
257         acumula_linhas = acumula_linhas + (time.time() - inicio_linhas)
258     if fator_assoc == 1:
259         acumula_assoc = acumula_assoc + (time.time() - inicio_assoc)
260     return 0
```

```
def pesquisa_binaria(array, valor):
    # ### FUNCAO ### Pesquisa Binaria Classica num Array/Lista, input array e valor, return indice ou -1 se nao existir
    inicio = 0
    fim = len(array)-1
    if fim == -1:
        return -1
    while inicio <= fim:
        meio = inicio + (fim - inicio) // 2    # Divisao Real, Arredonda para baixo
        if array[meio] == valor:           # Valor esta no meio
            return meio
        elif array[meio] < valor:           # Se valor e maior que o meio, ignora metade inferior
            inicio = meio + 1
        else:                              # Se for menor que o meio, ignora metade superior
            fim = meio - 1
    return -1                              # Nao existe


def print_array(array):
    # ### FUNCAO ### Transforma os dados num array numa string com espacos
    string = ""
    for num in array:
        string = string + " " + str(num)
    return string[1:]


def main():
    # ### FUNCAO ### Funcao Principal
    textos_relatorio = ["A", "B", "C", "D"]
    global acumula_linhas
    global acumula_assoc
    global acumula_rotacoes_simples
    for n_texto in textos_relatorio:
        print("# # # # # # TEXTO " + n_texto + " # # # # # #")
        acumula_texto = 0
        acumula_linhas = 0
        acumula_assoc = 0
        acumula_rotacoes_simples = 0
        for i in range(20):
            arvore_avl = ArvoreAVL()
            while msvcrt.kbhit():  # Clean stdin Windows
                msvcrt.getch()
            if n_texto == "A" or n_texto == "D":
                nome_fich = "./StdinsCalculaTempos/StdinTexto" + n_texto + "RelatorioN" + str(i) + ".txt"
            else:
                nome_fich = "./StdinsCalculaTempos/StdinTexto" + n_texto + "RelatorioN" + str(0) + ".txt"
            print("########" + nome_fich + "########")
            my_file = open(nome_fich, "r")
            my_stdin = my_file.read()
            my_file.close()
            sys.stdin = StringIO(my_stdin)
            if sys.stdin.readline() == CMD_IN_TEXTO:
                start_texto = time.time()
                input_texto(arvore_avl)
                end_texto = time.time()
                tempo_texto = end_texto - start_texto
            else:
                sys.exit("Erro - Sem Comando Incial: " + CMD_IN_TEXTO)
            input_cmd(arvore_avl)
            acumula_texto = acumula_texto + tempo_texto
            print("########################################\n")
        acumula_texto = acumula_texto / 20.0
        acumula_linhas = acumula_linhas / 20.0
        acumula_assoc = acumula_assoc / 20.0
        acumula_rotacoes_simples = acumula_rotacoes_simples // 20   # Porque contou 20 vezes
        print("* * * * * Tempo Medio em Segundos Input TEXTO = " + str(acumula_texto) + " * * * * *")
        print("* * * * * Rotacoes Simples - Input TEXTO = " + str(acumula_rotacoes_simples) + " * * * * *")
        print("* * * * * Tempo Medio em Segundos Input CMDs - LINHAS = " + str(acumula_linhas) + " * * * * *")
        print("* * * * * Tempo Medio em Segundos Input CMDs - ASSOC = " + str(acumula_assoc) + " * * * * *")
        print("# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #\n\n")
    return 0


if __name__ == '__main__':
    # ### START ###
    main()
```

**A2** – Parte 1/3 (As funções em falta são iguais ao A1)

```python
# #### NOTAS #### V1 Abordagem Top-Down, Nao-Recursiva, Percorre 1 vez | Possui Pesquisa Binaria em Arrays
# Arvore VP -> Raiz -> No (Contem Elemento/Dados) -> liga-se a outros Nos
# *** VERSAO RELATORIO ***


# #### BIBLIOTECAS ####
import sys
import time
import msvcrt
from io import StringIO


# #### CONSTANTES ####
CMD_IN_LINHAS = "LINHAS"
CMD_OUT_NULO = "-1"
CMD_IN_ASSOC = "ASSOC"
CMD_OUT_NAOENCONTRADA = "NAO ENCONTRADA."
CMD_OUT_ENCONTRADA = "ENCONTRADA."
CMD_IN_TERMINADO = "TCHAU\n"
CMD_IN_TERMINADO2 = "TCHAU"
CMD_IN_TEXTO = "TEXTO\n"
CMD_IN_FIM = "FIM.\n"
CMD_OUT_GUARDADO = "GUARDADO."
COR_VERMELHA = "v"
COR_PRETA = "p"
LADO_ESQ = "e"
LADO_DIR = "d"


# #### VARS GLOBAIS ####
acumula_linhas = 0
acumula_assoc = 0
acumula_rotacoes_simples = 0


# #### FUNCOES ####
class Elemento:                         # Elemento = Dados = Palavra + Ocorrencias
    def __init__(self, input_palavra, input_ocorrencias):
        self.palavra = input_palavra
        self.ocorrencias = []
        self.ocorrencias.append(input_ocorrencias)

    def add_ocorrencia(self, count):     # Funcao Adicionar Ocorrencias
        if not count == self.ocorrencias[-1]:
            self.ocorrencias.append(count)


class No:           # NO | Arvore VP -> Raiz -> No (Contem Elemento/Dados) -> liga-se a outros Nos
    def __init__(self, input_elemento=None, input_esquerda=None, input_direita=None, input_cor=COR_VERMELHA):
        self.elemento = input_elemento
        self.esquerda = input_esquerda
        self.direita = input_direita
        self.cor = input_cor


class ArvoreVP:           # Arvore VP
    def __init__(self, input_raiz=None):
        self.raiz = input_raiz

    def procura_palavra(self, input_palavra):
        # ### FUNCAO ### Procura Palavra na Arvore e return esse elemento, se nao existe retorna: None
        no = self.raiz
        while no is not None:
            if compara_str(input_palavra, no.elemento.palavra) == 0:
                return no.elemento
            elif compara_str(input_palavra, no.elemento.palavra) == 1:
                no = no.direita
            else:
                no = no.esquerda
        return None
```

```python
    def rotacao_simples_esquerda(self, input_no):
        # ### FUNCAO ### Rotacao Simples Esquerda (Direcao <-) | last == DIR
        global acumula_rotacoes_simples
        acumula_rotacoes_simples = acumula_rotacoes_simples + 1
        temp_no = input_no.direita
        input_no.direita = temp_no.esquerda
        temp_no.esquerda = input_no
        input_no.cor = COR_VERMELHA
        temp_no.cor = COR_PRETA
        return temp_no

    def rotacao_simples_direita(self, input_no):
        # ### FUNCAO ### Rotacao Simples Direita ( Direcao ->) | last == ESQ
        global acumula_rotacoes_simples
        acumula_rotacoes_simples = acumula_rotacoes_simples + 1
        temp_no = input_no.esquerda
        input_no.esquerda = temp_no.direita
        temp_no.direita = input_no
        input_no.cor = COR_VERMELHA
        temp_no.cor = COR_PRETA
        return temp_no

    def rotacao_dupla_direita_esquerda(self, input_no):
        # ### FUNCAO ### Rotacao Dupla Direita-Esquerda ( Direcao -> e <-) | last == DIR
        input_no.direita = self.rotacao_simples_direita(input_no.direita)   # last = ESQ
        return self.rotacao_simples_esquerda(input_no)          # last = DIR

    def rotacao_dupla_esquerda_direita(self, input_no):
        # ### FUNCAO ### Rotacao Dupla Esquerda-Direita ( Direcao <- e ->) | last == ESQ
        input_no.esquerda = self.rotacao_simples_esquerda(input_no.esquerda)    # last = DIR
        return self.rotacao_simples_direita(input_no)        # last = ESQ

    def inserir_elemento(self, input_elemento):
        # ### FUNCAO ### Inserir Elementos na Arvore VP
        if self.raiz is None:             # Primeiro Elemento da Arvore (Raiz)
            self.raiz = No(input_elemento)
        else:                             # Ja Ha Elementos na Arvore
            # Ponteiros: Familia-Nos: bisavo -> avo -> pai -> atual/filho
            no_avo = no_pai = None
            no_temp = No()                # No Temporario - Criacao
            no_bisavo = no_temp
            no_bisavo.direita = no_atual = self.raiz

            orientacao_atual = LADO_ESQ     # Entre Pai -> Filho
            orientacao_anterior = LADO_ESQ       # Entre Avo -> Pai
            orientacao_anterior2 = None            # Entre Bisavo -> Avo

```

```
        while True:                  # Navegando na Arvore
            if no_atual is None:            # Inserindo no Final da Arvore
                no_atual = No(input_elemento)
                if orientacao_atual == LADO_ESQ:
                    no_pai.esquerda = no_atual
                else:
                    no_pai.direita = no_atual
            else:
                if (no_atual.esquerda is not None and no_atual.direita is not None) and \
                        no_atual.esquerda.cor == COR_VERMELHA and no_atual.direita.cor == COR_VERMELHA:
                    # Se ambos os filhos do no atual sao vermelhos -> trocar para preto nos filhos, no atual fica vermelho
                    no_atual.cor = COR_VERMELHA
                    no_atual.esquerda.cor = COR_PRETA
                    no_atual.direita.cor = COR_PRETA

            # ------------------------------------------------------------------------
            # Resolver Violacao da Regra: Pai e Filho serem ambos Vermelho
            if (no_atual is not None and no_pai is not None) and no_atual.cor == COR_VERMELHA and \
                    no_pai.cor == COR_VERMELHA:
                # Se e filho sao vermelhos -> fazer rotacoes
                if no_bisavo.esquerda == no_avo:
                    # Indica a orientacao entre bisavo e avo, para guardar na direcao certa o return das rotacoes
                    orientacao_anterior2 = LADO_ESQ
                else:
                    orientacao_anterior2 = LADO_DIR

                if orientacao_anterior == LADO_ESQ:           # Orientacao Entre Avo e Pai
                    if orientacao_atual == LADO_ESQ:                    # Orientacao Entre Pai e Atual/Filho
                        # Se entre Avo-Pai-Filho/Atual -> Esq-Esq
                        if orientacao_anterior2 == LADO_ESQ:
                            no_bisavo.esquerda = self.rotacao_simples_direita(no_avo)
                        else:
                            no_bisavo.direita = self.rotacao_simples_direita(no_avo)
                    else:
                        # Se entre Avo-Pai-Filho/Atual -> Esq-Dir
                        if orientacao_anterior2 == LADO_ESQ:
                            no_bisavo.esquerda = self.rotacao_dupla_esquerda_direita(no_avo)
                        else:
                            no_bisavo.direita = self.rotacao_dupla_esquerda_direita(no_avo)
                else:
                    if orientacao_atual == LADO_DIR:           # Orientacao Entre Pai e Atual/Filho
                        # Se entre Avo-Pai-Filho/Atual -> Dir-Dir
                        if orientacao_anterior2 == LADO_ESQ:
                            no_bisavo.esquerda = self.rotacao_simples_esquerda(no_avo)
                        else:
                            no_bisavo.direita = self.rotacao_simples_esquerda(no_avo)
                    else:
                        # Se entre Avo-Pai-Filho/Atual -> Dir-Esq
                        if orientacao_anterior2 == LADO_ESQ:
                            no_bisavo.esquerda = self.rotacao_dupla_direita_esquerda(no_avo)
                        else:
                            no_bisavo.direita = self.rotacao_dupla_direita_esquerda(no_avo)
            # ------------------------------------------------------------------------

            if no_atual.elemento.palavra == input_elemento.palavra:
                break   # Condicao de Paragem do Ciclo

            # --------------------- Preparar para proxima iteracao do ciclo -----------------------
            if no_avo is not None:                  # manter no bisavo == raiz ate ter altura para tal
                no_bisavo = no_avo
            no_avo = no_pai
            no_pai = no_atual

            # Encontrar direcao do caminho a percorrer... Esq. Vs Dir.
            orientacao_anterior = orientacao_atual
            if compara_str(no_atual.elemento.palavra, input_elemento.palavra) == 1:   # str-no maior que str-input
                orientacao_atual = LADO_ESQ
                no_atual = no_atual.esquerda
            else:                       # str-input maior que str-no
                orientacao_atual = LADO_DIR
                no_atual = no_atual.direita

        self.raiz = no_temp.direita             # No fim do while, dentro do else, nova arvore na raiz
    self.raiz.cor = COR_PRETA           # colocar a cor da raiz de preto (regra), pode ter mudado nas rotacoes

```

**B** – Parte 1/4 (As funções em falta são iguais ao A1)

```python
# #### NOTAS #### V1 Abordagem: Percorre ate as folhas, insere, e depois traz-lo para a raz (splaying)
# Possui Pesquisa Binaria em Arrays
# Arvore Splay
# *** VERSAO RELATORIO ***


# #### BIBLIOTECAS ####
import sys
import time
import msvcrt
from io import StringIO


# #### CONSTANTES ####
CMD_IN_LINHAS = "LINHAS"
CMD_OUT_NULO = "-1"
CMD_IN_ASSOC = "ASSOC"
CMD_OUT_NAOENCONTRADA = "NAO ENCONTRADA."
CMD_OUT_ENCONTRADA = "ENCONTRADA."
CMD_IN_TERMINADO = "TCHAU\n"
CMD_IN_TERMINADO2 = "TCHAU"
CMD_IN_TEXTO = "TEXTO\n"
CMD_IN_FIM = "FIM.\n"
CMD_OUT_GUARDADO = "GUARDADO."


# #### VARS GLOBAIS ####
acumula_linhas = 0
acumula_assoc = 0
acumula_rotacoes_simples = 0
fator_rotacoes_input = 0


# #### FUNCOES ####
class Elemento:
    def __init__(self, input_palavra, input_ocorrencias):
        self.palavra = input_palavra
        self.ocorrencias = []
        self.ocorrencias.append(input_ocorrencias)

    def add_ocorrencia(self, count):
        if not count == self.ocorrencias[-1]:
            self.ocorrencias.append(count)


class No:
    def __init__(self, input_elemento=None, input_esquerda=None, input_direita=None, input_pai=None):
        self.elemento = input_elemento
        self.esquerda = input_esquerda
        self.direita = input_direita
        self.pai = input_pai


class ArvoreSplay:
    def __init__(self, input_raiz=None):
        self.raiz = input_raiz

```

```
58      def procura_palavra(self, input_palavra):
59          # ### FUNCAO ### Procura Palavra na Arvore e return esse elemento, se nao existe retorna: None
60          no = self.raiz
61          while no is not None:
62              if compara_str(input_palavra, no.elemento.palavra) == 0:  # Palavra Encontrada
63                  self.splaying(no)     # Traz-lo para a raiz
64                  return no.elemento
65              elif compara_str(input_palavra, no.elemento.palavra) == 1:  # Pesquisa, input MAIOR que no
66                  no = no.direita
67              else:
68                  no = no.esquerda
69          return None         # Palavra nao encontrada
70
71      def rotacao_esquerda(self, no_x):
72          # ### FUNCAO ### Rotacao Simples Esquerda (Direcao ->)
73          global acumula_rotacoes_simples
74          global fator_rotacoes_input
75          if fator_rotacoes_input == 1:
76              acumula_rotacoes_simples = acumula_rotacoes_simples + 1
77          no_y = no_x.direita
78          no_x.direita = no_y.esquerda
79          no_y.pai = no_x.pai
80          if no_y.esquerda is not None:
81              no_y.esquerda.pai = no_x    # no_x PAI do no_y-esquerdo
82          if no_y.pai is None:
83              self.raiz = no_y     # Se x era raiz, agora y e RAIZ
84          else:
85              if no_x == no_x.pai.esquerda:     # Se X era filho lado ESQ, set pai->y
86                  no_x.pai.esquerda = no_y
87              else:                               # Se X era filho lado DIR, set pai->y
88                  no_x.pai.direita = no_y
89          no_y.esquerda = no_x
90          no_x.pai = no_y        # X filho de y
91
92      def rotacao_direita(self, no_x):
93          # ### FUNCAO ### Rotacao Simples Direita (Direcao <-)
94          global acumula_rotacoes_simples
95          global fator_rotacoes_input
96          if fator_rotacoes_input == 1:
97              acumula_rotacoes_simples = acumula_rotacoes_simples + 1
98          no_y = no_x.esquerda
99          no_x.esquerda = no_y.direita
100         no_y.pai = no_x.pai
101         if no_y.direita is not None:
102             no_y.direita.pai = no_x   # no_x e PAI do no_y-direito
103         if no_y.pai is None:
104             self.raiz = no_y     # Se x era raiz, agora y e RAIZ
105         else:
106             if no_x == no_x.pai.direita:# Se X era filho lado DIR, set pai->y
107                 no_x.pai.direita = no_y
108             else:          # Se X era filho lado ESQ, set pai->y
109                 no_x.pai.esquerda = no_y
110         no_y.direita = no_x
111         no_x.pai = no_y             # X filho de y
```

**B** – Parte 3/4

```
113     def splaying(self, no_atual):
114         # ### FUNCAO ### Trazer o no_atual para a raiz atraves de rotacoes
115         while no_atual.pai is not None:    # Enquanto o no_atual nao e a raiz...
116             if no_atual.pai == self.raiz:      # No_atual filho do no-raiz
117                 if no_atual == self.raiz.esquerda:
118                     # no_atual FILHO ESQ | ZIG - Rotacao Simples DIR (Direcao <-)
119                     self.rotacao_direita(no_atual.pai)
120                 else:
121                     # no_atual FILHO DIR | ZIG - Rotacao Simples ESQ (Direcao ->)
122                     self.rotacao_esquerda(no_atual.pai)
123             else:               # No_atual ainda nao e filho do no-raiz
124                 no_pai = no_atual.pai         # Ponteiros: Familia-Nos: avo -> pai -> atual/filho
125                 no_avo = no_atual.pai.pai
126                 if no_avo.esquerda == no_pai:
127                     if no_pai.esquerda == no_atual:
128                         # avo-pai-no_atual -> ESQ - ESQ | ZIG ZIG - Rotacao Dupla DIR (Direcao <- <-)
129                         self.rotacao_direita(no_avo)  # Porque no_avo ou no_pai? ver slides Prof, g=avo e p=pai e x=no_atual
130                         self.rotacao_direita(no_pai)
131                     if no_pai.direita == no_atual:
132                         # avo-pai-no_atual -> ESQ - DIR | ZIG ZAG - Rotacao ESQ DIR (Direcao -> <-)
133                         self.rotacao_esquerda(no_pai)
134                         self.rotacao_direita(no_avo)
135                 if no_avo.direita == no_pai:
136                     if no_pai.esquerda == no_atual:
137                         # avo-pai-no_atual -> DIR - ESQ | ZIG ZAG - Rotacao DIR ESQ (Direcao <- ->)
138                         self.rotacao_direita(no_pai)
139                         self.rotacao_esquerda(no_avo)
140                     if no_pai.direita == no_atual:
141                         # avo-pai-no_atual -> DIR - DIR | ZIG ZIG - Rotacao Dupla ESQ (Direcao -> ->)
142                         self.rotacao_esquerda(no_avo)
143                         self.rotacao_esquerda(no_pai)
144 
145     def inserir_elemento(self, input_elemento):
146         # ### FUNCAO ### Inserir Elementos na Arvore Splay
147         novo_no = No(input_elemento)
148         if self.raiz is None:    # Se primeiro elemento na arvore
149             self.raiz = novo_no
150             return
151         no_pai = temp_no = self.raiz
152         while temp_no is not None:     # Encontra final da arvore (folha) a ser colocado novo_no
153             no_pai = temp_no           # Encontra Futuro No-Pai
154             if compara_str(temp_no.elemento.palavra, input_elemento.palavra) == 1:   # str1 (No da Arvore) MAIOR
155                 temp_no = temp_no.esquerda
156             else:                               # str2 (No Novo) MAIOR
157                 temp_no = temp_no.direita
158         novo_no.pai = no_pai
159         if compara_str(no_pai.elemento.palavra, input_elemento.palavra) == 1:    # str1 (No da Arvore) MAIOR
160             no_pai.esquerda = novo_no
161         else:                           # str2 (No Novo) MAIOR
162             no_pai.direita = novo_no
163         self.splaying(novo_no)           # Traz-lo para a raiz
164 
```

**B** – Parte 4/4

```
301 def main():
302     # ### FUNCAO ### Funcao Principal
303     textos_relatorio = ["A", "B", "C", "D"]
304     global acumula_linhas
305     global acumula_assoc
306     global acumula_rotacoes_simples
307     global fator_rotacoes_input
308     for n_texto in textos_relatorio:
309         print("# # # # # # TEXTO " + n_texto + " # # # # # #")
310         acumula_texto = 0
311         acumula_linhas = 0
312         acumula_assoc = 0
313         acumula_rotacoes_simples = 0
314         for i in range(20):
315             arvore_splay = ArvoreSplay()
316             while msvcrt.kbhit():  # Clean stdin Windows
317                 msvcrt.getch()
318             if n_texto == "A" or n_texto == "D":
319                 nome_fich = "./StdinsCalculaTempos/StdinTexto" + n_texto + "RelatorioN" + str(i) + ".txt"
320             else:
321                 nome_fich = "./StdinsCalculaTempos/StdinTexto" + n_texto + "RelatorioN" + str(0) + ".txt"
322             print("########" + nome_fich + "########")
323             my_file = open(nome_fich, "r")
324             my_stdin = my_file.read()
325             my_file.close()
326             sys.stdin = StringIO(my_stdin)
327             if sys.stdin.readline() == CMD_IN_TEXTO:
328                 start_texto = time.time()
329                 fator_rotacoes_input = 1
330                 input_texto(arvore_splay)
331                 fator_rotacoes_input = 0
332                 end_texto = time.time()
333                 tempo_texto = end_texto - start_texto
334             else:
335                 sys.exit("Erro - Sem Comando Incial: " + CMD_IN_TEXTO)
336             input_cmd(arvore_splay)
337             acumula_texto = acumula_texto + tempo_texto
338             print("########################################\n")
339         acumula_texto = acumula_texto / 20.0
340         acumula_linhas = acumula_linhas / 20.0
341         acumula_assoc = acumula_assoc / 20.0
342         acumula_rotacoes_simples = acumula_rotacoes_simples // 20  # Porque contou 20 vezes
343         print("* * * * * Tempo Medio em Segundos Input TEXTO = " + str(acumula_texto) + " * * * * *")
344         print("* * * * * Rotacoes Simples - Input TEXTO = " + str(acumula_rotacoes_simples) + " * * * * *")
345         print("* * * * * Tempo Medio em Segundos Input CMDs - LINHAS = " + str(acumula_linhas) + " * * * * *")
346         print("* * * * * Tempo Medio em Segundos Input CMDs - ASSOC = " + str(acumula_assoc) + " * * * * *")
347         print("# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #\n\n")
348     return 0
```

**[Extra]**
**Gerador de Ficheiros com o texto e comandos para "stdin" do trabalho**
Parte 1/1
(Código adaptado de A0)

```python
# Gera Stdins dos diversos textos com reaproveitamente do codigo de A0
# Le Pasta Inputs -> Gera e guarda em Outputs

# #### FUNCOES ####
def main():
    # ### FUNCAO ### Funcao Principal
    textos_relatorio = ["A", "B", "C", "D"]
    for n_texto in textos_relatorio:
        print("###### TEXTO " + n_texto + " ######")
        array_palavras = []  # Varias Palavras
        array_ocorrencias = [[]]  # Ocorrencias/N linhas - String -> Ex.: "1 2 3"
        while msvcrt.kbhit():  # Clean stdin Windows
            msvcrt.getch()
        nome_fich = "./Inputs/StdinTexto" + n_texto + ".txt"
        my_file = open(nome_fich, "r")
        my_stdin = my_file.read()
        my_file.close()
        sys.stdin = StringIO(my_stdin)
        if sys.stdin.readline() == CMD_IN_TEXTO:
            array_palavras, array_ocorrencias = input_texto(array_palavras, array_ocorrencias)
        else:
            sys.exit("Erro - Sem Comando Incial: " + CMD_IN_TEXTO)
        if n_texto == "A":
            for i in range(20):
                nome_fich = "./Outputs/StdinTexto" + n_texto + "RelatorioN" + str(i) + ".txt"
                my_file = open(nome_fich, "w")
                for line in my_stdin.splitlines():
                    if line == "FIM.":
                        my_file.write(line + "\n")
                        break
                    my_file.write(line + "\n")
                for j in range(50):
                    n = random.randint(0, len(array_palavras)-1)
                    my_file.write("LINHAS " + array_palavras[n] + "\n")
                fator = 0
                for j in range(50):
                    n = random.randint(0, len(array_palavras)-1)
                    if fator == 0:  # facil
                        fator = 1
                        ocorrencia = array_ocorrencias[n][(len(array_ocorrencias[n])-1)//2]
                    elif fator == 1:# dificil
                        fator = 2
                        ocorrencia = array_ocorrencias[n][0]
                    else:   # nao existe
                        fator = 0
                        ocorrencia = array_ocorrencias[n][-1] + 1
                    my_file.write("ASSOC " + array_palavras[n] + " " + str(ocorrencia) + "\n")
                my_file.write("TCHAU\n\n\n")
                my_file.close()
        elif n_texto == "B" or n_texto == "C":
            nome_fich = "./Outputs/StdinTexto" + n_texto + "RelatorioN" + str(0) + ".txt"
            my_file = open(nome_fich, "w")
            for line in my_stdin.splitlines():
                if line == "FIM.":
                    my_file.write(line + "\n")
                    break
                my_file.write(line + "\n")
            my_file.write("TCHAU\n\n\n")
            my_file.close()
        elif n_texto == "D":
            for i in range(20):
                nome_fich = "./Outputs/StdinTexto" + n_texto + "RelatorioN" + str(i) + ".txt"
                my_file = open(nome_fich, "w")
                for line in my_stdin.splitlines():
                    if line == "FIM.":
                        my_file.write(line + "\n")
                        break
                    my_file.write(line + "\n")
                lista_palavras = []
                for j in range(10):
                    n = random.randint(0, len(array_palavras)-1)
                    while array_palavras[n] in lista_palavras:
                        n = random.randint(0, len(array_palavras)-1)
                    lista_palavras.append(array_palavras[n])
                for j in range(50):
                    n = random.randint(0, len(lista_palavras)-1)
                    my_file.write("LINHAS " + lista_palavras[n] + "\n")
                my_file.write("TCHAU\n\n\n")
                my_file.close()
        print("#####################################")
    return 0
```